

Analyse offensive de PCB

Carnet d'expérimentations

H. B.

Préface

Ceci est un projet que j'avais commencé il y a plusieurs années, mais force est de constater que je n'ai guère le temps de le poursuivre. J'en fais donc don à la communauté cyber software et hardware. Il est à prendre tel quel, avec ses fautes, ses approximations, ses faiblesses et ses défauts. Il n'y aura, a priori, pas d'update prévu, pour les parties manquantes ou tronquées.

Les exemples de reverse engineering hardware abordés dans ce carnet sont parfois complets, parfois non, mais ont le mérite d'avoir été réalisés avec les moyens du bord.

L'ouvrage, si l'on peut tout de même l'appeler ainsi, commence avec une partie théorique, afin d'expliquer certaines notions de physique indispensables dans l'analyse de PCB. S'ensuit certains chapitres plus ou moins complets, et plus ou moins intéressants.

Il est à noter que l'objectif de cet ouvrage n'était pas de toucher au software. Ainsi toutes les exploitations hardware mises en avant s'arrêtent donc au moment où le software rentre en jeu. Frustrant ? Assurément ahah.

Ainsi, je vous souhaite tout de même une bonne lecture, et n'oubliez pas : "un grand pouvoir implique de grandes responsabilités".

Table des matières

1	Introduction	9
1.1	Notions d'électronique	9
1.1.1	Les grandeurs fondamentales	9
1.1.2	Qu'est ce qu'un signal ?	11
1.1.3	Comment mesurer un signal ?	12
1.1.4	Le bruit lors d'une mesure	14
1.1.5	Les composants usuels	16
1.2	Les protocoles de communication	24
1.2.1	UART	25
1.2.2	SPI	27
1.2.3	I2C	30
1.2.4	CAN	33
1.2.5	eMMC	35
1.2.6	Autres protocoles	37
1.3	Les interfaces de diagnostique	41
1.3.1	JTAG	41
1.3.2	SWD	42
1.4	Méthodologie d'analyse	43
2	Analyse d'un keylogger hardware	45
2.1	Analyse statique du PCB	46
2.1.1	Zone 1 - CPLD 5M160ZE64C5N	46
2.1.2	Zone 2 - SoC ESP8266EX	47
2.1.3	Zone 3 - Étage d'alimentation	49
2.1.4	Zone 4 - Flash NOR W25Q128FVSG	50
2.2	Analyse dynamique du PCB	51
2.2.1	Mesures des tensions de l'étage d'alimentation	51
2.2.2	Mesures des résistances	51
2.2.3	Mesures des capacités	52
2.2.4	— Vérification de la présence du JTAG	56
2.3	Attaques	56
2.3.1	Extraction du firmware	57
2.3.2	Clonage du PCB	63
3	Analyse d'un digicode	75
3.1	Analyse statique du PCB	76
3.1.1	Zone 1 - Étage d'alimentation	77
3.1.2	Zone 2 - Traitement des signaux	77
3.2	Analyse dynamique du PCB	79
3.2.1	Détermination des fréquences d'horloges	79

3.2.2	Vérification de la présence de l'UART	81
3.2.3	Détermination des signaux d'entrées	83
3.3	Attaques	84
3.3.1	Attaques temporelles sur la puissance consommée	84
3.3.2	— Interception I2C	87
3.3.3	— Extraction du firmware	87
4	Analyse d'un coffre fort	89
4.1	Analyse statique du PCB	90
4.2	Analyse dynamique du PCB	91
4.2.1	Mesure des tensions	91
4.2.2	— Caractéristiques des diodes de redressements	92
4.2.3	— Caractéristiques des diodes Zener	92
4.2.4	Détermination des signaux d'entrées	92
4.3	Attaques	93
4.3.1	Injection de glitch sur VCC	97
5	Analyse d'une carte à puce à contact	103
5.1	Théorie	104
5.1.1	Protocoles de communication	104
5.1.2	Le système de fichiers	107
5.2	Applications	109
5.2.1	Lecture du numéro IMSI d'une carte SIM	110
5.2.2	Validation d'un code PIN	112
5.3	Attaques	114
5.3.1	Sniffing de l'IO	115
5.3.2	Injection de glitch sur CLK	116
6	Analyse d'un router	117
6.1	Analyse statique du PCB	117
6.2	Analyse dynamique du PCB	117
6.3	Attaques	117
7	Analyse d'une camera wifi	119
7.1	Analyse statique du PCB	121
7.1.1	La face arrière	122
7.1.2	La face avant	125
7.2	Analyse dynamique du PCB	128
7.2.1	Vérification de la présence de l'UART	128
7.3	Attaques	131
7.3.1	Extraction du firmware	131
7.3.2	— Interception SPI	132
7.3.3	Attaque pin2pwn	132
8	Analyse d'un téléphone	137
8.1	Analyse statique du PCB	138
8.1.1	Dumbphone	138
8.1.2	Smartphone	139
8.2	Analyse dynamique du PCB	139
8.3	Attaques	139
8.3.1	Glitch sur CLK	139

8.3.2 Extraction d'une eMMC	140
9 Analyse d'une tablette	141
9.1 Analyse statique du PCB	141
9.2 Analyse dynamique du PCB	141
9.3 Attaques	141
9.3.1 Attaque DMA	141
A Annexes Électronique	143
A.1 Les différents types de boitiers SMD	143
A.2 Les différents types de vias	145
A.3 Annexe soudure	145
A.4 Nomenclature des labels associés aux composants	145
B Annexes Bootloader	147
B.1 Fonctionnement d'un bootloader	147
B.2 Les différents bootloaders	149
B.3 U-boot bootloader	150
C Annexes Carte à puce	153
C.1 Liste des codes de status	153
C.2 Liste d'instructions connues	162
C.3 Liste d'AIDs connus	166

Chapitre 1

Introduction

Bienvenue dans cet ouvrage dédié à l'analyse de **PCB** sous l'angle de la sécurité offensive. Dans cet univers en constante évolution, la sécurité des circuits imprimés est devenue une préoccupation majeure, compte tenu de l'omniprésence des **PCB** dans les appareils et systèmes critiques. Les technologies modernes sont de plus en plus vulnérables aux attaques sophistiquées et aux tentatives de compromission malveillantes.

Ce livre explore en profondeur les aspects de la sécurité offensive liés aux **PCB** sous forme d'expérimentation sur des appareils du commerce. Il offre un aperçu complet des méthodes, des techniques et des outils utilisés pour évaluer et exploiter les vulnérabilités potentielles de ces appareils électroniques. L'exploitation de ces attaques est parfois un succès, parfois non, mais la connaissance est toujours au rendez-vous. Chaque chapitre est soigneusement conçu pour fournir des connaissances pointues et pratiques, couvrant des sujets allant de l'analyse de la conception des circuits imprimés à la recherche de failles de sécurité, en passant par l'exploitation des dispositifs électroniques.

Que vous soyez un chercheur en sécurité, un ingénieur en électronique cherchant à renforcer la robustesse de vos conceptions ou un passionné désireux de comprendre les enjeux cruciaux de la sécurité des **PCB**, cet ouvrage vous guidera dans votre quête d'expertise. Vous apprendrez à identifier les faiblesses potentielles, à exploiter les vulnérabilités et à mettre en place des stratégies de défense pour renforcer la sécurité globale des circuits imprimés.

Nous vous invitons à explorer les chapitres de cet ouvrage, à vous immerger dans les mécanismes complexes de sécurité des **PCB**, et à découvrir comment la perspective de la sécurité offensive peut renforcer la résilience et la sûreté des systèmes électroniques modernes. Que ce voyage vous inspire et vous permette de devenir un acteur éclairé de la sécurité des **PCB** dans un monde de plus en plus connecté et exposé aux risques. Bonne lecture et bonnes découvertes !

1.1 Notions d'électronique

1.1.1 Les grandeurs fondamentales

Deux notions sont particulièrement importantes dans l'analyse d'un circuit électronique : la tension et le courant. La variation de ces paramètres sont le

signe que le circuit électronique est actif.

La **tension** (ou encore le potentiel électrique) entre deux points A et B sur un circuit peut être vu comme la force nécessaire qu'ont les électrons pour passer du point A au point B. Elle se mesure toujours entre 2 points et son unité de mesure est le **Volt** (V) (en l'honneur de Monsieur Alessandro Volta). Elle est représentée par la lettre **U**. La pression de l'eau dans un tuyau est une analogie qui fonctionne assez bien pour se représenter ce qu'est la tension. .

Le **courant** (ou encore l'intensité) est vu comme un flux d'électrons. Il est également possible de reprendre l'analogie précédente, cette fois-ci en comptant le nombre de molécules d'eau qui passent à travers une certaine section du tuyau. Le courant est donc un flux d'électrons à traversant une section donnée. Son unité est l'**Ampère** (A) (en l'honneur de Monsieur André-Marie Ampère) et est représentée par la lettre **I**.

La tension est donc toujours à visualiser comme une grandeur physique circulant entre 2 points, tandis que le courant comme une grandeur physique passant à travers un composant. Ainsi, dire que "la tension à travers la résistance est ..." n'a pas de sens. Il faudrait plutôt dire que "la tension aux bornes de la résistance est ...". Il existe une relation linéaire qui permet de lier le courant à la tension que l'on appelle la loi d'**Ohm** et qui s'exprime ainsi :

$$U = RI \quad (1.1)$$

On voit alors que la tension **U** et le courant **I** sont linéairement reliés grâce au paramètre **R**. Ce paramètre traduit la **résistance** électrique, c'est-à-dire la difficulté qu'ont les électrons à circuler entre le point A et le point B. Pour reprendre l'analogie avec un flux d'eau, la résistance représenterait le diamètre du tuyau qui augmente ou réduit la pression de l'eau, et donc qui traduit la difficulté qu'ont les molécules d'eau à circuler dans le tuyau. L'unité de mesure de la résistance électrique est l'**Ohm** (Ω). Cette relation fondamentale permet de :

- déterminer la valeur de la résistance d'un fil ou d'un composant en connaissant la tension et le courant.
- déterminer la valeur de l'intensité en connaissant la résistance et la tension.
- déterminer la tension en connaissant la résistance et l'intensité.

Prenons l'exemple d'un composant de résistance inconnue. Il a été mesuré à ces bornes une tension de 5 V, et un courant de 1 mA le traverse. On peut alors déterminer que la résistance de ce composant est de 5 k Ω ($R = 5/10^{-3}$).

Une autre relation existe entre le courant et la tension. Elle permet de calculer la **puissance** électrique circulant à travers un composant. Son unité est le **Watt** (W) et est représentée par la lettre **P**. La puissance électrique s'exprime ainsi :

$$P = U \cdot I = R \cdot I^2 = \frac{U^2}{R} \quad (1.2)$$

Suivant cette relation et l'exemple précédent, on peut estimer que la puissance électrique circulant à travers la résistance de $5 \text{ k}\Omega$ est de 5 mW . On dit aussi qu'elle dissipe 5 mW sous forme de chaleur. Le lecteur ayant des bases de physique notera que l'unité d'une puissance est généralement le **Joule** et non le **Watt**. Une relation existe entre le **Watt** et le **Joule** permettant de passer de l'une à l'autre. Ces deux grandeurs sont liées par le temps de la manière suivante :

$$1\text{W} = 1\text{J/s} \quad (1.3)$$

La tension se mesure avec un **Voltmètre**, le courant avec un **Ampèrmètre** et la résistance avec un **Ohmmètre**. Cependant, il n'est pas nécessaire de disposer de trois outils différents pour effectuer ces mesures. Généralement ces trois outils sont regroupés dans un seul appareil qui s'appelle un **multimètre**, comme celui présenté sur la figure 1.1.



FIGURE 1.1 – Un multimètre.

1.1.2 Qu'est ce qu'un signal ?

Un signal est un flux d'électrons couplé à une tension dans un circuit électronique. Cette tension peut varier périodiquement, on parle alors de courant alternatif (**AC** : *Alternate Current*), ou bien rester constante et on parle alors de courant continu (**DC** : *Direct Current*). La plupart des appareils électroniques modernes fonctionnent en courant continu. Cependant, il n'est pas rare de trouver des étages d'alimentation qui opère en courant alternatif. L'étage d'alimentation permet alors de redresser le courant alternatif afin d'obtenir un courant continu à la bonne tension pour alimenter le reste du circuit électronique. L'étage d'alimentation peut également être utilisé pour abaisser une tension continue afin d'être adaptée à une partie du circuit (voir la figure 1.2 qui représente l'étage d'alimentation d'un *switch* permettant alors d'abaisser une tension d'alimentation continue

de 9 V à 3,3 V). La physique des courants alternatifs (régime transitoire, oscillations harmoniques, etc) ne sera pas étudiée ici, mais les points importants seront abordés en tant voulu et au cas par cas.

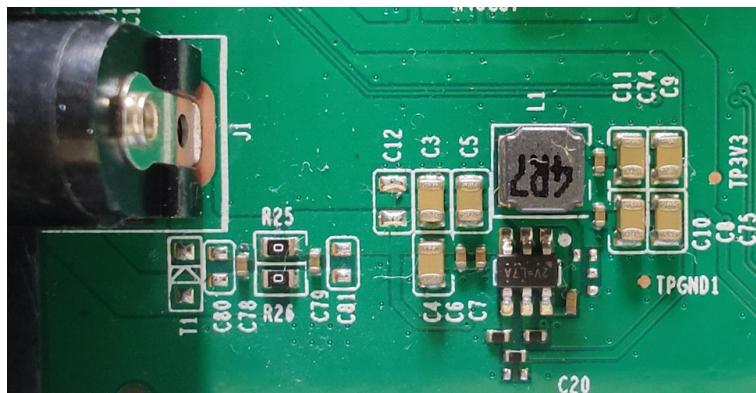


FIGURE 1.2 – Étage d'alimentation d'un *switch* qui abaisse une tension d'alimentation continue de 9 V à 3,3 V.

Bien qu'un courant continu, disons de 5 V, alimente un circuit électronique, cela ne signifie pas qu'il doive le rester au cours du temps. La transmission de l'information est une alternance non périodique de 0 logique et de 1 logique. Un tel signal est appelé signal numérique et son allure en fonction du temps est de la forme de celui présenté sur la figure 1.3.

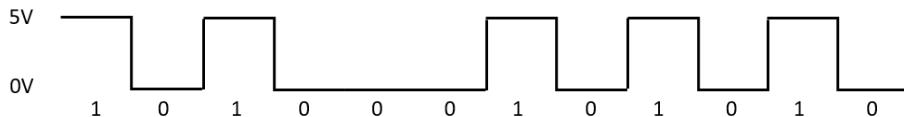


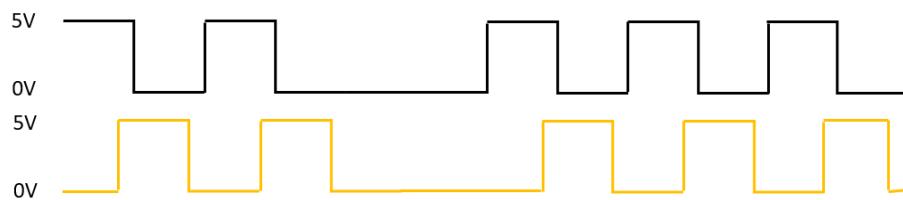
FIGURE 1.3 – Un signal typique sur 5 V.

On voit que ce signal varie entre 0 V et 5 V. Dans le cas présent, on dit que la valeur 5 V représente un 1 logique et que la valeur 0 V représente un 0 logique. Cette suite de 1 et de 0 représente l'information utile. Parfois il est nécessaire d'avoir des signaux négatifs qui évoluent entre 0 V et -5 V. Pour différencier ces signaux de ceux évoluant entre 0 V et 5 V, on introduit la notion de **polarité**. La polarité définit donc le signe du signal, peu importe que celui-ci soit continu ou alternatif.

A cette notion de polarité, doit s'ajouter une notion de **phase**. Deux signaux sont dits en phase lorsque ils arrivent au même point en même temps. Cette notion ne s'applique qu'aux signaux périodiques, en effet, il n'y a pas lieu de parler de phase pour un signal continu. Le déphasage d'un signal par rapport à un autre s'exprime en degré radian. A titre d'exemple, la figure 1.4 présente un signal déphasé de $\pi/2$ par rapport à un signal de référence.

1.1.3 Comment mesurer un signal ?

Pour mesurer un signal à un instant t , un multimètre est suffisant. Cependant, il n'est alors pas possible d'observer l'évolution temporelle du signal, et donc la

FIGURE 1.4 – Déphasage d'un signal de $\pi/2$.

transmission de l'information. Pour cela, il faut un appareil capable de prendre des mesures de tension périodiquement. L'intervalle temporel entre chaque mesure est qualifié par une fréquence que l'on appelle **fréquence d'échantillonnage** exprimé en **Hertz** (Hz).

Pour l'acquisition de signaux analogiques, un oscilloscope est l'outil idéal. En revanche, pour l'acquisition des signaux numériques, bien qu'un oscilloscope puisse être utilisé, il est préférable d'utiliser un analyseur logique. Le nombre de voies d'acquisitions disponible sur un oscilloscope est bien moindre que celui disponible sur un analyseur logique. En effet, un oscilloscope a le plus souvent 4 voies alors qu'il n'est pas rare d'avoir 8 à 16 voies pour un analyseur logique.

La figure 1.5 présente un analyseur logique bas de gamme. Il a 8 voies d'acquisitions et une fréquence d'échantillonnage de 24 Mhz par voie. D'après la formule 1.4 dans laquelle N est le nombre de mesures, f_{ech} la fréquence d'échantillonnage et t le temps, il apparaît que cet analyseur logique est capable de prendre 24 millions de mesures par voie toutes les secondes.



FIGURE 1.5 – Présentation d'un analyseur logique à 8 voies.

$$f_{ech} = N/t \quad (1.4)$$

Dans le cadre d'un oscilloscope comme celui présenté sur la figure 1.6, la fréquence d'échantillonage est de 8 GHz, ce qui équivaut à 8 milliards d'échantillons par seconde, soit près de 4000 fois plus que l'analyseur logique présenté ci-dessus.

Cependant ce taux d'échantillonnage n'est valable que pour les 4 voies, contrairement aux 8 de l'analyseur logique.



FIGURE 1.6 – Présentation d'un oscilloscope 4 voies.

Il apparaît alors évident qu'un oscilloscope sera préférable pour la capture de signaux rapides, tandis qu'un analyseur logique est suffisant pour des signaux plus lents.

Lors d'une mesure, il est important d'avoir un ordre de grandeur des amplitudes des signaux mesurés. Cela pour les raisons évoquées ci-dessous :

- En connaissant l'ordre de grandeur des amplitudes attendues, il est possible de sélectionner la plage d'échelle appropriée sur l'appareil de mesure. Si une échelle trop petite est choisie, le signal pourrait être écrasé et difficile à interpréter. À l'inverse, si une échelle trop grande est choisie, les détails importants du signal risquent de ne pas être visibles.
- Cela permet également d'éviter d'endommager l'appareil de mesure en s'assurant que le signal d'entrée ne dépasse pas les limites autorisées pour l'appareil en question. Un signal trop fort pourrait alors endommager les composants internes de l'appareil, le rendant inutilisable.
- Connaitre l'ordre de grandeur des amplitudes des signaux aide à interpréter correctement les résultats des mesures, ce qui peut être crucial pour diagnostiquer des problèmes ou analyser le comportement d'un circuit électronique.
- Cela permet également de repérer rapidement si une mesure semble incorrecte si celle-ci est significativement différente de ce qui était prévue.
- En utilisant la plage d'échelle appropriée dès le départ, les mesures sont effectuées plus rapidement et rend alors le processus de mesure plus efficace.

1.1.4 Le bruit lors d'une mesure

De manière générale, un circuit électronique est soumis à une multitude de bruits qui peuvent perturber les signaux. Il peut s'agir de bruits externes au

système, comme les rayonnements cosmiques, ou bien de bruits internes, comme les variations brusques de courant au sein des pistes du **PCB**. Le concepteur du **PCB** fait de son mieux pour limiter les bruits qu'il peut contrôler et qui peuvent perturber le bon fonctionnement du circuit électronique. Cela revient à limiter les rebonds de tension, la diaphonie, etc. En ce qui concerne les bruits externes, le concepteur peut avoir une influence sur certains d'entre eux en utilisant par exemple, des cages de Faraday. La cage de Faraday est une enceinte qui permet de bloquer les champs électromagnétiques extérieurs au système. Elle peut cependant être utilisée pour bloquer des bruits internes. En effet, lorsqu'une charge en mouvement traverse un conducteur, un champ magnétique est créé. Ce même champ magnétique peut alors interagir avec les autres pistes du **PCB** est induire en retour un champ électrique, lequel induira une perturbation dans les signaux. Pour limiter les bruits d'origines électromagnétiques, les dispositifs sensibles sont placés dans une cage de Faraday, comme celle que l'on peut voir sur la figure 1.7. On retrouve alors les cages de Faraday dans les appareils travaillant dans le domaine des radio-fréquences. La connaissance des différents bruits permet alors d'orienter le placement des composants sur le **PCB**. On aura alors tendance à regrouper les composants analogiques. De même pour les composants numériques, mais également pour les composants hautes-fréquences. Cela permet de mettre en place des stratégies de limitations des bruits.

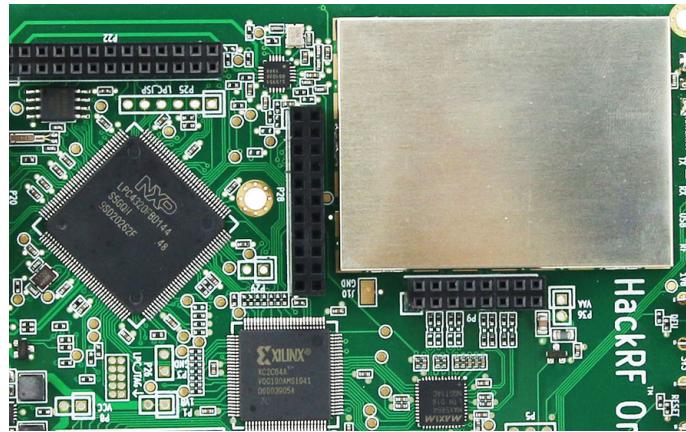


FIGURE 1.7 – Présentation d'une cage de Faraday

Dans le cadre d'une mesure d'un signal, on parlera alors des bruits externes induits par la présence de l'appareil de mesure. L'appareil de mesure, quel qu'il soit, est un appareil électronique qui induit alors un couplage entre lui-même et l'appareil de test. Il existe quatre types de couplages :

- **Conductif** : provient du partage d'un signal bruité entre plusieurs lignes d'alimentation. Un signal d'alimentation bruité par un effet thermique par exemple sera alors partagé à tout les composants partageant la même ligne d'alimentation.
- **Capacitif/Inductif** : provient de la variation des champs électriques/magnétiques. Comme énoncé plus haut, lorsque les lignes sont parcourues par un champ électrique, elles génèrent un champ électromagnétique en retour qui peut induire du bruit.
- **Radiatif** : Lorsqu'un signal de faible amplitude doit être fortement amplifié, il est alors possible d'y détecter un couplage radiatif issu de la

présence de champs électromagnétiques extérieurs (téléphone, wifi, ...).

Ainsi, lorsqu'un signal est mesuré et que celui-ci apparaît distordu, il faut se poser la question de savoir si celui-ci l'est naturellement, ou bien s'il l'est à cause de la présence de l'instrument de mesure.

1.1.5 Les composants usuels

La plupart des composants qui seront présentés dans cet ouvrage seront des composants **SMD** (*Surface Mounted Device*). En effet ceux-ci sont devenus incontournables de par le gain de place non négligeable qu'ils offrent ainsi que par la facilité de réalisation du **PCB** car il n'y a plus besoin de percer des trous pour les installer.

Les composants **SMD** sont délivrés sous forme de boîtiers de tailles différentes comme on peut le voir sur la figure 1.8. Certains boîtiers de composants passifs ainsi que leurs tailles sont présentés dans le tableau A.1 (voir annexe A.1 pour plus de détails concernant les boîtiers).

Nom du boîtier	taille (mm × mm)
0402	1,0 × 0,5 mm ²
0603	1,5 × 0,8 mm ²
0805	2,0 × 1,3 mm ²
1008	2,5 × 2,0 mm ²

TABLE 1.1 – Présentation de différents boîtiers **SMD** (résistances et condensateurs) ainsi que les tailles associées.



FIGURE 1.8 – Quelques exemples de résistances **SMD**.

La résistance

Comme il a été présenté dans la section 1.1.1, la résistance permet d'agir sur l'intensité et la tension. Ce comportement est déterminé grâce à la loi d'Ohm (voir équation 1.1). La résistance est de loin le composant le plus utilisé en électronique. Le terme "résistance" vient du fait que ce composant empêche plus ou moins la propagation des électrons dans le circuit électrique en fonction de la valeur de celle-ci. Elles sont constituées de couches métalliques, d'oxyde de métal, de

couches de carbone, ou encore de céramique. L'assemblage de ces couches permet d'obtenir toute une panoplie de valeurs de résistances. En résistant au mouvement des électrons, la résistance doit absorber l'excès d'énergie et la diffuser sous forme de chaleur. C'est pourquoi chaque résistance a une limite d'utilisation spécifiée en **Watt** ainsi qu'une tolérance à cette limite spécifiée en pourcentage. La valeur de la résistance, quant à elle, est exprimée en **Ohm**.

Il existe différents types de résistances comme on peut le voir sur la figure 1.9. Dans le cadre de cet ouvrage, on la rencontrera souvent soit comme résistance de rappel (*pull-down*), soit comme résistance de tirage (*pull-up*) sous forme **SMD**. Les résistances **SMD** sont noires avec la valeur de la résistance écrite dessus comme on peut le voir sur la figure 1.8.



FIGURE 1.9 – Présentation de différents types de résistances.

Les **résistances de rappel** et les **résistances de tirage** permettent de relier une ligne d'un bus soit à la terre (0 logique) soit à la source d'alimentation (1 logique). Lorsque la ligne d'un bus est reliée à la terre (*ground* : 0 V), on parle de résistance de rappel et, l'on parle de résistance de tirage lorsque la ligne est reliée à la source d'alimentation (**VCC**). Les résistances de rappel ont souvent une valeur située entre $50\ \Omega$ et $500\ \Omega$, tandis que les résistances de tirages ont souvent une valeur située entre $10\ k\Omega$ et $100\ k\Omega$.

Par exemple, les lignes **SDA** et **SCL** d'un bus **I²C** (voir section 1.2.3, figure 1.28) doivent être à l'état haut par défaut (1 logique), on parle de **collecteur ouvert** (ou drain ouvert), et doivent alors avoir une résistance de tirage sur chacune des deux lignes.

Le condensateur

Le condensateur (aussi appelé capacité) est un composant qui permet de stocker de l'électricité. Tout comme la résistance, le condensateur est un composant passif. Il est au moins tout aussi utilisé que la résistance. Ils sont constitués d'un assemblage de matériaux conducteur de courant et isolant. Les charges électriques sont alors stockées dans les feuillets conducteurs et ne peuvent donc pas traverser les feuillets isolants. La capacité d'un condensateur se note **C** et s'exprime en **Farad** (F). La relation suivante permet d'établir la charge électrique totale **Q** stockée en fonction de la capacité **C** et de la tension **U**.

$$Q = C \cdot U \quad (1.5)$$

Le condensateur est également relié à l'intensité, dans une toute autre mesure que la résistance. Le courant n'est plus proportionnel à la tension, mais proportionnel à la variation temporelle de la tension comme le montre l'équation différentielle ci-dessous :

$$I = C \cdot \frac{dU}{dt} \quad (1.6)$$

Du fait de la présence d'une dérivée temporelle, le condensateur est donc un composant dynamique (contrairement à la résistance) qui introduit alors des phénomènes transitoires dans le circuit électrique.

De par leur nature à stocker de l'énergie électrique, les condensateurs sont utilisés comme une source d'énergie auxiliaire qui permet de rattraper les défauts d'alimentation. Lorsque l'alimentation subit des variations minimes (lors d'une tentative d'injection de faute par variation de la tension d'alimentation par exemple) mais qui peuvent entraîner des perturbations dans le signal (du bruit), le condensateur permet de lisser le signal de manière à absorber ce défaut et donc à maintenir la tension optimale dans le circuit. Dans ce cas précis, on parle de condensateur de découplage.

Tout comme les résistances, il existe différents types de condensateurs (voir figure 1.10). Les condensateurs sont de couleur jaune/orange/marron lorsqu'ils sont du type **SMD**.



FIGURE 1.10 – Présentation de différents types de condensateurs.

La bobine

La bobine (aussi appelée inductance) est un composant qui, tout comme le condensateur, permet de stocker de l'énergie. Cette fois-ci, celle-ci n'est pas stockée sous forme de charge électrique, mais sous forme de champ magnétique. La bobine est également un élément passif et dynamique. À la différence du condensateur qui modifie la valeur de l'intensité suite à une variation temporelle de la tension, la bobine modifie la tension suite à une variation temporelle de

l'intensité. L'équation différentielle qui traduit cela est présentée ci-dessous :

$$U = L \cdot \frac{dI}{dt} \quad (1.7)$$

Dans cette équation, **L** est une constante qui est appelée l'**inductance** et est exprimée en **Henri** (H). Une bobine est constituée d'un enroulement de fil de cuivre isolé autour d'un cœur de ferrite. Au passage d'un courant continu, un champ magnétique est alors créé. Différents types de bobines existent, certaines d'entre elles sont présentés sur la figure 1.11 (les bobines dans un boîtier SMD sont le plus souvent de couleur grise).



FIGURE 1.11 – Présentation de différents types de bobines.

A ce stade, il est intéressant de regarder de nouveau la figure 1.2 (étage d'alimentation d'un *switch*) afin d'observer les résistances, les condensateurs et la bobine utilisés pour abaisser la tension d'alimentation de 9 V à 3,3 V.

Le cristal

Le cristal est un élément indispensable à tout circuit intégré nécessitant une fréquence d'horloge stabilisée. On l'appelle aussi oscillateur ou encore quartz car il est constitué d'un cristal de quartz. Ce cristal de quartz est taillé en fonction de la fréquence de résonance que l'on souhaite obtenir.

Le cristal de quartz est piézo-électrique, ce qui signifie qu'il génère des ondes acoustiques lorsqu'une tension lui est appliquée et inversement. Ainsi en propagant une onde mécanique à une fréquence résonnante, on obtient en retour une tension variant à cette même fréquence. L'onde acoustique est assurée par un champ électrique oscillant grâce à un circuit **RLC** (Résistance-Bobine-Condensateur).

Les fréquences standards des cristaux vont de 10 kHz à 30 MHz. Les fréquences sont fixées par le constructeur, mais il est néanmoins possible d'ajuster légèrement la fréquence de résonance à l'aide de condensateurs bien positionnés. La figure 1.12 en présente différents types.

Les cristaux sont, entre autre, utilisés pour effectuer des communications synchrones entre composants électroniques en fournissant un signal d'horloge sur



FIGURE 1.12 – Présentation des différents types de cristaux.

lequel baser la transmission/réception des informations.

Le transistor

Le transistor est l'acronyme de *TRANSfert resISTOR*. Il en existe plusieurs familles qui sont essentiellement vouées à 2 fonctions : la commutation ou l'amplification. Parmi les différents types de transistors on trouve principalement les transistors bipolaires et les transistors uni-polaires (voir figure 1.13). Le transistor le plus commun est le transistor bipolaire. Il est employé aussi bien dans le domaine des basses fréquences que des hautes fréquences, des faibles puissances que des hautes puissances.

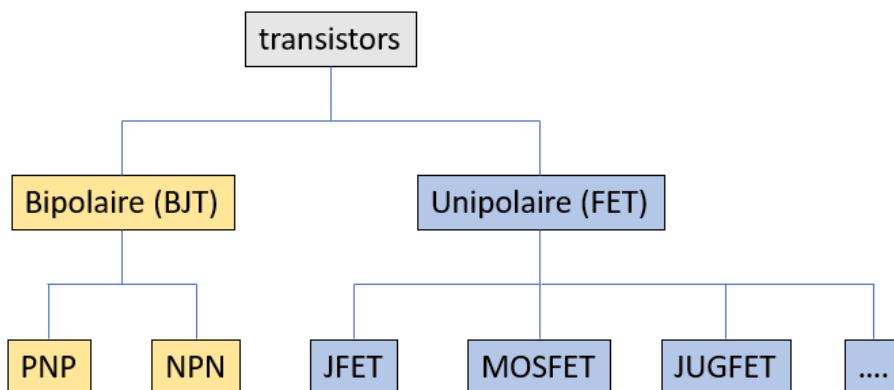


FIGURE 1.13 – Présentation des familles de transistors.

Les transistors ont trois pattes (parfois le boîtier en possède quatre). Lorsqu'ils sont utilisés en tant que commutateur, ils agissent comme un interrupteur commandé en fonction du signal qui est sur la troisième patte. Lorsqu'ils sont utilisés en tant qu'amplificateur, ils sont capables d'amplifier un courant en fonction du signal qui est sur la troisième patte. Différents types de transistors sont présentés sur la figure 1.14

La figure 1.15 (issue d'un digicode) présente deux transistors sur un **PCB**. Le transistor **Q3** est un transistor bipolaire de type **NPN**, tandis que le transistor **Q4** est un transistor bipolaire de type **PNP**.

La différence entre un transistor **NPN** et un transistor **PNP** se situe dans l'agencement des jonctions **PN**. En physique des semi-conducteurs, une jonction

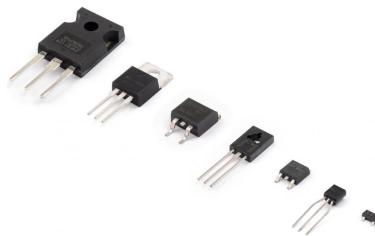


FIGURE 1.14 – Présentation des différents types de transistors.

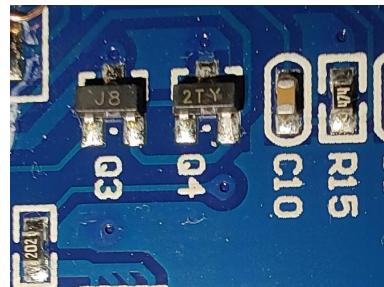


FIGURE 1.15 – Présentation de deux transistors issus d'un digicode.

PN est une zone d'un cristal de silicium dans laquelle le dopage en électrons et trous varient brusquement, laissant alors la jonction **PN** se comporter comme une diode).

Le régulateur de tension

Lorsque l'on intègre une source de tension à un circuit pour l'alimenter, il est nécessaire de s'assurer qu'elle soit la plus stable possible de manière à ne pas provoquer de perturbation dans le fonctionnement du circuit. Le régulateur de tension s'assure de cela. Celui qui sera présenté ici est le **LDO** (*Low-DropOut regulator*) (l'autre type étant le régulateur **DC-DC**). Le **LDO** est un régulateur de tension linéaire à courant continu.

Corrigeant les défauts d'alimentation, il doit avoir une tension de sortie variable et de préférence pas trop éloignée de la tension d'entrée. Des résistances et des capacités sont ajoutées au **LDO** de manière à régler finement la tension de sortie.

Le composant noir de la figure 1.2 (étage d'alimentation d'*switch*) est un **LDO**. On y voit clairement les composants (résistance, capacité) qui y sont reliés afin d'ajuster finement la tension de sortie. Le nombre de pattes des **LDO** peut varier selon les modèles, il n'est alors pas toujours facile de les repérer. Cependant, de par son action sur la stabilisation de la tension, il est le plus souvent localisé proche d'un étage d'alimentation.

La mémoire

Classiquement, la mémoire est catégorisée en deux grandes familles : les mémoires volatiles et les mémoires non volatiles. Les mémoires volatiles ne conservent pas les données en l'absence d'alimentation. On parle de mémoire **RAM** (*Random Ac-*

cess Memory). Cette famille de mémoire peut se subdiviser en deux : les **SRAM** (*Static RAM*) et les **DRAM** (*Dynamic RAM*). Si des données doivent survivre à une absence d'alimentation, comme par exemple un *bootloader* qui chargerait en mémoire **RAM** un système d'exploitation, il faudra utiliser une mémoire **NVM** (*Non Volatile Memory*) telle qu'une **flash** ou une **EEPROM** (*Electrically Erasable Programmable Read Only Memory*).

La différence principale entre la **SRAM** et la **DRAM** réside dans la conception même du stockage de l'information. Une **SRAM** stocke l'information dans des assemblages de transistors que l'on appelle *flip-flop*, tandis qu'une **DRAM** la stocke dans des condensateurs. Un condensateur se décharge en moins d'une seconde tandis qu'un transistor garde l'information tant qu'on ne le passe pas à un autre état. De par cette différence fondamentale, on comprend qu'une **DRAM** doit être réécrit régulièrement (plusieurs dizaines de fois par seconde) afin de conserver l'information. La figure 1.16 présente une **DRAM** issue d'un **ECU** (*Engine Control Unit*) automobile.



FIGURE 1.16 – Présentation d'une **DRAM**.

Il existe deux types de **flash** : la **NOR** (*Not OR*) et la **NAND** (*Not AND*). La différence étant dans l'architecture des cellules-mémoire les composants. L'une contient des portes logiques **NOR** et l'autre des portes logiques **NAND**. Les **NOR** sont plus rapides en lecture mais plus lentes en écriture que les **NAND**. De plus, la **NAND** a une capacité de stockage plus importante (jusqu'à 1 To, et 1 Go pour la **NOR**) et est le plus souvent utilisée lorsque le besoin d'écrire et d'effacer est important (clef **USB**, disque dur **SSD**, ...). En revanche la **NOR** autorise l'accès aléatoire, ce qui permet l'exécution de code. Pour de l'exécution de code sur une **NAND**, il faut prévoir une **RAM** à disposition. C'est pourquoi, il arrive que ces deux mémoires soient imbriquées dans un même composant comme on peut le voir sur la figure 1.17 qui embarque une **NAND** et une **RAM** (issue d'une caméra de surveillance). Une **NOR** typique contenant un *firmware* est présentée sur la figure 1.18 (issue d'un *router*).

Les mémoires sont lues/écrites/effacées par un micro-contrôleur en fonction des applications et des besoins du moment. Lorsque les lignes de communications entre la mémoire et le micro-contrôleur peuvent être déterminées, il est alors possible, *a priori*, d'agir sur la mémoire et donc d'agir sur le comportement du circuit électronique.



FIGURE 1.17 – Présentation d'un composant embarquant une **NAND** et une **RAM** issue d'une caméra de surveillance.

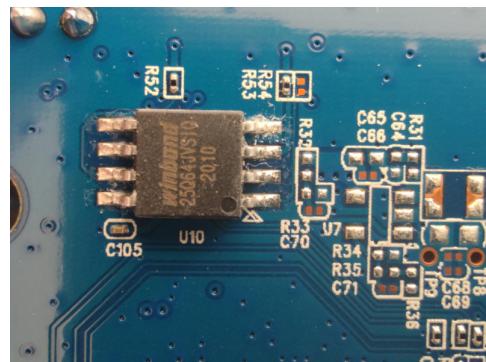


FIGURE 1.18 – Présentation d'une mémoire **flash** de type **NOR** issue d'un *router*.

Le micro-contrôleur

Le micro-contrôleur est un composant essentiel d'un circuit électronique. Il permet d'apporter au circuit une notion "d'intelligence" en étant capable d'exécuter du code à la manière d'un ordinateur. Il existe une multitude de micro-contrôleurs encapsulés dans des dizaines de boîtiers différents. Afin d'apporter cette "intelligence", le micro-contrôleur doit être capable de communiquer avec les différents composants du circuit électronique. Pour cela, différents protocoles de communication existent. Parmi ceux qui seront abordés dans cet ouvrage, on trouve l'**UART**, le **SPI**, l'**I²C**, et le **CAN**, mais il en existe d'autres comme le **PCI Express**, l'**Ethernet**, le **SATA**, l'**USB**, le **FireWire**, etc.

Pour exécuter du code, le micro-contrôleur est doté de différents éléments comme un **CPU** (*Central Processing Unit*), une mémoire de type **NVM**, une mémoire de type **SRAM**, un cristal, une interface de diagnostic (**JTAG**, **SWD**), une interface de programmation (**ICSP** : *In-Circuit Serial Programming*), ainsi que plusieurs bus de données internes permettant d'utiliser les protocoles de communication cités ci-dessus et présentés plus en détails dans la section 1.2. Cette architecture générale est présentée sur la figure 1.19.

Les micro-contrôleurs sont souvent associés à d'autres composants comme des **DRAM** et des **flash**. Le micro-contrôleur possède en **ROM** interne un *bootloader* qui charge en **RAM** le *firmware* localisé dans la **flash**. La figure 1.20 issue d'une carte mère d'une console de jeux vidéos présente un exemple de cette situation.

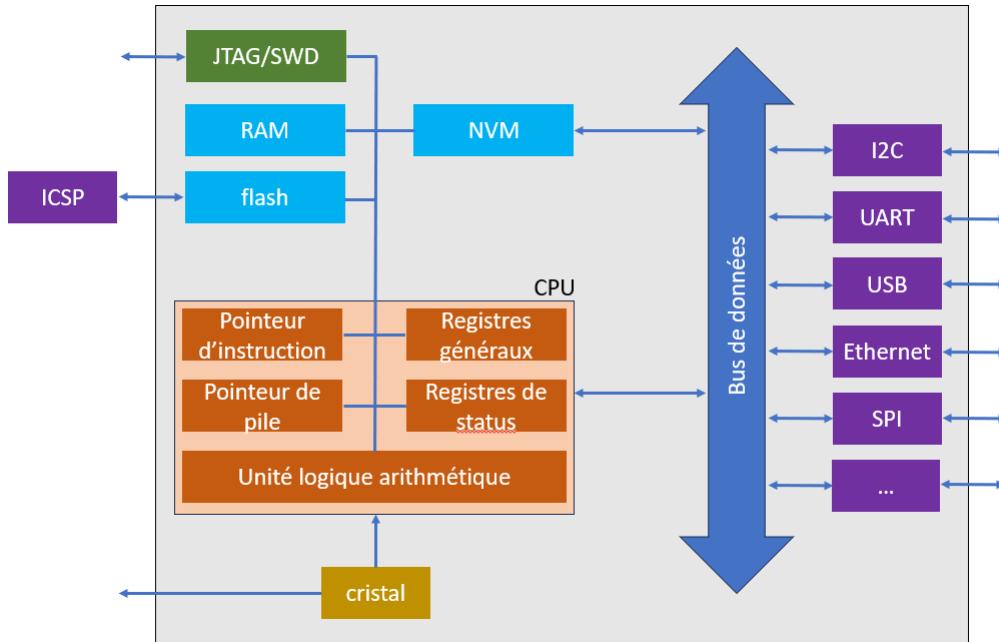


FIGURE 1.19 – Présentation générale de l'architecture d'un micro-contrôleur.



FIGURE 1.20 – Présentation d'un micro-contrôleur associé à une **DRAM** et une **flash NOR** (issue d'une carte mère d'une console de jeux vidéos).

1.2 Les protocoles de communication

Il existe deux types de bus de données sur lesquels les composants peuvent échanger de l'information par le biais de protocoles de communication. On distingue les bus séries des bus parallèles. Les principales différences sont :

1. Bus série

- Dans un bus série, les données sont transmises bit par bit, l'une après l'autre, sur une seule ligne de communication.
- Le transfert des données est séquentiel, c'est-à-dire que les bits sont envoyés l'un après l'autre dans un ordre prédéterminé.
- Un seul fil de communication est nécessaire pour transmettre les données.
- Les bus série sont souvent utilisés pour les communications longue distance, car ils nécessitent moins de câbles et sont plus faciles à gérer sur de longues distances.

2. Bus parallèle

- Dans un bus parallèle, les données sont transmises simultanément sur plusieurs lignes de communication (câbles) à la fois.
- Chaque bit des données est envoyé sur une ligne séparée, ce qui permet de transférer plusieurs bits en même temps.
- Les bus parallèles nécessitent un plus grand nombre de fils de communication en fonction du nombre de bits transférés simultanément.
- Historiquement, les bus parallèles étaient utilisés pour des transferts de données rapides entre les composants internes d'un ordinateur, comme entre le processeur et la mémoire.

En résumé, la principale différence entre un bus série et un bus parallèle réside dans la manière dont les données sont transmises. Les bus série transmettent les données bit par bit sur une seule ligne de communication, tandis que les bus parallèles transmettent plusieurs bits simultanément sur plusieurs lignes de communication. Les bus série sont souvent privilégiés pour les communications longue distance, tandis que les bus parallèles étaient utilisés pour des transferts internes rapides, bien qu'ils soient devenus moins courants avec l'évolution de la technologie.

Les principaux protocoles qui seront abordés dans cet ouvrage seront des protocoles transitant sur des bus séries.

1.2.1 UART

Le protocole **UART** (*Universal Asynchronous Receiver/Transmitter*) est une interface de communication série largement utilisée dans les systèmes électroniques pour permettre la transmission de données entre différents périphériques.

Contrairement à certaines autres interfaces série, l'**UART** utilise une communication asynchrone, ce qui signifie qu'il n'y a pas de signal de synchronisation continu entre l'émetteur et le récepteur. Les données sont transmises sous forme de trames, avec chaque trame comprenant un *bit* de départ, les *bits* de données (le nombre de *bits* dépend du nombre de *bits* de données spécifié, comme 8 *bits*), un ou plusieurs *bits* de parité facultatifs pour la détection d'erreurs et un ou plusieurs *bits* de *stop* pour délimiter la trame.

L'**UART** est utilisé pour permettre la communication série entre des composants électroniques, tels que des micro-contrôleurs, des capteurs, des modules sans fil, des modules **GPS**, des écrans **LCD**, des périphériques **Bluetooth**, etc. Il est souvent utilisé dans les systèmes embarqués et les dispositifs où la communication série simple et fiable est nécessaire.

L'**UART** est dit "asynchrone" car les deux parties qui communiquent peuvent avoir des horloges indépendantes, ce qui signifie qu'elles n'ont pas besoin d'être synchronisées exactement au même moment pour transmettre et recevoir des données. Cette "asynchronicité" rend l'**UART** simple à utiliser, mais nécessite une configuration correcte des paramètres de vitesse de transmission (*baud rate*), de bits de données, de bits de parité et de bits de stop pour assurer une communication réussie entre les appareils. Dans la littérature, ce protocole de commu-

nication est également nommé **Serial**, **RS-232** ou encore **TTL**. Si le protocole supporte aussi le mode synchrone, il est alors nommé **USART**.

Les niveaux de tension utilisés varient de **0 V** à **5 V** (ce que l'on appelle communément le **TTL** (*Transistor-Transistor Logic*)). Trois fils sont nécessaires pour utiliser l'**UART** :

- une ligne de transmission : **TX**.
- une ligne de réception : **RX**.
- une masse : **GND**.

Les trames de communication **UART** sont constituées de la manière suivante :

- un zéro logique qui signifie que le composant est prêt à transmettre. Il s'agit d'un *bit* de *start* qui est toujours à 0.
- les données à transmettre dont la taille est comprise entre 5 et 9 *bits*. Le *bit* de poids faible de chaque *byte* étant envoyé en premier, et le *bit* de poids fort en dernier.
- un *bit* de parité (optionnel) qui indique si le nombre total de 1 dans un *byte* est pair (**E** : *Even*) ou impair (**O** : *Odd*).
- un *bit* de *stop* qui est toujours à 1 et dont la durée (qui dépend de la vitesse de transmission établie entre les parties qui communiquent) varie entre 1, 1.5 et 2.

Cela peut être aperçu de manière schématique sur la figure. 1.21.

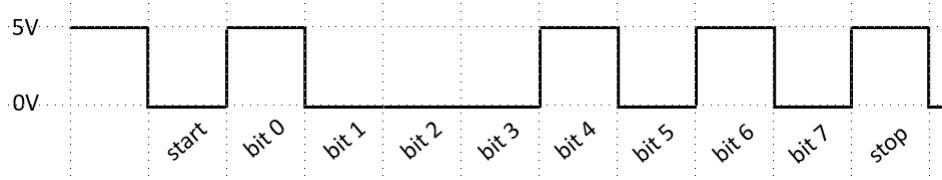


FIGURE 1.21 – Illustration d'une trame **UART**.

Ainsi pour communiquer en **UART** avec un composant comprenant l'**UART**, il est nécessaire de s'accorder sur une vitesse de transmission (qui ont été normalisées par multiples et sous-multiples de 9600 *bauds* : 2400, 4800, 9600, 19200, ... 115200, ...), le nombre de *bits* dans un *byte* (il est, de manière générale, admis qu'il vaut 8), la parité et la longueur du *bit* de *stop*. Cette configuration est alors écrit de la manière suivante : **115200 8N1** (115200 *bauds* pour la vitesse de transmission, 8 *bits* dans un *byte*, pas de *bit* de parité (N) et 1 *bit* de *stop*).

Le branchement des fils entre un composant **A** et un composant **B** est le suivant :

- le **TX** du composant **A** avec le **RX** du composant **B**.
- le **RX** du composant **A** avec le **TX** du composant **B**.
- le **GND** du composant **A** avec le **GND** du composant **B**.

Le protocole **UART** est souvent utilisé comme une console de *debug*. Il est donc fréquemment visible sur le **PCB** (voir figure 1.22 qui est issue d'un circuit

électronique d'un *router*). Bien que seulement trois fils soient nécessaires, le port **UART** est, de manière quasi-générale, rendu disponible avec quatre trous d'insertions (**TX**, **RX**, **GND** et **VCC**). La manière de déterminer la nature de ces trous d'insertions est explicitée dans la suite de cet ouvrage.

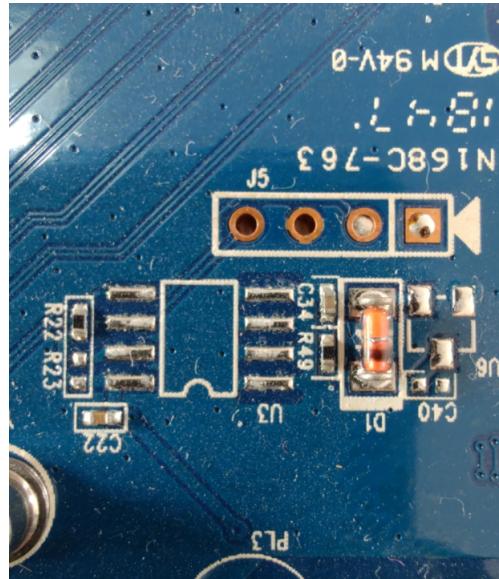


FIGURE 1.22 – Visualisation d'un port **UART** sur un **PCB** issu d'un *router*.

Ayant accès à ce port de *debug*, il est donc possible de discuter avec le composant grâce au protocole **UART**. Il est à noter que parfois l'écriture n'est pas autorisée. Il n'est alors possible que de lire ce qui est affiché sur la console de *debug*, ce qui peut néanmoins permettre d'obtenir des informations importantes sur le composant et sur le fonctionnement du **PCB** de manière générale.

1.2.2 SPI

Tout comme **UART** est un acronyme, **SPI** en est également un, il signifie *Serial Peripheral Interface*. Il s'agit d'une interface série développée par **Motorola** dans les années 80 et permettant à un contrôleur de communiquer avec des périphériques à travers un bus de données. Le contrôleur souvent appelé *master* contrôle la communication avec plusieurs périphériques appelés *slaves*. Le périphérique ne peut pas initier une communication, il ne peut que répondre à celle initiée par le contrôleur. De plus, il est à noter que plusieurs périphériques peuvent partager le même bus de données.

Quatre signaux logiques sont nécessaires pour communiquer en **SPI** :

- un signal d'horloge généré par le contrôleur : **SCLK** (*Serial CLock*).
- un signal de sortie du contrôleur vers l'entrée du périphérique : **MOSI** (*Master Output, Slave Input*).
- un signal d'entrée vers le contrôleur en provenance du périphérique : **MISO** (*Master Input, Slave Output*).
- un signal de sélection du périphérique : **SS** (*Slave Select* qui permet au contrôleur de sélectionner le périphérique avec lequel il souhaite communiquer).

Comme un signal d'horloge est nécessaire, on comprend qu'il s'agit d'une communication synchrone. Cela permet des transferts de données rapides et efficaces. Le contrôleur contrôle l'horloge, ce qui garantit que les périphériques sont parfaitement synchronisés. De plus, on peut également noter que le signal est *full-duplex*, ce qui signifie que l'information est transportée simultanément dans les deux sens.

Les branchements des fils entre un contrôleur et ses périphériques est le suivant :

- le **SCLK** du contrôleur avec le **SCLK** de tout les périphériques.
- le **MOSI** du contrôleur avec le **MOSI** de tout les périphériques.
- le **MISO** du contrôleur avec le **MISO** de tout les périphériques.
- le **\overline{SS}_1** du contrôleur avec le **\overline{SS}** du périphérique **1**, le **\overline{SS}_2** du contrôleur avec le **\overline{SS}** du périphérique **2**, ...

Ceux-ci sont représentés sur la figure 1.23.

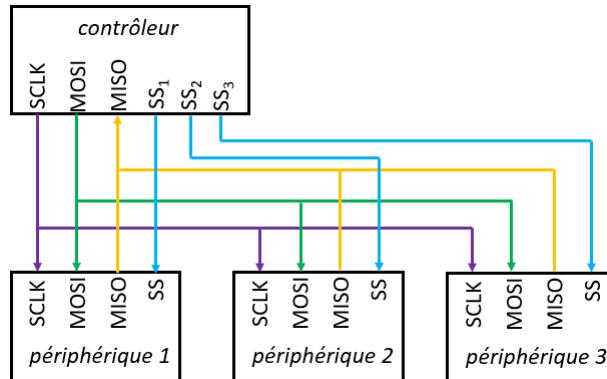


FIGURE 1.23 – Visualisation des branchements **SPI** entre un contrôleur et trois périphériques.

La barre au dessus de **\overline{SS}** signifie que ce pin est *active low*, ce qui indique que le voltage haut (souvent 3,3 V) correspond à la valeur *False* et que le voltage bas (0 V) correspond à la valeur *True*. Le périphérique attend donc que le **\overline{SS}** soit égal à 0 V afin de réceptionner le signal d'horloge (**SCLK**) ainsi que le signal de sortie du contrôleur (**MOSI**). A chaque coup d'horloge le contrôleur et le périphérique s'échangent alors 1 *bit*. Il faut donc 8 coups d'horloge pour échanger 1 *byte*.

La figure 1.24 issue d'un **PCB** d'un *router* présente un périphérique **SPI**. Il s'agit d'une mémoire **flash** permettant de stocker le *firmware* d'un micro-contrôleur. D'après la documentation de cette mémoire présentée sur la figure 1.25, il apparaît que la notation des pins est différente de celle présentée ci-dessus. En effet, la nomenclature est relativement permissive. Ainsi **\overline{SS}** peut aussi être nommé **/CS** ou encore ***CS**. **DO** (*Data Output*) correspond à **MOSI**, mais peut aussi être nommé **SDO**, **COP**I ou encore **SO**. **DI** (*Data Input*) correspond à **MISO**, mais peut aussi être nommé **SDI**, **CIP**O ou encore **SI**. **CLK** correspond à **SCLK** mais peut aussi être nommé **SCL** ou **SCK**.

Il existe quatre modes de communication **SPI**. Comme il a été énoncé précédemment, une communication **SPI** est basée sur la fréquence d'une horloge définie par le

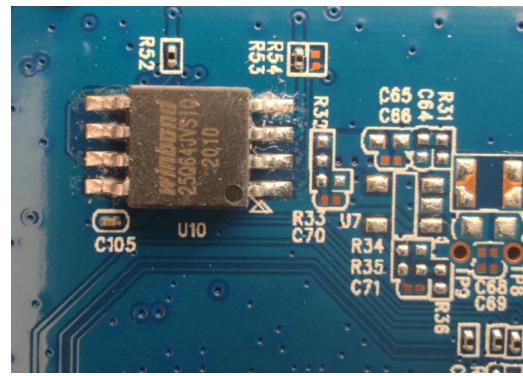


FIGURE 1.24 – Visualisation des pins d'une mémoire **flash SPI** issue d'un **PCB** d'un *router*.

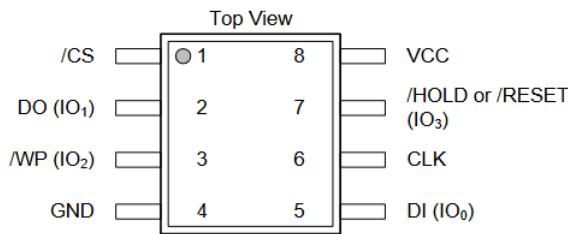


FIGURE 1.25 – Documentation d'une mémoire **flash SPI** présentant la description des pins.

contrôleur. Ces modes de communication sont définis par deux paramètres appelés **CPOL** (*Clock POLarity*) et **CPHA** (*Clock PHAse*) qui représentent respectivement la polarité de l'horloge et sa phase (voir section 1.1.2). Les quatre modes de communication représentés par le couple (**CPOL**, **CPHA**) sont représentés dans le tableau 1.2.

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

TABLE 1.2 – Détermination des quatre modes de communication **SPI**.

L'explication des différents modes est présentée ci-dessous :

- Le mode 0 est défini par le couple (**CPOL**, **CPHA**)=(0, 0). L'horloge est *active high* (le voltage haut correspond à *True*) et le transfert de données se fait sur le front montant du cycle d'horloge.
- Le mode 1 est défini par le couple (**CPOL**, **CPHA**)=(0, 1). L'horloge est *active high* également et le transfert de données se fait sur le front descendant du cycle d'horloge.
- Le mode 2 est défini par le couple (**CPOL**, **CPHA**)=(1, 0). L'horloge est *active low* (le voltage bas correspond à *True*) et le transfert de données se fait sur le front descendant du cycle d'horloge.
- Le mode 3 est défini par le couple (**CPOL**, **CPHA**)=(1, 1). L'horloge est *active low* également et le transfert de données se fait sur le front mon-

tant du cycle d'horloge.

Bien que les mémoires **flash SPI** aient la plupart du temps le *pinout* présenté sur la figure 1.25, il peut arriver que certaines aient un *pinout* différent. Par exemple, la **flash** présente sur la figure 1.26 (issue d'un casque audio) a un *pinout* totalement différent comme on peut le constater sur la figure 1.27. Dans le schéma présenté ici, le sigles correspondent à :

- S : Chip Select
- D : Serial Data Input
- Q : Serial Data Output
- C : Serial Clock
- ORG : Organization Select
- VCC : Supply Voltage
- VSS : Ground



FIGURE 1.26 – Visualisation des pins d'une mémoire **flash SPI** issue d'un **PCB** d'un casque audio.

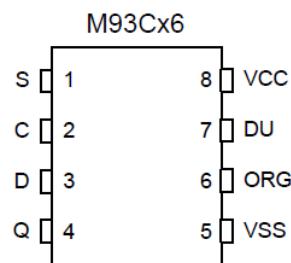


FIGURE 1.27 – Documentation d'un autre type de mémoire **flash SPI**.

Outre les noms de pins qui sont différents, leurs position l'est également ! Il est donc important de toujours vérifier la documentation d'un composant plutôt que de faire des hypothèses sur son *pinout*.

1.2.3 I²C

I²C est l'acronyme de *Inter-Integrated Circuit*. Il s'agit d'un protocole de communication développé par **Philips** (maintenant **NXP Semiconductors**) et qui permet de relier un micro-contrôleur (*Master*) à différents périphériques (*Slaves*) à travers un bus série synchrone bidirectionnel *half-duplex* (à comparer au protocole **SPI** qui est *full-duplex*). On note rapidement une grande ressemblance entre le

SPI et l'**I²C**. D'ailleurs la plupart des composants qui propose le **SPI**, propose également l'**I²C**, et inversement. La différence essentielle réside dans le fait que l'**I²C** est multi-contrôleurs, ce qui signifie qu'il peut y avoir plusieurs contrôleurs sur le même bus.

Les échanges de données ont toujours lieu entre un contrôleur et un (ou plusieurs) périphérique, mais jamais entre un contrôleur et un autre contrôleur. Les branchements reliant un contrôleur à un périphérique sont :

- une ligne de données bidirectionnelle : **SDA** (*Serial DAta*).
- une ligne d'horloge de synchronisation bidirectionnelle : **SCL** (*Serial CLock*).
- une ligne de masse communes aux différents composants : **GND** (*Ground*).

L'**I²C** utilise un mode dit *Open Drain* pour les lignes **SDA** et **SCL**, ce qui signifie que les lignes sont "tirées" à l'état bas par les composants, mais ils "flottent" à l'état haut. Plus concrètement, aux lignes représentant **SDA** et **SCL** sont associées une résistance de tirage **R_p** (voir section 1.1.5) qui permet de "tirer" la ligne au niveau de tension **VCC**. Cela signifie que si aucun composant n'émet de données sur la ligne, celle-ci est alors au niveau 1 logique. Il est alors possible pour un équipement d'émettre des données sur la ligne **SDA** en la mettant au 0 logique. Ainsi lorsque **SDA** est au 0 logique et **SCL** est au 1 logique, il est alors possible pour un équipement de commencer le transfert de données. Il s'agit d'un signal dit de *start*. Lorsque **SDA** revient au 1 logique et que **SCL** est toujours au 1 logique, la communication est alors terminée et le bus est donc de nouveau libre pour établir une nouvelle communication (signal dit de *stop*).

Un schéma de deux contrôleurs et **N** périphériques avec les résistances de tirage est présenté sur la figure 1.28.

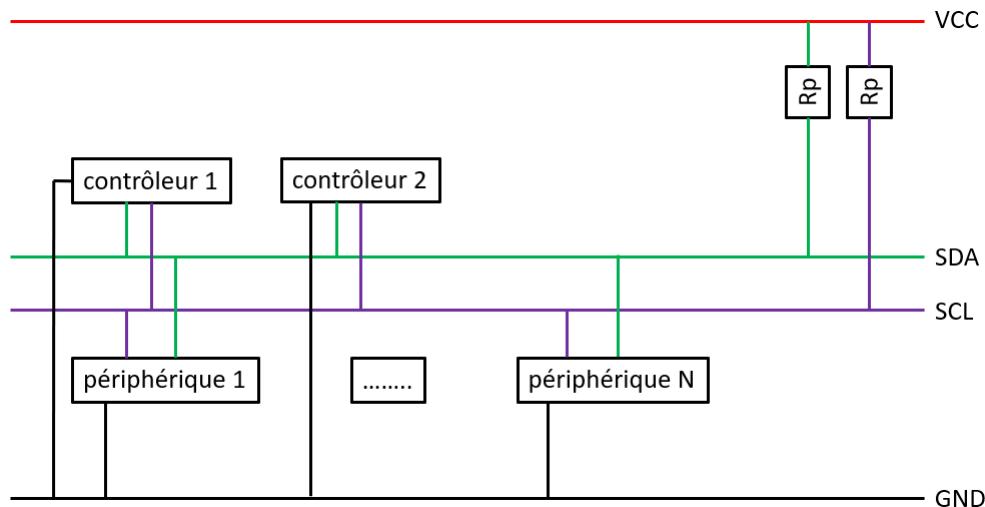


FIGURE 1.28 – Schéma de câblage I²C.

L'une des caractéristiques distinctives de l'**I²C** est qu'il utilise un mécanisme d'adressage pour identifier les périphériques sur le bus. Une fois le signal de *start* envoyé, le contrôleur peut alors envoyer sur la ligne **SDA** une adresse unique associée à un équipement. Chaque périphérique a sa propre adresse codée sur 7 bits. De manière générale, une partie de cette adresse est codée en dur dans le

périphérique, tandis que l'autre partie est programmable de manière à différencier des composants identiques. Un huitième *bit* peut être envisagé comme faisant partie de l'adressage. Il s'agit d'un *bit* qui permet de définir l'action à réaliser : Lecture/Écriture, et est alors nommé **bit R/W** (*Read / Write*). Bien que cela ne soit pas tout à fait vrai, comme l'a compris le lecteur, on peut alors imaginer que chaque périphérique possède deux adresses, une qui permet de lire et l'autre qui permet d'écrire.

Les signaux de *start* et de *stop* sont présentés sur les figures 1.29 et 1.30.

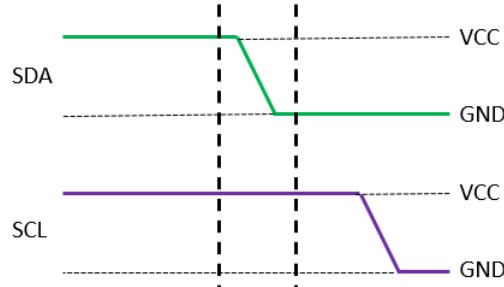


FIGURE 1.29 – Signal de *start* I²C.

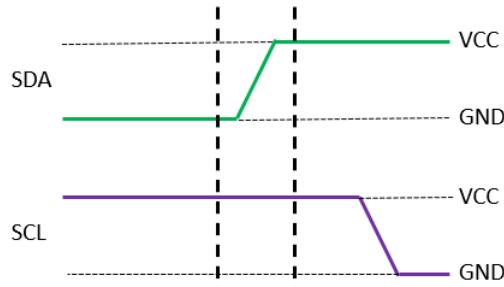


FIGURE 1.30 – Signal de *stop* I²C.

Lorsque le contrôleur a envoyé l'adresse du périphérique avec lequel il souhaite communiquer, il doit attendre la validation du périphérique qui se traduit par l'envoi d'un signal dit d'acquittement (*acknowledgment*). Ce *bit* d'acquittement est présenté sur la figure 1.31. Suite à cela, il est alors possible de lire/écrire des données sur la ligne **SDA**.

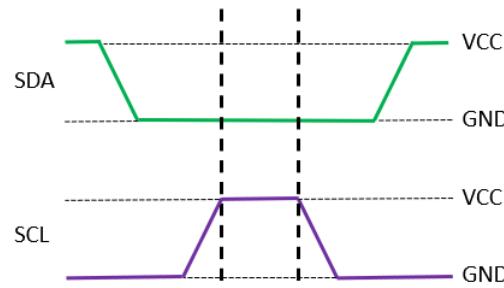


FIGURE 1.31 – Signal d'acquittement I²C.

Comme le protocole de communication **I²C** considère la possibilité d'existence de plusieurs contrôleur, il apparait alors possible de se brancher sur les lignes **SDA**

et **SCL** afin de se faire passer pour un contrôleur légitime, et alors, de discuter avec les périphériques.

1.2.4 CAN

CAN est l'acronyme de *Controller Area Network* [20]. Il a été développé par l'équipementier automobile **Bosch** dans les années 80. Le **CAN** est un bus série très répandu dans l'automobile mais aussi dans l'aéronautique. Il a été mis en place pour faire face à la complexité croissante des systèmes automobiles et au besoin de connecter plusieurs micro-contrôleurs sur le même bus de données. Ce bus de données est bidirectionnel *half-duplex* dans l'automobile, mais il est unidirectionnel dans l'aéronautique.

Le fait que le **CAN** permette de relier plusieurs micro-contrôleurs est un avantage dans la connectivité et le diagnostic de certaines pannes, mais est également un inconvénient car il apparaît alors possible d'interagir avec n'importe quel micro-contrôleur depuis un micro-contrôleur compromis par un attaquant. Il est alors possible, dans le cas d'un **CAN** automobile, d'interagir avec le micro-contrôleur qui gère la sécurité depuis le micro-contrôleur qui gère le multimédia par exemple.

Chaque équipement connecté sur le bus de données **CAN** est appelé un nœud, qui de part la définition du **CAN**, peut dialoguer avec tous les autres noeuds. Deux fils sont nécessaires pour discuter sur le **CAN** :

- le **CANH** (*CAN High*).
- le **CANL** (*CAN Low*).

Un schéma des noeuds et la visualisation du **CANH** et du **CANL** est présenté sur la figure 1.32.

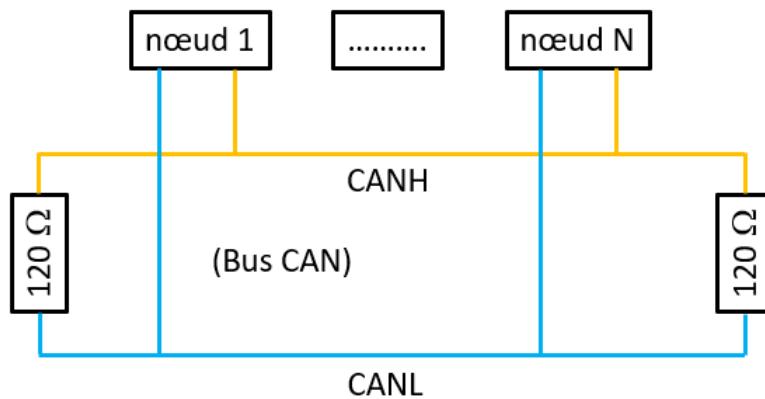


FIGURE 1.32 – Visualisation du bus CAN.

Chaque nœud est constitué d'un micro-contrôleur et d'un contrôleur **CAN** ainsi que d'un organe de transmission qui transforme les *bits* en tension. Les tensions de l'organe de transmission, aussi appelé le *transceiver*, sont de 5 V. Un nœud **CAN** typique est présenté sur la figure 1.33.

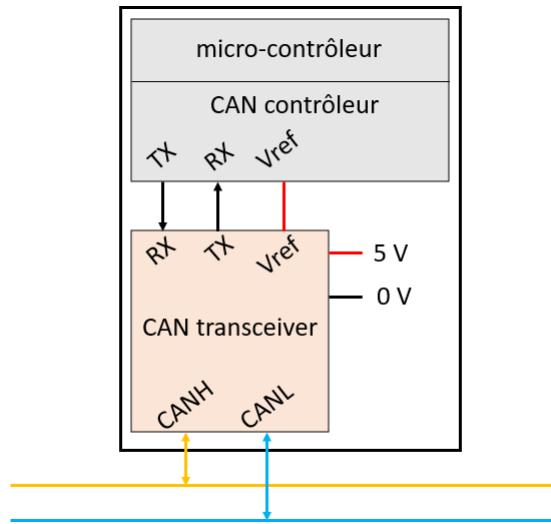


FIGURE 1.33 – Représentation des composants d'un nœud CAN typique.

La figure 1.34 issue d'un **ECU** (*Engine Control Unit*) automobile présente deux *transceivers* capables de transformer un signal issu d'un micro-contrôleur en données accessibles depuis le bus **CAN**. On voit très bien les deux lignes correspondant aux **CANH** et **CANL** sur chaque entrée et sortie des *transceivers*.

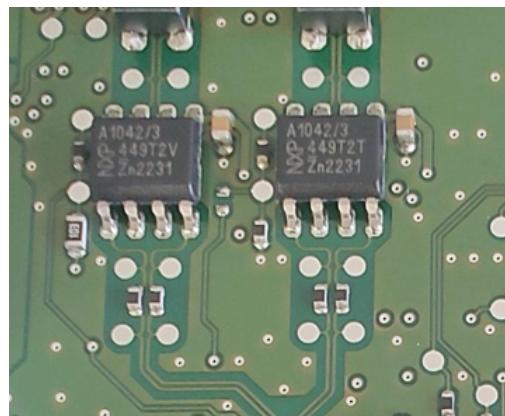


FIGURE 1.34 – Visualisation des **CANH** et **CANL** sur deux *transceiver* d'un **ECU** automobile.

Il existe deux types de vitesse. Une vitesse basse appelée **LS-CAN** (*Low Speed CAN*) et une vitesse haute appelée **HS-CAN** (*High Speed CAN*). Le nombre de nœuds du **LS-CAN** est limité à 20, tandis que celui du **HS-CAN** est limité à 30. Au repos, le voltage du **CAN** est de 2,5 V. Lorsqu'un signal assimilé à un 1 logique arrive sur un bus **HS-CAN**, le **CANH** passe à 3,5 V et le **CANL** à 1,5 V, alors que sur un bus **LS-CAN**, le **CANH** passe à 5 V et le **CANL** à 0 V.

Afin d'être résilient au bruit et aux erreurs, le **CANH** et **CANL** sont antagonistes. Cela signifie que lorsque le **CANH** vaut un 1 logique, le **CANL** vaut un 0 logique, et réciproquement. C'est une manière de vérifier que la valeur qui est lue sur l'une des lignes est correcte.

Comme tout protocole de communication nécessite un bus de données, une communication ne peut avoir lieu que si le bus est disponible. Une trame de

données est constituée des 7 éléments suivants :

- **SOF** (*Start Of Frame*) qui vaut 1 bit et qui indique le début d'une trame.
- **AF** (*Arbitration Field*) qui est constitué de 4 bits définissant le type de l'objet de communication sur la trame, puis d'un identifiant sur 7 bits (pour la version 2.0A du **CAN** et, 11 bits pour la version 2.0B) et d'un bit **RTR** (*Remote Transmission Request*).
- **CF** (*Control Field*) qui est composé de 6 bits. Les deux premiers sont réservés et les quatre suivants constituent le **DLC** (*Data Length Code*) indiquant le nombre d'octets du **DF** (*Data Field*).
- **DF** (*Data Field*) correspond aux données transmises sur le bus et vaut de 0 à 8 bytes.
- **CRC** (*Cyclic Redundancy Code*) qui est composé de 16 bits et qui permet de vérifier l'intégrité des données envoyées sur le bus.
- **ACK** (*ACKnowledgement*) qui est composé de 2 bits et qui permet d'accepter la réception d'un message.
- **EOF** (*End Of Frame*) qui contient 7 bits et qui permet de terminer un message sur le bus.

1.2.5 eMMC

eMMC est l'acronyme de *embedded Multi Media Controller*. Il s'agit de l'optimisation des cartes **MMC** (voir figure 1.35). L'eMMC a un facteur de forme réduit et des performances améliorées. C'est une mémoire de type **NVM** (voir section 1.1.5) qui peut être lue et écrite et qui est dans un boîtier **BGA** (*Ball Grid Array*) comme on peut le voir sur la figure 1.36 issue d'un **PCB** d'un téléphone. Elle est essentiellement utilisée dans les technologies qui nécessitent performance et espace restreint, comme par exemple les téléphones, les caméras de surveillance, les **ECU** automobiles,

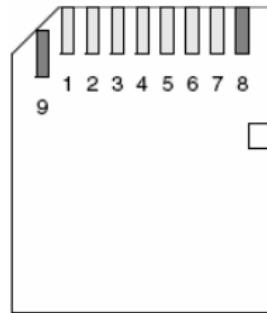


FIGURE 1.35 – Schéma d'une carte **MMC**.

L'eMMC est davantage une norme de stockage *flash* qu'un protocole de communication. Une eMMC est essentiellement une **flash NAND** avec un contrôleur d'erreurs embarqué. Dans le cas de la **flash NAND**, le contrôleur qui contrôlait la **flash** devait aussi se charger de contrôler les éventuelles erreurs. La figure 1.37 présente cette réflexion [11].

L'interface de communication de l'eMMC est basée sur une communication série, généralement en utilisant le protocole **SD** (*Secure Digital*) pour la transmission de données. Ce protocole de communication fait intervenir un contrôleur



FIGURE 1.36 – Visualisation d'une eMMC issue d'un PCB d'un téléphone.

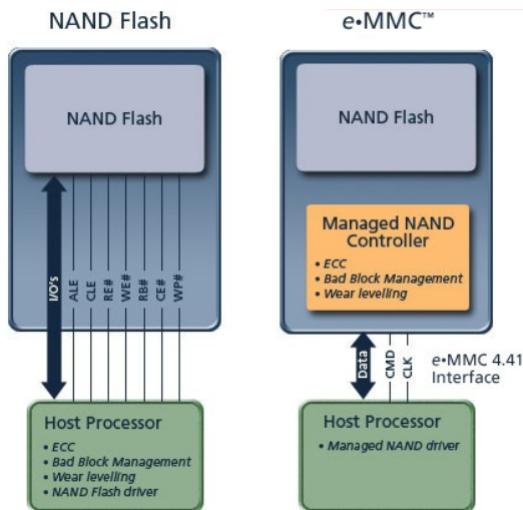


FIGURE 1.37 – Comparaison d'une flash NAND et d'une eMMC [11].

(le plus souvent un micro-contrôleur) qui va se charger de communiquer avec un périphérique (une mémoire eMMC). Toutes les communications sont initiées par le contrôleur. Seize fils sont nécessaires pour relier un contrôleur à une eMMC :

- **CLK (CLock)** qui permet de délivrer un signal d'horloge.
- **RCLK** est un signal émit par le périphérique qui est utilisé par **DAT₀₋₇** et **CMD**. Il est calqué sur le signal d'horloge et permet de transmettre l'information sur les fronts montants et descendants. Une résistance de tirage de 50-100 kΩ est nécessaire.
- **DAT₀₋₇** correspond aux 8 lignes de données. Chaque ligne à une résistance de tirage d'environ 100 kΩ.
- **CMD (CoMmanD)** permet d'initialiser le périphérique depuis le contrôleur, mais également de recevoir les réponses du périphérique depuis le contrôleur.
- **RST (ReSeT)** est un signal de remise à zéro du contrôleur.
- **VCC** est la ligne d'alimentation pour le périphérique.
- **VCCQ** est la ligne d'alimentation qui permet de fournir l'énergie nécessaire à la transmission d'information sur les lignes de données.
- **VSS/VSSQ** sont les lignes de terre pour le périphérique et les lignes de données.

Un schéma d'un *pinout* typique d'une eMMC sur lequel on voit les *pins* énoncés ci-dessus est présenté sur la figure 1.38. Durant chaque cycle d'horloge, 1

bit de **CMD** est transféré. En revanche, pour les signaux de données (**DAT₀₋₇**), 1 ou 2 *bits* peuvent être transférés en fonction de la configuration.

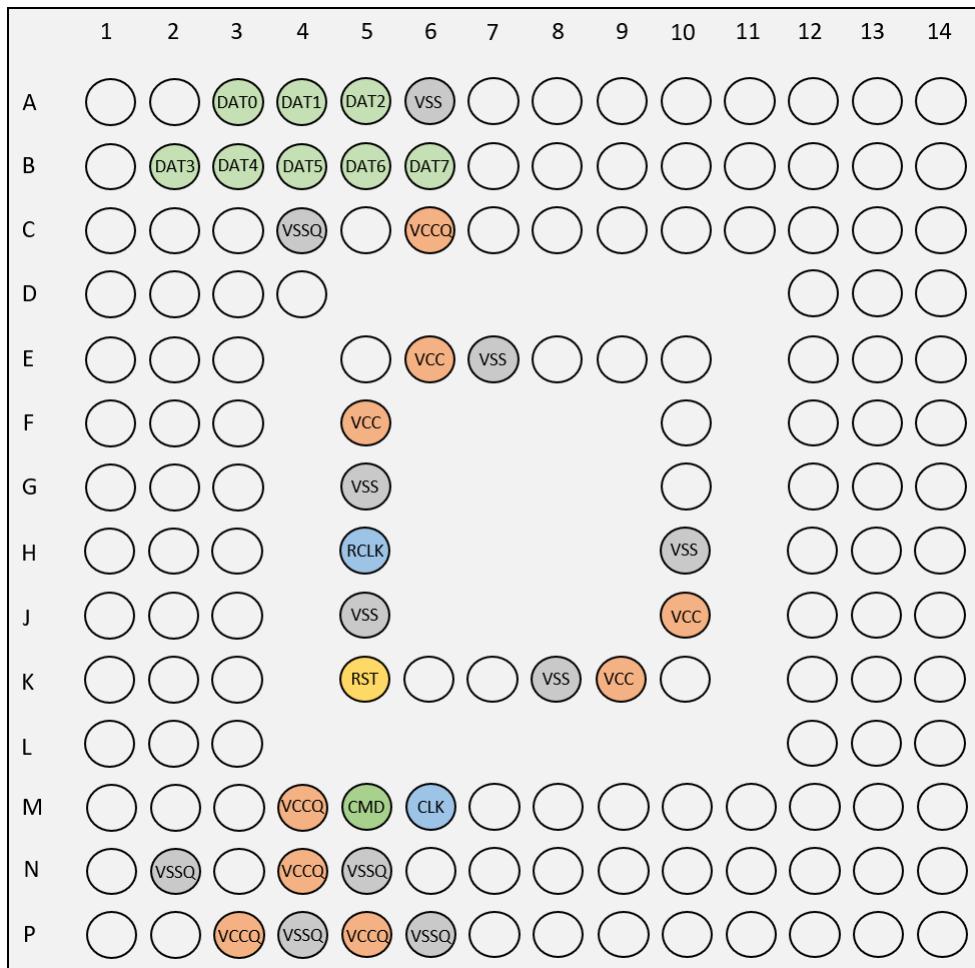


FIGURE 1.38 – Schéma d'un *pinout* typique d'une eMMC.

En raison de sa conception intégrée et de ses coûts abordables, l'eMMC est largement utilisé dans les appareils grand public qui nécessitent une mémoire de stockage fiable mais qui n'ont pas besoin de performances extrêmes. Cependant, pour les applications nécessitant des vitesses de transfert plus élevées, les disques **SSD** (*Solid State Drives*) basés sur la mémoire *flash NAND* sont souvent préférés, car ils offrent des performances supérieures grâce à des interfaces de communication plus rapides comme **SATA** ou **PCI Express**.

1.2.6 Autres protocoles

1-Wire

Le protocole **1-Wire** est un protocole de communication série utilisé pour interconnecter des dispositifs électroniques avec un seul fil de communication. Conçu par la société **Dallas Semiconductor**, maintenant connue sous le nom de **Maxim Integrated**, le protocole **1-Wire** permet de transférer des données, d'alimenter et de contrôler des périphériques à l'aide d'un seul fil de données. Ce fil unique relie tous les périphériques en série, formant ainsi un réseau **1-Wire**.

Ce protocole utilise une méthode de communication à *half-duplex*, où les données sont transmises dans les deux sens sur le même fil, permettant ainsi une mise en œuvre simple et économique. Les dispositifs *1-Wire* sont utilisés dans une variété d'applications, allant des capteurs et des mémoires aux dispositifs d'identification tels que les puces d'identification **RFID** (*Radio Frequency IDentification*). Sa conception à un seul fil en fait une solution attrayante pour les systèmes où l'espace est limité ou lorsque le câblage doit être réduit au minimum.

RS-232

Le protocole **RS-232** est un protocole de communication série largement utilisé pour établir des connexions entre des équipements électroniques, tels que des ordinateurs, des modems, des imprimantes et d'autres périphériques. Développé par l'**Electronic Industries Alliance** dans les années 1960, le protocole **RS-232** définit les normes de communication série pour les équipements électroniques.

Ce protocole utilise une communication asynchrone. Les données sont transmises bit par bit, avec un bit de départ et un ou plusieurs bits d'arrêt pour délimiter chaque octet de données. Il utilise des niveaux de tension positifs et négatifs pour représenter les bits logiques 1 et 0, respectivement.

Une connexion **RS-232** se compose généralement de deux fils de communication, l'un pour la transmission de données (**TX**) et l'autre pour la réception de données (**RX**). Ce protocole définit également les signaux de commande pour contrôler le flux de données, tels que les signaux de demande de transmission (**RTS** : *Request to Send*) et de prêt à recevoir (**CTS** : *Clear to Send*).

Bien que le protocole **RS-232** ait été largement utilisé dans le passé pour les connexions série entre ordinateurs et périphériques, il est devenu moins courant avec l'avènement des technologies de communication plus rapides et plus modernes, telles que l'**USB** (*Universal Serial Bus*) et les connexions **Ethernet**. Cependant, il reste encore utilisé dans certains domaines industriels et pour des applications spécifiques où une connexion série fiable et simple est nécessaire.

SATA

Le protocole **SATA** (*Serial ATA*) est une norme de communication série utilisée pour connecter des dispositifs de stockage de données, tels que les disques durs et les lecteurs **SSD** (*Solid State Drive*), à des ordinateurs et d'autres équipements électroniques. **SATA** est conçu pour remplacer les anciennes interfaces de disque dur parallèles, telles que l'**IDE** (*Integrated Drive Electronics*) et l'**ATA** (*Advanced Technology Attachment*), en offrant des avantages de performance et de fiabilité supérieurs.

Le protocole **SATA** utilise une communication série, ce qui signifie que les données sont transmises bit par bit sur un seul fil de communication. Contrairement aux interfaces parallèles, le **SATA** utilise des fils de données plus simples et plus minces, ce qui facilite la conception des câbles et réduit l'encombrement. En outre, le protocole **SATA** prend en charge des taux de transfert de données

plus élevés que les interfaces parallèles, ce qui permet des transferts de données plus rapides.

Les câbles **SATA** sont généralement dotés de connecteurs rectangulaires à 7 broches pour les périphériques de stockage, et les connexions sont conçues pour être à chaud, ce qui signifie que les périphériques **SATA** peuvent être connectés ou déconnectés alors que le système est en marche.

Le protocole **SATA** est devenu la norme dominante pour les connexions de stockage dans les ordinateurs et les serveurs en raison de ses performances élevées, de sa fiabilité et de sa facilité d'utilisation. Il existe différentes versions de **SATA**, telles que **SATA I** (1.5 Gbps), **SATA II** (3 Gbps), **SATA III** (6 Gbps) et **SATA Express**, qui permettent des taux de transfert de plus en plus rapides pour répondre aux besoins croissants de stockage de données et d'applications gourmandes en performances.

PCI Express

Le protocole **PCI Express** est une norme de communication série utilisée pour interconnecter des composants internes d'un ordinateur, tels que les cartes graphiques, les cartes réseau, les cartes d'extension et les **SSD**, à la carte mère. Le **PCI Express** a remplacé les anciennes interfaces de bus parallèles, telles que le bus **PCI** (*Peripheral Component Interconnect*) et l'**AGP** (*Accelerated Graphics Port*), en raison de sa grande évolutivité et de ses performances élevées.

Contrairement aux interfaces de bus parallèles, le protocole **PCI Express** utilise une communication série, ce qui signifie que les données sont transmises bit par bit sur des voies de communication distinctes, appelées **voies PCI Express**. Chaque **voie PCI Express** est bidirectionnelle et peut transférer des données à grande vitesse, ce qui permet des taux de transfert de données élevés.

En raison de ses performances élevées et de sa grande flexibilité, le protocole **PCI Express** est largement utilisé dans les ordinateurs modernes, des ordinateurs de bureau aux serveurs en passant par les systèmes embarqués. Au fil du temps, le **PCI Express** a évolué pour offrir des taux de transfert de données de plus en plus rapides, comme **PCI Express 2.0**, **PCI Express 3.0**, **PCI Express 4.0** et **PCI Express 5.0**, permettant de répondre aux besoins croissants des applications gourmandes en bande passante et en vitesse de transfert.

USB

Le protocole **USB** (*Universal Serial Bus*) est une norme de communication série largement utilisée pour connecter des périphériques électroniques à des ordinateurs et à d'autres appareils. Introduit pour la première fois en 1996, l'**USB** est devenu l'une des interfaces les plus répandues dans le monde de l'informatique et de l'électronique grand public en raison de sa simplicité, de sa polyvalence et de sa commodité.

Le protocole **USB** utilise une communication série, ce qui signifie que les données sont transmises bit par bit sur des fils de communication distincts. Une connexion **USB** standard comprend généralement quatre fils : deux fils pour les données (**D₊** et **D₋**) et deux fils pour l'alimentation électrique (**VCC** et **GND**).

L'**USB** est conçu pour être à chaud et à brancher, ce qui signifie que les périphériques **USB** peuvent être connectés ou déconnectés à tout moment sans avoir besoin de redémarrer l'ordinateur ou l'appareil hôte. Cela facilite grandement l'ajout et le retrait de périphériques sans interrompre le fonctionnement du système.

L'**USB** est utilisé pour connecter une grande variété de périphériques, tels que les claviers, les souris, les imprimantes, les scanners, les disques durs externes, les caméras, les smartphones, les tablettes, les lecteurs de musique, les appareils photo, les contrôleurs de jeux, et bien plus encore. L'évolution de l'**USB** au fil des années a conduit à différentes versions, telles que l'**USB 1.1**, l'**USB 2.0**, l'**USB 3.0**, l'**USB 3.1** et l'**USB 3.2**, avec des taux de transfert de données de plus en plus rapides et des améliorations de la puissance délivrée aux périphériques.

FireWire

Le protocole **FireWire**, également connu sous le nom d'**IEEE 1394**, est une norme de communication série haut débit utilisée pour connecter des périphériques électroniques à des ordinateurs et à d'autres appareils. **FireWire** a été développé par **Apple** dans les années 1990 et a été adopté comme norme **IEEE 1394** pour une utilisation plus large dans l'industrie de l'électronique.

Tout comme l'**USB**, le protocole **FireWire** utilise une communication série, mais il se distingue par sa vitesse de transfert de données élevée. Il existe différentes versions de **FireWire**, telles que **FireWire 400** avec des taux de transfert allant jusqu'à 400 Mbps et **FireWire 800** avec des taux de transfert allant jusqu'à 800 Mbps. Certaines versions plus récentes de **FireWire**, comme **FireWire S800T**, ont été développées pour atteindre des vitesses de transfert encore plus élevées.

Ethernet

Le protocole **Ethernet** est une norme de communication utilisée pour interconnecter des périphériques informatiques dans un réseau local ou un réseau étendu. Il a été développé au début des années 1970 et est maintenant largement utilisé dans le monde entier pour permettre la communication entre les ordinateurs, les serveurs, les routeurs, les commutateurs et d'autres équipements réseau.

Le protocole **Ethernet** utilise une communication série basée sur des trames, ce qui signifie que les données sont divisées en paquets appelés **trames Ethernet**. Chaque trame contient des adresses source et de destination, ainsi que les données à transférer. Les **trames Ethernet** sont ensuite transmises sur le réseau en utilisant des câbles **Ethernet**.

Ethernet est souvent associé à des câbles à paires torsadées, mais il peut également être utilisé avec des câbles coaxiaux ou à fibre optique. La technologie la plus couramment utilisée aujourd’hui est l’**Ethernet** à paires torsadées, comme le **Gigabit Ethernet** ou le **10 Gigabit Ethernet**, qui offre des vitesses de transfert de données allant jusqu’à plusieurs gigabits par seconde.

Le protocole **Ethernet** est utilisé pour permettre une communication fiable et rapide entre les appareils dans un réseau. Il est basé sur le principe **CSMA/CD** (*Carrier Sense Multiple Access with Collision Detection*), qui permet aux appareils de détecter et de gérer les collisions de données qui se produisent lorsque plusieurs appareils tentent de transmettre des données en même temps.

Ethernet est devenu la norme pour les réseaux locaux et d’entreprise en raison de sa simplicité, de sa fiabilité et de sa large adoption. Il est également largement utilisé dans les infrastructures **Internet**, où il sert de base pour de nombreux réseaux et inter-connecte les différents nœuds du réseau **Internet** mondial. **Ethernet** continue de se développer avec l’évolution des technologies, offrant des débits de plus en plus élevés et de nouvelles fonctionnalités pour répondre aux besoins croissants en matière de communication et de connectivité.

1.3 Les interfaces de diagnostic

1.3.1 JTAG

L’interface **JTAG** (*Join Test Action Group*) a été mise en place dans les années 80 par le groupe *Join Test Action Group* pour résoudre les problèmes de connections entre différents composants d’un circuit électronique. Avec l’apparition des composants **SMD**, il devenait difficile de déterminer la cause d’une panne dans un circuit. Il a alors été imaginé une interface qui relierait chaque composants grâce à un bus série de 4 fils :

- **TMS** (*Test Mode Select*) qui permet de sélectionner le composant à diagnostiquer. Il doit avoir une résistance de tirage d’environ $100\text{ k}\Omega$.
- **TCK** (*Test ClocK*) qui est une ligne d’horloge. Une résistance de rappel d’environ $100\text{ k}\Omega$ est nécessaire.
- **TDI** (*Test Data In*) qui est utilisé pour écrire des données sur le composant à diagnostiquer. Une résistance de tirage d’environ $100\text{ k}\Omega$ est requise.
- **TDO** (*Test Data Out*) qui est utilisé pour lire des données depuis le composant à diagnostiquer.

Ces 4 fils, qui font partie d’un bus série synchrone, permettent d’agir sur un registre d’instruction et un registre de données qui sont accessibles à travers une machine à état qui contrôle l’interface **JTAG**. Comme une machine à état est impliquée dans l’interface **JTAG**, il peut être intéressant d’inclure, entre autre, des lignes supplémentaires au bus (optionnelle) : **TRST** (*Test ReSeT* qui est *active low*) qui permet de remettre la machine à état à son état initial, ou bien **RTCK** (*Returned Test Clock*) ou encore **NRST** (*Neutral Test Reset*). La machine à état permet de traiter les registres d’instruction et de données comme un registre à décalage, et ainsi enchaîner les différents composants les uns à la suite des autres.

La figure 1.39 présente cet enchainement.

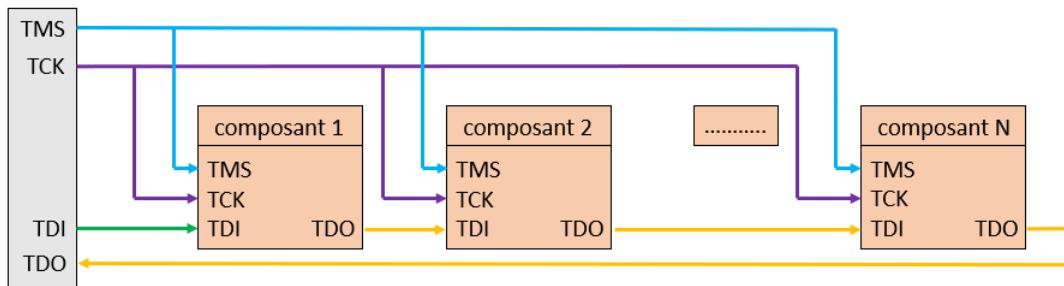


FIGURE 1.39 – Visualisation des bus utilisés pour l’interface **JTAG**.

L’interface **JTAG** est souvent rendue disponible à travers un connecteur comme on peut le voir sur la figure 1.40. Bien que l’interface **JTAG** nécessite un minimum de 4 fils comme cela a pu être explicité, il n’est pas rare d’en voir davantage en fonction des lignes supplémentaires optionnelles. Cependant, sur la figure 1.40 issue d’un **PCB** d’un disque dur **SATA**, on constate que le nombre de pattes disponibles est bien trop élevé pour que toutes correspondent au **JTAG**. Il est donc possible que d’autres interfaces de diagnostiques soient présentes sur ce même connecteur. Il est à noter qu’il n’est pas rare que le connecteur associé à l’interface **JTAG** ait 14 ou 20 pattes.

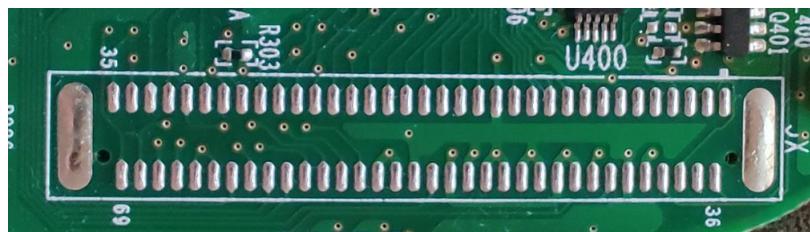


FIGURE 1.40 – Visualisation d’un emplacement pour un connecteur associé à l’interface **JTAG** issu d’un disque dur **SATA**.

Le registre d’instruction de la machine à état permet de déterminer l’action à effectuer, comme par exemple la lecture (ou l’écriture) d’une zone-mémoire. Le registre de données est ensuite utilisé comme un paramètre supplémentaire au registre d’instruction. Dans le cas d’une lecture de zone-mémoire, le registre de données peut contenir l’adresse à laquelle lire.

1.3.2 SWD

SWD est l’acronyme de *Single Wire Debug*. C’est une interface de *debug* utilisée par la famille de processeur **ARM Cortex** [3]. Contrairement au **JTAG**, l’interface **SWD** ne requiert qu’un seul bus de données : **SDWIO**. Les fils à câbler pour utiliser l’interface **SWD** sont les suivant :

- **SWDIO** (*SWD Input Output*) qui permet de lire/écrire sur le bus de données. Il doit avoir une résistance de tirage d’environ 100 kΩ.

- **SWDCLK** (*SWD CLocK*) qui est un signal d'horloge nécessaire à la transmission de l'information sur le bus de données. Une résistance de rappel d'environ $100\text{ k}\Omega$ est nécessaire.
- **RESET** est une ligne *active low* permettant la remise à zéro du microprocesseur.

L'interface **SWD** a pour objectif de remplacer l'interface **JTAG** et de rendre le diagnostic plus simple sur les processeurs **ARM Cortex**, notamment en réduisant le nombre de bus nécessaires. Le **SWD** s'interface avec un bus interne appelé **DAP** (*Debug Access Port*). Le **DAP** permet donc d'accéder à différents ports d'accès appelés **AP** (*Access Port*). A la différence du **JTAG**, le **SWD** ne permet pas d'enchaîner les composants.

1.4 Méthodologie d'analyse

La méthodologie d'analyse de **PCB** du point de vue sécurité offensive repose sur une approche systématique et rigoureuse visant à identifier les vulnérabilités potentielles et à évaluer la sécurité des circuits imprimés. L'analyse statique implique une étude approfondie de la conception des **PCB**, examinant les schémas, les connexions, les composants et les couches du circuit pour repérer les faiblesses éventuelles. Cette phase permet de détecter des erreurs de conception, des points d'accès sensibles et des emplacements propices à des intrusions malveillantes, mais aussi d'émettre des hypothèses qui seront à valider dans la phase d'analyse dynamique.

L'analyse dynamique, quant à elle, se concentre sur la manipulation du **PCB** en action, en simulant des scénarios réels et en observant les comportements du circuit lorsqu'il est en fonctionnement. Cela peut inclure des tests de résistance aux surtensions, des attaques de rétro-ingénierie sur les signaux émis ou reçus, et des mesures de consommation d'énergie pour détecter des activités suspectes.

Une fois les vulnérabilités identifiées, l'élaboration des plans d'attaques consiste à évaluer les différentes approches possibles pour exploiter ces failles. Les chercheurs en sécurité offensive se penchent sur des méthodes telles que l'injection de signaux, l'utilisation de sondes spécifiques, l'exploitation de débordements de mémoire ou d'autres vecteurs d'attaque sophistiqués.

La réalisation des attaques constitue l'étape où les chercheurs mettent en œuvre les plans élaborés pour évaluer leur faisabilité et déterminer les conséquences potentielles. Ils peuvent simuler des attaques sur des prototypes ou des échantillons de **PCB** pour évaluer leur impact sur les systèmes électroniques.

Enfin, la conclusion de l'analyse de **PCB** en sécurité offensive se concentre sur la synthèse des résultats obtenus. Les chercheurs évaluent l'efficacité des attaques réussies, énoncent des recommandations pour renforcer la sécurité du circuit, et mettent en évidence les enseignements tirés de l'expérience. Cette étape cruciale permet de mieux comprendre les points faibles et de proposer des mesures de défense adéquates pour protéger les systèmes électroniques contre les menaces potentielles.

En adoptant cette méthodologie rigoureuse, les professionnels de la sécurité offensive peuvent contribuer à améliorer la robustesse et la fiabilité des **PCB**, renforçant ainsi la sécurité globale des technologies électroniques dans un environnement de plus en plus exposé aux risques et aux attaques malveillantes.

Chapitre 2

Analyse d'un keylogger hardware

Le *keylogger hardware* (ou encore *keygrabber hardware*) est un appareil physique (à comparer avec le *keylogger software* qui s'installe sur un système d'exploitation) qui se branche entre le clavier et l'ordinateur. Ainsi les frappes-clavier sont enregistrés directement dans le *keylogger*. De manière générale, les *keylogger hardware* propose une interface sans fil (wifi, bluetooth, radio) de manière à récupérer le contenu des frappes-clavier sans avoir besoin de récupérer le dispositif. La figure 2.1 présente l'allure d'un tel appareil.



FIGURE 2.1 – Allure d'un *keylogger hardware*.

On distingue que cet appareil possède un port **USB** femelle (à gauche) et un port **USB** mâle (à droite) de manière à venir se positionner entre le clavier et l'ordinateur. Une fois la coque de protection enlevée, le **PCB** présenté sur la figure 2.2 est alors visible.

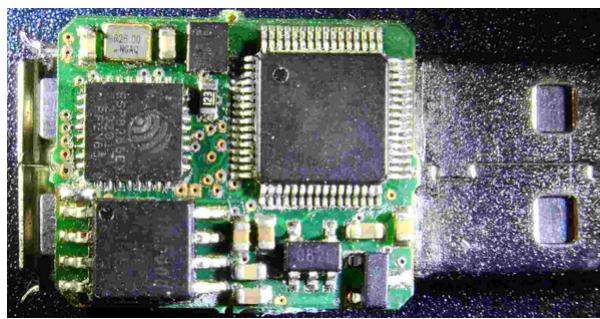


FIGURE 2.2 – Visualisation du **PCB** du *keylogger hardware*.

Le **PCB** peut être divisé en 4 zones. La première zone, en haut à droite, contient un **CPLD Intel Altera 5M160ZE64C5N** dans un boîtier **TQFP-64** cadencé à 152 MHz et alimenté en courant continu de 1,8 V. Un **SOC Espressif ESP8266EX** dans un boîtier **QFN-32** alimenté en 3,3 V et proposant une interface wifi est positionné en haut à gauche dans la seconde zone. En bas à droite,

dans la troisième zone, on peut y voir un étage d'alimentation, permettant d'abaisser la tension de 5 V issue du port **USB**. Et enfin, dans la quatrième zone, en bas à gauche, se trouve une mémoire **flash NOR Winbond W25Q128FVSG** de 128 Mbit (soit 16 Mo) dans un boîtier **SOP-8** alimentée en courant continu de 3,3 V. Chaque zone sera explicitée en détails dans les sections suivantes.

Il est possible de réaliser un masque du **PCB** comme cela est présenté sur la figure 2.3. Sur ce masque sont représentés les différents composants d'après la nomenclature détaillée dans l'annexe A.4, mais aussi les différents vias visibles (certains peuvent se cacher sous les composants). Ce masque permettra de nommer les composants et de savoir où ils sont placés. Il sera également possible de compléter ce masque en y ajoutant les données issues de l'analyse statique et dynamique (voir sections 2.1 et 2.2).

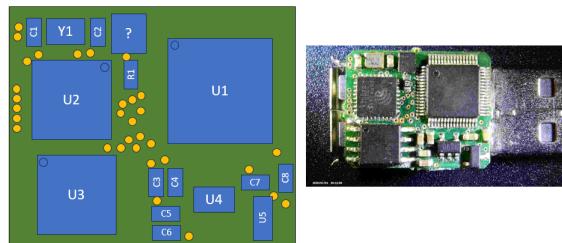


FIGURE 2.3 – Réalisation d'un masque du **PCB** du *keylogger hardware*.

2.1 Analyse statique du PCB

2.1.1 Zone 1 - CPLD 5M160ZE64C5N

L'**Altera 5M160ZE64C5N** (voir figure 2.4 qui représente un zoom sur le composant **U1**) est un **CPLD** (*Complex Programmable Logic Device*). C'est un circuit intégré constitué de blocs fonctionnels programmables qui permet d'effectuer de la logique combinatoire ou séquentielle. Ces blocs sont appelés des macro-cellules (*macrocells* en anglais). Chaque macro-cellule contient un certains nombre d'éléments logiques (portes **OR/AND**). L'utilisateur doit programmer le circuit avant de pouvoir l'utiliser. Le **CPLD** est considéré comme une version édulcorée d'un **FPGA** (*Field Programmable Gate Array*) lequel possède un nombre bien plus important de cellules élémentaires.

Dans le cas du *keylogger hardware*, l'**Altera 5M160ZE64C5N** est programmé pour agir sur la *stack USB*. C'est donc ce composant qui récupère le signal d'entrée du clavier, le décode afin de récupérer la touche qui a été tapée, puis le renvoie sur le port USB de sortie afin que l'ordinateur puisse afficher la lettre. Un **CPLD** est de manière générale moins rapide qu'un **FPGA** ou qu'un **ASIC** (*Application-Specific Integrated Circuit*), mais reste ce qu'il y a de plus rapide pour éviter les latences lors du traitement du signal issu du clavier. Ainsi, l'utilisateur ne se rend pas compte que ces frappes-clavier sont en train d'être interceptées.

En regardant la documentation de ce composant [2], il est possible de retranscrire son empreinte et d'y associer les pins. L'empreinte est présentée sur la



FIGURE 2.4 – Zoom sur le CPLD Altera 5M160ZE64C5N du *keylogger hardware*.

figure 2.5. On y voit des pins généraux qui correspondent à des entrées/sorties, des pins d'alimentation, mais aussi les pins nommés **TMS**, **TDI**, **TCK** et **TDO**. Ces pins sont utilisés par le protocole de diagnostic **JTAG** (voir section 1.3.1).

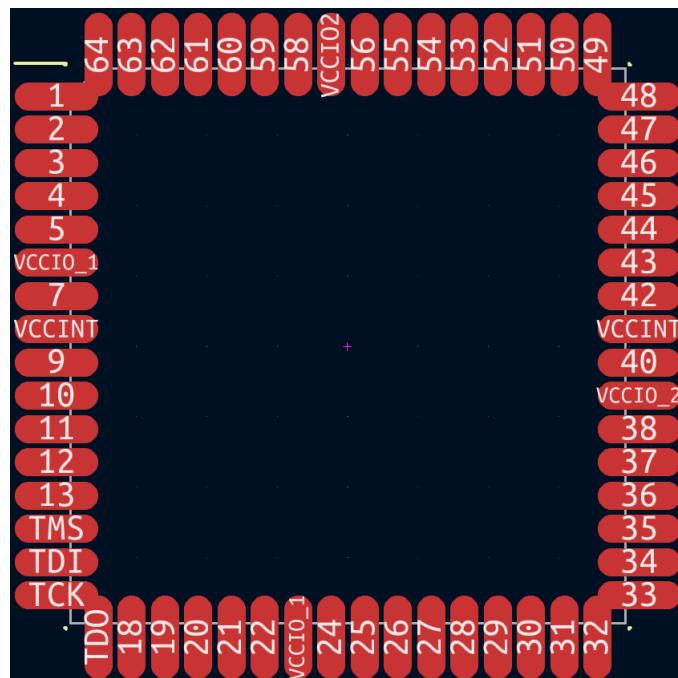


FIGURE 2.5 – Empreinte d'un CPLD Altera 5M160ZE64C5N.

Juste à côté du **CPLD**, on peut voir en bas à gauche de celui-ci, des vias. Ceux-ci sont situés à proximité des pins correspondant au **JTAG**. Un test de continuité est alors effectué afin de s'assurer qu'à chaque via correspond bien un pin associé au **JTAG**. Ces résultats ont été retranscrits sur le masque que l'on peut voir sur la figure 2.6.

2.1.2 Zone 2 - SoC ESP8266EX

L'**ESP8266EX**, est un **SoC** (*System on Chip*) très utilisé par les *hobbyists* dans le domaine des systèmes embarqués. En effet, ce composant (voir figure 2.7 qui représente un zoom sur le composant **U2**) contient un **CPU**, une **SRAM**, une **ROM**, mais peut aussi communiquer avec une **flash** externe comme c'est le

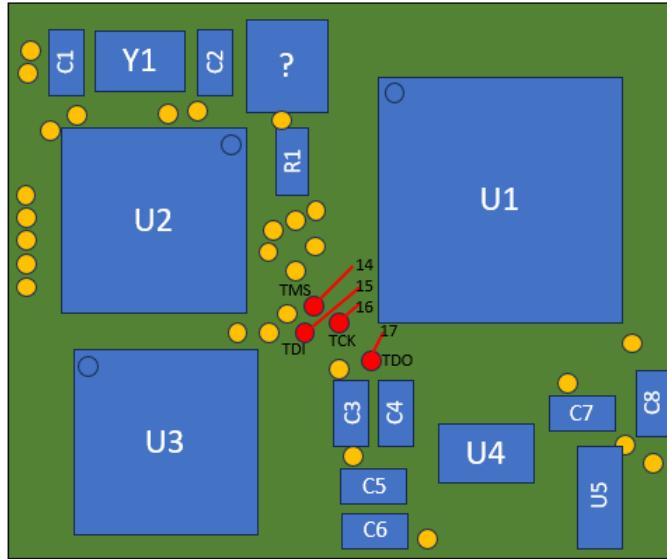


FIGURE 2.6 – Visualisation des pins correspondants au **JTAG** sur le masque du *keylogger hardware*.

cas dans le *keylogger*. L'**ESP8266EX** gère aussi le wifi et la radio (2,4 GHz) et peut également proposer un signal d'horloge, et communiquer suivant toute une série de protocoles. Ici, l'**ESP8266EX** est utilisé pour proposer un point d'accès wifi auquel se connectera l'attaquant souhaitant récupérer les frappes-clavier.

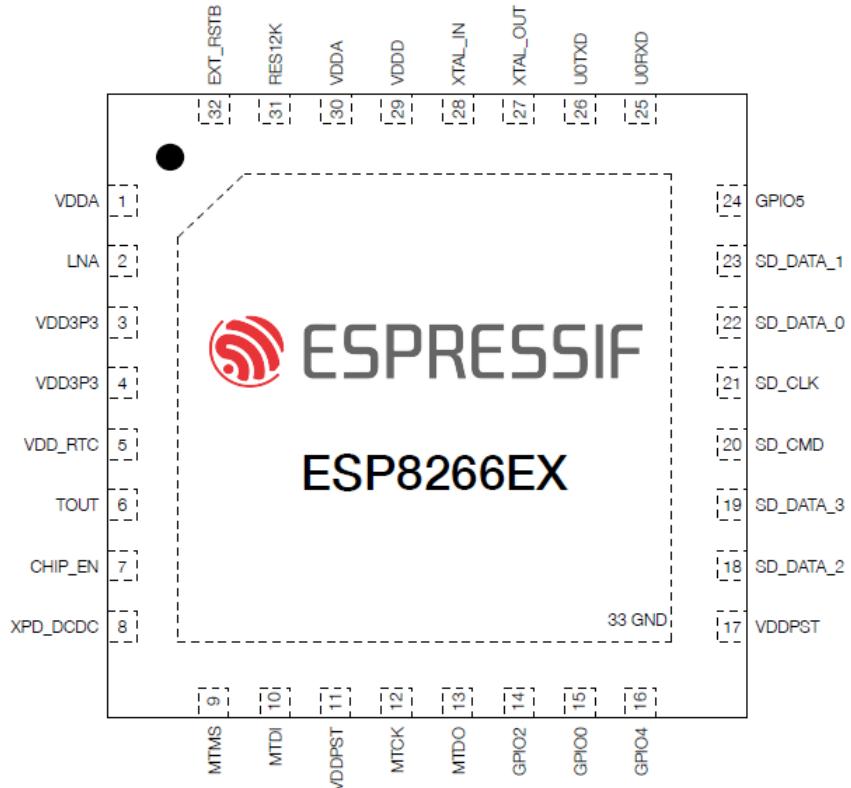


FIGURE 2.7 – Zoom sur l'esp8266 du *keylogger hardware*.

Le *pinout* de l'**ESP8266EX** est présenté sur la figure 2.8. La documentation [5] indique que durant la phase démarrage, **UART0** devrait afficher des informations depuis les pins **U0RXD** et **U0TXD**. Cependant la forme du boîtier rend difficile les mesures de tests de continuité entre ces pins et les vias disponibles. Le *baudrate* de l'**UART** dépend de la fréquence d'horloge fournie par le cristal externe **Y1** ($f = 40$ MHz équivaut à un *baudrate* de 115200, et $f = 26$ MHz équivaut à un *baudrate* de 74880)

La documentation indique également que le pin 31 doit être relié à une résistance de $12\text{ k}\Omega$, ce qui est bien le cas car on peut voir que le composant **R1** est une résistance sur laquelle est indiqué **123**, ce qui correspond à $12000\ \Omega$ soit $12\text{ k}\Omega$.

En ce qui concerne les vias situés tout à gauche, on peut se rendre compte qu'ils semblent reliés au pins situés en face, à savoir (de haut en bas) : **MTCK** (12), **MTDO** (13), **GPIO2** (14), **GPIO0** (15) et **GPIO4** (16). Encore une fois,

FIGURE 2.8 – Pinout d'un **ESP8266EX**.

l'encombrement stérique inhérent à la forme du boîtier et à la petitesse du **PCB** empêchent de vérifier cela par un test de continuité. Cependant, ces pins étant d'une grande importance sur l'**ESP8266**, on peut estimer que cette hypothèse est valide. D'après la documentation, le pin **GPIO4** est associé au **PWM** (*Pulse-Width Modulation*), les pins **GPIO0**, **GPIO2** sont utilisés pour choisir le mode de démarrage, **MTDO** est utilisé pour sélectionner le mode **SDIO** (*Secure Digital Input/Output*), et **MTCK** est utilisé comme signal d'horloge pour le **SPI**. On peut donc estimer qu'il s'agit bien d'un port utilisé pour mettre à jour le *firmware*. Le pin **GPIO0** est utilisé pour mettre à jour le *firmware* par **USB** lorsque qu'une résistance de rappel lui est associée durant la phase de *boot*. En mode d'exécution normal, il est censé être associé à une résistance de tirage. Cette dernière n'est pas visible, les concepteurs ont dû ne pas la mettre intentionnellement, probablement pour limiter le nombre de composants sur le **PCB** de manière à le garder le plus petit possible.

La figure 2.9 permet de voir l'ajout de ces résultats sur le masque.

2.1.3 Zone 3 - Étage d'alimentation

Souvent lorsque l'on met en place un étage d'alimentation, on utilise un **LDO** (voir section 1.1.5) comme régulateur de tension. Cependant, d'autres méthodes existent comme utiliser une **diode Zener** associée à une **transistor**. Dans le cadre de ce *keylogger hardware* un **LDO** dans un boîtier **SOT-23-5** (composant **U4**) est utilisé afin d'alimenter l'**ESP8266EX**, la **flash** et **VCCIO1/2** (6, 23, 39 et 57) du **CPLD** et une **diode Zener** dans un boîtier **SOT-23** (composant

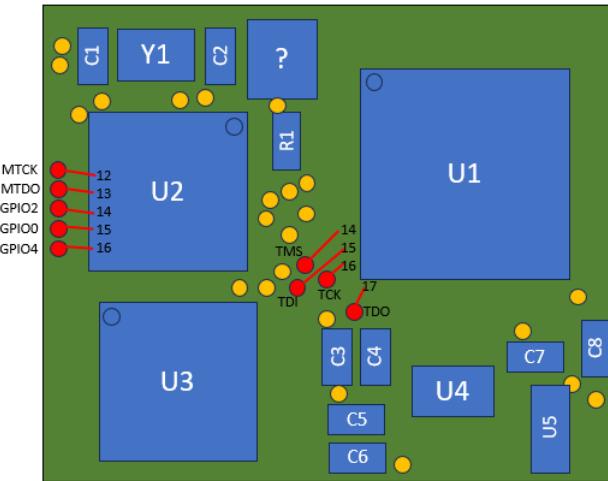


FIGURE 2.9 – Visualisation des pins d'intérêt de l'**ESP8266EX** sur le masque du *keylogger hardware*.

U5) est utilisée pour alimenter le **CPLD** sur la patte **VCCINT** (8 et 41) (voir figure 2.10).



FIGURE 2.10 – Zoom sur l'étage d'alimentation du *keylogger hardware*.

Les tensions aux bornes des pattes du **LDO** seront présentés dans l'analyse dynamique. En ce qui concerne l'utilisation d'une **diode Zener** comme régulateur de tension, il faut noter que celle-ci ne fonctionne pas tout à fait comme une diode classique qui ne laisse passer le courant que dans un seul sens. La **diode Zener** peut laisser passer le courant dans le sens inverse si la tension sur la patte de sortie est suffisamment importante. Le terme "suffisamment" fait référence à une tension seuil qui est appelée "tension d'avalanche" ou "tension Zener". Si la tension est supérieure ou égale à la tension Zener, la **diode Zener** maintient la tension à cette valeur, en l'occurrence 1,8 V sur la patte d'entrée qui se retrouve être la patte de sortie (vu qu'on est sur signal qui va en sens inverse). Afin de servir de régulateur de tension, elle doit donc être montée en sens inverse de manière à laisser passer le courant en sens inverse de la diode (de la sortie vers l'entrée). Les quelques condensateurs qui sont présents sont utilisés comme condensateurs de découplages pour s'assurer d'avoir un signal le plus propre possible en entrée des composants majeurs (**CPLD**, **ESP8266EX** et la **flash NOR**).

2.1.4 Zone 4 - Flash NOR W25Q128FVSG

La documentation [25] indique que la **flash NOR** (composant **U3**) est une **flash SPI** de 16 Mo. Elle communique donc avec le **Soc ESP8266EX** via un bus **SPI**. Son *pinout* est présenté sur la figure 2.11.

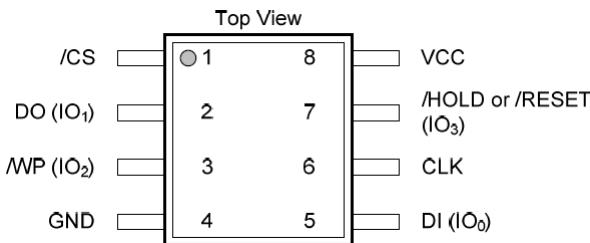


FIGURE 2.11 – Pinout de la flash W25Q128FVSG du keylogger hardware.

Un test de continuité permet d'affirmer qu'elle est bien alimentée en 3,3 V par la sortie du **LDO**. Mais également qu'est est bien reliée au **Soc ESP8266EX**. En effet, les pins **CLK**, **DI** et **DO** de la **flash** sont reliés au pins **SD_CLK**, **SD_DATA_1** et **SD_DATA_2** de l'**ESP8266EX**.

2.2 Analyse dynamique du PCB

2.2.1 Mesures des tensions de l'étage d'alimentation

Pour effectuer des mesures correctes de tension, il faut commencer par trouver une masse. Un **PCB** possède généralement un ou plusieurs plan de masse. Il est possible de gratter la surface jusqu'à atteindre ce plan de masse. Cependant, il existe des approches plus simples et tout aussi efficaces. Par exemple, la partie extérieure d'un port **USB** est métallique et doit être reliée à la masse pour éviter d'éventuels signaux parasites sur les lignes de données. On peut donc y pointer une des sondes du multimètre. Un autre exemple est de regarder la visserie. Dans le cas du *keylogger hardware*, il n'y a pas de visserie, mais prenons l'exemple du *switch* présenté en figure 2.12 contenu dans un boîtier métallique. Le **PCB** doit être maintenu à son boîtier par des vis, et ne voulant pas que le boîtier se transforme en antenne, il est nécessaire de le relier à un plan de masse par l'intermédiaire des vis. On y voit la zone cuivrée sur laquelle est fixé le boîtier. Il s'agit d'une zone de masse que l'on peut utiliser comme référence pour effectuer des mesures de tension. En ce qui concerne le *keylogger hardware* étudié ici, la masse a été prise sur le boîtier **USB**.

Il suffit ensuite de pointer la sonde de tension du multimètre sur les différents pins de l'étage d'alimentation. Les résultats sont présentés sur la figure 2.13. Cela permet de vérifier les tensions qui sont envoyées au **CPLD** sur **VCCIO1** et **VCCINT**, mais aussi sur l'**ESP8266EX** sur **VDDA** et **VDD3P3**.

2.2.2 Mesures des résistances

Ce *keylogger hardware* n'a qu'une seule résistance (**R1**). Les résistances **SMD** sont toujours de couleur noire et leur valeur est généralement inscrite dessus. Ici on peut lire **123**. Ce marquage à trois chiffres permet de connaître la valeur de la résistance. Les deux premiers chiffres représentent la valeur, le troisième est le multiplicateur (en puissance de dix). Ainsi, le marquage **123** se traduit par $12 \times 10^3 = 12 \text{ k}\Omega$.



FIGURE 2.12 – Visualisation de la visserie pour atteindre une zone de masse.

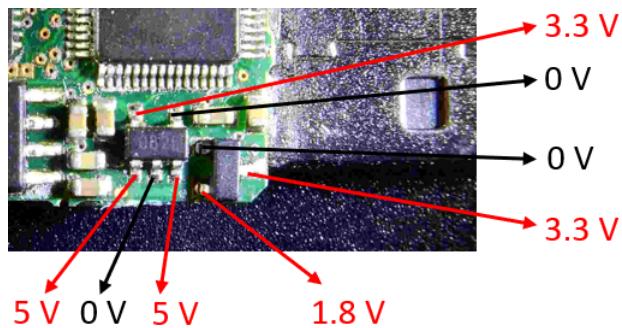


FIGURE 2.13 – Visualisation des différentes tensions de l'étage d'alimentation du keylogger hardware.

Cela a également été vérifié avec un multimètre positionné sur le mode ohmmètre (voir figure 2.14), et l'on détermine expérimentalement que la résistance **R1** vaut $11,9 \text{ k}\Omega$, soit $\sim 12 \text{ k}\Omega$, ce qui correspond à ce que la documentation de l'**ESP8266EX** spécifie.

2.2.3 Mesures des capacités

Il existe plusieurs méthodes pour mesurer la capacité d'un condensateur. Voici quelques-unes des méthodes couramment utilisées :

- Multimètre** : L'utilisation d'un multimètre est la méthode la plus simple et la plus courante pour mesurer la capacité d'un condensateur. Les multimètres numériques modernes disposent souvent d'une fonction de mesure de capacité. Il suffit de connecter les sondes du multimètre aux bornes du condensateur et le multimètre affichera la valeur de capacité.
- Pont de mesure** : Un pont de mesure est un appareil spécifique conçu pour mesurer la capacité des condensateurs. Il utilise un circuit de pont équilibré pour effectuer la mesure de capacité. Cette méthode est plus précise que l'utilisation d'un multimètre, mais elle est également plus coûteuse et moins courante.



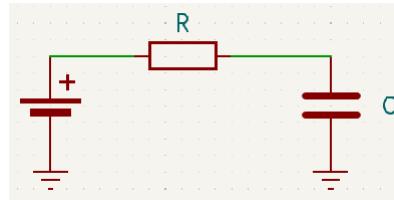
FIGURE 2.14 – Vérification de la valeur de la résistance associée au pin **RES12K** sur l'**ESP8266EX**.

3. **Méthode de charge et de décharge** : Cette méthode implique de charger complètement le condensateur avec une tension connue, puis de le décharger à travers une résistance connue et de mesurer le temps nécessaire pour que la tension chute à un certain seuil. À partir de ces mesures, la capacité du condensateur peut être calculée en utilisant l'équation $Q = C \cdot V$, où **Q** est la charge stockée, **C** est la capacité et **V** est la tension.
4. **Méthode de mesure de la constante de temps** : Cette méthode consiste à charger le condensateur avec une tension connue et à mesurer le temps nécessaire pour que la tension atteigne un pourcentage donné de la tension d'entrée. À partir de cette mesure, la capacité du condensateur peut être calculée en utilisant l'équation $C = \tau / R$, où **C** est la capacité, τ est la constante de temps (en secondes) et **R** est la résistance utilisée dans le circuit.

Il est important de noter que la précision des mesures dépendra de la méthode utilisée et de la qualité de l'équipement de mesure. L'utilisation d'un équipement de mesure approprié et la prise en compte des tolérances des composants sont essentielles pour obtenir des mesures précises de la capacité du condensateur.

La méthode utilisée ici est la mesure de la constante de temps. Comme énoncé ci-dessus, cette méthode nécessite de réaliser un circuit contenant le condensateur dont on cherche la capacité associé à une résistance connue en série. Il s'agit d'un montage que l'on appelle plus communément **montage RC** que l'on peut voir sur la figure 2.15.

Les quelques lignes qui suivent peuvent paraître abruptes pour le non physicien. Elles peuvent donc être omises par le lecteur qui ne s'intéressera alors qu'au résultat. Ceci étant dit, l'équation différentielle 1.6 présentée dans la section 1.1.5 (et retranscrite ci-dessous) est une équation différentielle du premier ordre associée au condensateur. La loi d'Ohm (voir section 1.1.5) stipule également que $U = R \cdot I$. On a ainsi :

FIGURE 2.15 – Visualisation d'un circuit **RC**.

$$I = C \cdot \frac{dU}{dt} = -\frac{U}{R} \quad (2.1)$$

Les solutions de cette équation sont simples et connues comme on peut le voir ci-dessous :

$$U(t) = U_0 + Ae^{-t/RC} \quad (2.2)$$

La constante **A** est déterminée grâce aux conditions initiales. Si à temps initial $t = 0$ on choisit $U(0) = 0$, on a $A = -U_0$, et donc les solutions 2.2 s'écrivent alors :

$$U(t) = U_0(1 - e^{-t/RC}) \quad (2.3)$$

Dans cette équation, R est la valeur de la résistance, C est la valeur du condensateur que l'on cherche à connaître, t est le temps, $U(t)$ est la tension aux bornes du condensateur à chaque instant, et U_0 est la tension d'alimentation. La constante de temps d'un circuit **RC** est alors :

$$\tau = R \cdot C \quad (2.4)$$

Ainsi, connaissant τ et R , il apparaît possible de déterminer la capacité du condensateur. Physiquement, la constante de temps représente le temps nécessaire pour que la tension à travers le condensateur atteigne environ 63,2% de sa valeur finale lorsque le circuit est soumis à une tension continue. En effet, avec un oscilloscope, on ne peut mesurer que des tensions et des temps. Pour obtenir la valeur de la constante de temps à l'aide de l'oscilloscope, il faut alors trouver un moyen d'obtenir la valeur de la tension à $t = \tau$. En reprenant l'équation 2.3 à $t = \tau$, on détermine que l'on atteint bien 63,2% de la charge complète du condensateur comme cela est présenté ci-dessous :

$$\begin{aligned}\frac{U(t)}{U_0} &= 1 - e^{-t/RC} \\ \frac{U(t = \tau)}{U_0} &= 1 - e^{-1} \approx 0,632\end{aligned}\tag{2.5}$$

Pour retirer les condensateurs du *keylogger hardware*, un pistolet à air chaud ainsi qu'une pince (*tweezer*) et du flux sont nécessaires (voir figure 2.16b et annexe A.3). Une résistance de $12\text{ k}\Omega$ est utilisée dans le montage **RC** que l'on peut voir sur la figure 2.16a. Lorsque l'on applique une tension d'entrée $V_{in} = 5\text{ V}$ au montage, le condensateur va se charger. Lorsque l'on coupe le circuit, le condensateur se décharge. L'oscilloscope permet d'observer cette charge/décharge et de mesurer la constante de temps lorsque la charge atteint 63,2% de la charge maximale. La difficulté réside dans la coupure du circuit : il faut qu'elle soit la plus abrupte possible. C'est pourquoi il n'est pas recommandé de couper manuellement l'alimentation avec un interrupteur, par exemple. Pour effectuer cette interruption, le montage est alimenté avec un signal carré de fréquence suffisante pour laisser le temps au condensateur de se charger et de se décharger complètement. Un résultat typique est présenté sur la figure 2.17.



(a)



(b)

FIGURE 2.16 – Sur la figure 2.16a, le montage **RC**. Sur la figure 2.16b, un des condensateurs dessoudé (celui du bas à gauche) du *keylogger hardware*.

Concernant la mesure, la tension est prise aux bornes du condensateur. La tension d'alimentation étant de 5 V , un curseur vertical a été placé à $3,16\text{ V}$ (63,2% de 5 V). Les curseurs horizontaux ont été placés au croisement des curseurs verticaux et de la courbe jaune, délimitant ainsi un intervalle de temps correspondant à la constante de temps. Cet intervalle de temps vaut $\Delta t = 14\text{ ms}$. Ainsi, d'après l'équation 2.5, on en déduit que la capacité du condensateur est alors de $1,17\text{ }\mu\text{F}$:

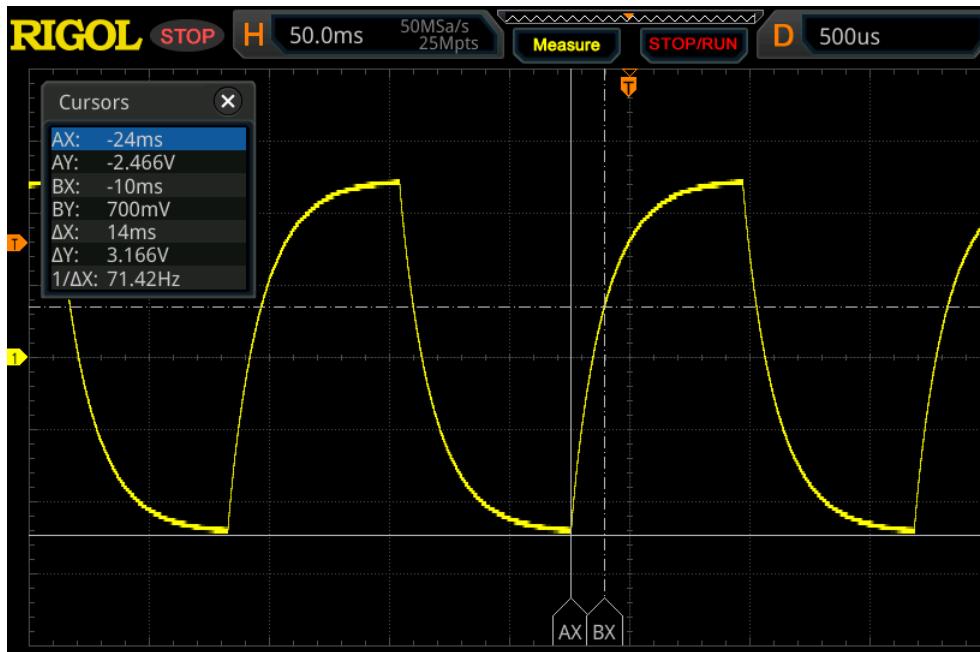


FIGURE 2.17 – Visualisation de la constante de temps d'un montage **RC**.

$$C = \frac{\tau}{R} = \frac{14 \cdot 10^{-3}}{12 \cdot 10^3} \approx 1,17 \mu\text{F} \quad (2.6)$$

2.2.4 — Vérification de la présence du JTAG

A faire

2.3 Attaques

Une attaque intéressante pour ce type de dispositif est la récupération du *firmware*. En effet, on peut imaginer avoir retrouvé cet appareil sur les lieux d'un incident, et souhaiter en déterminer le fonctionnement, ou bien décider de cloner cet appareil (le **PCB** et le *firmware*) afin de ne plus être dépendant du fabricant. Dans les deux cas, la récupération du *firmware* est une étape essentielle. Trois méthodes sont envisagées pour cela. La première tire parti du fait que le **SoC** est un **ESP8266EX** qui est un composant *opensource*. Il existe donc des outils *opensources* qui permettent de le *flasher*. Comme il a été présenté lors de l'analyse statique, le mode de démarrage de l'**ESP8266EX** est déterminé par le câblage du **GPIO0**. La documentation [9] indique que pour lire le contenu d'une *flash* de 2 MB, il faut utiliser la commande suivante :

```
1 esptool.py -p /dev/ttyUSB0 -b 460800 read_flash 0 0x200000
   ↳ flash_contents.bin
```

Mais il est possible de détecter automatiquement la taille de la *flash* en utilisant la commande :

```
1 esptool.py -p /dev/ttyUSB0 -b 460800 read_flash 0 ALL
   ↳ flash_contents.bin
```

Une deuxième méthode consiste à utiliser le **JTAG**, mais la difficulté à souder proprement les fils sur ce **PCB** amène une troisième méthode, plus générique. Il s'agit d'extraire directement le *firmware* depuis la *flash* en utilisant le protocole **SPI**. Le détail de ces étapes est présenté dans la section suivante.

2.3.1 Extraction du firmware

Pour extraire le *firmware* d'un boîtier **SOP-8** ou **SOP-16**, des pinces spéciales comme celles présentées sur la figure 2.18 sont souvent utilisées.



FIGURE 2.18 – Pinces **SOP-8** et **SOP-16** permettant de se connecter sur des boîtiers **SOP** sans les dessouder.

Dans le cas présent l'encombrement stérique est trop important. Il n'est donc pas possible de placer correctement la pince. Il faut alors dessouder la *flash* et la placer dans un *socket SOIC/SOP/DIP* comme celui présenté sur la figure 2.19. Afin de dessouder proprement et de pouvoir ressoudler la *flash* une fois celle-ci lue, il est important d'utiliser du flux pour éviter la surchauffe du composant mais aussi l'oxydation des soudures (voir annexe A.3).

Si l'auditeur n'a pas de *socket* disponible, il est aussi possible de souder directement sur les pins du boîtier afin d'y relier les fils à un appareil qui communiquera ensuite avec la *flash* en **SPI**. Ce mécanisme est présenté sur la figure 2.20 sur laquelle on peut voir la *flash* dessoudée du *keylogger* (figure 2.20a), mais aussi reliée à un **Hydrabus** (figure 2.20b, en haut l'**Hydrabus**, en base la *flash* et au



FIGURE 2.19 – Présentation d'un *socket* SOP-8 pour la lecture/écriture de *flash*.

milieu une interface entre les deux).

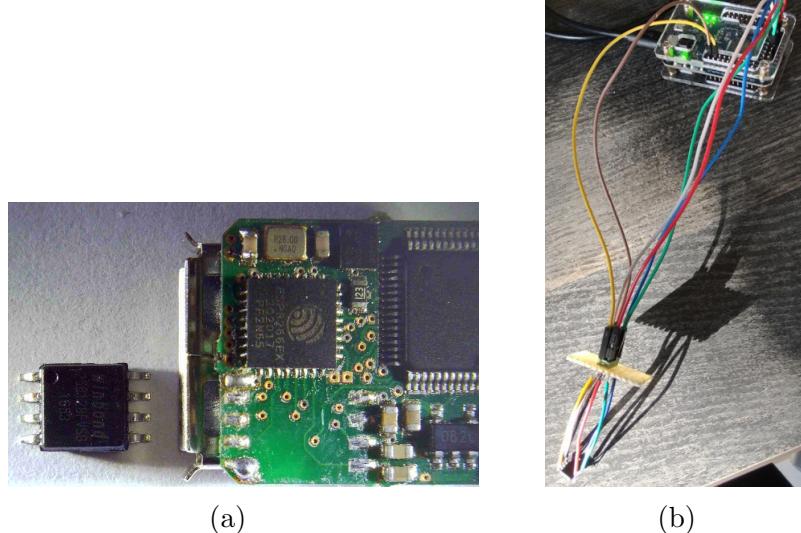


FIGURE 2.20 – Visualisation de la *flash* dessoudée puis connectée à l'**Hydrabus**.

Utilisation de l'Hydrabus pour lire la flash

Sur une machine **Linux**, une fois que l'**Hydrabus** est connecté, il est possible de voir apparaître le périphérique dans les *logs*. Pour les voir, taper la commande suivante :

```
1 sudo dmesg
```

Les *logs* présentées sur la figure 2.21 indiquent alors la présence de l'**Hydrabus**. On voit qu'il est disponible via le port /dev/ttyACM0. Il est donc possible d'interagir avec lui en utilisant une console série comme par exemple **minicom**, **picocom** ou encore **screen**. Pour cela, exécuter la commande suivante :

```
1 sudo minicom -D /dev/ttyACM0
```

Il est alors possible de lui spécifier le protocole à utiliser en tapant **spi**, puis de déterminer les pins à utiliser pour le **SPI** en tapant **show pins** comme cela

```
[ 197.328315] usb 2-2: new full-speed USB device number 3 using ohci-pci
[ 197.886151] usb 2-2: New USB device found, idVendor=1d50, idProduct=60a7, bcdDevice= 2.00
[ 197.886157] usb 2-2: New USB device strings: Mfr=1, Product=2, SerialNumber=3
[ 197.886159] usb 2-2: Product: HydraBus 1.0 COM Port1
[ 197.886160] usb 2-2: Manufacturer: Openmoko, Inc.
[ 197.886161] usb 2-2: SerialNumber: 005100225853501120383658
[ 197.955936] cdc_acm 2-2:1.0: ttyACM0: USB ACM device
[ 197.956074] usbcore: registered new interface driver cdc_acm
[ 197.956076] cdc_acm: USB Abstract Control Model driver for USB modems and ISDN adapters
```

FIGURE 2.21 – Visualisation des *logs* sur une machine **Linux** faisant apparaître le port de disponibilité de l'**Hydrabus**.

peut être vu sur la figure 2.22.

```
> spi
Device: SPI1
GPIO resistor: floating
Mode: master
Frequency: 320khz (650khz, 1.31mhz, 2.62mhz, 5.25mhz, 10.50mhz, 21mhz, 42mhz)
Polarity: 0
Phase: 0
Bit order: MSB first
spi1>
spi1>
spi1>
spi1>
spi1> show pins
CS: PA15
SCK: PB3
MISO: PB4
MOSI: PB5
```

FIGURE 2.22 – Détermination des pins à utiliser sur l'**Hydrabus** pour communiquer en **SPI**.

Comme présenté dans la section 1.2.2, pour communiquer en **SPI** avec une *flash*, il est nécessaire de connecter un certain nombre de câbles. Quatre câbles sont nécessaires : **SCLK**, **MOSI**, **MISO** et **SS**. Comme la *flash* n'est plus sur le **PCB**, il est nécessaire de l'alimenter et donc d'ajouter deux autres câbles : la masse (**GND**) et la tension (**VCC**). D'après la figure 2.22, on voit alors la correspondance suivante :

- **SCLK** → **PB3**.
- **MOSI** → **PB5**.
- **MISO** → **PB4**.
- **SS** → **PA15**.

Avant d'extraire le contenu de la *flash*, il est nécessaire de s'assurer que l'**Hydrabus** communique bien avec elle. Pour cela, il est intéressant de lire l'identificateur unique (**UID**) de la *flash* qui d'après la documentation est retourné par la *flash* lorsque le contrôleur envoie la commande **0x4B**, **0x00**, **0x00**, **0x00**, **0x00**. D'après la figure 2.23, l'**UID** est alors **0x00**, **0x31**, **0xD2**, **0x66**, **0x2C**, **0x30**, **0x19**, **0x31**.

Quelques explications s'imposent concernant la commande :

- [: condition de *start*. En **SPI**, cela signifie *Enable Chip Select*.
-] : condition de *stop*. En **SPI**, cela signifie *Disable Chip Select*.

```

spi1> [ 0x4B 0x00:4 hd:8 ]
/CS ENABLED
WRITE: 0x4B 0x00 0x00 0x00 0x00
00 31 D2 66 2C 30 19 31
| .1.f,0.1
/CS DISABLED

```

FIGURE 2.23 – Récupération de l'UID de la *flash*.

- : **x** : répétition **x** fois.
- **hd : x** : permet de lire **x bytes (hex dump)**.

L'utilisation de l'**Hydrabus** en mode **SPI** est disponible dans la documentation officielle [22]. Il est alors possible de lire secteur par secteur la *flash* en interagissant avec l'**Hydrabus**. D'après la documentation, une lecture de la *flash* se fait avec la commande **0x03** suivie de l'adresse codée sur 24 bits (3 octets) à laquelle on souhaite lire. Le script **Python 3** suivant permet de se connecter sur le périphérique **/dev/ttyACM0** et de lire chaque secteur à partir de l'adresse **0x0**.

```

1 import serial
2 import sys
3
4 output_filename = "flash_dump.bin"
5 number_sectors = 1024 # modify that value depending on the
→ flash size (1024 = 4MB -> 1024*4096)
6 start_address = 0x0
7 sector_size = 0x1000 # 1 sector = 4096 byte = 0x1000
8
9 global hydrabus
10 hydrabus = serial.Serial('/dev/ttyACM0', 115200)
11
12 def hex_to_bin(num, padding):
13     return num.to_bytes(padding, byteorder='big')
14
15 def calc_hex_addr(addr, add):
16     address = addr + add
17     return hex_to_bin(address, 3)
18
19 # open in binary mode
20 for i in range(20):
21     hydrabus.write(b'\x00')
22 if b"BBI01" not in hydrabus.read(5):
23     print("[KO] opening binary mode]")
24     sys.exit(1)
25
26 hydrabus.reset_input_buffer()
27
28 # switch to SPI mode
29 hydrabus.write(b'\x01')
30 if b"SPI1" not in hydrabus.read(4):

```

```
31     print("[KO] SPI mode")
32     sys.exit(1)
33
34 # default polarity, default clock phase
35 hydrabus.write(b'\x81')
36 if b'\x01' not in hydrabus.read(1):
37     print("[KO] default config: polarity, phase")
38     sys.exit(1)
39
40 # start dumping here
41 print('[OK] reading ' + str(number_sectors) + ' sectors
   ↪ ...')
42
43 data_dumped = bytearray()
44 sector = 0 # start with the first sector
45 while sector < number_sectors:
46     # write-then-read: write 4 bytes (1 read cmd + 3 read
        ↪ addr), read sector_size bytes
47     hydrabus.write(b'\x04\x00\x04' +
        ↪ hex_to_bin(sector_size, 2))
48
49     # to read: 0x03 + address
50     # if hast read: 0x0B + address
51     hydrabus.write(b'\x03' + calc_hex_addr(start_address,
        ↪ sector * sector_size))
52
53     # if read OK, hydrabus send 1, read that
54     hydrabus.read(1)
55
56     # then read the response, display it and incremente
        ↪ sector
57     data_dumped += hydrabus.read(sector_size)
58     print('[+] read sector ' + str(sector))
59
60     sector += 1
61
62 # clean up hydrabus
63 hydrabus.write(b'\x00') # binary mode
64 hydrabus.write(b'\x0F\n') # console mode
65
66 # Write content to file
67 with open(output_filename, 'wb+') as f:
68     f.write(data_dumped)
69
70 print('[OK] flash dumped to ' + output_filename)
71 sys.exit(1)
```

Vérification du contenu de la flash

Il est recommandé d'effectuer au minimum deux fois l'extraction de la *flash* afin de s'assurer que le fichier n'a pas été compromis lors de cette étape. Un moyen simple de vérifier la concordance des deux fichiers est d'en calculer les *hashes* (par exemple **MD5** ou **SHA1**) et de les comparer. Les commandes permettant de calculer un de tels *hashes* sont :

Il est possible de vérifier que le *dump* est complet en utilisant l'outil *opensource esptool* [10] d'**Espressif** comme cela est présenté sur la figure 2.24. On voit que le *dump* contient 3 segments chargés en mémoire aux adresses **0x40100000** (segment de code), **0x3FFE8000** (segment de données) et **0x3FFE87F0** (segment de données). Ces segments sont accessibles aux *offset* **0x08**, **0x7CA8** et **0x8498** respectivement. On apprend aussi que le point d'entrée du *firmware* est à l'adresse **0x40100004**. On valide également que le fichier est correct grâce au *checksum* qui indique **valide**.

```
(shellchocolat)→ →esptool git:(master) ./esptool.py --chip esp8266 image_info .. /dump4.bin
esptool.py v4.7-dev
File size: 16777216 (bytes)
Image version: 1
Entry point: 40100004
3 segments

Segment 1: len 0x07c98 load 0x40100000 file_offs 0x00000008 [IRAM]
Segment 2: len 0x007e8 load 0x3ffe8000 file_offs 0x00007ca8 [DRAM]
Segment 3: len 0x01b64 load 0x3ffe87f0 file_offs 0x00008498 [DRAM]
Checksum: 7a (valid)
```

FIGURE 2.24 – Vérification de l'intégrité du *dump* avec l'outil *opensource esptool*.

On peut utiliser l'outil **binwalk** afin d'avoir un aperçu du contenu du *dump*. La figure 2.25 permet de constater que la taille du *dump* fait bien 16 Mo grâce à la commande **ll**, mais aussi d'apercevoir les différents types de fichiers qui y sont contenus grâce à l'outil **binwalk**. On s'aperçoit qu'un exécutable **Windows** serait présent à l'intérieur. On peut récupérer ce binaire avec l'outil **binwalk** directement ou avec la commande **dd**, selon les préférences. Une fois récupéré, on peut l'exécuter sur une machine **Windows** et on obtient la fenêtre présentée sur la figure 2.26. Il y a donc effectivement un binaire **Windows** tout à fait fonctionnel embarqué dedans ce qui semble, pour le moins, étrange.

DECIMAL	HEXADECIMAL	DESCRIPTION
35760	0x8BB0	AES S-Box
36016	0x8CB0	AES Inverse S-Box
39984	0x9C30	PEM RSA private key
40084	0x9C94	PEM certificate
318320	0x4DB70	SHA256 hash constants, little endian
350696	0x559E8	Microsoft executable, portable (PE)
358665	0x57909	Copyright string: "CopyrightAttribute"
363323	0x58B3B	XML document, version: "1.0"

FIGURE 2.25 – Vérification du contenu du *dump* avec l'outil **binwalk**.

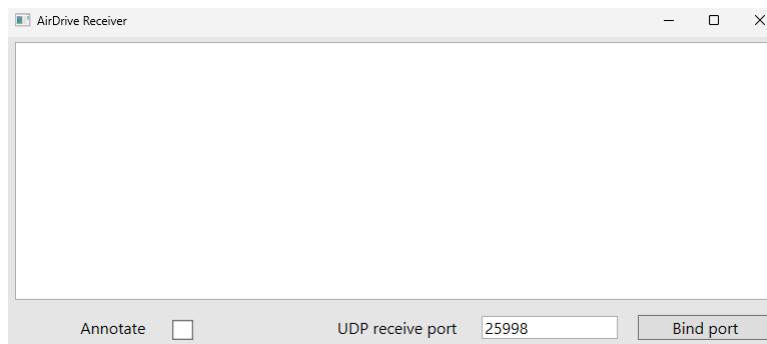


FIGURE 2.26 – Exécution de l'exécutable Windows embarqué dans le *dump*.

Il est alors possible d'analyser le *firmware* avec des outils comme **IDA** [14] ou **Ghidra** [17], mais aussi des outils plus classiques comme **strings**, **qemu** (**qemu-xtensa-static**), **readelf**, etc. Bien que l'analyse du *firmware* soit extrêmement intéressante, nous n'irons pas plus loin dans cet ouvrage consacré à l'analyse de **PCB**.

2.3.2 Clonage du PCB

Le clonage de **PCB** est une manière de reproduire un circuit électronique de manière à pouvoir le produire soi-même sans passer par le fabricant. En fonction du nombre de séries produites, cela peut ne pas être rentable financièrement. En effet, la création du **PCB** et l'achat des composants en petite quantité revient plus cher qu'en grande quantité. De plus, le positionnement des composants et la soudure de ceux-ci peuvent s'avérer complexes.

Le *keylogger hardware* est un bon exercice de clonage car c'est un appareil de conception simple. En revanche l'encombrement stérique étant ce qu'il est, la soudure des composants n'est pas chose aisée. Cette section permet de présenter les différentes étapes de création d'un circuit électronique. L'outil utilisé pour cela est un outil *opensource* : **KiCad** (<https://www.kicad.org/>). Il est disponible sur les plateformes suivantes : **Windows**, **macOS** et **Linux**. La version 7.0 est utilisée dans cette section. Les mesures des composants effectuées dans l'analyse dynamique sont importantes afin de copier de manière fidèle le **PCB** d'origine. Les schémas présentés dans la présente section sont volontairement erronés/incomplets, et ce, afin de ne pas voler la propriété intellectuel du fabricant de ce *keylogger hardware*. Ils sont cependant suffisamment fidèles pour proposer au lecteur une compréhension du processus de clonage de **PCB**.

Édition de schématique

La schématique (schéma électronique) permet de traduire sous forme de symboles les composants, l'alimentation et les différents signaux du circuit électronique. Une fois un projet **KiCad** créé, il est possible d'ouvrir l'**éditeur de schématique** comme cela peut se voir sur la figure 2.27. Une nouvelle fenêtre s'ouvre sur laquelle il sera possible de placer les composants et de les relier entre eux.

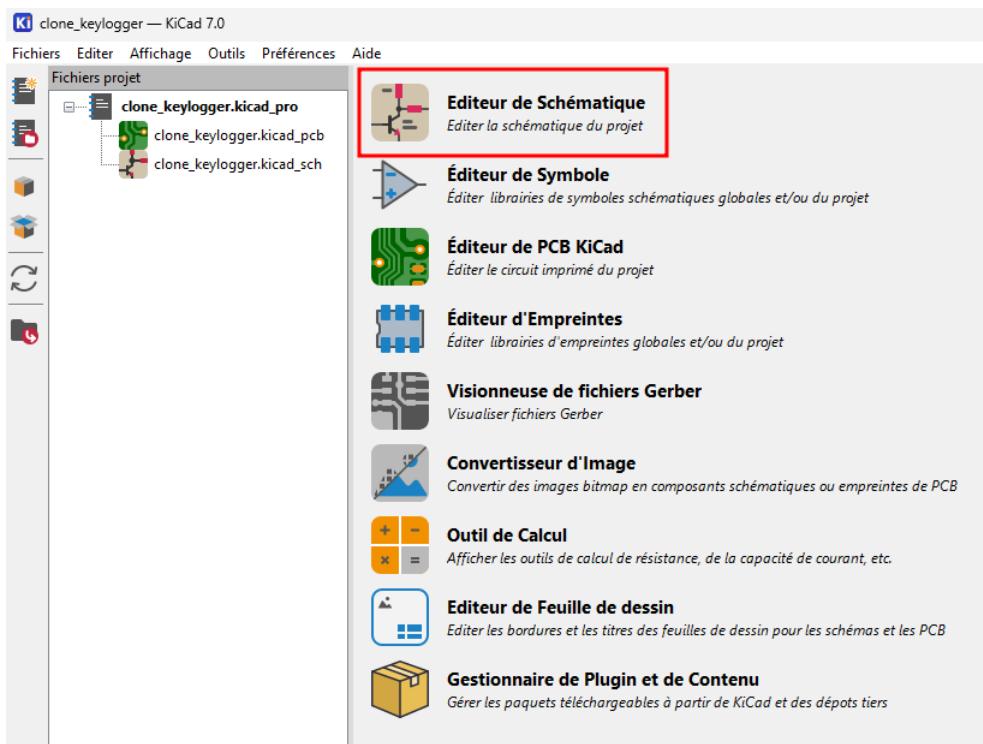


FIGURE 2.27 – Accéder à l'éditeur de schématique sur KiCad.

Pour ajouter un symbole sur le plan, il faut cliquer sur le bouton représentant un amplificateur opérationnel comme cela peut se voir sur la figure 2.28. Il est également possible d'appuyer sur la lettre **a**.

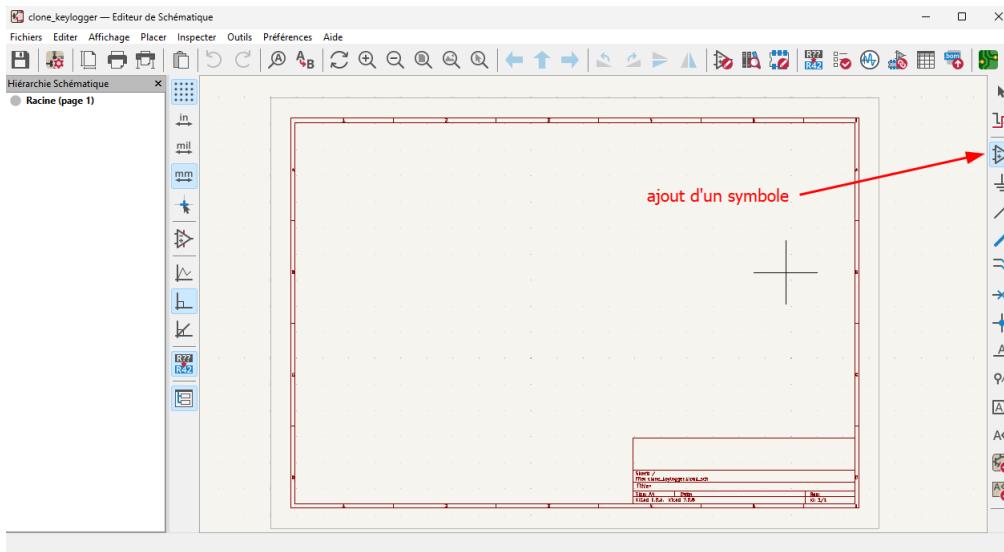


FIGURE 2.28 – Ajout d'un symbole sur la schématique.

Dans la fenêtre qui vient de s'ouvrir, il est possible de rechercher les composants dans une bibliothèque. Prenons l'exemple du composant **ESP8266EX** comme cela est présenté sur la figure 2.29. On voit que celui-ci est disponible (en haut à gauche), on y voit son schéma (en haut à droite), son empreinte (en bas à droite), ainsi qu'une description du composant (en bas à gauche). Il est donc possible de le positionner sur le plan en cliquant sur **OK**.

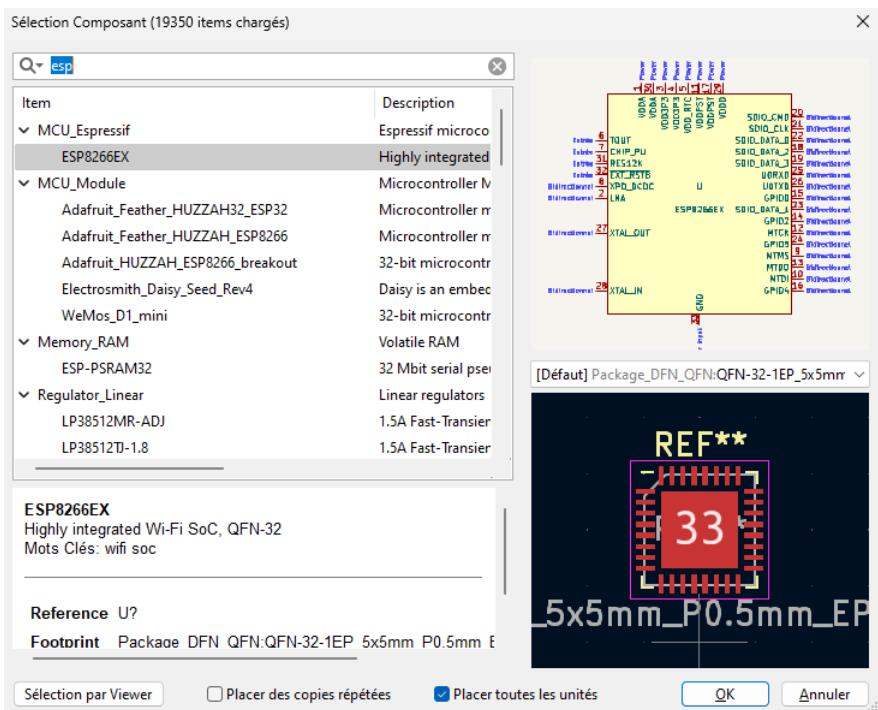


FIGURE 2.29 – Recherche du composant **ESP8266EX** dans la bibliothèque de **KiCad**.

En regardant le composant de plus près (figure 2.30), on peut voir que les pattes du composants ne sont pas réparties de la même manière que sur le composant physique. En effet, le but de la schématique est de pouvoir relier facilement les différentes pattes des composants, mais pas de les retranscrire fidèlement. C'est le rôle de l'éditeur d'empreinte. L'empreinte de l'**ESP8266EX** étant déjà dans la bibliothèque, on peut faire confiance à la personne qui l'y a mise. Il est cependant possible de la modifier si jamais cela était souhaité. On verra par la suite comment créer une empreinte de zéro.

Le composant a été labellisé **U1**, cependant, sur le masque qui a été créé dans les sections précédentes, l'**ESP8266EX** a été labellisé **U2**. Afin de coller à ce masque, il est possible de double-cliquer sur le label afin de le modifier (figure 2.31).

Il est également possible de placer des ports **USB**. Il faut deux emplacement : un port **USB** mâle (branchement à l'ordinateur) et un femelle (branchement du clavier). Un port **USB** a 5 pattes : le boitier de protection qui est relié à la masse, une patte de masse, une patte de tension (5 V), et deux pattes de données (D₊ et D₋). Pour relier ces pattes, des labels globaux seront utilisés. Pour ajouter un label global, il faut cliquer sur le bouton (ou le raccourci **Ctrl+L**) présenté sur la figure 2.32.

Le label global est le plus souvent utilisé pour représenter un bus, soit d'alimentation, soit de données. Pour ajouter un symbole de tension (3,3 V ou la masse), il faut cliquer sur le symbole d'alimentation (voir figure 2.32). Comme il ne s'agit que de schématique, il faut ensuite spécifier que ces labels globaux seront associés à de la puissance. Il faut pour cela ajouté un symbole d'alimentation spécifique appelé **PWR_FLAG**. Ces étapes sont présentées sur la figure 2.33.

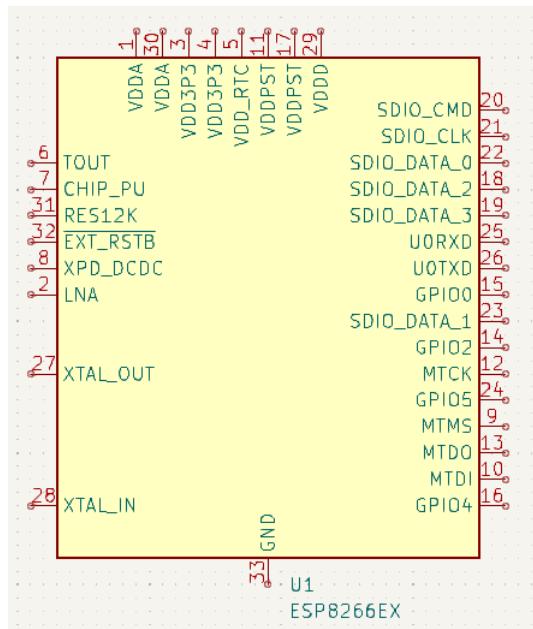


FIGURE 2.30 – Visualisation du composant **ESP8266EX** une fois placé sur la schématique.

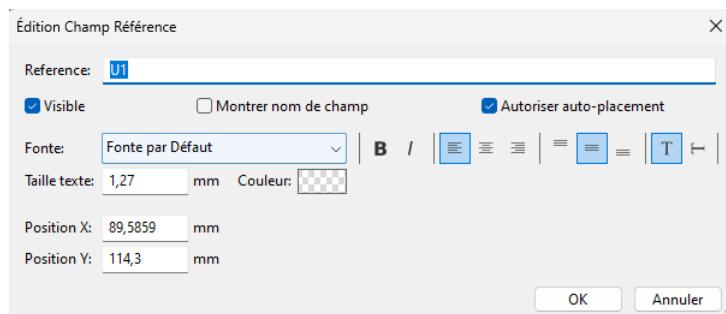


FIGURE 2.31 – Modification du label d'un composant.

On voit que les labels globaux peuvent avoir une orientation afin de spécifier la direction du signal. Il est à noter qu'une résistance de quelques **Ohms** est généralement nécessaire sur les bus de données **USB** mais qu'elles n'ont pas été mises ici dans un souci de gain de place sur le **PCB**.

Pour que l'**ESP8266EX** fonctionne, il faut lui associer un signal d'horloge. D'après la documentation, une fréquence située entre 24 MHz et 52 MHz est requise. Une valeur typique de 27 MHz est choisie. Le cristal doit être associé à deux condensateurs afin d'avoir un signal d'horloge le plus stable possible. D'ailleurs, ces deux condensateurs n'ont pas été enlevé du *keylogger* malgré le souhait de gagner de la place. Des condensateurs de 10 pF ont été ajoutés au cristal, mais il faut noter que leurs valeurs dépend fortement du cristal choisi. Afin de tirer des liens entre l'**ESP8266EX** et le cristal, il faut cliquer sur le bouton représentant un fil (ou le raccourci **Ctrl+W**) comme cela est présenté sur la figure 2.34.

En ce qui concerne la *flash* externe, on n'est pas obligé d'utiliser exactement le même modèle. Ici, j'ai utilisé une *flash* **Winbond W25Q128JVS** qui utilise, tout comme la **Winbond W25Q128FVSG** le protocole **SPI** et qui fait 16 Mo.

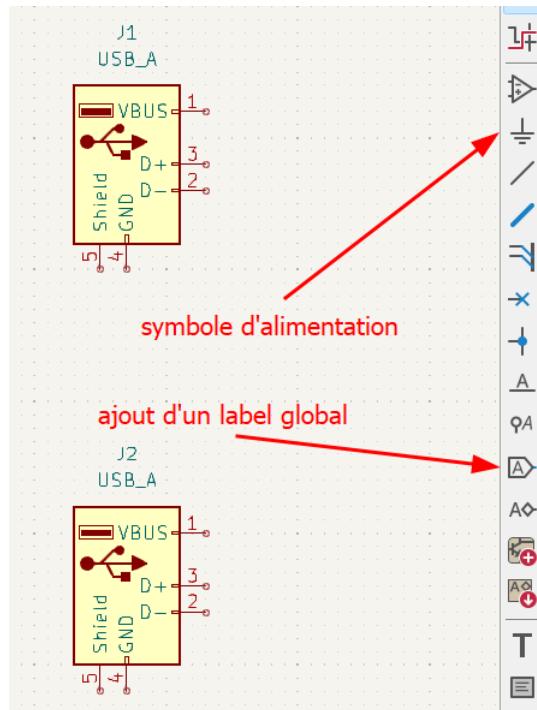


FIGURE 2.32 – Ajout d'un label global.

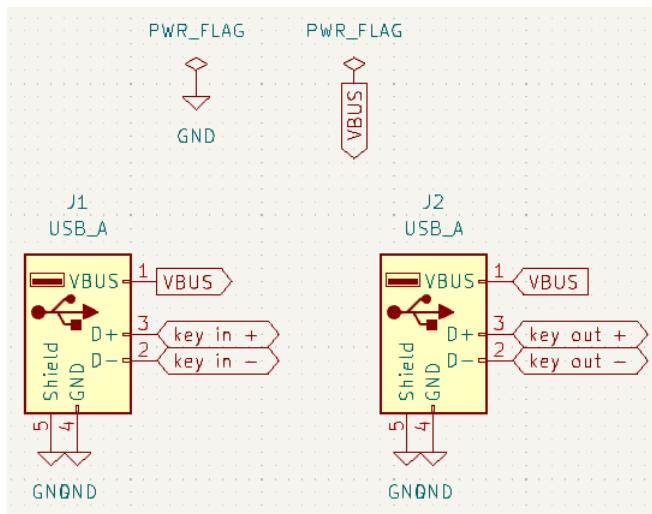


FIGURE 2.33 – Mise en place des ports USB mâle et femelle.

D'après la documentation de l'**ESP8266EX**, on peut alors effectuer les branchements comme présentés sur la figure 2.35.

Pour ajouter un composant qui n'est pas existant dans la bibliothèque de **KiCad**, il faut le créer manuellement. Pour cela, cliquer sur l'**éditeur de symbole** (voir figure 2.36), puis ajouter un nouveau symbole comme présenté sur la figure 2.37.

Il est ensuite possible de renseigner le nom du symbole comme présenté sur la figure 2.38. suite à cela, il est possible de dessiner le nouveau symbole qui sera alors disponible dans la schématique. Il faut alors se souvenir qu'aucune fidélité par rapport au composant physique n'est demandé dans la création d'un symbole. En effet, le composant physique est carré, tandis que le composant que je

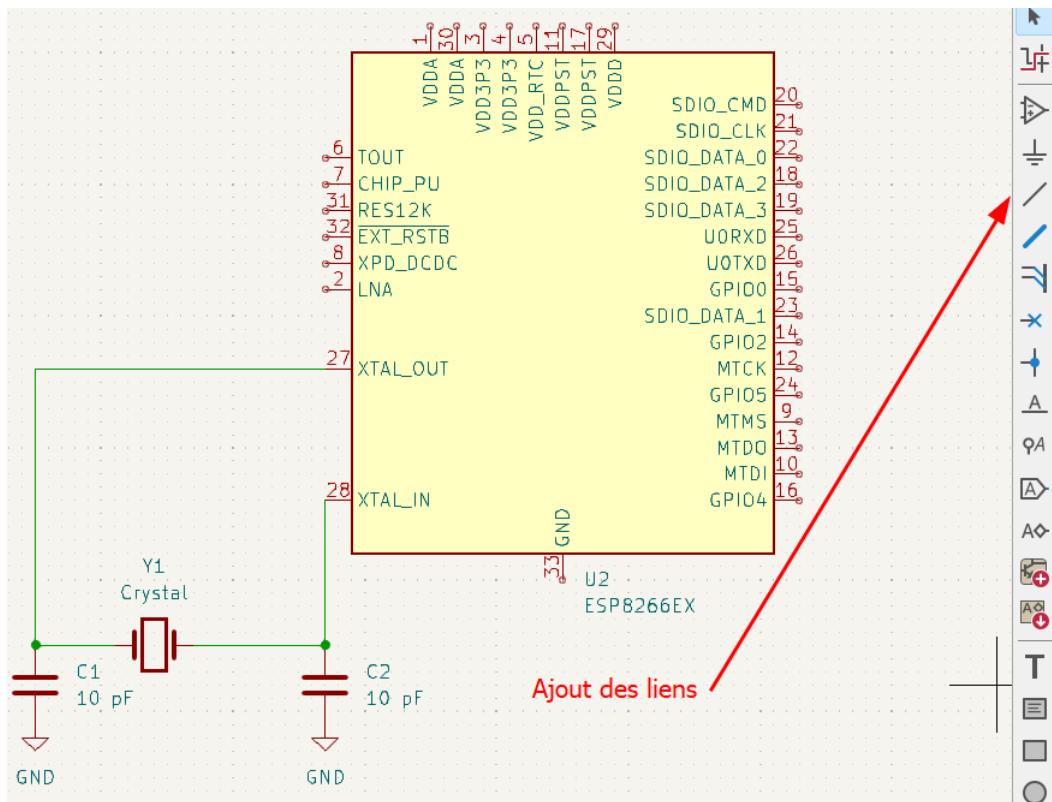


FIGURE 2.34 – Création de lien et visualisation de la mise en place du cristal.

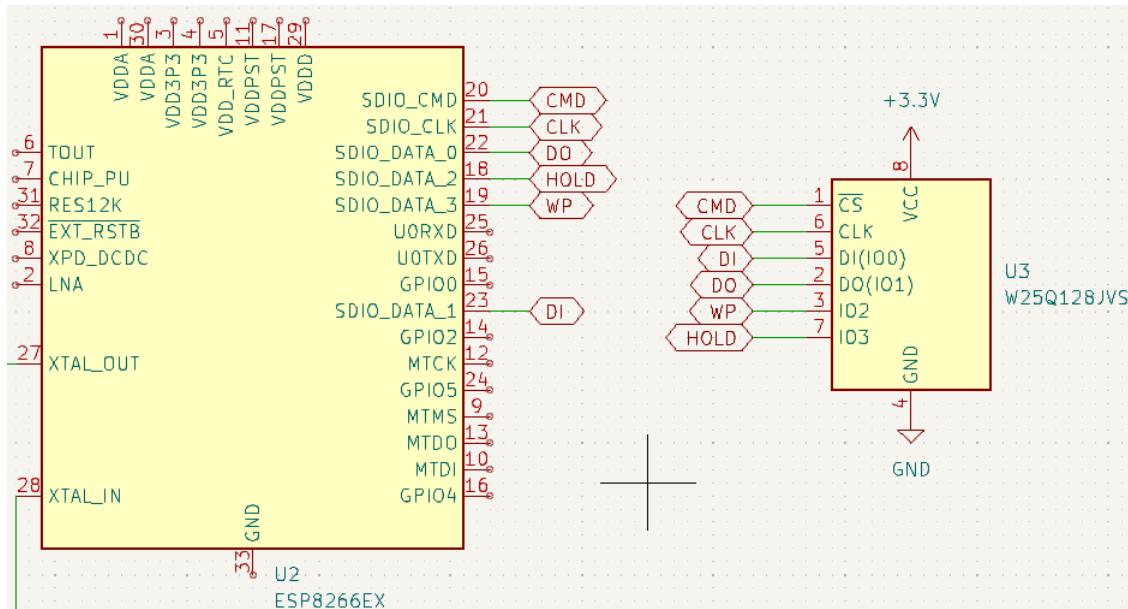
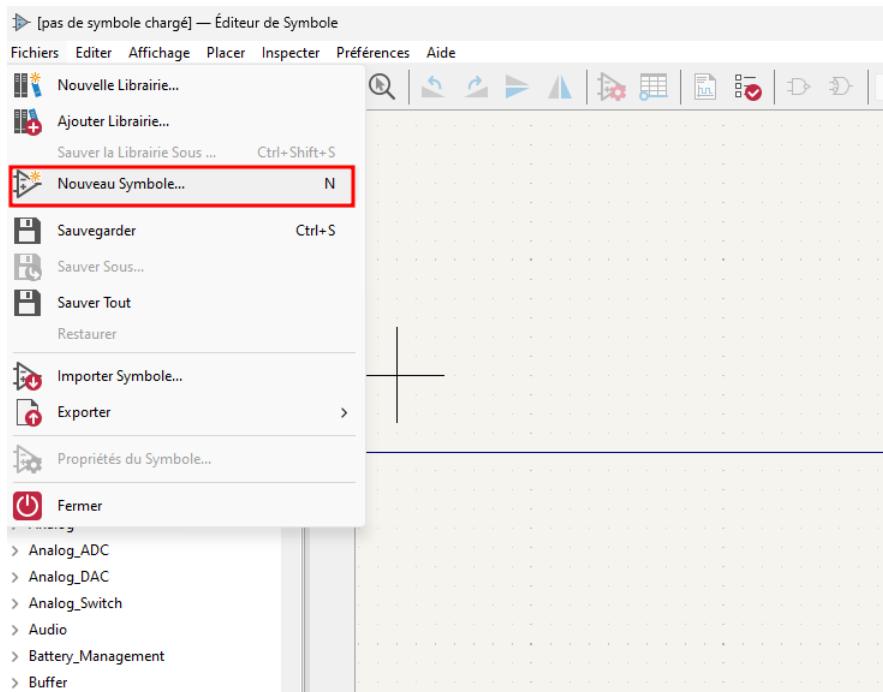


FIGURE 2.35 – Branchement de la flash à l'ESP8266EX.

créerais sera de forme rectangulaire. C'est l'empreinte de ce symbole qui devra alors correspondre à la réalité. Cela sera vu ensuite.

Pour créer une forme rectangulaire, il faut cliquer sur le bouton en forme de rectangle comme présenté sur la figure 2.39. Il faut ensuite cliquer sur le bouton ajouter un pin, et ajouter l'ensemble des pins comme cela est présenté sur la figure 2.40.

FIGURE 2.36 – Accéder à l'éditeur de symbole sur **KiCad**.FIGURE 2.37 – Ajouter un nouveau symbole à une bibliothèque **KiCad**.

Suite à cela, il est possible d'utiliser ce symbole. Pour cela, il faut spécifier à **KiCad** le chemin vers lequel chercher la nouvelle bibliothèque associé au projet en cours. Pour cela, dans l'éditeur de schématique, cliquer sur **Préférences**, puis sur **Configurer les librairies de symboles**. L'ensemble des composants peuvent donc être positionner sur la schématique. Une fois que tout est mis en place, l'éditeur de schématique offre la possibilité de vérifier que l'ensemble des pins sont bien reliés. Pour cela, il faut utiliser le bouton **Exécute le test des règles électriques** comme présenté sur la figure 2.41. Ce test est indispensable car il permet de vérifier qu'aucun pin n'est libre. Il ne vérifie cependant pas si les connections sont correctes. La figure 2.42 présente un test échoué dans lequel il y a 55 erreurs à corriger. Il faut absolument que ce test soit validé avant de passer

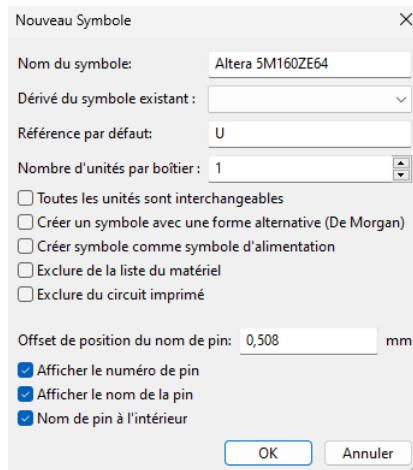


FIGURE 2.38 – Renseigner le nom du nouveau symbole dans **KiCad**.

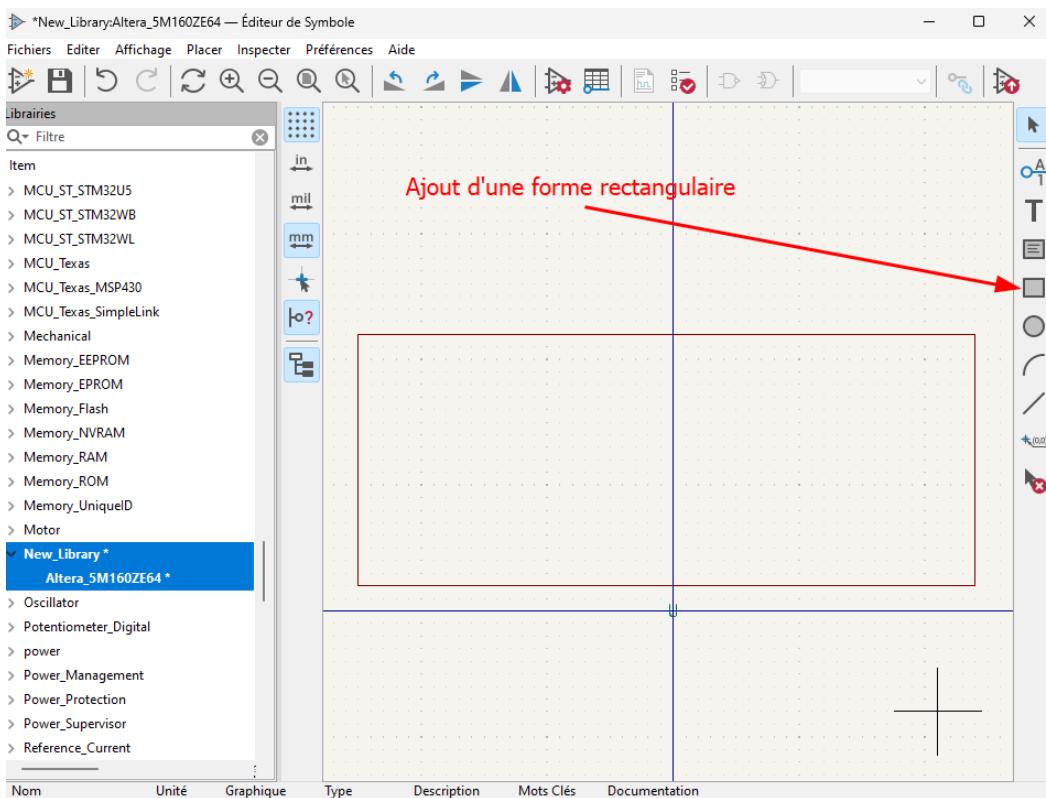


FIGURE 2.39 – Création d'une forme rectangulaire dans l'éditeur de symbole.

à la suite du processus de création d'un circuit électronique.

Génération de la netliste

La **netliste** consiste à associer à chaque composant une empreinte. Dans les composants que nous avons placé à l'étape précédente, tous ont une empreinte, sauf le **CPLD** que nous avons dû créer nous même car il n'était pas présent dans les bibliothèques par défaut. Pour générer la **netliste**, il faut cliquer sur le bouton **Outil d'association d'empreintes** comme cela est présenté sur la figure 2.43. La fenêtre présentée sur la figure 2.44 apparaît alors. Trois panneaux sont disponibles. Le panneau du milieu permet de visualiser l'ensemble des composants qui

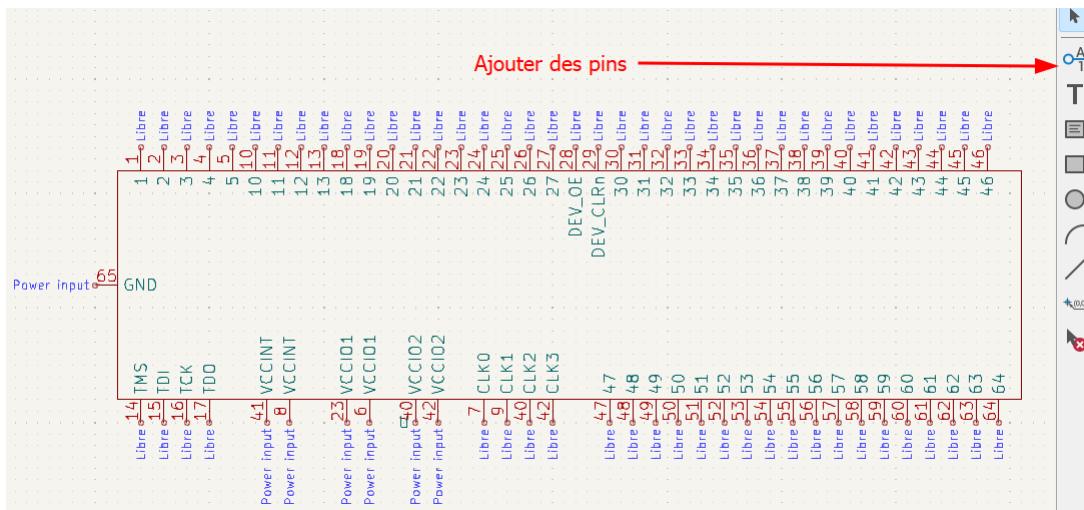


FIGURE 2.40 – Ajouter des pins à un symbole.

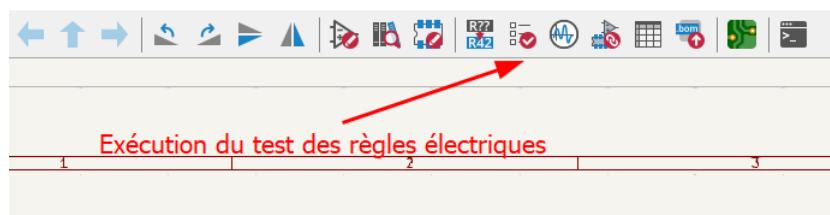


FIGURE 2.41 – Bouton permettant d'exécuter le test des règles électriques sur KiCad.

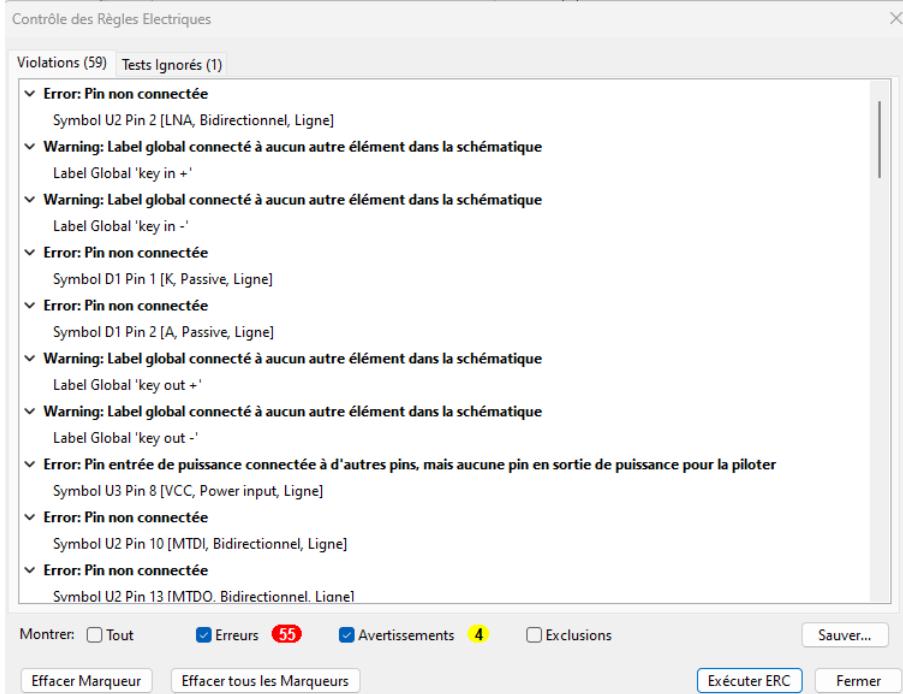


FIGURE 2.42 – Test de règles électriques en erreur.

sont placés sur la schématique. On peut voir que seul les composants **U2**, **U3** et **U4** ont une empreinte associée. Pour associer un composant à sa forme réelle, il faut sélectionner le composant dans la bibliothèque d'empreinte sur le panneau de gauche, puis choisir la bonne empreinte sur le panneau de droite.



FIGURE 2.43 – Accéder à l'outil d'association d'empreintes sur KiCad.

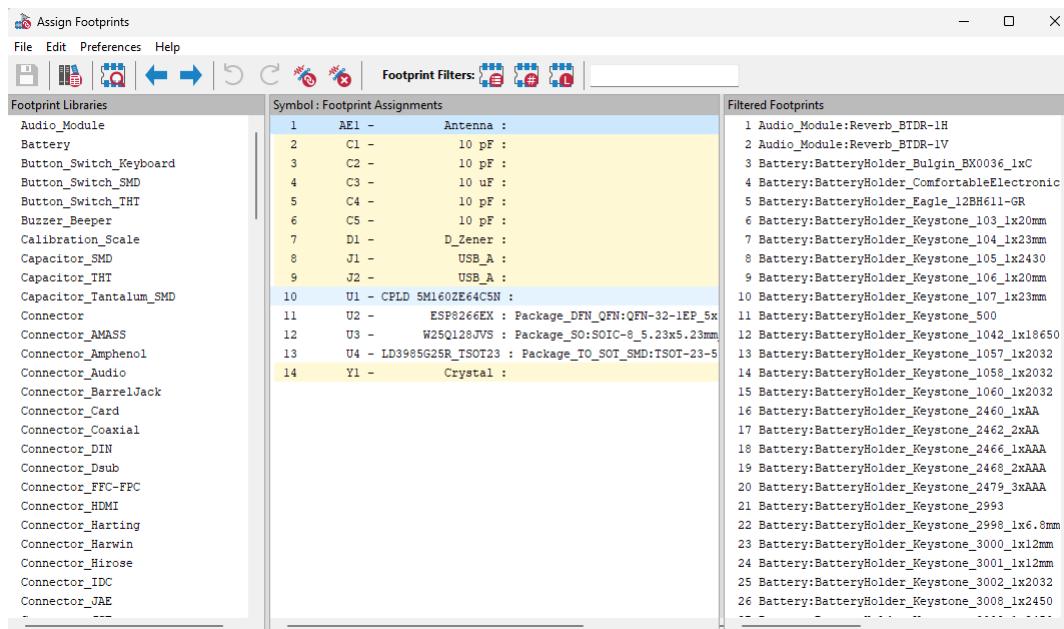


FIGURE 2.44 – Visualisation des différents panneaux de l'outil d'association d'empreintes.

Prenons l'exemple du condensateur **C1**. On peut choisir des condensateurs **SMD** (*Surface Mounted Device*) ou **THT** (*Through Hole Technology*). Le *keylogger* est un appareil relativement petit, il est donc préférable de prendre des composants de surface (**SMD**). Les boîtiers des composants **SMD** les plus petits soudable à la main sont, par expérience, les **0603** (1,5 mm × 0,8 mm). La figure 2.45 présente la sélection de ce type de boîtier.

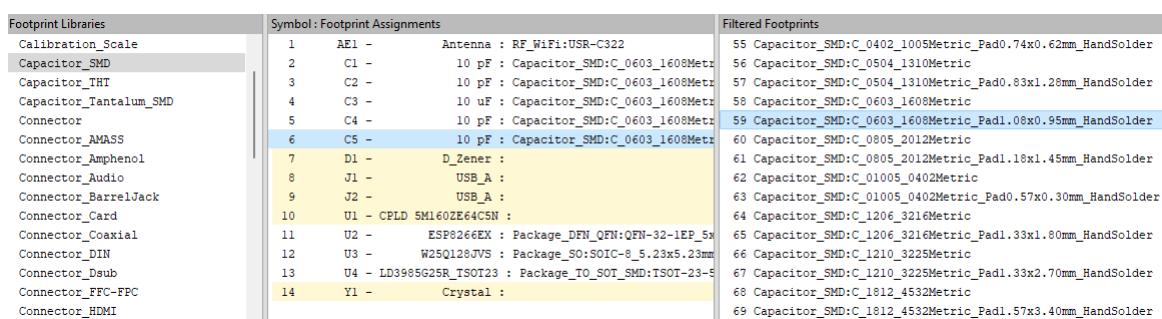


FIGURE 2.45 – Sélection d'un boîtier **0603** pour un condensateur **SMD**.

Comme il a été énoncé, il faut créer l'empreinte du **CPLD**. Pour cela, il faut cliquer sur l'**éditeur d'empreinte** comme cela est présenté sur la figure 2.46.

Pour créer une nouvelle empreinte, cliquer sur le bouton **Créer une nouvelle empreinte** ou sur **Ctrl+N**.



FIGURE 2.46 – Accéder à l’éditeur d’empreintes sur **KiCad**.

Cependant, comme le **CPLD** est un boîtier standard **TQFP64**, il est possible de se servir d’une empreinte déjà existante.

Édition de PCB

Une fois que les tests électriques sont validés et que les empreintes ont toutes été associées aux composants, il est possible de démarrer l’étape d’édition du **PCB**. Pour cela, il faut cliquer sur le bouton **Éditeur de PCB KiCad** comme cela est présenté sur la figure 2.47.

Cliquer sur **Outils** puis sur **Mise à jour du PCB à partir du schéma** comme sur la figure 2.48 afin d’importer la **netliste**. Suite à cela, **KiCad** présente les empreintes). Il ne reste plus qu’à les disposer correctement. Pour cela, il faut créer la forme générale du **PCB**, qui est le plus souvent rectangulaire, puis agencer les composants de la manière la plus optimale possible. **KiCad** permet de faire des **PCB** multi-couche. Pour ce projet, on ne se contentera que d’une seule couche. Bien qu’il eut pu être nécessaire de placer les port **USB** sur le recto pour être le plus fidèle possible au *keylogger* d’origine. La figure 2.49 présente les composants assemblés.

Une fois les composants positionnés, il faut tracer les routes entre chaque composant. L’espace étant réduit, il sera nécessaire d’utiliser des micro-vias pour passer sur la face arrière du **PCB**. On peut aussi utiliser des vias simples pour sortir le **JTAG** du **CPLD** et le **GPIO0** de l’**ESP8266EX**. Une fois les routes proprement tracées, il reste à mettre en place un plan de masse. Pour cela, cliquer sur **Placer** puis sur **Ajouter zone remplie**. Attention, il faut s’assurer de ne

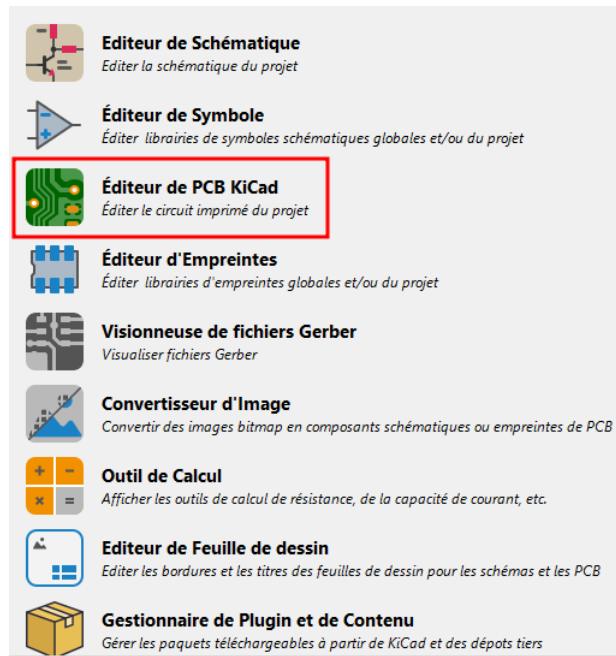


FIGURE 2.47 – Accéder à l’éditeur de PCB sur KiCad.

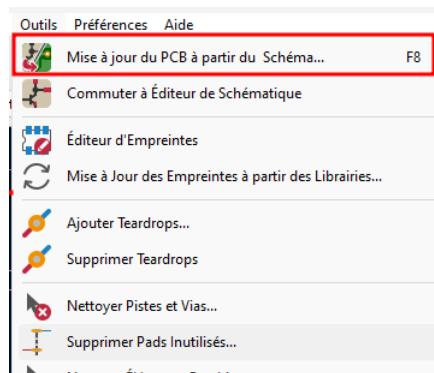


FIGURE 2.48 – Import de la netliste.

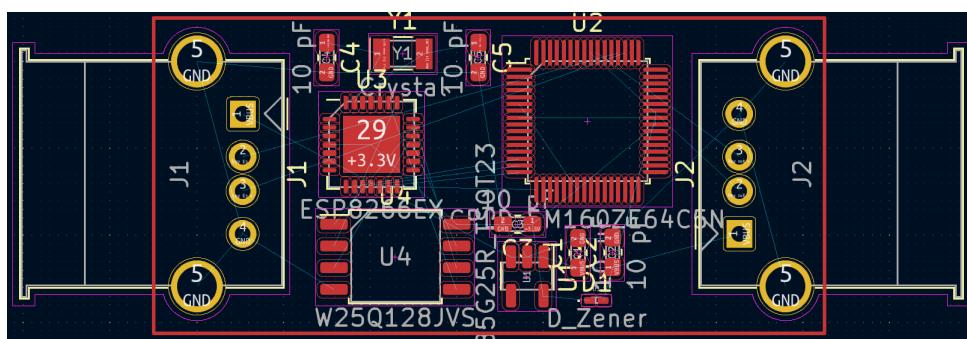


FIGURE 2.49 – Présentation des empreintes dans l’éditeur de PCB sans les routes.

jamais mettre de plan de masse sous une antenne. L'**ESP8266EX** possédant une antenne, il faut prendre les mesures qui conviennent.

Chapitre 3

Analyse d'un digicode

Le digicode est un appareil de contrôle d'accès. Il est très utilisé en entreprise, mais aussi pour le particulier dans le contrôle d'entrée des résidences privées. Le digicode analysé dans cette section est présenté sur la figure 3.1.



FIGURE 3.1 – Face avant d'un digicode.

Pour rentrer dans le bâtiment, il est nécessaire de taper le bon code, qui a été préalablement rentré en mémoire, ou bien d'avoir un badge **RFID** (*Radio Frequency IDentification*). Pour sortir du bâtiment, il suffit d'appuyer sur un interrupteur (qui ne se trouve qu'à l'intérieur du bâtiment pour des raisons évidentes) pour ouvrir la porte. Lorsque le boîtier du digicode est ouvert, une alarme peut se déclencher. Lorsqu'un mauvais code est rentré, une lumière rouge s'allume sur la face avant, et lorsqu'un code correct est rentré, c'est une lumière qui s'allume.

Le digicode est alimenté par le biais d'une alimentation externe en 12 V continue et peut consommer jusqu'à 3 A. Une fois le boîtier ouvert, on y trouve un **PCB** très clair. Les deux faces du **PCB** sont présentées sur la figure 3.2. On y voit plusieurs circuits intégrés. Il y a très peu de chance qu'ils soient alimentés en 12 V. Il doit donc y avoir un (ou plusieurs) étage d'alimentation.

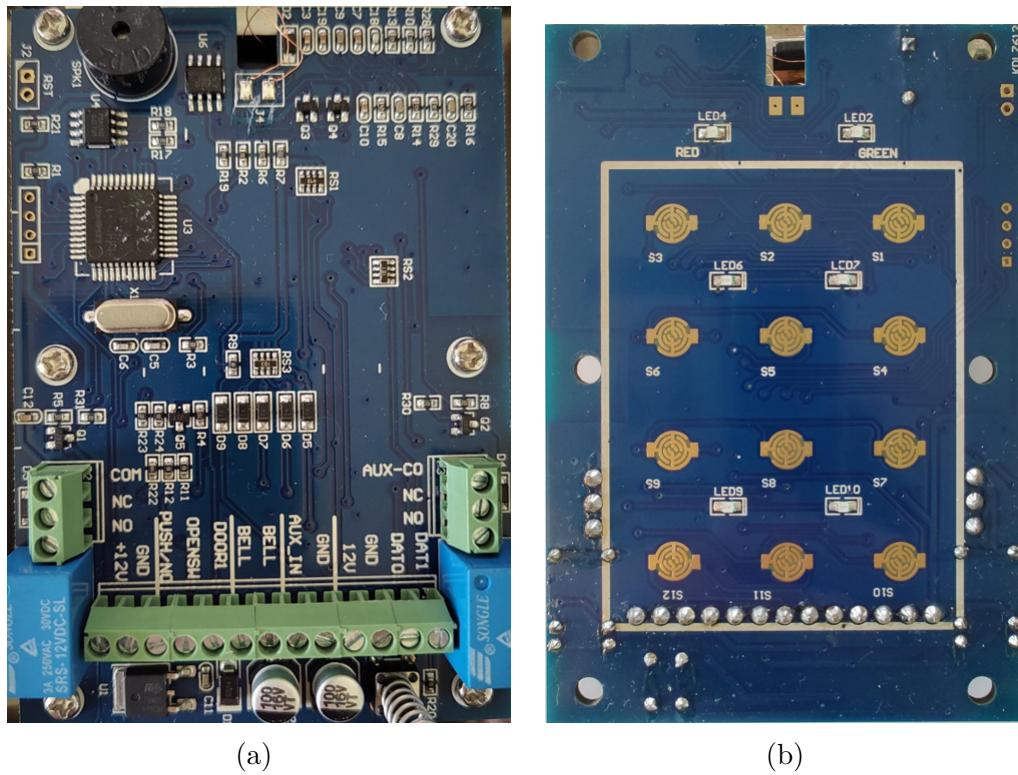


FIGURE 3.2 – Sur la figure 3.2a, la face arrière du digicode. Sur la figure 3.2b, la face avant.

On constate que l'ensemble des composants sont déjà labellisés, il n'apparaît alors pas nécessaire de faire un masque afin d'y faire référence par la suite.

3.1 Analyse statique du PCB

D'après la figure 3.2, on voit que la face avant du **PCB** ne contient que très peu d'éléments. Quelques **leds** et les emplacements pour les touches du clavier numérique. Cette face n'est pas la plus intéressante. Ce sera donc la face arrière qui sera analysée. Elle est présentée en détails sur la figure 3.3.

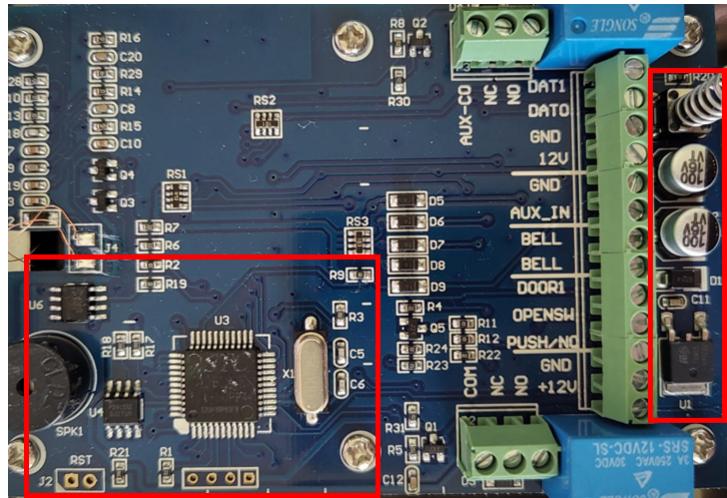


FIGURE 3.3 – Visualisation de la face arrière du **PCB** du digicode.

Plusieurs éléments sont visibles sur ce **PCB**. On y voit au milieu à gauche, deux fils de cuivre labellisés **J4**. Il s'agit de l'antenne **RFID** permettant d'ouvrir la porte à l'aide d'un badge plutôt que de taper une combinaison de chiffres. En bas à gauche, un gros composant cylindrique noir labellisé **SPK1** (*SPeaKer*) qui est un *buzzer* qui permet d'émettre une alarme sonore (à comparer aux alarmes silencieuses) lorsque le boîtier est ouvert par un intrus. Tout en haut à droite se trouve un ressort attaché à un bouton poussoir. Ce ressort est plaqué contre la face arrière du boîtier. Ainsi lorsque celui-ci est ouvert, le ressort se relâche et le bouton se déclenche. C'est le mécanisme anti-intrusion qui permet de déclencher l'alarme. Les deux gros composants bleu sont des relais qui supporte une tension d'entrée de 12 V. En bas à gauche se trouve un micro-contrôleur **STC 10F12XE** labellisé **U3** associé à un cristal labellisé **X1** d'une fréquence de résonance de 24 MHz. À sa gauche, deux circuits intégrés. Le composant **U6** est un amplificateur opérationnel, et le composant **U4** est une **EEPROM P24C512 I²C** de chez **Puya Semiconductor** de 512 kbit. Également, en bas à gauche, on peut y voir un port de 2 trous labellisé **J2/RST**, et un port de quatre trous sans label.

Suite à cet état des lieux, deux grandes zones seront mise en évidence. Tout d'abord, à droite un étage d'alimentation, puis en bas à gauche une zone dite de "traitement des signaux".

3.1.1 Zone 1 - Étage d'alimentation

Le circuit intégré **U1** (en bas de l'étage d'alimentation) est un régulateur de tension monolithique de la marque **Toshiba** (**TA78M05F**). Il possède une limitation interne du courant interne de court-circuit et une protection contre les surcharges thermiques. Il peut délivrer jusqu'à 0,5 A et permet d'abaisser la tension jusqu'à 5 V. La sortie est sur la patte de gauche du composant tandis que l'entrée en 12 V est sur la patte de droite.

Des condensateurs sont utilisés pour stabiliser la tension en sortie du régulateur, mais on y voit également une diode (**D1**) qui assure que le courant ne peut circuler que dans un sens depuis la sortie du composant. Cela permet d'éviter des effets de rebonds et de griller le régulateur de tension. Un test de continuité permet de valider ces affirmations et également de s'assurer que la borne 12 V du bornier alimente bien le **TA78M05F** sur sa patte d'entrée.

3.1.2 Zone 2 - Traitement des signaux

Le composant **U3** est un micro-contrôleur **STC 10F12XE** dans un boîtier **LQFP-44**. Sa tension d'alimentation est située entre 3,3 V et 5,5 V. Il possède une **flash** interne de 12 kbit, une **EEPROM** interne de 1 kbit et, une **SRAM** de 512 kbit. Son *pinout* est tiré de la documentation du composant [24] et est présenté sur la figure 3.4.

Le composant **X1** situé à droite du micro-contrôleur est un cristal oscillant à 24 MHz (il est écrit dessus : 24.000 ce qui correspond à 24 MHZ). Un test de continuité permet de vérifier qu'il est bien relié aux pins 14 et 15 (**XTAL2** et

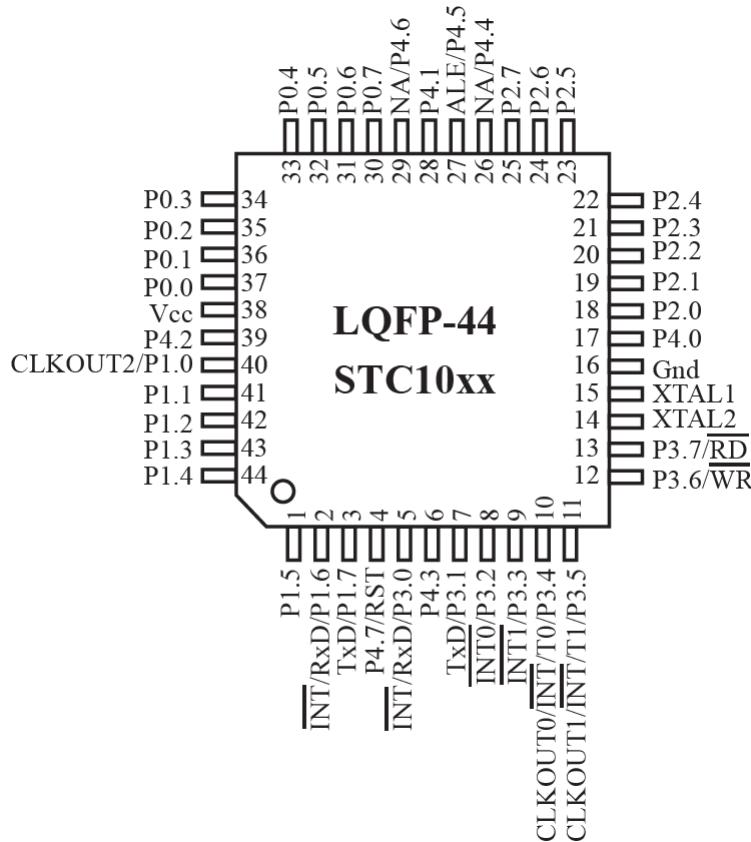


FIGURE 3.4 – *Pinout* du micro-contrôleur **10F12XE** du digicode.

XTAL1 respectivement). Généralement les pins d'un cristal sont toujours reliés à des condensateurs. Ces condensateurs permettent de régler finement la fréquence de résonance. Dans le cas présent, la documentation stipule que les capacités des condensateurs **C5** et **C6** doivent être inférieure ou égale à 47 pF.

Le port de quatre trous situé sous le micro-contrôleur est intéressant. On peut penser intuitivement qu'il s'agit d'un port **UART**. Cela mérite une vérification. Souvent les ports **UART**, bien que nécessitant seulement deux bus de données (**RX** et **TX**), sont représentés avec quatre ports. La masse et la tension y sont ajoutées (**GND** et **VCC** respectivement). La liste ci-dessous présente le branchement de ce port (d'après un test de continuité) :

- Le premier pin (celui de gauche) de ce port est relié au pin 4 (**RST**) du micro-contrôleur en passant par la résistance **R1**.
- Le second pin est relié au pin 5 (**RxD**) du micro-contrôleur.
- Le troisième pin est relié au pin 7 (**TxD**) du micro-contrôleur.
- Le quatrième pin n'est pas relié au micro-contrôleur, mais à la borne 12 V du bornier (près de l'étage d'alimentation).

D'après les branchements, on peut estimer que le second et le troisième pin peuvent être associés à de l'**UART**. Il faudra le vérifier dynamiquement. En ce qui concerne le deuxième port labellisé **J2** et qui est situé à gauche du premier. Les pins sont branchés comme suit :

- Le premier pin (celui de gauche) est relié à la masse du bornier (**GND**).

- Le second pin est relié au premier pin de l'autre port par le biais de la résistance **R21**.

D'après le *pinout* du micro-contrôleur, on peut également voir que le pin 3 semble être associé à **TxD** et que le pin 2 à **RxD**. En suivant les traces et en vérifiant à l'aide d'un test de continuité, on se rend compte qu'ils sont reliés aux diodes **D5** et **D8**. Il faudra aussi déterminer dynamiquement la nature de ces signaux.

Le composant **U4** est également un élément intéressant car il s'agit d'une **EEPROM P24C512 I²C** [23] dans un boîtier **SOP8**. La figure 3.5 présente son *pinout*. On détermine alors les branchements pertinents (voir section 1.2.3) suivants :

- **SDA** est relié au pin 41 (**P1.1**) du micro-contrôleur.
- **SCL** est relié au pin 42 (**P1.2**) du micro-contrôleur.

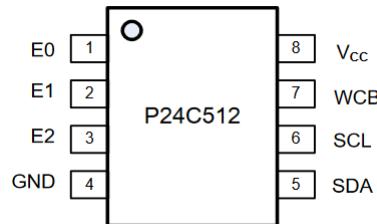


FIGURE 3.5 – *Pinout* de l'**EEPROM P24C512** du digicode.

3.2 Analyse dynamique du PCB

3.2.1 Détermination des fréquences d'horloges

Il a été déterminé par l'analyse statique que le cristal vibrait à 24 MHz. Cela peut se confirmer de manière dynamique (dans le cas où la fréquence n'est pas indiquée sur le cristal) en utilisant un oscilloscope. Le signal en provenance du cristal a été enregistré et est présenté sur la figure 3.6. On constate qu'il a une amplitude d'environ 2 V et oscille bien à 24 MHz. Ainsi on peut dire que le micro-contrôleur **STC10F12XE** est cadencé à 24 MHz.

On a déterminé que l'**EEPROM** communiquait en **I²C**. Il est alors nécessaire d'avoir un signal d'horloge pour utiliser ce protocole. Le micro-contrôleur génère un signal d'horloge sur le pin 10 (**CLKOUT0**). Il est possible de le mesurer de la même manière que pour le cristal. La capture de ce signal est présentée sur la figure 3.7. On constate que la fréquence est nettement plus faible que celle du cristal car elle vaut 125 kHz, ce qui est de l'ordre de grandeur des fréquences utilisées pour le protocole **I²C**.

Autant le signal du cristal est propre, autant celui généré par le micro-contrôleur l'est beaucoup moins.

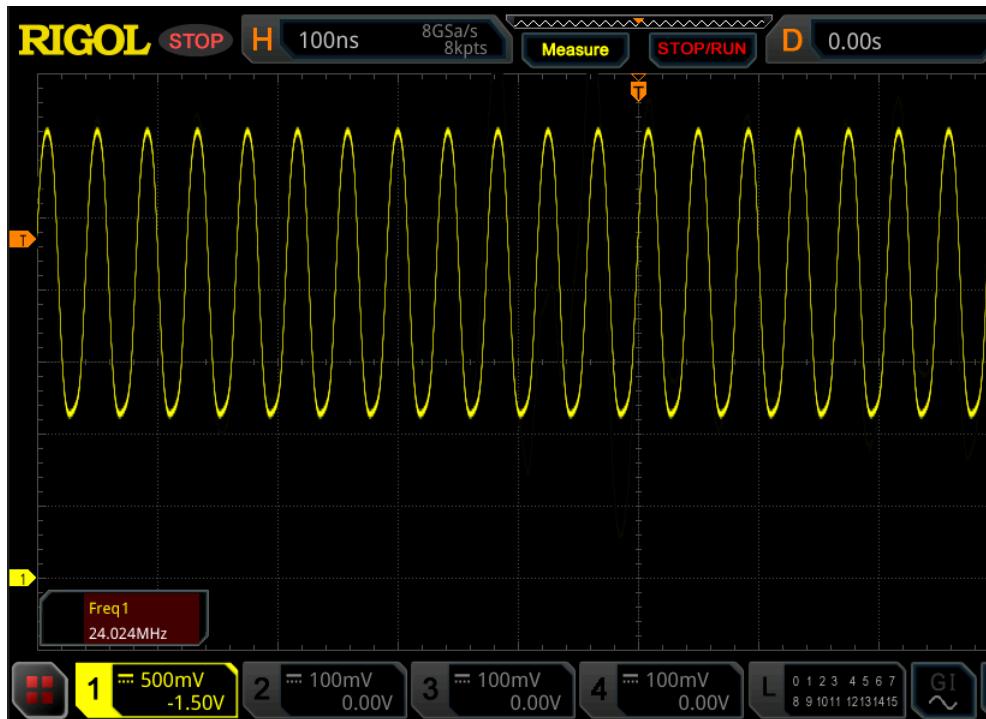


FIGURE 3.6 – Vérification de la fréquence d’oscillation du cristal associé au micro-contrôleur STC10F12XE.

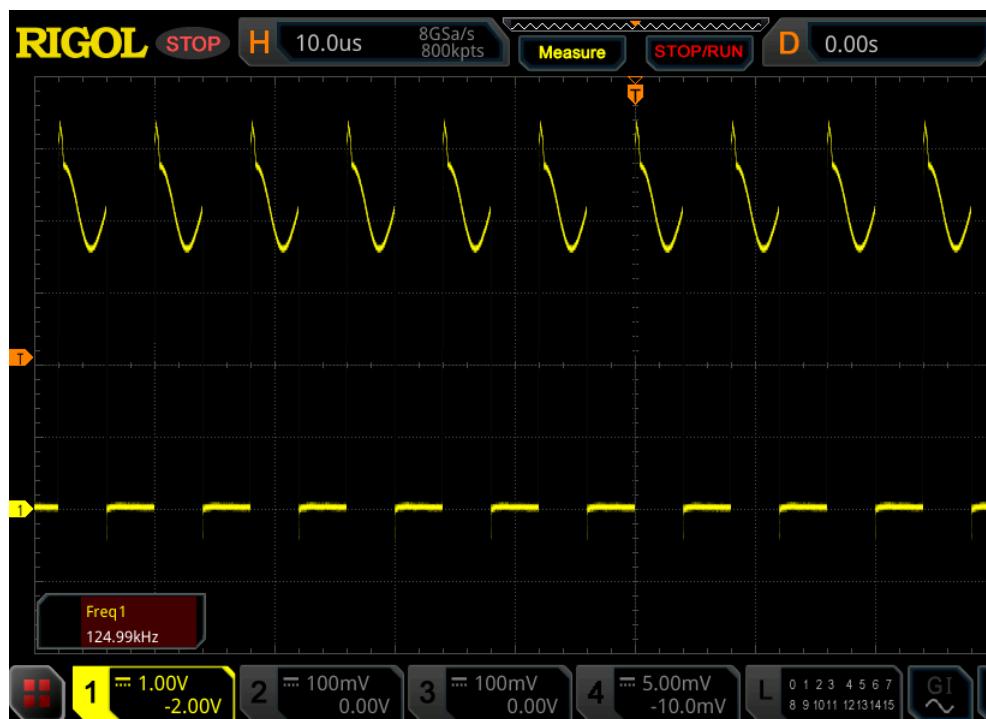


FIGURE 3.7 – Détermination du signal d’horloge I²C utilisé pour communiquer avec l’EEPROM P24C512.

Un autre signal d’horloge est délivré par le micro-contrôleur. Il est délivré par le pin 44 et vaut 1 Hz, ce qui équivaut à changement d’état par seconde. Ce signal est donc probablement utilisé pour effectuer une mesure du temps. Le pin 44 est également disponible sur un via qui se trouve juste à côté (voir figure 3.11). La mesure de ce signal d’horloge a alors été effectuée à la fois sur la patte du micro-

contrôleur et sur le via par l'intermédiaire d'un fil soudé dessus (voir figure 3.8). Cela permet de mettre en évidence l'impact d'un fil soudé sur la mesure d'un signal. Les deux résultats sont présentés sur la figure 3.9. La figure 3.9a représente la mesure du signal directement sur le pin 44, tandis que la figure 3.9b représente la mesure du signal sur le fil soudé au via.



FIGURE 3.8 – Mesure d'un signal d'horloge directement sur le pin 44 du digicode. On y voit également le fil soudé sur le via associé au pin 44.

On voit clairement apparaître un niveau de bruit plus élevé sur la mesure du signal effectuée sur le fil. Comme il a été vu dans la section 1.1.4, différentes sources de bruit peuvent nuire à la qualité de la mesure. Le passage du courant dans un fil (qui de plus, n'est pas isolé) induit, de par sa simple présence, plusieurs types de bruits (capacitif, inductif, radiatif) qui vont venir perturber la mesure du signal. Dans le cas présent, cela ne pose pas vraiment de problème, mais il est bon de noter que tous fils soudés sur un **PCB** va induire du bruit dans les mesures.

3.2.2 Vérification de la présence de l'UART

Durant l'analyse statique, il a été émis l'hypothèse que le port de 4 trous pouvait être associé à de l'**UART**. Pour vérifier cela, un analyseur de signaux a été branché sur les pins associés à **RX** et **TX**. Le schéma de montage est présenté sur la figure 3.10.

Des pins ont été soudés sur les trois premiers trous, mais pas sur le quatrième. En effet, il a été montré lors de l'analyse statique que ce trou était relié au 12 V. L'analyseur logique ne supportant pas une telle tension, aucun pin n'a été soudé sur ce trou pour éviter toute maladresse. Une fois les branchements effectués, il est possible de brancher l'alimentation du digicode et de constater que **TX** et **RX** ne semblent pas utilisés. On estime alors, que dans le cas présent, **RX** et **TX** sont associés à l'**ISP** (*In-System Programming*), ce qui est en accord avec la

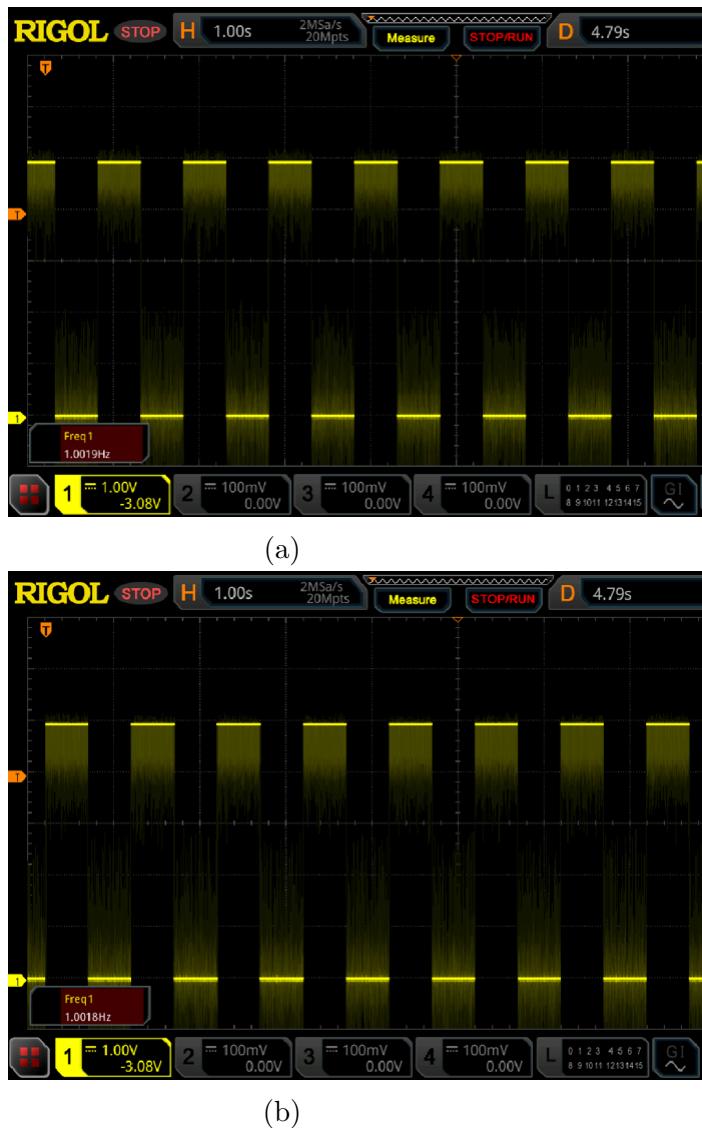


FIGURE 3.9 – Visualisation du bruit lors de la mesure d'un signal sur un fil soudé sur un via. Sur la figure 3.9a, la mesure est effectuée sur la patte du micro-contrôleur ; sur la figure 3.9b, la mesure est effectué sur un fil soudé sur le via.

documentation. L'**ISP** est une fonctionnalité qui permet à un micro-contrôleur d'être reprogrammé alors qu'il est déjà en place sur le **PCB**.

Il a également été montré lors de l'analyse statique que les pins 2 et 3 pouvaient être susceptibles d'être associé à l'**UART**. Il est possible d'y accéder à l'aide d'un via. Pour cela il est nécessaire d'enlever délicatement la résine du **PCB** avec un scalpel de manière à faire apparaître le cuivre sur lequel on pourra souder un fil. Cela est présenté sur la figure 3.11 (sur la figure 3.11a, l'apparition du cuivre au niveau du via; sur la figure 3.11b un fil soudé sur ce même via). Il est nécessaire d'utiliser du flux pour une soudure aussi fine que celle-ci.

Après vérification, il apparaît que ces pins ne sont pas utilisés pour l'**UART**. En effet, la documentation indique que lorsque les pins 5 et 7 du micro-contrôleur sont utilisés pour l'**ISP**, et que le micro-contrôleur n'a pas besoin d'utiliser l'**UART**, il est recommandé de brancher les pins 3 et 4 à **VCC** à travers une diode, ce qui est effectivement le cas. On en conclut qu'il n'y a pas d'**UART**



FIGURE 3.10 – Mesure du signal sur le port de 4 trous du digicode.

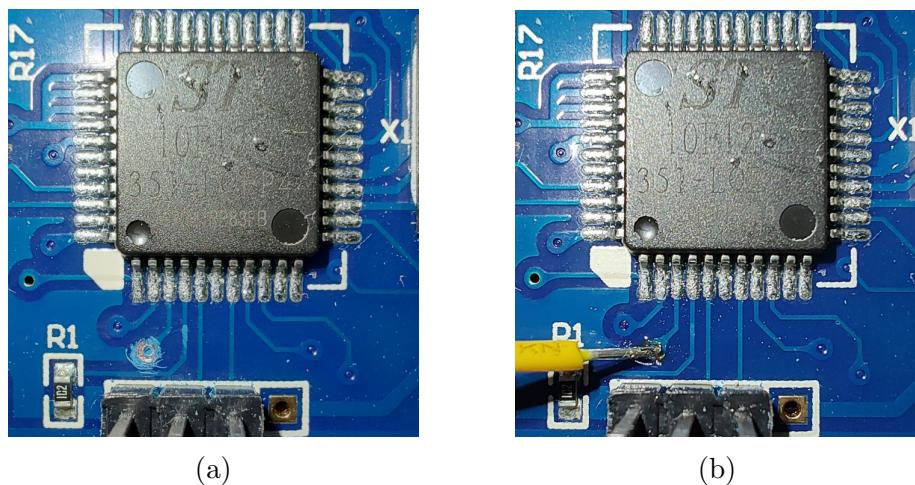


FIGURE 3.11 – Sur la figure 3.11a, apparition du cuivre sous un via par grattage au scalpel. Sur la figure 3.11a, un fil soudé sur ce même via.

disponible sur le digicode.

3.2.3 Détermination des signaux d'entrées

Les signaux d'entrées sont les signaux en provenance des touches du digicode. L'utilisateur doit appuyer sur ces touches pour ouvrir la porte. Il doit donc y avoir un signal électrique correspondant à chaque touche. En effet, le micro-contrôleur doit pouvoir discriminer le signal d'un bouton d'un autre. La figure 3.12 présente les vias permettant d'intercepter les signaux en provenance des boutons. Leurs correspondances sur les pattes du micro-contrôleur ont également été mis en avant.

Quatre vias servent à amener le signal des touches vers le micro-contrôleur.

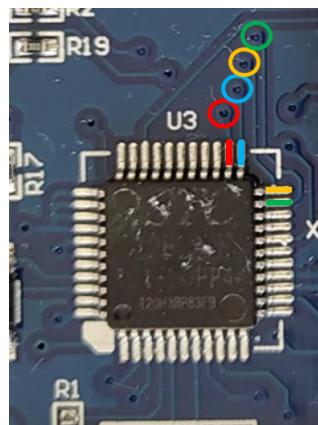


FIGURE 3.12 – Visualisation des via sur le **PCB** et des pattes du micro-contrôleur sur lesquels peuvent être interceptés les signaux issus boutons du digicode.

Les signaux en provenance de la première ligne (touche 1,2 et 3) arrivent sur la patte rouge. La patte bleue récupère les signaux issus de la seconde ligne. La patte jaune récupère les signaux de la troisième ligne, et la patte verte, les signaux de la quatrième ligne.

3.3 Attaques

D'après les résultats obtenus lors de l'analyse du **PCB**, différentes attaques peuvent être imaginées. Certaines seront menées avec plus ou moins de succès dans cette section. Une liste non exhaustive est présentée ci-dessous :

- Mimer l'appui sur l'interrupteur situé à l'intérieur du bâtiment qui permet d'ouvrir la porte.
- Intercepter les signaux issus des touches afin de l'enregistrer et de pouvoir le rejouer par la suite.
- Analyser la consommation du micro-contrôleur lorsqu'un code pin correct/incorrect est tapé.
- Intercepter les communications **I²C** entre le micro-contrôleur et l'EEPROM.
- etc.

3.3.1 Attaques temporelles sur la puissance consommée

Les attaques par canaux auxiliaires, telles que les attaques temporelles sur la puissance consommée, sont des méthodes sophistiquées et subtiles de compromission de la sécurité matérielle. Ces attaques exploitent les variations minuscules mais significatives dans la consommation de puissance d'un dispositif électronique, tel qu'un micro-contrôleur ou un système embarqué, pour divulguer des informations sensibles, notamment des clés cryptographiques.

Lors d'une attaque temporelle sur la puissance consommée, un attaquant mesure la consommation de courant du dispositif cible pendant son fonctionnement, en particulier lorsqu'il exécute des opérations cryptographiques ou des algorithmes sensibles. En observant les fluctuations de la consommation de puissance au fil du temps, l'attaquant peut déduire des informations sur les données traitées

par le dispositif. Par exemple, lorsqu'une opération cryptographique dépend de certaines valeurs de bits, les variations dans les temps de calcul se reflètent dans les variations de la consommation d'énergie. En analysant ces variations, un attaquant peut éventuellement déduire des informations clés, telles que des bits de clé, en fonction des modèles observés. Ces attaques peuvent être particulièrement efficaces contre les systèmes mal protégés ou insuffisamment conçus pour résister à de telles intrusions. Pour atténuer ces risques, les concepteurs de matériel doivent mettre en œuvre des contre-mesures telles que l'inclusion de techniques de masquage de puissance, le brouillage de la consommation de courant, ou l'utilisation d'opérations cryptographiques qui minimisent les variations de temps.

Dans le cas particulier qui nous intéresse, il ne s'agit pas d'une opération cryptographique, mais d'une validation de code pin. Il a été montré que ce type d'attaque sur des mécanismes de validation de mot de passe [15] peuvent être efficaces. Pour effectuer ce type d'attaque, la connaissance des pins du micro-contrôleur est primordiale. On sait que le micro-contrôleur est alimenté par le pin 38 avec une tension d'amplitude comprise entre 3,3 V et 5,5 V. De plus, on a pu déterminer grâce à un test de continuité que les pins 34, 35, 36 et 37 (P0.3, P0.2, P0.1 et P0.0 respectivement) sont reliés entre eux. Un via est situé proche de ces pins, et un autre via est également disponible pour le pin 38 (**VCC**). En grattant la surface du **PCB**, il est alors possible d'accéder à ces signaux. Les pins 34, 35, 36 et 37 nous permettent d'obtenir un signal dit d'**IO** (*Input-Output*) qui prend des valeurs différentes lorsqu'une touche du digicode est appuyée. Cet **IO** permet donc de savoir à quel moment il est utile de regarder la tension sur le pin 38. Sans cette information, il aurait été très compliqué de déterminer la zone temporelle qui correspond au traitement du code pin par le micro-contrôleur.

Le pin correct a été défini à **1234**. Pour vérifier les variations de temps que prend le micro-contrôleur pour effectuer la vérification du pin, il faut enregistrer les traces de l'oscilloscope lorsque des codes incorrects sont entrés. Pour être exhaustif, les mesures suivantes ont été effectuées : **5678** (aucune chiffre correct), **1235** (3 chiffres corrects), **1111** (1 chiffre correct) et **1234** (tous les chiffres corrects). Pour chaque combinaison, une trace de l'**IO** et de **VCC** sont enregistrées. Une trace typique pour le code **5678** est présentée sur la figure 3.13 (en jaune, l'**IO** et en rose **VCC**).

On constate que le digicode est en attente (*idle*) au niveau haut. Lorsqu'une touche est appuyée, l'**IO** passe au niveau bas pendant 100 ms. Ce sera le cas quelles que soient les combinaisons. Suite au 4^{ème} chiffre du code pin, le micro-contrôleur prend un certain temps (étape de traitement du code pin) pour vérifier le code pin. Temps pendant lequel l'**IO** est à l'état haut. Suite à cette routine de validation (le pin **5678** étant incorrect), l'**IO** passe à l'état bas (la porte reste fermée) puis le micro-contrôleur repasse en attente. On observe également des variations d'amplitude dans le niveau de tension, mais ce qui nous intéresse dans cette attaque, sont les variations temporelles, notamment dans l'étape de validation du code pin. La figure 3.14 présente la trace obtenue lors de la soumission d'un code pin correct (**1234**).

Seul le dernier chiffre (**4**) du code pin est visible. On voit bien l'**IO** qui est à l'état bas pendant 100 ms. Lors d'un code pin correct, l'**IO** reste à l'état haut

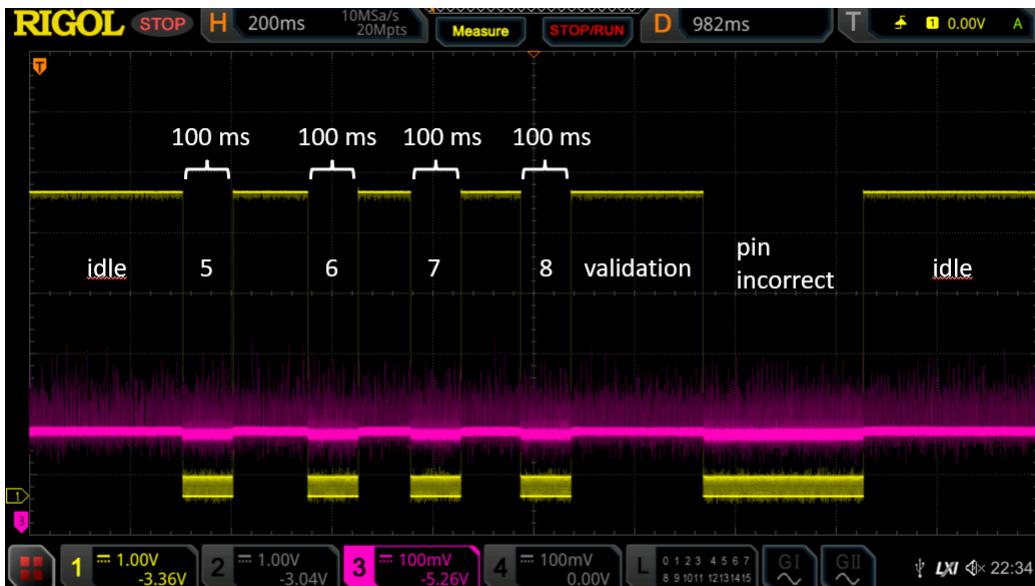


FIGURE 3.13 – Trace de IO (jaune) et VCC (rose) lors de la soumission d'un code pin incorrect (**5678**).

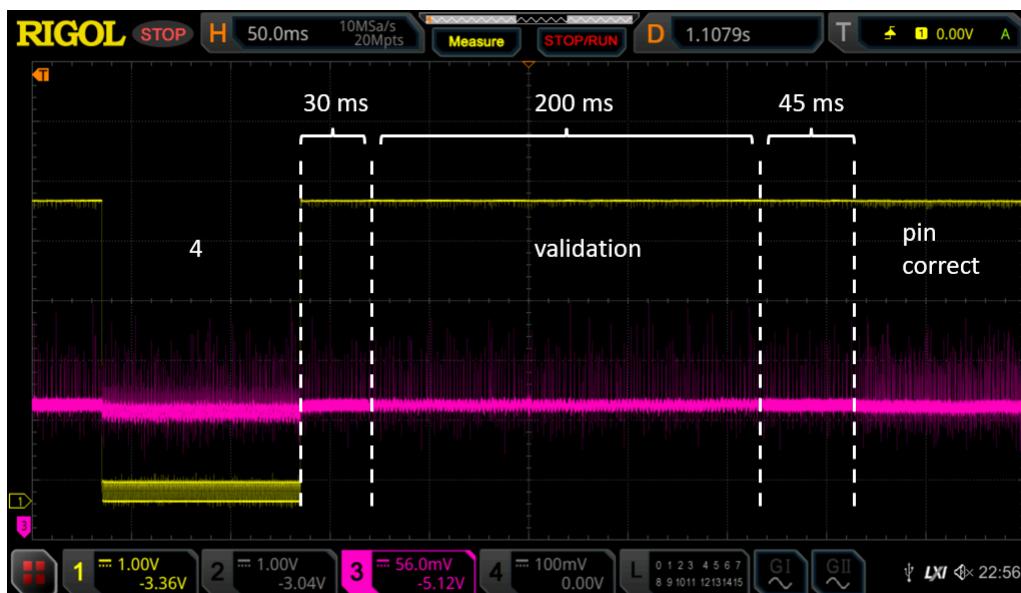


FIGURE 3.14 – Trace de IO (jaune) et VCC (rose) lors de la soumission d'un code pin correct (**1234**). Le dernier chiffre (**4**) est visible ainsi que l'étape de validation.

en suivant l'étape de validation comme cela peut être constaté. En revanche, sur cette figure zoomée sur l'étape de validation, on y constate des variations temporelles. Trois zones peuvent être identifiées. La première zone dure 30 ms, la seconde 200 ms et la troisième 45 ms. Ces trois zones sont toujours présentes quelles que soient les combinaisons soumises. De plus, les temps associés à chaque zone reste également le même. On en conclut que la vérification du code pin par le micro-contrôleur se fait en temps constant. Ainsi, le digicode analysé apparaît résilient aux attaques temporelles sur la puissance consommée.

3.3.2 — Interception I2C

3.3.3 — Extraction du firmware

Chapitre 4

Analyse d'un coffre fort

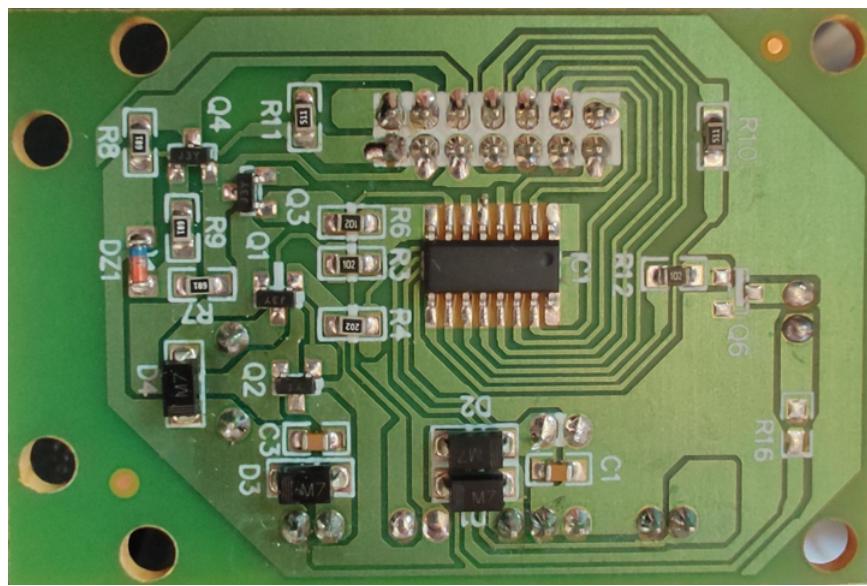
Le coffre fort étudié ici est un coffre fort que l'on retrouve typiquement dans les chambres d'hôtel. La figure 4.1 en présente la face avant. On y retrouve sur la droite un digicode (à comparer avec le digicode présenté dans la section 3), sur la gauche une poignée d'ouverture et, au milieu une serrure de secours réservée à l'équipe gérante de l'hôtel pour ouvrir le coffre si le client oublie son code pin. La serrure est normalement masquée par un cache en plastique. A titre d'information, il s'agit d'une serrure tubulaire basique que l'on peut plus ou moins aisément ouvrir avec des crochets parapluie.



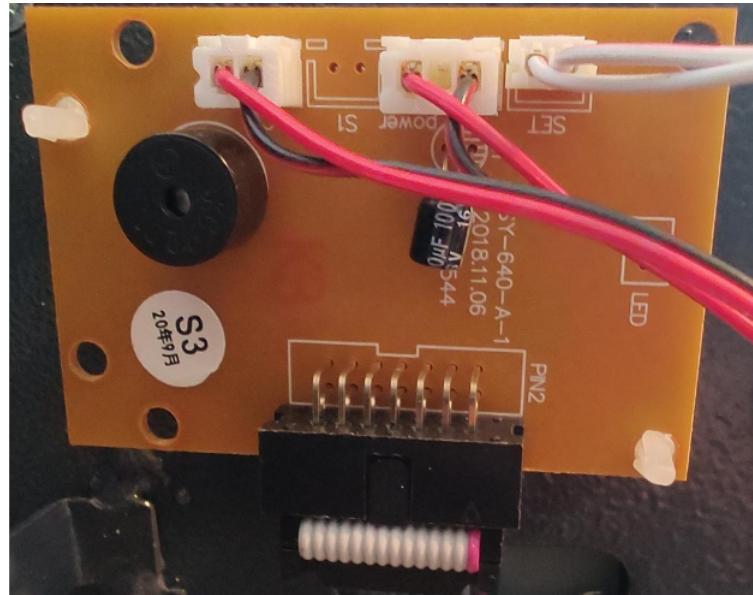
FIGURE 4.1 – Face avant d'un coffre fort de chambre d'hôtel.

La carte électronique qui gère l'ouverture du coffre est située derrière la porte. Lorsque celle-ci est ouverte, il est possible d'y accéder en enlevant le cache de protection. Le **PCB** est présenté sur la figure 4.2 (face arrière sur la figure 4.2a et face avant sur la figure 4.2b).

L'alimentation du coffre est fournie par 4 piles **AA R6 1,5 V** en provenance du connecteur rectangulaire du milieu de la figure 4.2b. Le connecteur rectangulaire de gauche permet d'alimenter le moteur qui bloque la rotation de la poignée, et le connecteur rectangulaire de droite est relié à un bouton situé à l'intérieur de la porte (permet la programmation du code pin). En bas de la face avant, une nappe permet de récupérer la pression des touches du digicode. On y voit



(a)



(b)

FIGURE 4.2 – Sur la figure 4.2a, la face arrière du coffre fort. Sur la figure 4.2b, la face avant.

également un *buzzer* (composant noir) au même format que celui présenté dans la section 3.1, ainsi qu'un condensateur $100 \mu\text{F}$. Ceci étant dit, l'analyse statique se concentrera sur la face arrière du **PCB**.

4.1 Analyse statique du PCB

Après avoir vu la face avant, on peut faire la correspondance entre la nappe associée aux boutons (en haut sur la figure 4.2a) et les connecteurs rectangulaires (en bas sur la même figure). Par association, en bas à droite se trouve les deux pins du connecteur associé au bouton de programmation. Sur sa gauche, les trois pins (dont seulement deux sont utilisés) du connecteur d'alimentation. Encore à

gauche, deux pins associé à un connecteur qui est absent, et enfin, totalement sur la gauche, les pins associés au connecteur d'ouverture du verrou.

L'ensemble des composants étant déjà labellisés, il n'y a pas besoin de faire de masque pour y faire référence dans la suite de cette étude. Le lecteur y trouvera des composants qu'il a déjà abordé dans cet ouvrage. Parmi ceux-ci, un grand nombre de résistances (**R**), quelques transistors (**Q**) et deux condensateurs (**C**). Des composants sont également vu pour la première fois comme, la diode Zener **DZ1** sur la gauche, les quatre diodes de redressement (**D1, D2, D3 et D4**), et enfin un circuit intégré inconnu (**IC1**) au milieu du **PCB**.

Deux étages d'alimentations ont été observés. Le premier (voir figure 4.3a) qui permet d'alimenter l'**IC1** (traitement des données), et le second (voir figure 4.3b) qui permet d'alimenter le moteur du verrou. Dans ce type de configuration, il est normal d'avoir deux plans de masse distincts et deux lignes d'alimentation qui ont pu être déterminé grâce à des tests de continuité.

Il a été vu qu'un bouton permettait la programmation du digicode en reliant un pin de l'**IC1** à la masse. La figure 4.4 permet de mettre en évidence la ligne permettant cela ainsi que le pin du micro-contrôleur responsable de cela.

L'**IC1** et la nappe du digicode contiennent peu de pins (16 et 14 respectivement). Il est intéressant de connaître les traces menant à chacun des pins. La figure 4.5 présente cela de manière détaillée.

4.2 Analyse dynamique du PCB

4.2.1 Mesure des tensions

Les tensions mesurées dans cette section sont prises à l'instant t_0 , c'est-à-dire lorsque l'**IC1** est en mode attente. Aucun code pin n'a encore été transmis. Cela permet d'avoir des niveaux de tensions de références qui peuvent être déterminés soit avec un multimètre soit avec un oscilloscope (voir figure 4.6).

La tension aux bornes de l'étage d'alimentation est importante. En effet, on a vu que cet étage d'alimentation (celui qui est sur la gauche sur la figure 4.2a) gère l'ouverture/fermeture du verrou. Lorsque le coffre fort est en mode attente, la tension aux bornes de cet étage est nulle. En revanche, lorsqu'un code pin correcte est rentré, la tension passe 5,5 V puis s'amortit à 4,6 V en ≈ 30 ms permettant alors de libérer le verrou (5,5 V), puis de le maintenir ouvert (4,5 V). Celui-ci reste ouvert pendant 6 s. Ces étapes sont visibles sur les figures 4.7 et 4.8.

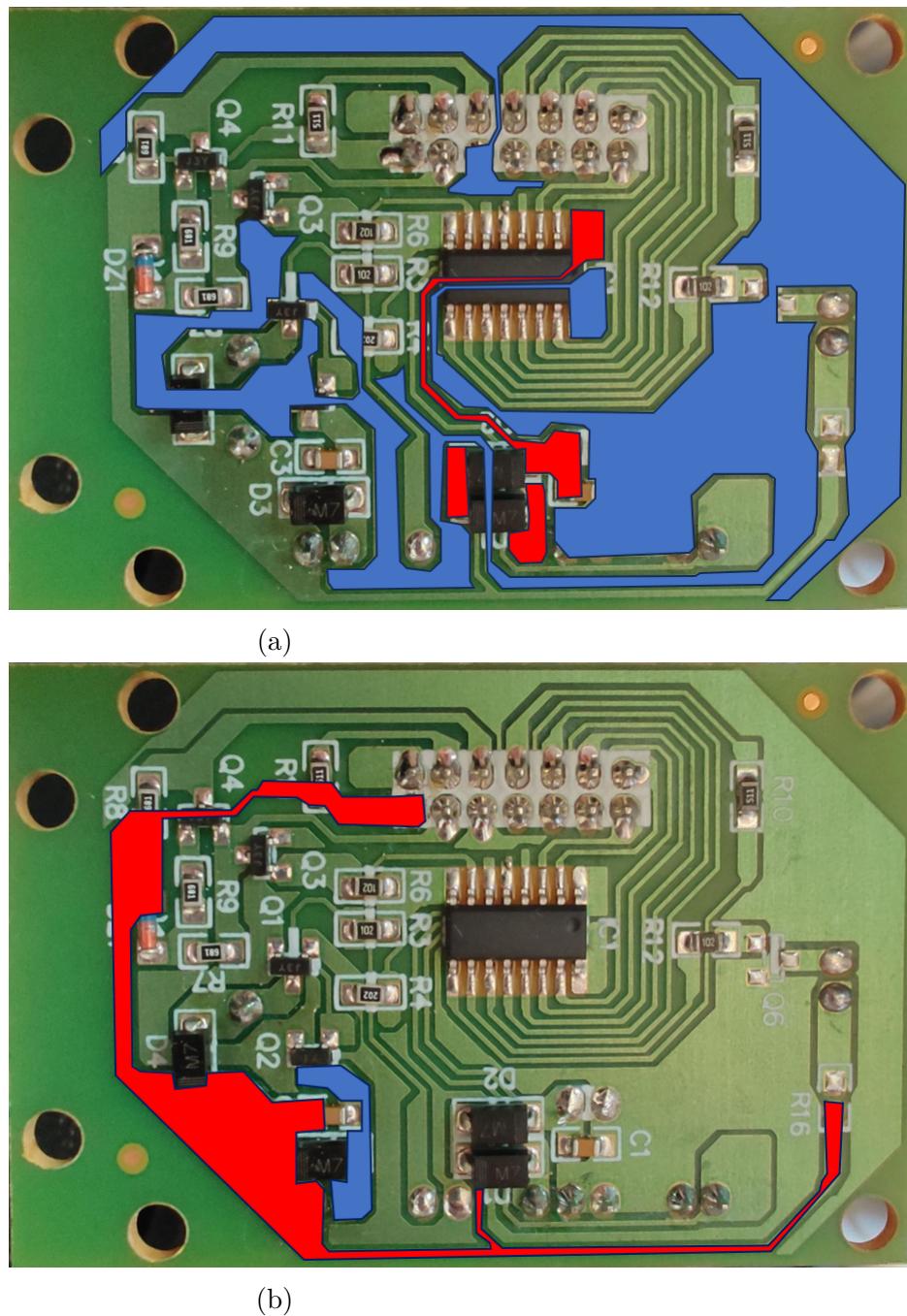


FIGURE 4.3 – Sur la figure 4.3a, le premier étage d'alimentation. Sur la figure 4.3b, le second étage d'alimentation.

4.2.2 — Caractéristiques des diodes de redressements

4.2.3 — Caractéristiques des diodes Zener

4.2.4 Détermination des signaux d'entrées

L'objectif de cette section est de déterminer comment est transmis l'information de la pression d'une touche du digicode au micro-contrôleur (**IC1**). Pour cela, il faut intercepter les signaux en provenance de la nappe qui se trouve sur la face avant du **PCB**. La broche qui y est relié permet facilement d'atteindre les signaux de la première rangée. En revanche les signaux de la seconde rangée sont difficilement accessible. Il a donc été opté de souder des pins sur la face arrière

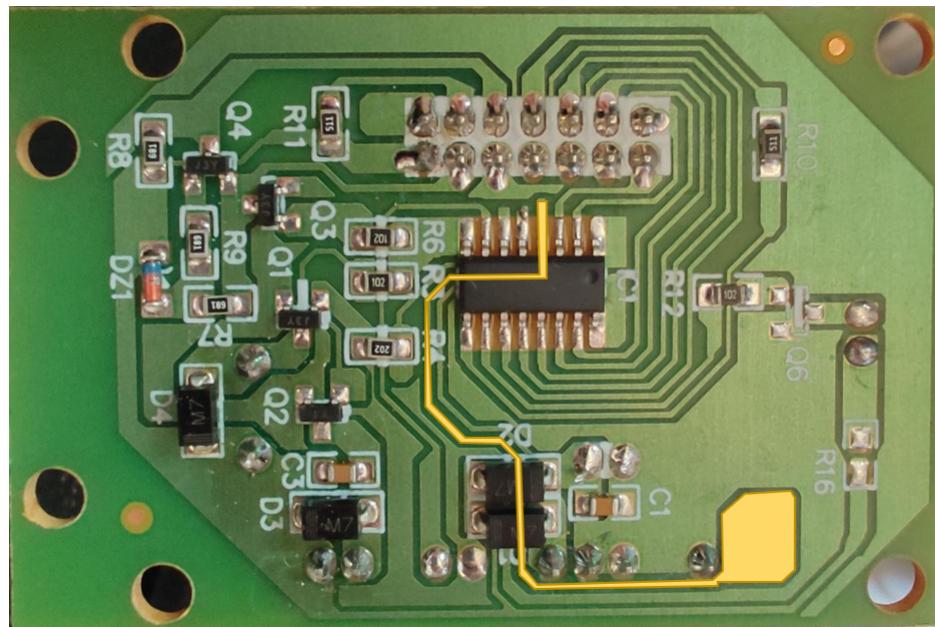


FIGURE 4.4 – Visualisation de la ligne permettant la programmation du digicode.

du **PCB** comme cela peut se voir sur la figure 4.9. Ainsi, l'ensemble des pins est facilement accessible.

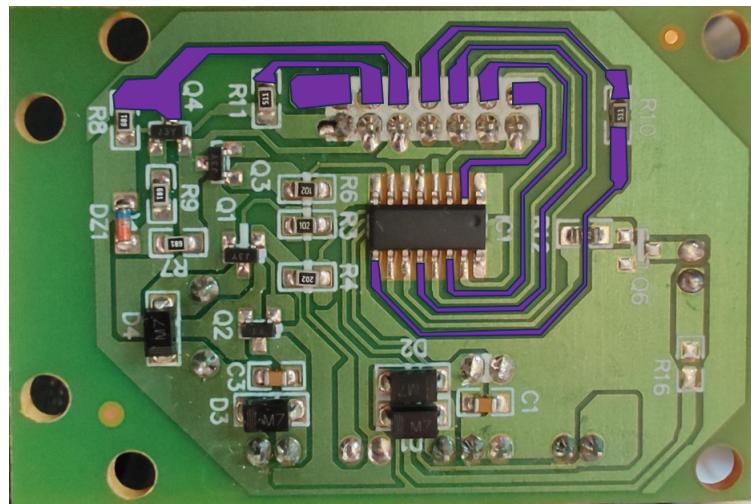
En reprenant la correspondance entre les pins de la broche et ceux du microcontrôleur, on peut se limiter à 8 pins en provenance de la broche. La figure 4.10 permet de nommer les pins de la broches de manière à pouvoir comprendre la discussion qui suit. En utilisant un analyseur logique, chaque touche est enregistré permettant ainsi de déterminer comment est construit le signal. On détermine que chaque colonne sur le digicode est gérée par un pin. Ainsi la première colonne est gérée par le pin 7 qui reste à l'état bas tant que le bouton est appuyé. De même, la deuxième colonne est gérée par le pin 3 et, la troisième colonne par le pin 1.

Une mécanique similaire est effectuée pour déterminer les lignes. 3 pins sont nécessaires pour déterminer une ligne. La première ligne est déterminée par le pin 2 à l'état bas et les pins 5 et 6 à l'état haut. Pour la seconde ligne, le pin 2 et 5 sont à l'état haut et le pin 6 est à l'état bas. Pour la troisième ligne, les pins 2 et 6 sont à l'état haut, et le pin 5 à l'état bas. Et pour finir, la quatrième ligne est définie par les pins 2, 5 et 6 à l'état haut.

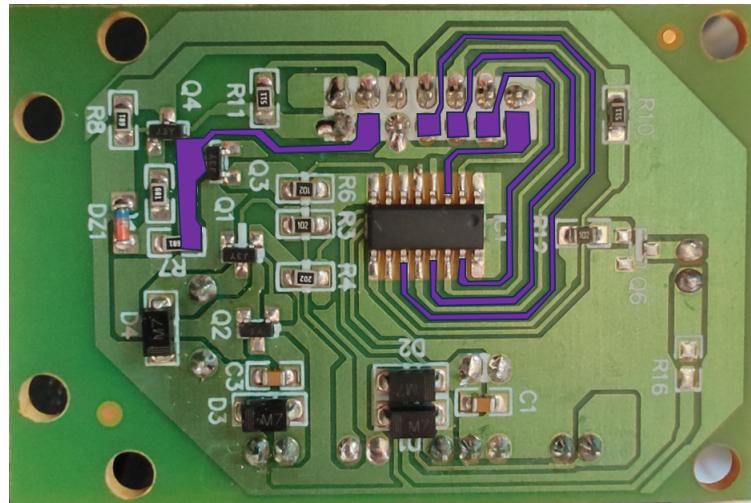
Connaissant cela, il est alors possible de connaître le chiffre appuyé lorsque l'on connaît la trace. La figure 4.11 présente une trace typique de capture du chiffre 5. Le lecteur est encouragé à le déterminer par lui-même.

4.3 Attaques

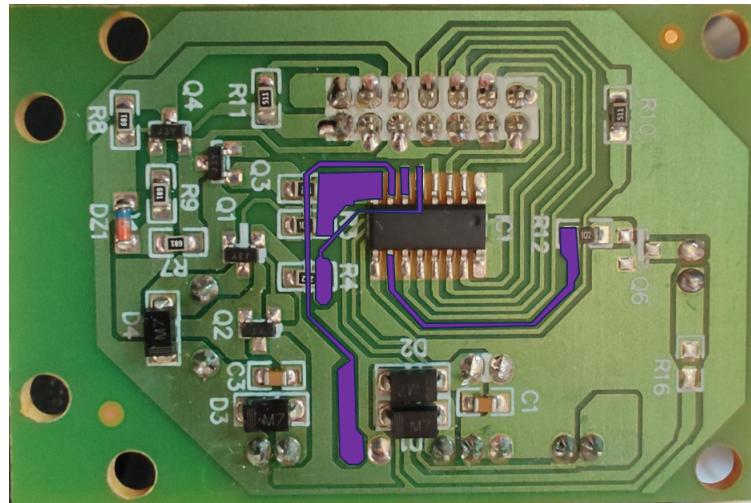
De par les résultats des analyses statique et dynamique, des attaques peuvent être imaginées assez facilement :



(a)



(b)



(c)

FIGURE 4.5 – Visualisation des traces menant aux pins de l'**IC1** et à la nappe du digicode.

- Interception du signal correspondant aux touches du digicode.
- Simulation de l'ouverture du verrou en appliquant une tension de 5,5 V sur l'étage d'alimentation.

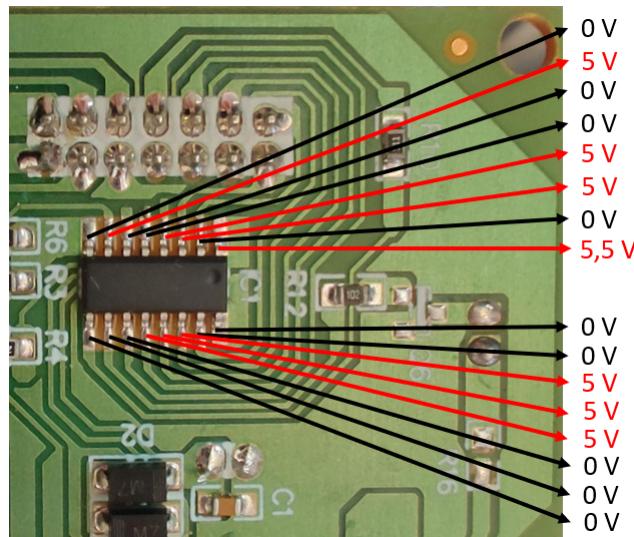


FIGURE 4.6 – Tensions de références sur l'IC1 du coffre fort.



FIGURE 4.7 – Visualisation de la tension sur l'étage d'alimentation d'ouverture/fermeture du verrou lorsqu'un code pin correct est rentré.

— Re-programmation du micro-contrôleur pour modifier le code pin.

Évidemment ces attaques ne sont réalisables que lorsque le coffre est ouvert (ou si un implant a été positionné en avance de phase). S'il ne l'est pas, il faut percer le coffre au bon endroit afin de pouvoir accéder au **PCB**. De plus, l'attaque de la serrure de secours reste toujours une option.

Une attaque matérielle du type "injection de fautes" vaut le coup d'être menée sur ce type de dispositif. Il s'agit d'une attaque de *glitch* sur la tension. Le principe physique est simple, il consiste à couper l'alimentation du micro-contrôleur lorsque celui-ci effectue sa routine de validation du pin. Si le micro-contrôleur est privé d'alimentation trop longtemps, il s'arrête et redémarre. Il faut alors couper suffisamment longtemps pour passer l'ensemble de la routine de vérification mais pas pour redémarrer le micro-contrôleur. Les paramètres physiques sur lesquels il

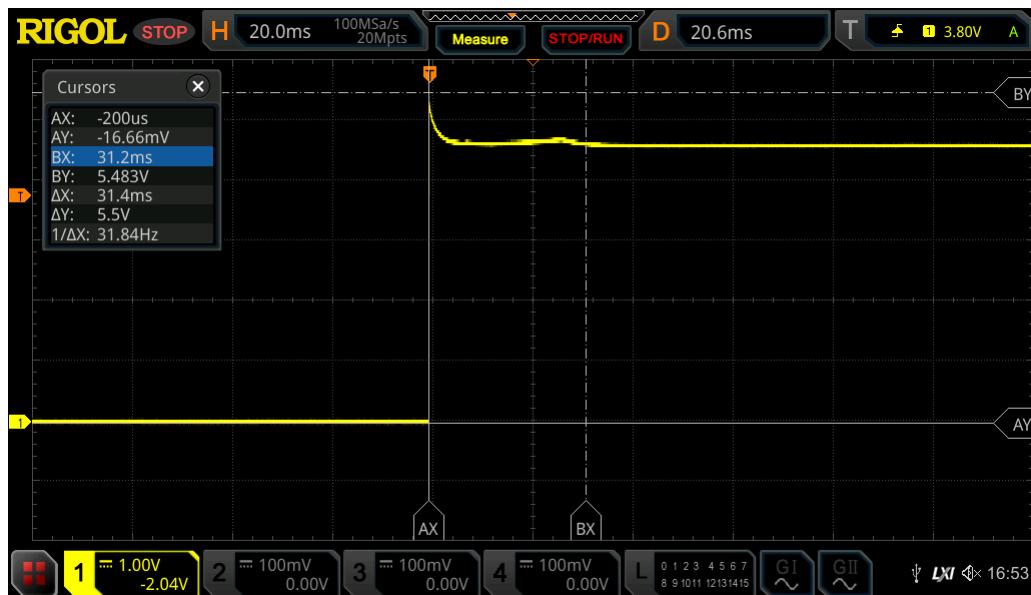


FIGURE 4.8 – Zoom sur la tension d'ouverture du verrou. On y voit le pic de tension à 5,5 V qui permet l'ouverture, puis la tension à 4,5 V qui permet de maintenir le verrou ouvert pendant 6 s.

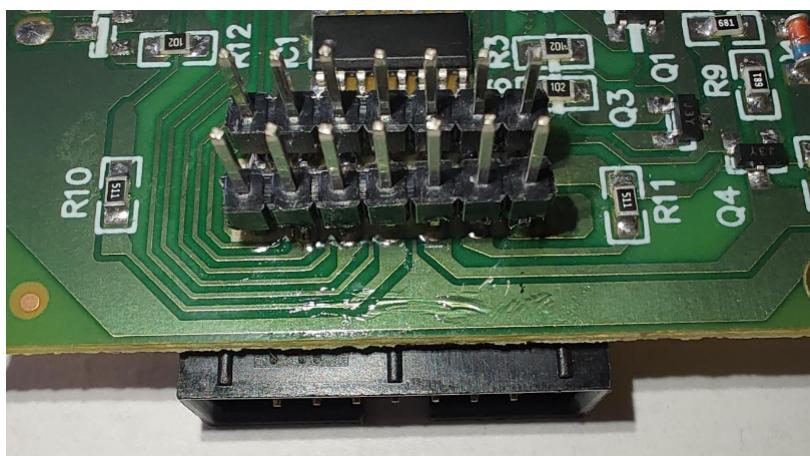


FIGURE 4.9 – Soudage de pins sur la face arrière du **PCB** pour atteindre l'ensemble des pins de la broche reliée au digicode du coffre fort.

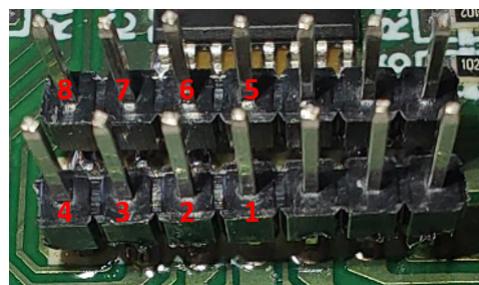


FIGURE 4.10 – Numérotation des pins de la broche qui sont reliés au micro-contrôleur.

faut agir sont donc l'amplitude du *glitch* et la durée de celui-ci. Cette technique est présentée plus en détail dans la section ci-dessous.

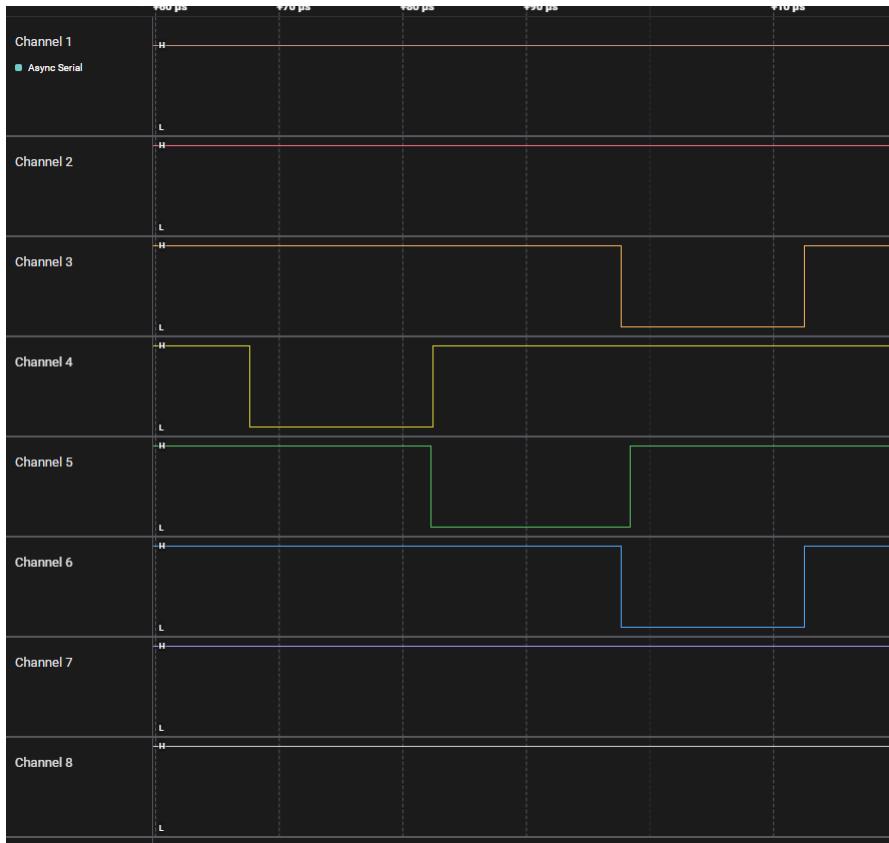


FIGURE 4.11 – Le pin 3 est à l'état bas, ce qui suggère la deuxième colonne du digicode. Les pins 2 et 5 sont à l'état haut et le pin 6 est à l'état bas, ce qui suggère la deuxième ligne. Il s'agit donc du chiffre 5.

4.3.1 Injection de glitch sur VCC

La technique d'injection de fautes par *glitch* sur la tension [26] est une méthode avancée et ciblée utilisée en sécurité matérielle pour compromettre la fiabilité ou le comportement attendu d'un dispositif électronique. Elle implique la création intentionnelle de perturbations temporaires et contrôlées dans la tension d'alimentation d'un système pendant son fonctionnement. Ces perturbations sont conçues pour induire des erreurs transitoires dans le dispositif, susceptibles de perturber son fonctionnement normal et potentiellement de révéler des informations sensibles ou de contourner des mécanismes de sécurité.

Pour exécuter une attaque par injection de fautes par *glitch* sur la tension, un attaquant utilise généralement un générateur de signaux capable de générer des impulsions de tension précises et synchronisées [6]. En identifiant les moments opportuns dans l'exécution du dispositif où une injection de *glitch* pourrait perturber le traitement en cours, l'attaquant tente de manipuler les opérations internes du dispositif, telles que les vérifications de sécurité ou les calculs cryptographiques. Si le *glitch* induit une modification suffisante dans le comportement du système, il pourrait conduire à des conséquences imprévues, telles que des erreurs de calcul ou la révélation de données sensibles, facilitant ainsi une compromission.

La réussite d'une attaque par injection de fautes par *glitch* sur la tension repose sur une compréhension approfondie du dispositif cible, de ses caractéristiques électriques et de ses points faibles potentiels. Les concepteurs de systèmes sécurisés

doivent mettre en œuvre des contre-mesures pour atténuer les risques de telles attaques, telles que l'ajout de circuits de détection de *glitches*, la mise en place de contrôles de tension robustes, ou l'utilisation d'algorithmes de sécurité résistants aux variations de tension.

L'injection de fautes par *glitch* sur la tension est une technique sophistiquée qui exploite les faiblesses électroniques pour manipuler le fonctionnement d'un dispositif électronique. Cette méthode met en évidence l'importance cruciale d'une conception de sécurité matérielle robuste et réfléchie pour prévenir et contrer de telles attaques potentiellement dévastatrices.

Un **MOSFET** (*Metal Oxyde Semiconductor Field Effect Transistor*) peut être utilisé pour générer un *glitch* de tension contrôlé dans un dispositif électronique de la manière suivante :

1. **Configuration du MOSFET** : Le **MOSFET** est configuré en tant que commutateur entre la source d'alimentation du dispositif cible (**VCC**) et la masse (**GND**). Le canal du **MOSFET** agit comme un interrupteur contrôlé par une tension de commande (**Gate**).
2. **Contrôle de la tension de Gate** : En appliquant une tension de commande au terminal **Gate** du **MOSFET**, vous pouvez ouvrir ou fermer le canal du **MOSFET**. Lorsque le **MOSFET** est activé, le courant peut circuler à travers le canal, ce qui perturbe la tension d'alimentation du dispositif cible.
3. **Génération du Glitch** : Pour générer un *glitch*, vous pouvez contrôler précisément le moment où le **MOSFET** est activé et désactivé. En synchronisant cette activation avec des opérations critiques du dispositif cible, telles que des opérations cryptographiques, vous pouvez induire une perturbation temporaire dans la tension d'alimentation, ce qui peut entraîner des erreurs ou des comportements imprévus.
4. **Analyse des Effets** : En surveillant les réponses du dispositif cible aux *glitches* générés, vous pouvez analyser les effets de ces perturbations sur son fonctionnement. Les variations dans les comportements observés peuvent potentiellement révéler des informations sensibles, telles que des données cryptographiques ou des clés, en fonction des modèles observés.

Un **ChipWhisperer Lite** [16] a été utilisé pour générer les *glitches* de cette section. Un exemple de **glitch** multiples sur la tension est présenté en figure ???. Dans le cas étudié ici, l'objectif du *glitch* est de perturber la routine de vérification du code pin. De manière générale, un *glitch* doit être inférieur à 1 ms. Plus ce temps est petit, plus il est possible de cibler des portions petites de codes et donc d'être précis dans l'injection de fautes. Pour cibler temporairement le *glitch*, il faut un signal de déclenchement à partir duquel on peut envoyer le *glitch* (avec un délai par rapport à ce signal de déclenchement). Typiquement, on souhaite envoyer le *glitch* après avoir tapé le code pin, mais avant la validation de la routine de vérification du code pin. Lors d'un audit, il est parfois possible de demander au commanditaire la manière de programmer le micro-contrôleur de manière à fournir un signal de déclenchement à travers un **IO**. Dans le cas présent, cela n'est pas possible, il faut alors trouver une manière d'obtenir un tel signal. Connaissant

la manière dont les signaux issus des boutons du digicode sont interprétés (voir section 4.2.4), il est possible d'envoyer un code pin, puis de déclencher le *glitch* après l'envoi du dernier chiffre.

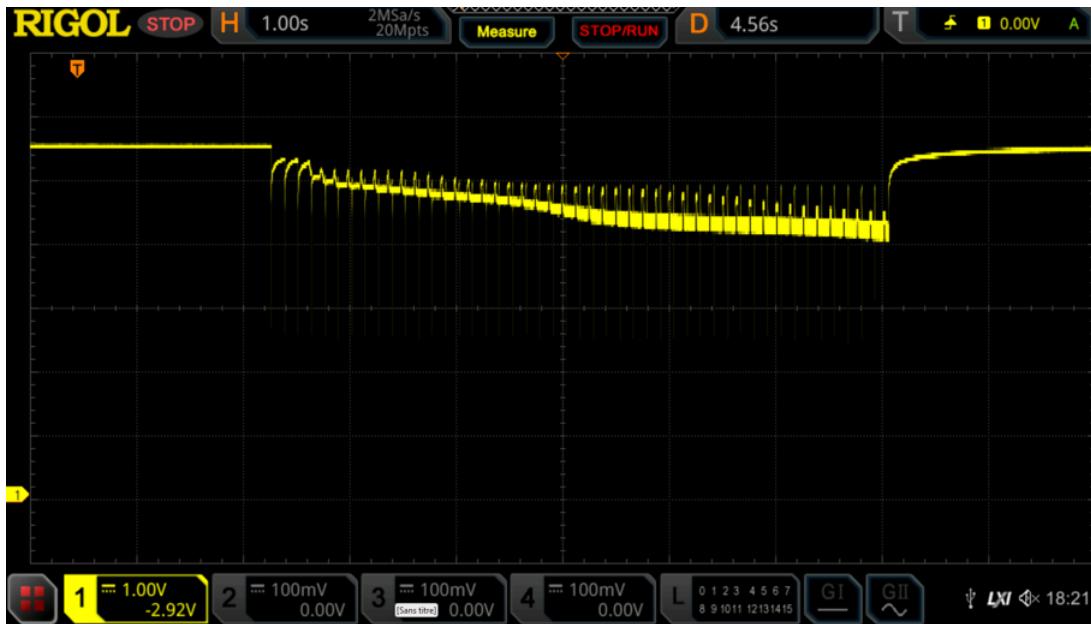


FIGURE 4.12 – Visualisation de plusieurs *glitches* sur la tension d'alimentation d'un coffre fort.

Commençons par regarder l'allure de la tension lorsqu'un code pin OK est soumis (voir figure 4.13), puis lorsqu'un pin KO est soumis (voir figure 4.14). On peut observer que la tension varie lorsque l'**IC1** traite l'information concernant chaque pin (1, 2, 3 et 4), mais également lorsqu'il traite l'information concernant la touche de validation (*). On en déduit alors l'étape de validation du pin.

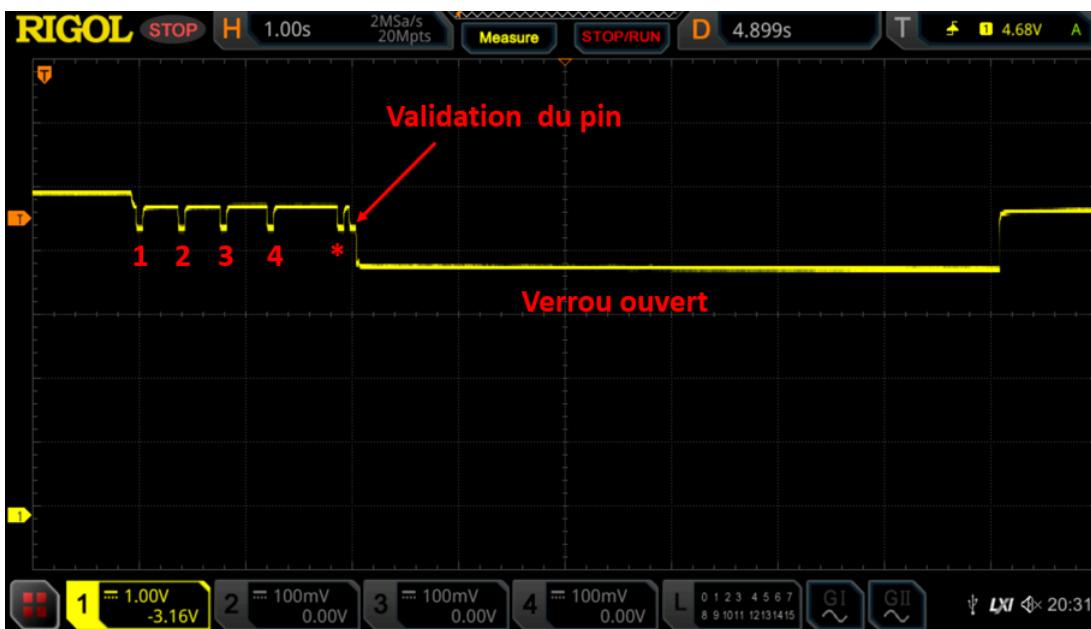


FIGURE 4.13 – Visualisation de la tension sur l'**IC1** du coffre lorsqu'un pin OK est soumis.

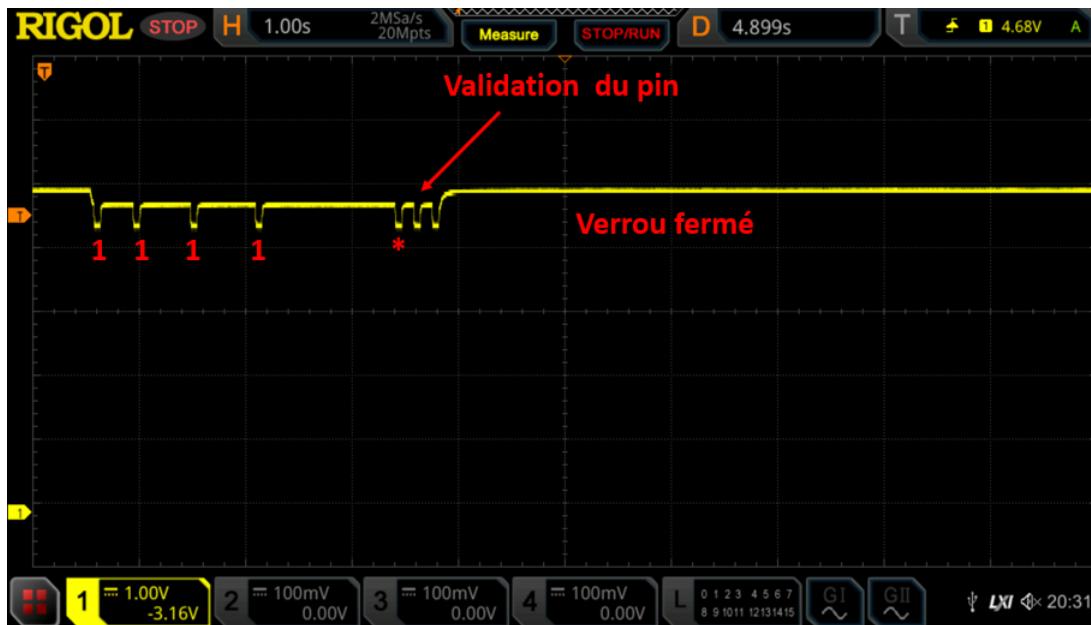


FIGURE 4.14 – Visualisation de la tension sur l'**IC1** du coffre lorsqu'un pin KO est soumis.

La divulgation des routines effectuées par le micro-contrôleur permet de déterminer le moment idéal pour induire un *glitch*. La figure 4.15 permet d'observer cela. Le code pin rentré est correct, cependant le verrou reste fermé signifiant alors que le *glitch* a bien agit sur la routine de validation du pin.

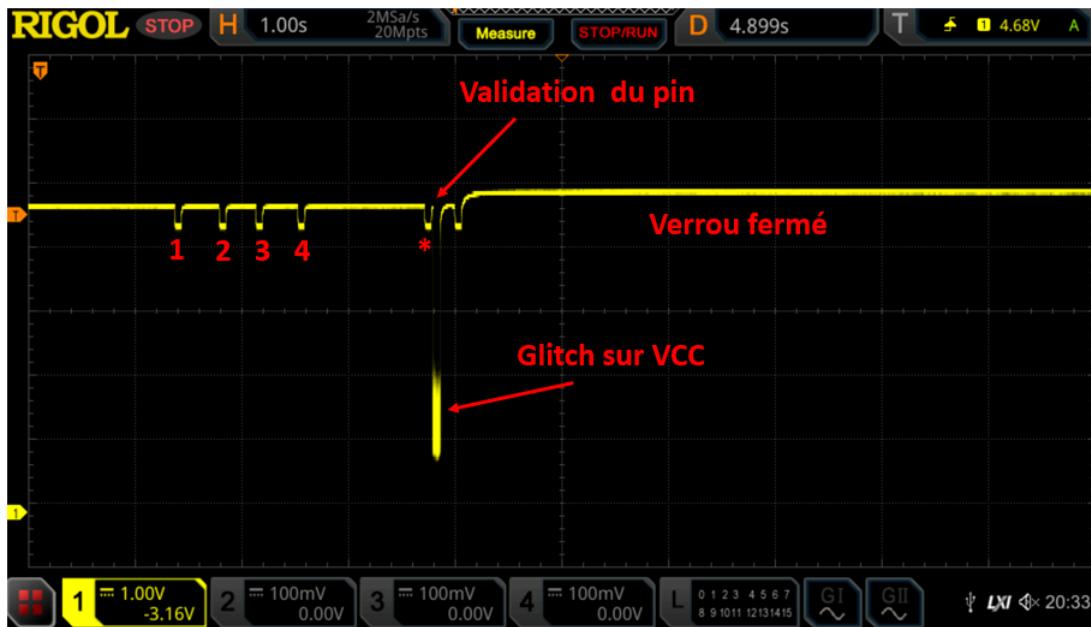


FIGURE 4.15 – Visualisation d'un *glitch* sur la tension lors de la routine de validation d'un pin.

Il n'a pas été en mesure d'effectuer l'étape inverse, c'est-à-dire forcer l'ouverture du verrou lorsque le code pin est KO. Certes les figures ci-dessus permettent de s'assurer que la carte électronique n'est pas protégée contre le *glitch* sur la tension, mais la technique du *glitch* n'est pas une solution miracle. Prenons les bouts de codes C suivant qui représentent deux manières de valider un code pin :

```

1
2  /*
3   * 1st method of pin validation
4  */
5  int pin_validation_1() {
6      if ( pin == 1234 ) {
7          open_lock();
8      }
9
10     close_lock();
11
12     return EXIT_SUCCESS;
13 }
14
15 /*
16  * 2d method of pin validation
17 */
18 int pin_validation_2() {
19     if ( pin != 1234 ) {
20         close_lock();
21         return EXIT_FAILURE;
22     }
23
24     open_lock();
25     close_lock();
26
27     return EXIT_SUCCESS;
28 }
29

```

La première méthode de validation d'un code pin (**pin_validation_1()**) vérifie si le code pin est bien égal à celui qui est renseigné par l'utilisateur (voir ligne 6). Une fois la routine d'ouverture du verrou effectuée, la fonction **close_lock()** permet de le refermer. La seconde méthode de validation (**pin_validation_2()**) vérifie que le code pin renseigné par l'utilisateur est différent du véritable code pin (voir ligne 19). Si tel est le cas, le verrou reste fermé sinon, le verrou s'ouvre grâce à la fonction **open_lock()**. Une des deux méthodes est vulnérable au *glitch*. Il s'agit de la seconde. En effet, il est possible de sauter les instructions qui exécutent la validation du code pin en coupant l'alimentation, puis d'arriver au moment où le micro-contrôleur exécute la fonction **open_lock()**, ce qui ouvre le verrou même en présence d'un code pin faux. Dans le cas de la première méthode, si l'attaquant saute la condition de vérification du code pin alors la fonction **close_lock()** est exécutée quoiqu'il arrive et le verrou reste fermé. Dans le cas présent, bien que la carte électronique soit vulnérable au *glitch* sur la tension comme cela a pu être présenté, la routine de vérification du code pin, elle, ne l'est pas. Il n'est donc, *a priori*, pas possible d'ouvrir ce coffre fort avec la technique du *glitch* car nous sommes apparemment en présence de la première méthode de validation.

Cependant, d'autres attaques d'injection de fautes sont possibles pour contourner cette première méthode de validation. Il est notamment possible d'effectuer des attaques d'injection **LASER** ou **EM** (ÉlectroMagnétique) afin de tenter de modifier par exemple le *byte* = en *byte* !, ce qui reviendrait à ouvrir le coffre fort tant que la combinaison est incorrecte. Évidemment, ce type d'attaque nécessite un gros budget. Il est également possible de *glitcher* uniquement la comparaison et le saut conditionnel. Cependant cela requiert une précision extrême car il s'agit alors de compromettre l'exécution de seulement deux instructions soit quelques cycles d'horloges uniquement.

Chapitre 5

Analyse d'une carte à puce à contact

Une carte à puce [12] est une carte à circuit intégré qui contient un micro-processeur. Ce micro-processeur a des pattes d'entrées/sorties qui se retrouvent sur la carte au niveau des contacts métalliques. Les cartes à puce sont utilisées pour stocker et traiter des informations de manière sécurisée, et elles sont couramment utilisées dans divers domaines tels que la banque (carte de débit, carte de retrait), les télécommunications (carte **SIM**), l'identification et l'accès (carte de chambre d'hôtel, carte d'authentification **Kerberos**). Le micro-processeur intégré peut exécuter des instructions et des programmes. Il permet de réaliser des opérations de calcul, de stocker des données et de gérer des transactions. La carte à puce possède une mémoire intégrée qui permet de stocker des données, des clefs de sécurité, des informations d'identification ou d'autres informations pertinentes pour l'utilisation de la carte.

Les caractéristiques physique de la carte à puce sont définies par plusieurs normes. Les plus courantes sont présentées dans la liste ci-dessous :

- **ISO 7816** : Cette norme définit les caractéristiques physiques des cartes à puce, comme leur taille, leur épaisseur, leur emplacement des contacts électriques et les protocoles de communication de base. Elle est largement utilisée pour les cartes à puce de type ID, bancaires et SIM.
- **EMV** : Cette norme est utilisée principalement dans les cartes de paiement et les terminaux de paiement électronique. Elle spécifie les protocoles de communication et de sécurité nécessaires pour les transactions de paiement sécurisées par carte à puce.
- **ISO 14443** : Cette norme définit les protocoles de communication pour les cartes à puce sans contact (**RFID** : *Radio Frequency IDentification*) et les lecteurs. Elle est souvent utilisée pour les cartes de transport en commun et les cartes d'accès.
- **ISO 7810** : Cette norme définit les dimensions physiques standard des cartes d'identification, incluant les cartes à puce.
- **ISO 8583** : Bien qu'elle ne concerne pas directement les cartes à puce, cette norme définit les messages utilisés dans les transactions financières électroniques, y compris celles effectuées avec des cartes à puce.
- **GlobalPlatform** : Cette organisation développe des spécifications pour

les infrastructures de gestion de cartes à puce multi-applications et multi-domaines, ce qui permet d'utiliser une même carte pour différents services (paiement, identification, etc.).

- **JCOP (Java Card OpenPlatform)** : Bien qu'il ne s'agisse pas strictement d'une norme, **JCOP** est une plateforme de cartes à puce basée sur **Java Card**, largement utilisée pour développer des applications sur des cartes à puce.

La plupart des cartes à puce du marché suivent la norme **ISO 7816** qui est composée de 15 parties. La norme entière ne sera pas retranscrite ici, mais les points importants seront présentés dans les sections qui suivent.

Il faut bien noter que la carte à puce est un micro-ordinateur à part entière qui possède un **CPU**, une **RAM**, une **ROM** et une **EEPROM**. Elle a un système de fichiers qui permet d'organiser les données dans la mémoire et est donc capable de stocker et d'exécuter des applications (*applet*) et même d'effectuer des calculs cryptographique comme du **RSA** (*Rivest-Shamir-Adleman*), **DES** (*Data Encryption Standard*) ou de l'**AES** (Advanced Encryption Standard).

5.1 Théorie

5.1.1 Protocoles de communication

Le terminal fournit l'alimentation ainsi qu'un signal d'horloge. Les données sont échangées par une liaison série dont le *baudrate* est souvent compris entre 9600 bauds et 230400 bauds. La liaison est *half-duplex*, ce qui implique que la carte ne répond qu'aux commandes qui lui sont envoyées. Rien n'est initié depuis le **CPU** de la carte sans que la demande n'ait été effectuée par le terminal. On verra par la suite que cela n'est pas tout à fait vrai car la norme prévoit que la carte doit émettre spontanément un bloc d'octet lors d'une remise à zéro.

Selon la norme **ISO 7816**, le terminal produit des requêtes que l'on appelle **APDU** (*Application Protocol Data Unit*). Ces requêtes (ou messages) suivent donc la norme et sont constitués de plusieurs blocs d'octets. La réponse est tout aussi codifiée. Un **APDU** comporte au moins 5 *bytes* (**CLA**, **INS**, **P1**, **P2** et **P3**). Des *bytes* optionnels peuvent également être ajoutés. La longueur **L_c** de ces *bytes* optionnels est définie dans l'octet **P3**. Une fois l'**APDU** reçu, la carte émet un message de réponse qui comprend sa longueur **L_e**, les *bytes* d'informations et un code de status **SW** (*Status Word*). Le code de status est composé de deux octets : **SW1** et **SW2**. La liste ci-dessous détaille la signification des octets composant un **APDU** et sa réponse :

1. Requête APDU

- **CLA (CLAss)** : la classe indique la structure et le format des commandes et des réponses
- **INS (INSTRUCTION)** : l'instruction de la commande
- **P1, P2** : les paramètres de l'instruction
- **P3 ou L_c** : le nombre d'octets des *bytes* optionnels

- **Octets optionnels** : Les octets optionnels dont la longueur est spécifié par le paramètre **P3**.
- La figure 5.1 permet de visualiser cela.

2. Réponse APDU

- **L_e** : la longueur du message de réponse
- **Message** : le message dont la longueur est spécifiée par **L_e**
- **SW (Status Word)** : un code de status permettant de déterminer si la requête a été traitée avec succès ou erreur.

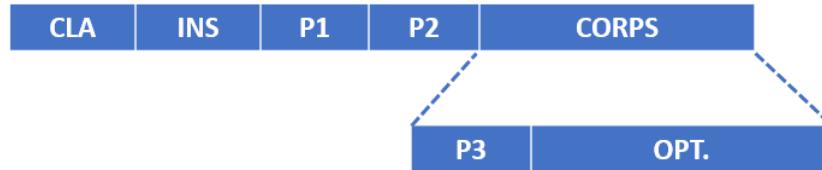


FIGURE 5.1 – Visualisation d'une requête APDU.

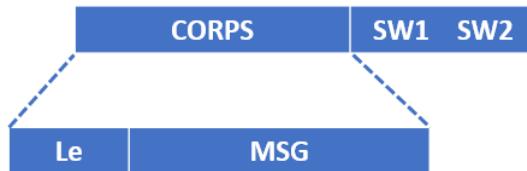


FIGURE 5.2 – Visualisation d'une réponse APDU.

Comme il a été énoncé plus haut, la carte est censée émettre un bloc d'octets lors d'une remise à zéro (le seul cas où la carte émet un message sans qu'il n'y ait eu d'**APDU**). Ainsi dès que la carte est mise sous tension, elle envoie un message d'initialisation appelé : réponse au *reset* (**ATR** : *Answer To Reset*). L'**ATR** fournit un nombre varié de paramètres liés à la carte dont le numéro de série de la puce, le *baudrate*, et le protocole de transport. Les deux protocoles de transports les plus utilisés sont le T = 0 et le T = 1. Ces protocoles définissent la manière dont les commandes et les réponses sont transmises entre la carte et le terminal. Les principales différences entre les deux sont présentées ci-dessous :

1. Protocole T=0

- **Communication caractère par caractère** : Le protocole **T=0** utilise une communication caractère par caractère. Chaque caractère (octet) est transmis séparément entre la carte et le lecteur, avec des bits de départ et d'arrêt ainsi qu'un *bit* de parité facultatif.
- **Contrôle de flux par la carte** : Dans **T=0**, la carte contrôle le flux de données en attendant des signaux de demande (**ACK** : *ACKnowledge*) du lecteur avant d'envoyer chaque caractère.
- **Longueur variable des commandes et des réponses** : Les commandes et les réponses peuvent avoir des longueurs variables, ce qui permet une flexibilité dans les échanges de données.
- **Gestion des erreurs** : Les erreurs sont gérées à travers des procédures de retransmission et de demande de répétition de caractères.

- **Convient aux transmissions à basse vitesse : T=0** est généralement utilisé pour les communications à basse vitesse.

2. Protocole T=1

- **Transmission par blocs** : Le protocole **T=1** utilise une communication par blocs, où plusieurs caractères sont regroupés en blocs et transmis en une seule fois.
- **Contrôle de flux par le lecteur** : Dans **T=1**, le lecteur contrôle le flux de données en utilisant des séquences d'échange pour demander et confirmer la transmission des blocs.
- **Longueur fixe des blocs** : Les blocs ont une longueur fixe, ce qui peut entraîner un certain gaspillage d'espace si les données ne correspondent pas exactement à la taille des blocs.
- **Gestion des erreurs** : Le protocole **T=1** utilise des mécanismes de détection et de correction d'erreurs pour assurer la fiabilité des données transmises.
- **Convient aux transmissions à haute vitesse** : **T=1** est souvent utilisé pour les communications à des vitesses plus élevées que **T=0**.

Il est à noter qu'il existe d'autres protocoles moins usuels :

- **T=2/3** : Réservé pour les opérations *full duplex*
- **T=4** : Réservé pour les transmissions *half duplex* améliorées
- **T=5..13** : Réservé pour un usage future
- **T=14** : Protocole non-ISO
- **T=15** : Réservé pour un usage future

L'**ATR** est l'unique moyen qu'a le terminal pour identifier une carte. Le tableau 5.1 donne certains exemples d'**ATR** et une liste plus fournie est disponible dans la référence [18]. Le code de status d'une commande acceptée et réalisée avec succès est **90 00**. Au contraire lorsque des erreurs se sont présentées, le code de status sera de la forme **SW1 SW2**. Une liste étouffée des code de status pour la norme **EMV** est consultable dans l'annexe [?].

ATR	Type
3B 02 10 50	carte Visa
3B 02 14 50 11	carte Master Visa
3B 04 A2 13 10 91	carte Siemens SLE 4432/42
3B 16 95 D0 01 6C FD 0D 00	carte SIM Virgin Mobile
3F 67 25 00 2A 20 00 0F 68 90 00	carte SNCF Grand Voyageur

TABLE 5.1 – Exemples d'**ATR**.

Pour communiquer simplement avec une carte, il suffit alors d'utiliser le protocole **T=0** (communication caractère par caractère), de lire l'**ATR**, puis d'envoyer des **APDUs**. Les réponses aux **APDUs** seront alors le résultat de la commande. Évidemment, il n'apparaît pas aisément d'envoyer un **APDU** correct. En effet, les cartes à puces étant des objets orientés sur la sécurité, les classes (**CLA**) et les instructions (**INS**) ne sont pas forcément publiques. De plus, comme dans toute communication, connaître la commande ne suffit pas à l'utiliser. Il faut également

connaître les paramètres qui lui sont associés. Suite à cela, il faut également être en mesure de comprendre la réponse. Quand il s'agit de texte pur, il n'y a pas trop de difficulté, mais lorsqu'il s'agit d'une réponse hexadécimale pure, des difficultés commencent à apparaître.

5.1.2 Le système de fichiers

Une carte à puce doit pouvoir contenir des données comme par exemple un historique de transactions, des certificats, des données d'identité ou de santé, des codes d'accès, des applications, etc. Ces données ne sont pas stockées de manière erratique dans la mémoire, mais dans un système de fichiers. Le système de fichiers d'une carte à puce est une structure hiérarchique qui permet d'organiser et de gérer les données stockées sur la carte. Il peut être basé, sans que ce soit toujours le cas, sur des normes spécifiques telles que les normes **ISO 7816** pour les cartes à puce à contact ou **ISO 14443** pour les cartes à puce sans contact. Suivant la norme **ISO 7816**, le système de fichier est conçu comme suit :

- **MF (*Master File*)** : C'est le répertoire racine qui contient les répertoires et les fichiers.
- **DF (*Dedicated Files*)** : Ce sont des répertoires spécifiques pour chaque application ou groupe d'applications.
- **EF (*Elementary Files*)** : Ce sont les fichiers individuels qui stockent les données spécifiques, tels que les informations personnelles, les clés, les certificats, etc.
- **Répertoires parents et enfants** : Le système de fichiers peut comporter une hiérarchie de répertoires parent/enfant, permettant d'organiser les données de manière logique.
- **ACL (*Access Control Lists*)** : Des règles de contrôle d'accès peuvent être définies pour restreindre l'accès à certains fichiers ou répertoires en fonction des droits d'utilisateur.
- **Fichiers transparents et linéaires** : Les fichiers transparents stockent des données dans un format brut, tandis que les fichiers linéaires stockent des données dans un format structuré avec des enregistrements.

Chaque type de carte à puce (carte de paiement, carte SIM, carte d'identité électronique, etc.) peut avoir son propre système de fichiers adapté à ses besoins et aux normes qui la régissent. La structure exacte et les règles de gestion des fichiers peuvent varier en fonction de l'application et du type de carte à puce. Le formalisme présenté ci-dessus n'est que le cas le plus général. La figure 5.3 permet de visualiser l'allure du système de fichiers d'une carte à puce suivant la norme **ISO 7816**.

Il n'y a qu'un seul **MF** tandis qu'il peut y avoir plusieurs **DF**. Un **DF** ne contient pas forcément un **EF** et peut donc être "vide". Un **EF** est le niveau le plus bas dans l'architecture de ce système de fichiers (il peut y en avoir plusieurs dans un **DF**). Comme cela a été présenté, une **ACL** existe et empêche donc l'accès à certains **DF** et/ou **EF** d'être accédés sans les autorisations nécessaires. Parfois un code pin est suffisant pour accéder à certains **DF/EF** et parfois il faut un code constructeur spécifique. Ainsi, le fait de posséder la carte n'offre

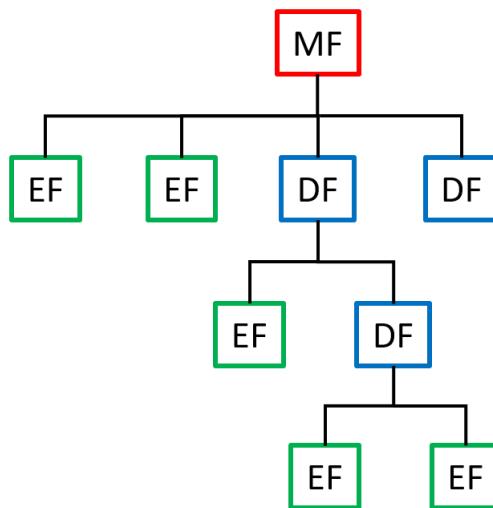


FIGURE 5.3 – Visualisation du système de fichiers d'une carte à puce.

par forcément la possibilité de la modifier dans son entièreté (par exemple, en ce qui concerne l'accès à certains certificats ou de données d'identités/santés). Les fichiers (**MF/DF/EF**) ont des noms. Par manque de place dans la mémoire, ces noms de fichiers ne sont composés que de deux octets. Ainsi le **MF** a le "nom" **3F 00** et il peut être vu comme le répertoire racine. Les **DF** et les **EF** peuvent être vu comme les dossiers et les fichiers respectivement. Par exemple, pour sélectionner le **MF** d'une carte **SIM**, l'**APDU** suivant peut être utilisé :

```
1 A0 A4 00 00 02 3F 00
```

Dans ce cas là, la classe pour la carte **SIM** est **A0**, **A4** est l'instruction de sélection qui ne nécessite aucun paramètre (**P1=00** et **P2=00**). La valeur de **P3** est **02** car le fichier **MF** (**3F 00**) est composé de 2 *bytes*.

Une carte à puce peut (et doit) contenir des applications qui permettent d'effectuer certains calculs. Il s'agit le plus souvent de routine de chiffrement. La norme **ISO 7816** définit le système de numérotation et les procédures d'enregistrement et d'attribution des identifiants des applications (**AID** : *Application IDentifier*). A chaque application est associé un unique **AID**. Un **AID** est composé de plusieurs *bytes* allant de 5 à 16. Les 5 premiers *bytes* représentent le numéro d'enregistrement du fournisseur d'application (**RID** : *Registered Application Provider IDentifier*). Les *bytes* suivants composent un identifiant optionnel (**PIX** : *Proprietary Application Identifier eXtension*). Ainsi, une application peut se nommer : **A0 00 00 00 03 10 10** (avec le **RID=A0 00 00 00 03** et le **PIX=10 10**). Le **RID** signifie dans ce cas particulier qu'il s'agit d'une application **Master Card** et le **PIX** qu'il s'agit d'une application concernant les carte de crédit/débit. Pour sélectionner une telle application, il suffit d'utiliser la même commande que pour sélectionner le **MF** (avec le paramètre **P1=04** signifiant qu'il s'agit de la sélection d'une application) :

```
1 A0 A4 04 00 07 A0 00 00 00 03 10 10
```

Cette application retourne un code de status **61 29**, ce qui signifie que le retour de l'application renvoie **0x29 bytes**. Pour les lire, il faut donc le demander à la carte à l'aide de l'**APDU** suivant :

```
1 00 C0 00 00 29
```

Ici, **C0** est l'instruction de récupération d'une réponse qui ne prend aucun paramètre (**P1=00** et **P2=00**) et qui attend en **P3** la longueur à réceptionner (ici **0x29**). On se rend compte que la connaissance des *bytes* de classe et d'instruction est une condition indispensable à la navigation dans le système de fichiers et à l'utilisation des applications contenues dans la carte.

5.2 Applications

La théorie nécessaire pour lire une carte à puce à contact étant présentée, il sera question dans cette section d'une mise en pratique. Pour lire une carte à puce, il faut se munir d'un lecteur **PC/SC** comme celui présenté sur la figure 5.4. Il s'agit d'un lecteur **Omnikey 3121** permettant de lire un grand nombre de cartes à puce (5 V, 3 V, 1,8 V, norme ISO 7816, classes A, B et C).



FIGURE 5.4 – Lecteur de carte à puce **Omnikey 3121**.

L'ordinateur sur lequel sera connecté ce lecteur a un OS Linux sur lequel le paquet **pcsc-tools** a été installé. Il ne faut pas oublier de démarrer le service **pcscd.socket**. Les commandes suivantes sont donc à taper dans un terminal :

```

1 sudo apt-get install libpcsclite-dev pcscd pcsc-tools
   ↳ libusb-dev
2 sudo systemctl start pcscd.socket
3 sudo systemctl enable pcscd.socket

```

Le paquet **pcsc-tools** contient les utilitaires suivants :

- **ATR_analysis** : permet d'analyser un **ATR** et de déterminer le type de carte associé à l'aide de la liste présente dans le fichier **smartcard_list.txt**.
- **scriptor** : permet d'envoyer des commandes à la carte à l'aide d'un fichier **batch** ou depuis **stdin**.
- **gscriptor** : la même chose que **scriptor** mais en mode graphique.
- **pcsc_scan** : permet de scanner les cartes depuis le lecteur connecté à la machine.

5.2.1 Lecture du numéro IMSI d'une carte SIM

Armés des instructions connues présentées dans l'annexe C.2 et des codes de status de l'annexe C.1, il est possible de lire par exemple le numéro **IMSI** [1] (*International Mobile Subscriber Identity*) d'une carte **SIM**. Il s'agit d'un numéro unique qui permet d'identifier une carte sur le réseau de téléphonie mobile et n'est, *a priori*, pas connu de l'utilisateur.

Un adaptateur de carte **SIM** a été utilisé de manière à pouvoir insérer la **nano SIM** dans le lecteur de carte à puce comme cela peut se voir sur la figure 5.5. Il est possible d'utiliser l'utilitaire **pcsc_scan** pour valider que la carte **SIM** est bien reconnue. La figure 5.6 permet de constater la présence d'une carte **Lyca-mobile Prepaid** basée sur la valeur de l'**ATR**.



FIGURE 5.5 – Carte **nano SIM** dans son adaptateur.

Le numéro **IMSI** est situé dans le fichier élémentaire **EF=6F07** qui est lui-même situé dans le **DF=7F20** (aussi appelé **GSM**). Ce **DF** est accessible depuis le **MF**. Pour accéder au numéro **IMSI**, il faut alors sélectionner le **MF**, puis le **DF** et enfin le **EF**. Suite à cela, il faudra demander à la carte de lire ce **EF**. Les commandes à exécuter dans l'utilitaire **scriptor** (ou **gscriptor**) sont donc (dans

```

Reader 0: HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21 Smart Card Reader] 00 00
Event number: 16
Card state: Card inserted, Shared Mode,
ATR: 3B 9F 95 80 1F C7 80 31 E0 73 FE 21 13 67 22 28 00 40 01 00 01 91

ATR: 3B 9F 95 80 1F C7 80 31 E0 73 FE 21 13 67 22 28 00 40 01 00 01 91
+ TS = 3B → Direct Convention
+ T0 = 9F, Y(1): 1001, K: 15 (historical bytes)
TA(1) = 95 → Fi=512, Di=16, 32 cycles/ETU
    125000 bits/s at 4 MHz, fMax for Fi = 5 MHz ⇒ 156250 bits/s
TD(1) = 80 → Y(i+1) = 1000, Protocol T = 0

TD(2) = 1F → Y(i+1) = 0001, Protocol T = 15 - Global interface bytes following

TA(3) = C7 → Clock stop: no preference - Class accepted by the card: (3G) A 5V B 3V C 1.8V
+ Historical bytes: 80 31 E0 73 FE 21 13 67 22 28 00 40 01 00 01
Category indicator byte: 80 (compact TLV data object)
Tag: 3, len: 1 (card service data byte)
Card service data byte: E0
- Application selection: by full DF name
- Application selection: by partial DF name
- BER-TLV data objects available in EF.DIR
- EF.DIR and EF.ATR access services: by GET RECORD(s) command
- Card with MF

Tag: 7, len: 3 (card capabilities)
Selection methods: FE
- DF selection by full DF name
- DF selection by partial DF name
- DF selection by path
- DF selection by file identifier
- Implicit DF selection
- Short EF identifier supported
- Record number supported
Data coding byte: 21
- Behaviour of write functions: proprietary
- Value 'FF' for the first byte of BER-TLV tag fields: invalid
- Data unit in quartets: 2
Command chaining, length fields and logical channels: 13
- Logical channel number assignment: by the card
- Maximum number of logical channels: 4
Tag: 6, len: 7 (pre-issuing data)
Data: 22 28 00 40 01 00 01
+ TCK = 91 (correct checksum)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B 9F 95 80 1F C7 80 31 E0 73 FE 21 13 67 22 28 00 40 01 00 01 91
    Lycamobile Prepaid SIM-Card (Telecommunication)
    http://lycamobile.at

```

FIGURE 5.6 – Détection d'une carte à puce **Lyca mobile** grâce à la valeur de l'ATR.

l'ordre) :

```

1 > A0 A4 00 00 02 3F 00
2 < 9F 22
3
4 > A0 A4 00 00 02 7F 20
5 < 9F 22

```

La première ligne signifie la demande de sélectionner le **MF=3F00**. La classe **0xA0** est celle nécessaire pour les cartes **SIM**. L'instruction **0xA4** est celle qui permet la sélection d'un élément. Cette instruction ne prend aucun paramètre (**P1=P2=00**). La longueur (**P3**) vaut **0x02** car les *bytes* optionnels qui correspondent au **MF** ont une longueur de deux *bytes*. Suite à l'exécution de cette commande, la carte répond par **9F 22**, ce qui signifie **OK, la longueur de la réponse à lire est de 0x22 bytes**. La ligne 4 permet ensuite de sélectionner le **DF=7F20** et le code retour indique que la réponse fait également **0x22 bytes**. Pour la lire, il faut alors utiliser l'instruction **0xC0** comme suit :

```

1 > A0 C0 00 00 22
2 < 00 00 33 BC 7F 20 02 00 00 00 00 00 15 B1 00 2F 05 00 83
→ 8A 83 8A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 90 00

```

On peut constater que la réponse fait bien **0x22 bytes** (34) auquel il faut ajouter le code retour (à la fin). Ce code de status vaut ici **90 00** ce qui signifie que tout c'est bien déroulé. Il est ensuite nécessaire de sélectionner le **EF=6F07** (qui correspond au numéro **IMSI**) de la manière suivante :

```

1 > A0 A4 00 00 02 6F 07
2 < 9F 0F
3
4 > A0 C0 00 00 0F
5 < 00 00 00 09 6F 07 04 40 14 F0 14 01 02 00 00 90 00

```

La réponse ci-dessus n'est pas documentée. On peut cependant en déduire certaines choses. **09** est la longueur en *bytes* du fichier à lire, **6F 07** est le fichier en question. **04** est le type de fichier (**MF=01**, **DF=02** et **04=EF**). D'après la documentation du système de télécommunication mobile [1], on valide que la longueur du fichier **IMSI** fait bien neuf *bytes*. Il est alors possible de le lire avec l'instruction **0xB0** comme suit :

```

1 > A0 B0 00 00 09
2 < 08 30 80 52 XX XX XX XX 95 90 00

```

Une partie du numéro **IMSI** a été masqué par des **XX**. Le premier *byte* de cette réponse (**0x08**) est la longueur du fichier, et le reste est le numéro **IMSI** avec le *byte* de poids faible au début et le *byte* de poids fort à la fin. Ainsi le numéro **IMSI** de cette carte **SIM** est : **95 XX XX XX XX 52 80 30**.

La liste des différents **DF** et **EF** connus et présents dans une carte **SIM** ainsi que le moyen de les consulter est présentée dans la documentation technique du système de télécommunication mobile [1].

5.2.2 Validation d'un code PIN

Il existe des cartes à puces vierge et bon marché permettant d'effectuer certains tests tout en respectant la loi. Celle présentée sur la figure 5.7 a une **EE-PROM** de 256 *bytes* dont 26 *bytes* sont réservés au fabricant, 2 *bytes* correspondent à une référence de l'émetteur, 4 *bytes* permettent d'identifier la carte, ce qui laisse 224 *bytes* de stockage. Il est possible de protéger cette zone mémoire par un code **PIN** à 3 *bytes*. Les 224 *bytes* sont suffisants pour stocker un montant, un code d'authentification, etc.



FIGURE 5.7 – Carte à puce vierge supportant la norme ISO 7816.

L'utilitaire **pcsc_scan** retrouve facilement le type de carte à l'aide de l'ATR comme cela peut se voir sur la figure 5.8.

```
Reader 0: HID Global OMNIKEY 3x21 Smart Card Reader [OMNIKEY 3x21 Smart Card Reader] 00 00
Event number: 18
Card state: Card inserted,
ATR: 3B 04 A2 13 10 91

ATR: 3B 04 A2 13 10 91
+ TS = 3B → Direct Convention
+ T0 = 04, Y(1): 0000, K: 4 (historical bytes)
+ Historical bytes: A2 13 10 91
  Category indicator byte: A2 (proprietary format)

Possibly identified card (using /usr/share/pcsc/smartcard_list.txt):
3B 04 A2 13 10 91
    PM2P Chipkarte SLE 4442, Code FFFFFF
```

FIGURE 5.8 – Détection d'une carte à puce **SLE4442** grâce à la valeur de l'ATR.

Cette carte est vierge et il est alors possible de la vérifier en lisant le contenu avec l'instruction **INS=B0** comme suit :

La zone destinée au stockage-utilisateur est vide (rempli de **0xFF**). Il est maintenant possible d'y écrire du contenu avec l'instruction **INS=D6**. Il faut auparavant lui spécifier le code **PIN** par défaut (**0xFF 0xFF 0xFF**) avec l'instruction **INS=20**. Il est ainsi possible d'y écrire, par exemple, 13 bytes (**P3=0D**) à l'adresse **P2=42** comme suit :

```

1 > FF 20 00 00 03 FF FF FF
2 < 90 00
3
4 > FF D6 00 42 0D 53 48 45 4C 4C 43 48 4F 43 4F 4C 41 54
5 < 90 00
6
7 > FF B0 00 00 FF
8 < A2 13 10 91 FF FF 81 15 FF FF
    ↳ FF FF D2 76 00 00 04 00 FF FF
    ↳ FF FF
    ↳ FF 53 48 45 4C 4C 43 48
    ↳ 4F 43 4F 4C 41 54 FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF
    ↳ FF FF FF 90 00

```

La ligne 7 permet de lire le contenu de la carte et de vérifier que les 13 bytes ont bien été inscrits dans la mémoire.

5.3 Attaques

Après avoir appréhendé le fonctionnement des cartes à puce fonctionnant sous la norme **ISO 7816**, différentes attaques sont susceptibles d'être menées. Il est par exemple possible de *bruteforcer* les différents **DF** et **EF** mais aussi les différents **AIDs** et **INS** en se basant sur la valeur de code retour pour déterminer si l'élément existe ou non. Il est également possible d'effectuer des attaques de *fuzzing* de manière à compromettre le flot d'exécution d'un **AID** ciblé. Cependant, ces attaques relèvent davantage du *software* que du *hardware*, et ne seront donc pas présentées dans cet ouvrage.

Au niveau du *hardware*, il apparaît intéressant de comprendre à quel signal correspond chaque *pads* de la carte. La figure 5.9 présente les différents signaux associés aux *pad*. La distance et la position de ceux-ci sont définies par la norme **ISO 7816**. Connaissant l'utilité des différents *pads*, il apparaît que certains auront

une importance plus grande que d'autres dans les attaques qui seront menées.

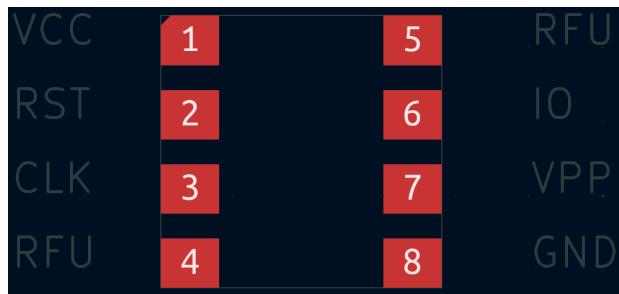


FIGURE 5.9 – Présentation des signaux associés à chaque *pad* d'une carte à puce.

Ainsi, le signal d'horloge (**CLK**), l'alimentation (**VCC**) et les signaux d'entrées et de sorties (**IO**) permettront d'attaquer et de vérifier si l'attaque a réussi ou non. Il est ainsi possible d'agir sur l'**IO** pour les attaques *software* énoncées plus haut, mais il sera aussi utilisé dans les attaques *hardware* comme moyen de vérification. Les *pads* **CLK** et **VCC** seront utilisés pour effectuer des attaques de *glitch* sur le signal d'horloge et sur l'alimentation respectivement.

Il est également possible d'effectuer des attaques plus complexes en enlevant les *pads* et en agissant directement sur la puce qui se trouve dessous. Des attaques **LASER** (*Light Amplification by Stimulated Emission of Radiation*) ou **EM** (*Electro Magnetic*) sont alors possibles. La complexité et le coût de ces attaques étant bien trop élevées, elles ne seront pas abordées dans cet ouvrage.

5.3.1 Sniffing de l'IO

Pour intercepter les données circulant sur une ligne de la carte, il faut pouvoir y avoir accès. La carte étant insérée dans le lecteur, les connecteurs ne sont donc plus accessibles. Il est possible de récupérer le signal en soudant des fils sur les *pads* comme sur la figure 5.10. Les fils doivent être le plus fins possible. Ceux qui ont été choisis ont un diamètre de 0,1 mm. S'ils sont trop épais la carte risque de ne plus rentrer dans le lecteur. Il est également important que les points de soudure ne soient pas trop épais, pour la même raison que précédemment. Des fils de 0,1 mm sont susceptible de s'accrocher à l'intérieur du lecteur et donc de se casser. Il est donc important de les scotcher de manière à ce qu'ils soient bien maintenus.

Cette approche, bien qu'efficace, à l'inconvénient de devoir souder sur la puce. Il est alors préférable de développer une interface sur laquelle on fera ressortir les lignes d'intérêts. La figure 5.11 présente un prototype d'une telle interface. Sur la gauche se trouve un connecteur permettant de connecter la carte. Sur la droite se trouve les connecteurs qui iront dans le lecteur et, au milieu, deux rangées de *pins* permettant d'intercepter le signal.

Cette carte permet alors de surveiller l'ensemble des signaux transitant entre la carte et le lecteur. Elle peut également être utilisée afin d'insérer un *glitch* sur le signal d'horloge (voir section 5.3.2).

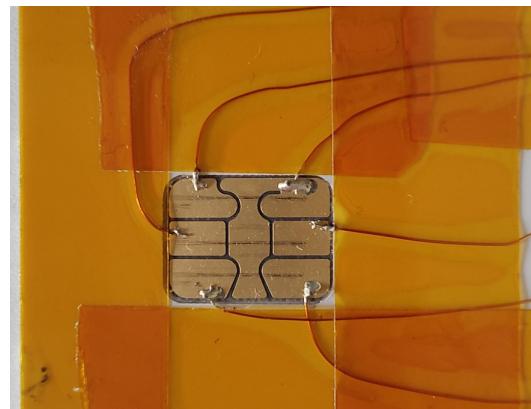


FIGURE 5.10 – Présentation des fils soudés sur la carte à puce.

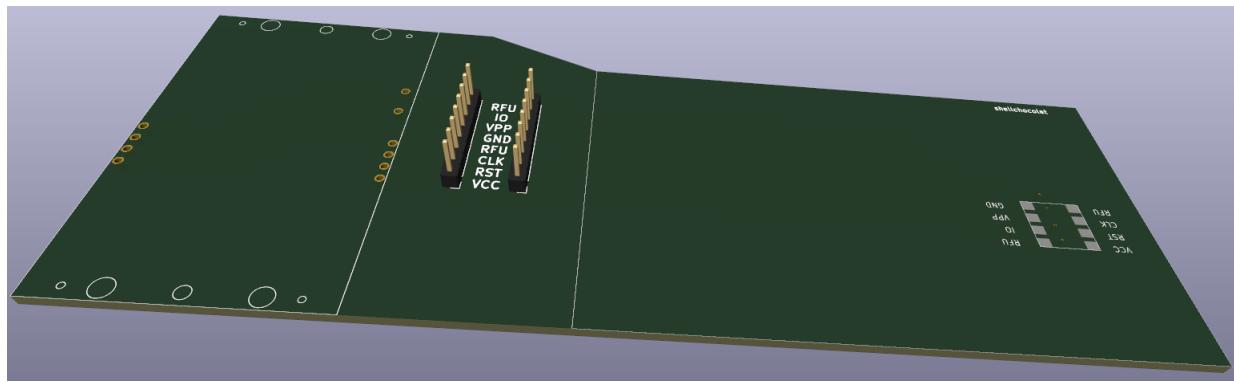


FIGURE 5.11 – Présentation d'une interface permettant d'intercepter les signaux d'une carte à puce.

5.3.2 Injection de glitch sur CLK

Chapitre 6

Analyse d'un router

6.1 Analyse statique du PCB

6.2 Analyse dynamique du PCB

6.3 Attaques

Chapitre 7

Analyse d'une camera wifi

Une caméra WiFi, également appelée caméra IP (*Internet Protocol*), est une caméra de surveillance qui se connecte à un réseau local ou à Internet via une connexion sans fil WiFi. Contrairement aux caméras de sécurité traditionnelles qui nécessitent souvent des câbles pour la transmission des données, les caméras WiFi sont plus flexibles car elles peuvent être installées à des endroits variés sans avoir à se soucier de la disponibilité de prises de courant ou de câbles réseau.

Ces caméras offrent plusieurs fonctionnalités :

- **Surveillance à distance** : Elles permettent aux utilisateurs de surveiller en temps réel ce qui se passe dans une zone spécifique, en utilisant un smartphone, une tablette ou un ordinateur connecté à Internet.
- **Alertes et notifications** : Elles sont équipées de capteurs de mouvement et/ou de son, ce qui leur permet de détecter des événements inhabituels. Lorsqu'un mouvement ou un son est détecté, la caméra peut envoyer des notifications à l'utilisateur via une application ou par e-mail.
- **Stockage des enregistrements** : Certaines caméras WiFi permettent d'enregistrer les vidéos capturées, soit localement sur une carte mémoire intégrée, soit dans le *cloud*. Cela permet de revoir les enregistrements ultérieurement en cas de besoin.
- **Audio bidirectionnel** : De nombreuses caméras WiFi sont équipées de microphones et de haut-parleurs, ce qui permet aux utilisateurs de communiquer à distance avec les personnes présentes dans la zone surveillée. Cela peut être utilisé pour des conversations bidirectionnelles ou pour dissuader les intrus.
- **Vision nocturne** : Beaucoup de ces caméras sont équipées de capteurs infrarouges qui permettent de voir dans l'obscurité, offrant ainsi une surveillance continue même la nuit.
- **Angle de vue variable** : Elles sont disponibles avec différents angles de vue, allant des angles étroits pour surveiller des zones spécifiques aux angles plus larges pour surveiller des pièces entières.
- **Intégration avec d'autres systèmes** : Certaines caméras WiFi peuvent être intégrées à des systèmes de domotique plus larges, ce qui permet de créer des scénarios d'automatisation en fonction de la détection de mouvement ou d'autres événements.

Les caméras WiFi offrent donc une solution de surveillance flexible et pratique,

permettant aux utilisateurs de garder un œil sur leurs biens ou leurs espaces à distance, que ce soit pour des raisons de sécurité, de surveillance de bébé, ou pour simplement avoir un aperçu de ce qui se passe chez eux en leur absence. La caméra analysée est présentée sur la figure 7.1. Il est possible d'orienter la caméra de gauche à droite et de haut en bas nécessitant deux moteurs pas-à-pas. Elle est alimenté par un chargeur **USB** en 5 V. Il est également possible de la contrôler via le WiFi mais aussi via le câble Ethernet dont on aperçoit le port. De l'autre côté de la caméra se trouve un emplacement pour une carte micro-SD laissant la possibilité d'effectuer des enregistrements.



FIGURE 7.1 – Visuel extérieur de la caméra WiFi analysée.

Lorsque l'on démonte la caméra, plusieurs éléments sont visibles. LA figure 7.2 présente les éléments contenu dans la base de la caméra. On y distingue un étage d'alimentation, un transformateur **LAN** (pour l'Ethernet), un moteur pas-à-pas (en haut) permettant de tourner la caméra de droite à gauche et un haut parleur (en haut à droite).

Dans la tête de la caméra se trouve le **PCB** principal (voir figure 7.3a pour la face avant et figure 7.3b pour la face arrière). Sur la face avant se trouve l'emplacement pour la carte micro-SD (à droite), en haut se trouve le second moteur pas-à-pas permettant l'orientation verticale de la caméra. On y trouve également divers connecteurs permettant d'alimenter les différents composant, mais aussi d'en récupérer les signaux utiles au fonctionnement de la caméra. La face arrière contient le capteur **CCD** (*Charge Coupled Device*) (au milieu), ainsi que l'optique nécessaire. On y trouve aussi une *flash* (en bas à gauche) dans un boîtier **SOP-8**, deux circuits intégrés inconnus pour le moment (en haut à gauche et en bas à droite), une pile servant probablement au maintien de l'horloge interne et/ou des paramètres de configuration (en cas de perte temporaire d'alimentation), et une antenne WiFi (en haut à droite).

L'analyse sera axée sur le **PCB** principal laissant de côté l'étage d'alimentation, le haut parleur et les moteurs pas-à-pas.



FIGURE 7.2 – L'étage d'alimentation en base, un moteur pas-à-pas en haut et sur la droite un haut parleur.

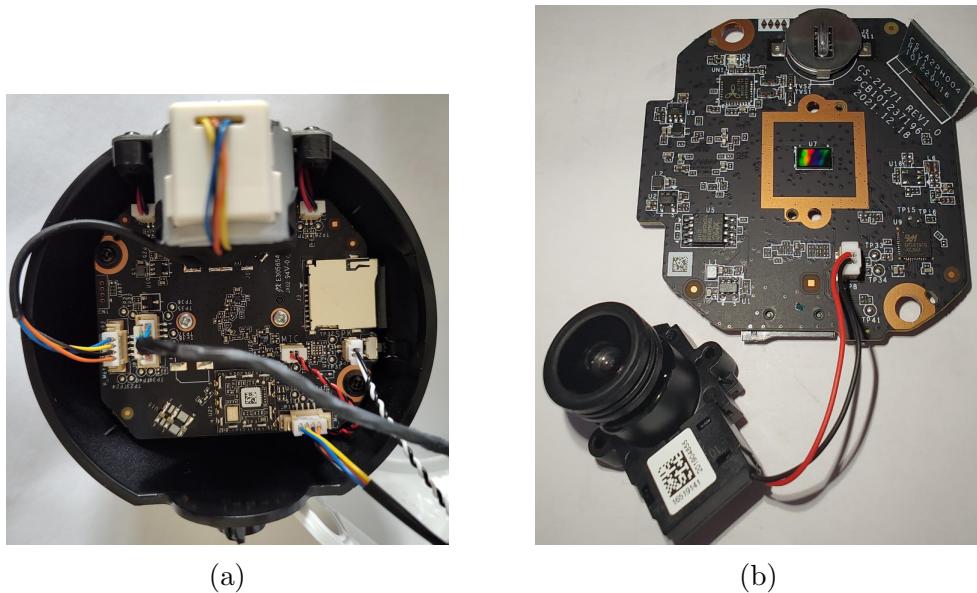


FIGURE 7.3 – Sur la figure 7.3a, la face avant du **PCB** de la caméra. Sur la figure 7.3b, la face arrière.

7.1 Analyse statique du PCB

Les composants passifs sont extrêmement petits sur ce **PCB**. A titre d'exemple les résistances sont dans des boîtiers **01005** ($0,4 \text{ mm} \times 0,2 \text{ mm}$). Il est donc très difficile de faire des mesures de valeurs de ces composants. Même les tests de continuité ne sont pas évident avec un multimètre classique. Ces sections aura pour objectif de déterminer les différents circuit intégré afin d'en connaître les fonctionnalités.

7.1.1 La face arrière

Sur la face arrière présentée en gros plan sur la figure 7.4, on peut voir que chaque composant est labellisé, ce qui nous permettra d'y faire référence dans cette étude.

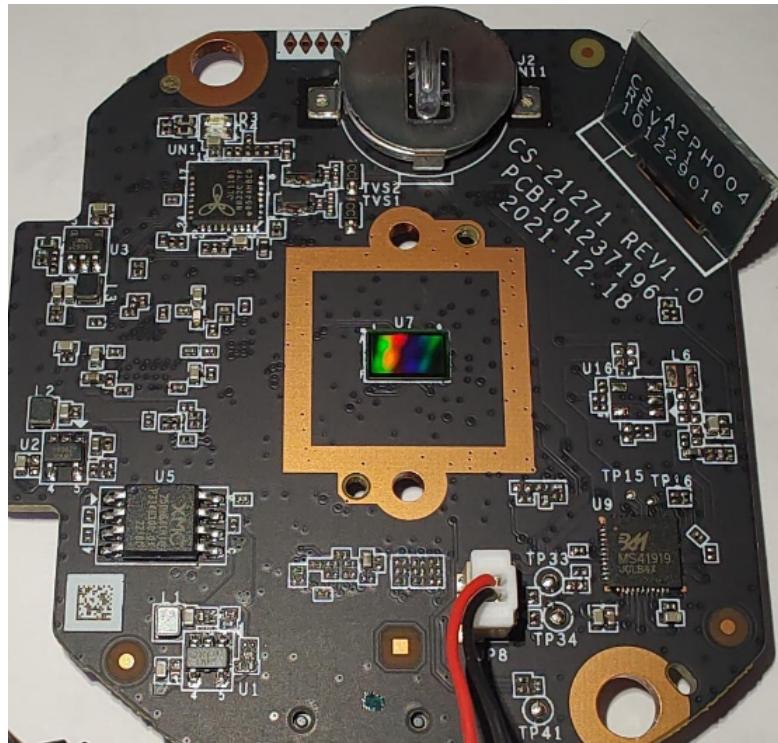


FIGURE 7.4 – Gros plan sur la face arrière du **PCB** principal de la caméra.

La flash

Le composant **U5** est une **flash NOR SPI** (en bas à gauche, voir aussi figure 7.5) de chez **XMC** dont la référence est **25QH64CH10**. Elle est dans un boîtier très classique qui est le **SOP-8**. Ce type de composant est généralement utilisé pour stocker un *firmware*. Il faudra le vérifier en extrayant son contenu.

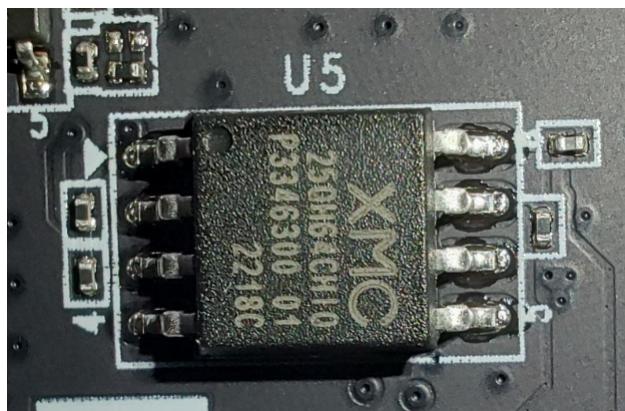


FIGURE 7.5 – Visualisation de la **flash** de la caméra.

Les convertisseurs DC-DC

Les composants **U1** (en bas à gauche), **U2** (à gauche de la *flash*) et **U3** (en haut à gauche) sont identiques. Ce sont des convertisseurs **DC-DC** dans des boîtiers **SOT-23-5**. Le convertisseur **DC-DC** est aussi appelé (en anglais) *Synchronous Step-Down Convertor* ou encore convertisseur **Buck** synchrone. Il est utilisé pour abaisser la tension continue (**DC**) d'une source d'alimentation à une tension inférieure et constante, en fournissant un courant plus élevé à la charge tout en maintenant une efficacité élevée.

La principale caractéristique qui distingue un convertisseur **Buck** synchrone d'un convertisseur **Buck** non synchrone (à diode) est l'utilisation de transistors (généralement des **MOSFET**) pour la commutation des circuits au lieu de diodes. Les transistors permettent un contrôle plus précis du processus de conversion de tension.

Ci-dessous sont présentées quelques caractéristiques et avantages clefs d'un convertisseur **Buck** synchrone :

- **Efficacité améliorée** : Étant donné que les transistors sont utilisés à la fois pour l'interrupteur principal (le transistor haut) et pour le diode de récupération (le transistor bas), les pertes de commutation sont réduites, ce qui se traduit par une efficacité plus élevée par rapport aux convertisseurs **Buck** non synchrones.
- **Réduction des pertes de conduction** : Les diodes de récupération (dans les convertisseurs **Buck** non synchrones) ont des pertes de conduction, alors que les transistors utilisés dans les convertisseurs **Buck** synchrones réduisent ces pertes, améliorant ainsi l'efficacité globale du convertisseur.
- **Contrôle de la commutation** : Les transistors permettent un meilleur contrôle de la fréquence de commutation et de la durée des phases de marche et d'arrêt, ce qui facilite le réglage fin de la tension de sortie.
- **Réduction des courants parasites** : Les diodes de récupération dans les convertisseurs **Buck** non synchrones peuvent générer des courants parasites indésirables, tandis que les transistors utilisés dans les convertisseurs **Buck** synchrones réduisent ce problème.
- **Meilleure réponse aux variations de charge** : Les convertisseurs **Buck** synchrones ont généralement une meilleure réponse aux variations de charge grâce à leur meilleure efficacité et à leur contrôle de commutation plus précis.

Les convertisseurs **Buck** synchrones sont couramment utilisés dans une variété d'applications, notamment les alimentations électriques de carte mère d'ordinateur, les systèmes d'alimentation pour appareils portables, les applications industrielles et de télécommunications, ainsi que dans d'autres domaines où la conversion efficace de tension est nécessaire. Dans le cas présent, ces convertisseurs sont utilisés comme étage d'alimentation intermédiaire permettant d'abaisser la tension de 5 V aux tensions requises pour les différents circuits intégrés.

Afin de fonctionner correctement, ce composant nécessite de lui associer des composants passifs (voir figure 7.6a). Comme on peut le voir, on y distingue une bobine (**L1**), des condensateurs et des résistances. L'ensemble ayant pour objectif

d'abaisser la tension au seuil souhaité. Un montage typique est présenté sur la figure 7.6b.

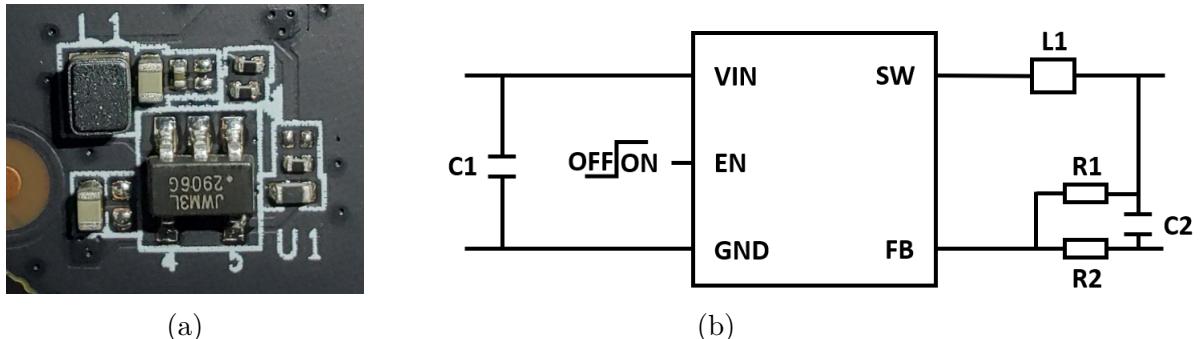


FIGURE 7.6 – Sur la figure 7.6a, le montage réducteur de tension continue utilisé sur la caméra. Sur la figure 7.6b, un montage minimal présentant les composants essentiels pour ce type de réducteur de tension.

Le pilote de lentille optique

Le composant **U9** (en bas à droite) est un pilote permettant d'orienter la lentille (à l'aide d'un moteur pas-à-pas) dans la partie optique de la caméra. Il s'agit d'un **MS41919** de la société **Rui Meng** dans un boîtier **QFN-44**. La seule documentation disponible est en chinois et très peu d'informations ont pu en être tirés. Ce composant permet de contrôler la position de la lentille 256 "micro-pas" à l'aide de quatre lignes de données. Ce composant est visible sur la figure 7.7a et son schéma typique d'utilisation est présenté sur la figure 7.7b.

Le transceiver Ethernet

Le composant **UN1** (en haut à gauche) est un émetteur-récepteur Ethernet aussi appelé transceiver Ethernet (*TRANSmitter / reCEIVER*). En d'autres termes, c'est un composant qui permet à un équipement informatique de se connecter à un réseau Ethernet en envoyant et en recevant des données via des câbles ou d'autres types de médias de transmission.

Les principales fonctions d'un transceiver Ethernet sont les suivantes :

- **Conversion de signaux électriques en signaux optiques (si nécessaire) :** Certains transceivers Ethernet prennent en charge la conversion des signaux électriques en signaux optiques, notamment dans les réseaux fibre optique. Ces transceivers sont appelés des transceivers optiques.
- **Modulation et démodulation des signaux :** Les transceivers modulent les données provenant d'un équipement réseau en signaux appropriés pour la transmission sur le support de câble ou optique, puis démodulent les signaux reçus en données utilisables par l'équipement.
- **Réglage de la vitesse et du mode de fonctionnement :** Les transceivers peuvent généralement être configurés pour prendre en charge différentes vitesses de transmission (comme 10/100/1000 Mbps) et différents modes de fonctionnement (duplex intégral ou semi-duplex).

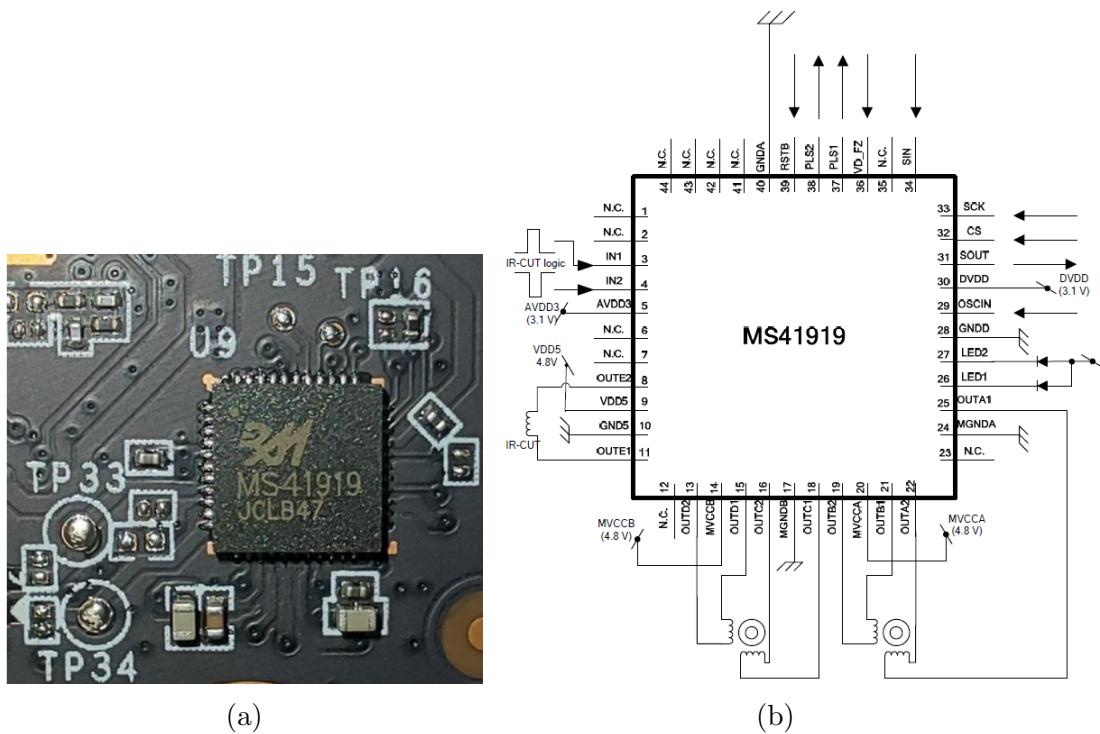


FIGURE 7.7 – Sur la figure 7.7a, le pilote de lentille ainsi que ces composants passifs associés. Sur la figure 7.7b, un montage typique d'utilisation de ce composant.

- **Détection de collision (dans les réseaux à accès multiple)** : Dans les anciens réseaux Ethernet utilisant la méthode CSMA/CD (Carrier Sense Multiple Access with Collision Detection), les transceivers détectent les collisions sur le réseau et prennent les mesures nécessaires pour les gérer.
 - **Interface avec l'équipement réseau** : Les transceivers sont équipés de ports de connexion (comme des connecteurs **RJ45** pour les câbles cuivre) qui permettent la connexion physique à l'équipement réseau.

Il assure la transmission et la réception de données tout en gérant des aspects tels que la vitesse, le mode de fonctionnement et la gestion des collisions (le cas échéant). Le transceiver utilisé ici (voir figure 7.8) est un **JL1101** de chez **JL-Semi** dans un boîtier **QFN-32**.

7.1.2 La face avant

La face avant avait un moteur qui masquait une partie du **PCB**. La figure 7.9 présente cette même face sans le moteur de manière à voir apparaître un circuit intégré propriétaire labellisé **U4** (au milieu à droite) dans un boîtier **QFN-88**. Aucune documentation n'a donc pu être trouvée à son sujet. Il s'agit probablement du micro-contrôleur principal. Sur la face arrière, juste derrière ce circuit intégré se trouve la *flash* laissant alors penser qu'il la lit pour récupérer le *firmware*.

Il n'est pas nécessaire de représenter l'emplacement de la carte micro-SD en bas du PCB.

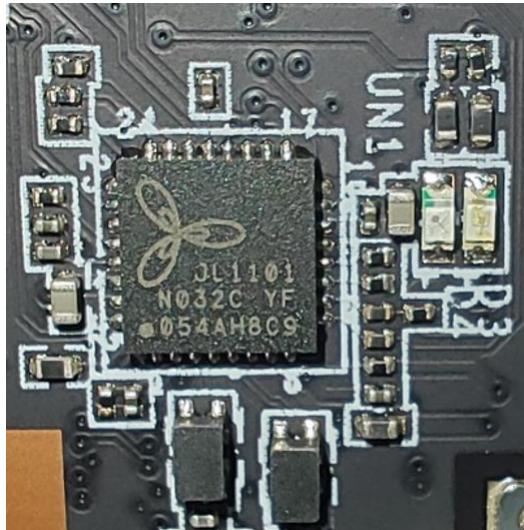


FIGURE 7.8 – Visualisation du transceiver Ethernet **JL1101** utilisé dans la caméra.

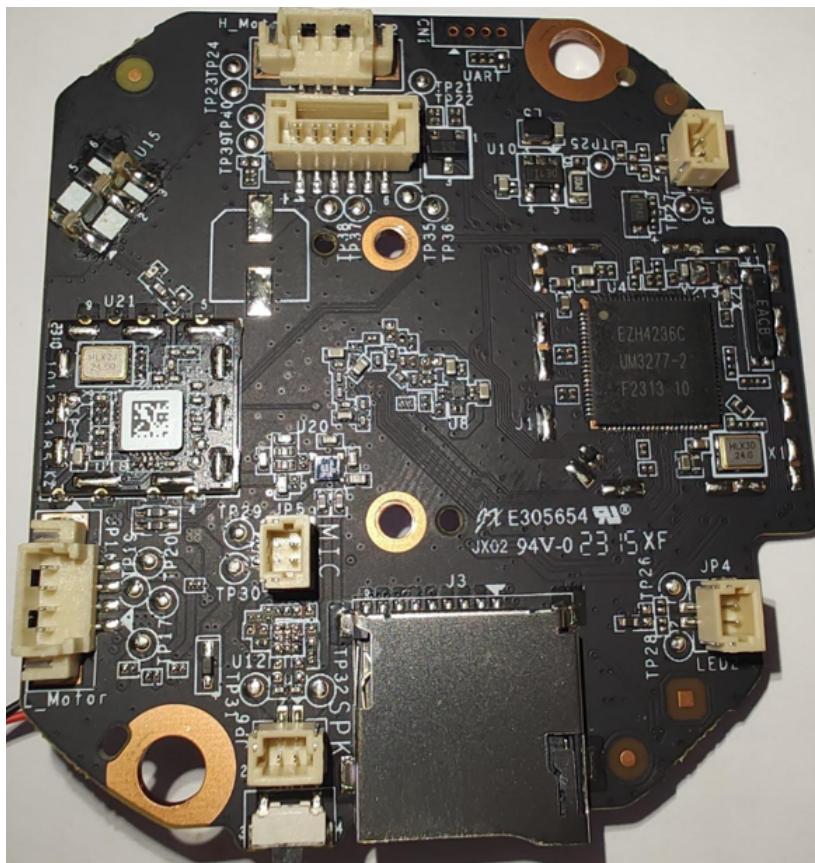


FIGURE 7.9 – Face avant du **PCB** sans le moteur pas-à-pas permettant la rotation sur l'axe vertical de la caméra.

Le port UART

Le port **UART** est bien visible (en haut, et sur la figure 7.10). On y voit bien les résistances associées à la transmission (**TX**) et la réception (**RX**) reliées aux deux trous du milieu (une troisième résistance est cependant utilisée, à quel

but ?). Le trou de gauche est une masse. On peut le deviner car les masses ont généralement des traces plus épaisses, et on voit parfaitement les traces entourant le trou. Le trou de droite est la tension. On peut le déterminer car la trace est plus épaisse que les traces des données (**RX** et **TX**). Évidemment une analyse dynamique permettra de confirmer cela.

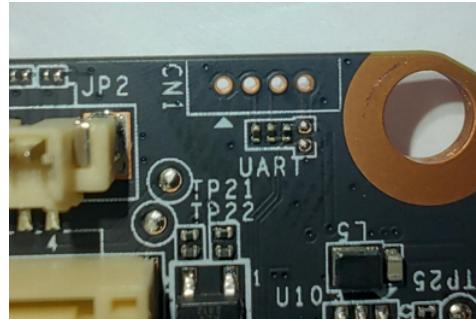


FIGURE 7.10 – Présentation du port **UART** sur la face avant de la caméra.

Le module WiFi

Le composant de gauche est un **PCB** en lui-même qui est soudé sur le **PCB** principal. Il s'agit d'un module WiFi **ATBM6032** de chez **AltoBeam**. Il est visible sur la figurez 7.11. Il fonctionne en 3,3 V et supporte plusieurs OS tels que Linux/Android/RTOS. Il est situé juste à "côté" (sur la face arrière) de l'antenne qui lui est associée.

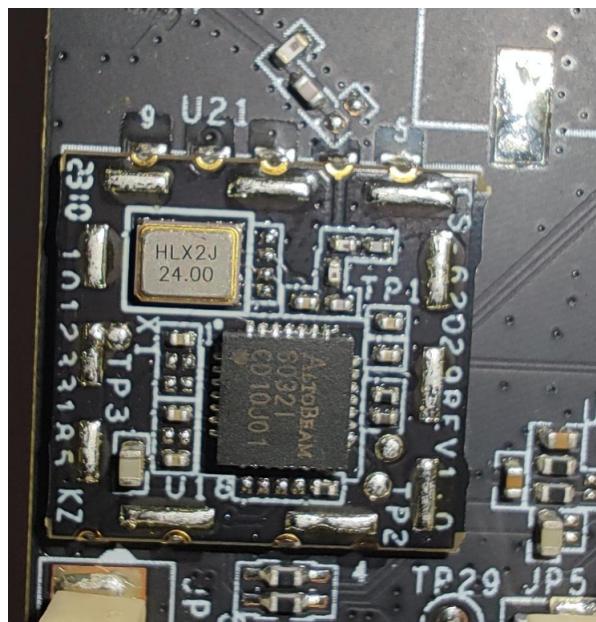


FIGURE 7.11 – Module WiFi de la caméra.

Les points de test

Il a put être remarqué plusieurs points de test labellisés **TP** (*Test Point*). Ces points de test sont situés au niveau de tous les connecteurs. A chaque broche

d'un connecteur est associé un point de test. Cela permet de vérifier et de valider les niveaux des signaux qui arrivent aux différents éléments. Aucun point de test en dehors de ceux associés aux connecteurs n'a été relevé indiquant qu'aucun dispositif de *debug* n'a été mis en place en dehors du port **UART** présenté précédemment.

7.2 Analyse dynamique du PCB

7.2.1 Vérification de la présence de l'UART

Bien que l'**UART** soit utilisé comme protocole de communication entre différents périphériques, le fait pour le concepteur de mettre à disposition un port dédié ne peut signifier qu'une chose : interface de *debugging*. Il est donc primordiale de déterminer si ce port est fonctionnel ou non. D'après l'analyse statique, on se doute qu'il l'est car des résistances sont présentes sur les lignes de données. Une mesure avec un voltmètre permet de constater que le trou de droite est bien relié à une tension de 3,3 V, et que le trou de gauche est relié à la masse. Pour déterminer la nature des deux autres trous, il faut souder des fils sur les trous puis les connecter à un analyseur de signaux comme cela est présenté sur la figure 7.12.

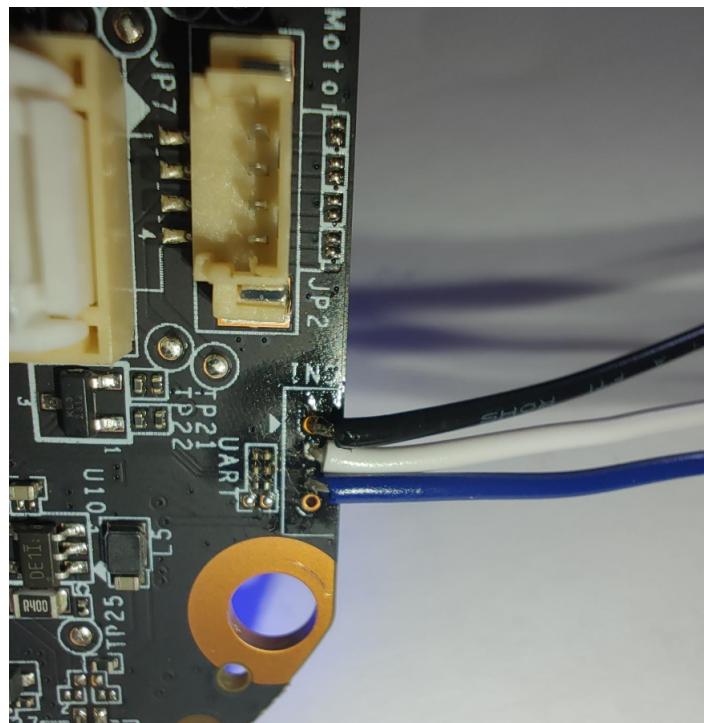


FIGURE 7.12 – Fils soudés aux présumés trous de transmission et de réception de l'**UART** (blanc et bleu respectivement), et à la masse en noir.

Pour interpréter les données présentées sur la figure 7.13, il faut connaître les paramètres utilisés par le micro-contrôleur pour communiquer en **UART**. D'après ce qui a été vu dans la section 1.2.1, il faut estimer les valeurs de la vitesse de transmission, le nombre de *bits* dans un *byte*, la parité et la longueur du *bit* de *stop*. À part la vitesse de transmission (*baudrate*), ces valeurs sont relativement standard et l'on peut estimer qu'ils valent 8 pour le nombre de *bits* dans un *byte*,

pas de bit de parité et 1 *bit* de *stop*. En ce qui concerne le *baudrate*, il est possible de tester les valeurs les plus usuelles ou bien le calculer pour être certains de ce que l'on observe.



FIGURE 7.13 – Une partie de la capture des données issues de la ligne **RX** du port **UART** de la caméra.

Estimation du baudrate

Reprendons la définition du *baudrate*. Le *baudrate* (taux de transmission en bauds) est la vitesse à laquelle les données sont transmises sur une ligne de communication série. Il est mesuré en bauds (symboles par seconde) et détermine la fréquence à laquelle les signaux électriques représentant les données sont transmis. Ainsi, la formule générale pour calculer le *baudrate* est :

$$\text{baudrate} = \frac{v}{N} \quad (7.1)$$

avec v la vitesse de transmission des *bits* en Nbits/s (nombre de *bits* par seconde), et N le nombre de *bits* dans un *byte*. Ainsi, en prenant un agrandissement de la figure 7.13 que l'on représente sur la figure 7.14, on peut déterminer que 17 *bytes* ont été transmis durant 146,9 μ s. On détermine que la vitesse de transmission des *bits* est $v = 17 \times 8 / (146,9 \times 10^{-6}) = 925799,9$ bits/s (car hypothèse très classique qu'un *byte* est codé sur 8 *bits*) et donc que le *baudrate* vaut 115725 s^{-1} (*baudrate* = $925799,9 / 8$). Il s'agit là d'une estimation sur un laps de temps très court. On peut cependant arrondir cette valeur à la valeur très courante de 1157200.



FIGURE 7.14 – Détermination du *baudrate* de l'UART.

Analyse des logs

En utilisant cette valeur pour le *baudrate* (configuration **UART** : 115200 8N1), il est alors possible de décoder le contenu du signal comme on peut le voir sur la figure 7.15 présentant une partie des *logs* transmis sur la ligne **RX**.

```

    Use nor flash.
ROM:   Init DDR..Training done.
R

U-Boot 2010.06-svn172490 (Sep 20 2022 - 15:20:42)

DRAM:  64 MiB
MMC:  FH MMC: 0
master [ctl : mem] = [0 : 0]
SF: Got idcode 20 40 17 20 40
product name:CS_XP1
Using SZ_8M TYPE_RT flash partition choice.
MMC FLASH INIT: No card on slot!
No mmc storage device found!
load_update_file fail
Net:   fh_gmac_initialize
FH EMAC
Hit Ctrl+u to stop autoboot:  2 \x08\x08\x08 1 \x08\x08\x08 0
load rt app

header_data.u32Magic is 0xa7b4c9f8
header_data.u32header_len is 0x10
header_data.u32RawDataLen is 0x96000
Done load!
Thread Operating System3.1.3 build Jan 16 2023 - 14:07:36
SDK V2.1.2-g100a56b
svn version is 183211
DSP_FASTBOOT1_StartVpu OK!
data read from 0xa =0x8, 0xb = 0x41
[SFUD] Find a XMC XM25QH64C flash chip. Size is 8388608 bytes.
[SFUD] fh_flash flash device is initialize success.
\x1B[32m[I/FAL] RT-Thread Flash Abstraction Layer(V0.4.0) initialize success.\x1B[0m
fl_load_disp_text code_index: 6
get_code_part_info part_idx: 6
load_code part_idx: 6
efuse_clk warning: div failed 7
ez_srand_init: pts_seed=0x34388b, srand_seed=0x8265c8d2
VBus loaded: 255 out blocks, 255 in blocks
##exe sd card mmcSD_detect:973 sd_hw_power off
##exe try detect SD card
\x1B[32m[I/DFS] Device File System initialized!
\x1B[0m
jffs2 System dfs_mount ok!
jffs2 System first initialized!
Mount res partition!
mtd7 dfs_mount ok!
mtd7 File System first initialized!
lwIP-2.0.2 initialized!
\x1B[32m[I/SAL_SKT] Socket Abstraction Layer initialize success.\x1B[0m
get_init_mac_addr: 78:c1:ae:e3:db:a8
ac_mclk warning: div failed 11
[17 11:31:31] [DEVLOG] [ERROR]--- main_init ycm ---
```

FIGURE 7.15 – Visualisation d'une partie des *logs* transmis sur la ligne RX de l'UART de la caméra.

Sur cette partie des *logs*, on constate que le *bootloader* (voir annexe B.1) utilisé est **U-Boot** [7] (voir annexe B.3) qui *opensource* et qui a été compilé le 20 septembre 2022. On constate également que 64 MB de **DRAM** sont utilisées. Comme il n'y a pas de composant associée spécifiquement à la **DRAM** sur ce **PCB**, on en conclut que cette **DRAM** est inclue dans le composant propriétaire. On peut également apercevoir que le système d'exploitation utilisé est **RT-Thread** [19] qui est également *opensource* et largement utilisé dans l'IoT. La version de l'**OS** est 3.1.3 compilé le 16 janvier 2023. On voit également que la *flash* **XM25QH64C** est initialisée et que le système de fichiers utilisé est **Jefferson** [13] qui est également *opensource* et largement utilisé dans les projets IoT.

Souvent une invite de commandes est disponible via l'UART lorsque l'**OS** et le système de fichiers sont chargés. Ce n'est pas le cas ici.

7.3 Attaques

7.3.1 Extraction du firmware

Il a été vu dans l'analyse statique qu'une **flash NOR SPI** était présente sur le **PCB**. Il a également été montré lors de l'analyse dynamique que le microcontrôleur chargeait le contenu de cette *flash* en mémoire qui doit alors contenir le *firmware* de la caméra. Le firmware d'une flash SPI a déjà été extrait dans la section 2 en la dessoudant. L'espace était en effet trop restreint pour pouvoir utiliser une pince **SOP-8**. Ici, la pince **SOP-8** est utilisée avec succès. La figure 7.16 présente le positionnement de la pince sur le composant. L'autre bout de la pince étant relié à un **Hydrabus**.

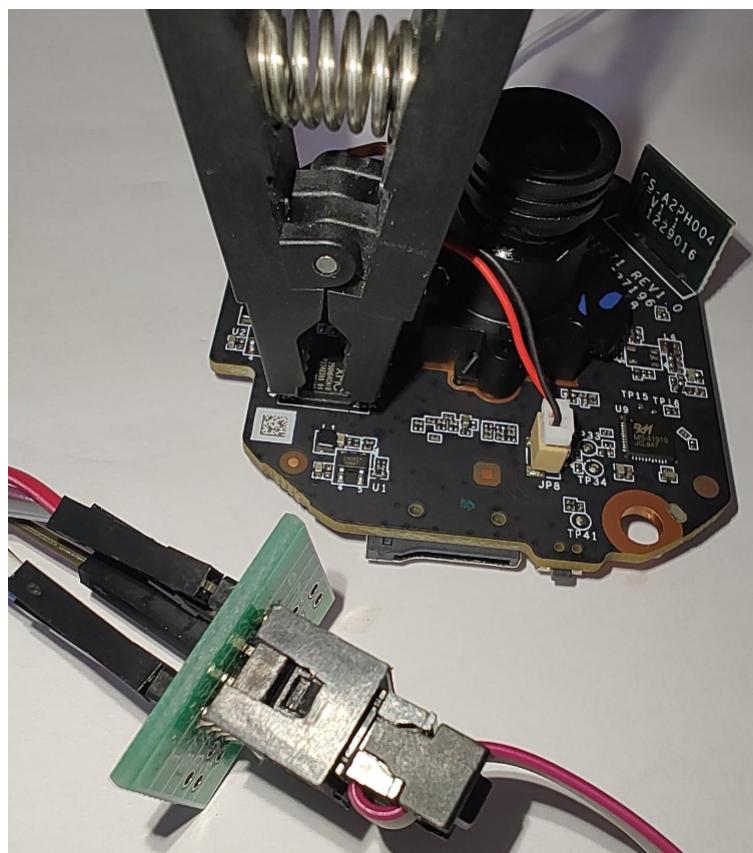


FIGURE 7.16 – Positionnement de la pince **SOP-8** sur la *flash* de la caméra.

Le firmware a été extrait trois fois pour s'assurer qu'il ne soit pas corrompu. Le script d'extraction est le même que celui présenté dans la section sus mentionnée. Les *hashes MD5* permettent d'assurer de l'intégrité des fichiers extraits. Comme on peut le voir sur la figure 7.17 le premier fichier n'est pas identique aux deux autres. On établit alors que le premier fichier est corrompu.

Il a été montré dans l'analyse dynamique que le système de fichiers utilisé est **Jefferson**. On peut vérifier cette information sur le *firmware* extrait en utilisant l'outil **binwalk** comme cela peut se voir sur la figure 7.18.

Une fois le système de fichiers extrait, il est alors possible de naviguer dedans puis de trouver des vulnérabilités software ou de configuration. Cela n'est cepen-

```
(shellchocolat)→ camera md5sum camera_flash_dump_1.bin
4afad88c33b0dee4de49b31022bde6e6  camera_flash_dump_1.bin
(shellchocolat)→ → camera md5sum camera_flash_dump_2.bin
ffd66e042e62b7b97367768f4aa6c6fc  camera_flash_dump_2.bin
(shellchocolat)→ → camera md5sum camera_flash_dump_3.bin
ffd66e042e62b7b97367768f4aa6c6fc  camera_flash_dump_3.bin
[...]
```

FIGURE 7.17 – Vérification de l'intégrité des fichiers extraits en calculant le *hash MD5* de ceux-ci.

(shellchocolat)→ → camera binwalk camera_flash_dump_2.bin		
DECIMAL	HEXADECIMAL	DESCRIPTION
88197	0x15885	Certificate in DER format (x509 v3), he
215780	0x34AE4	CRC32 polynomial table, little endian
461965	0x70C8D	Certificate in DER format (x509 v3), he
462089	0x70D09	Certificate in DER format (x509 v3), he
462849	0x71001	Certificate in DER format (x509 v3), he
683376	0xA6D70	CRC32 polynomial table, little endian
757776	0xB9010	LZMA compressed data, properties: 0x5D,
1572880	0x180010	LZO compressed data
1973573	0x1E1D45	Certificate in DER format (x509 v3), he
1973697	0x1E1DC1	Certificate in DER format (x509 v3), he
1974457	0x1E20B9	Certificate in DER format (x509 v3), he
2375008	0x243D60	LZO compressed data
2391224	0x247CB8	CRC32 polynomial table, little endian
2856195	0x2B9503	eCos RTOS string reference: "ecost"
2856210	0x2B9512	eCos RTOS string reference: "ecost"
2859399	0x2BA187	eCos RTOS string reference: "ecost_[cha
2859454	0x2BA1BE	eCos RTOS string reference: "ecost_[cha
2871312	0x2BD010	LZO compressed data
2888355	0x2C12A3	SHA256 hash constants, little endian
2895382	0x2C2E16	AES S-Box
2902806	0x2C4B16	AES Inverse S-Box
3016327	0x2E0687	mcrypt 2.5 encrypted data, algorithm: "
3035021	0x2E4F8D	AES S-Box
3079260	0x2EFC5C	DES PC1 table
3170110	0x305F3E	AES S-Box
3186418	0x309EF2	AES Inverse S-Box
4948983	0x4B83F7	mcrypt 2.5 encrypted data, algorithm: "
5697546	0x56F00A	Unix path: /home/voice/didi.aac
6459471	0x62904F	AES Inverse S-Box
6459727	0x62914F	AES S-Box
7012364	0x6B000C	JFFS2 filesystem, little endian
7045156	0x6B8024	JFFS2 filesystem, little endian
7045280	0x6B80A0	JFFS2 filesystem, little endian
7143424	0x6D0000	JFFS2 filesystem, little endian
7363640	0x705C38	Zlib compressed data, compressed

FIGURE 7.18 – Vérification du système de fichiers utilisé. Il s'agit bien d'un **Jefferson**.

dant plus du ressort de l'analyse offensive de **PCB**.

7.3.2 – Interception SPI

7.3.3 Attaque pin2pwn

Il a été montré dans lors de l'analyse dynamique qu'aucune invite de commandes n'était proposée par le système d'exploitation sur l'**UART**. Cependant, en reprenant la figure 7.15, on peut apercevoir qu'il est possible d'appuyer sur les touches **Ctrl+u** (code hexadécimal **0x15**) pour interrompre le démarrage automatique. L'utilisateur a 2 s pour faire cela. Il n'a pas été possible d'interrompre

le démarrage. En effet, la ligne **TX** permettant de transmettre des données et la ligne **RX** permettant d'en recevoir sont reliées comme sur la figure 7.19 via le port **UART**. On voit que la ligne **RX** arrive bien au micro-contrôleur, mais que la ligne **TX** est en parallèle avec la tension. Ainsi le signal sur la ligne **TX** est forcée à l'état haut (voir la section 1.1.5 pour la notion de collecteur ouvert), ce qui empêche d'envoyer des données sur cette ligne depuis la console **UART**. La solution consiste à enlever la résistance dénommée **R3**.

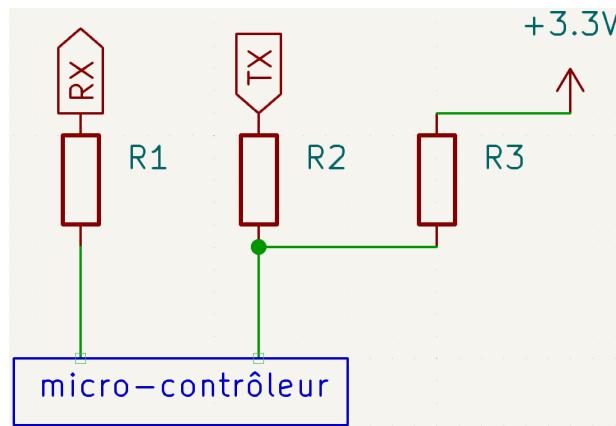


FIGURE 7.19 – Visualisation du schéma reliant les lignes **RX** et **TX** au micro-contrôleur.

Une fois la résistance enlevée (voir figure 7.20), il est alors possible d'accéder au *shell* de **U-Boot** en appuyant sur les touches **Ctrl+u**. Cependant, ce n'est pas cette technique que je souhaitais présenter. Il peut arriver que la configuration de **U-Boot** soit bien réalisée et qu'il ne soit pas possible d'accéder au *shell* car le *timer* est alors fixé à 0 (au lieu de 2 s dans le cas présent). La suite de cette étude fait l'hypothèse qu'il n'est pas possible d'appuyer sur des touches pour interrompre le démarrage.

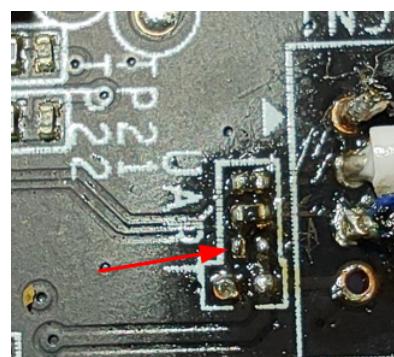


FIGURE 7.20 – Suppression de la résistance qui bloque la transmission de données.

La technique présentée ici s'appelle **pin2pwn**. Cette technique implique l'utilisation de signaux matériels, tels que la manipulation d'une broche de la mémoire *flash*, pour provoquer une erreur ou un comportement non prévu dans l'exécution de **U-Boot**, ce qui pourrait permettre d'obtenir un accès au *shell* de débogage ou à un mode de diagnostic.

De manière générale, lorsque **U-Boot** rencontre une erreur lors de linitialisation des divers périphériques, il propose un *shell* à lutilisateur via la console **UART**. Il est ainsi possible de modifier la configuration de **U-Boot** de manière à corriger lerreur. Ainsi, cette attaque est inhérente à **U-Boot**. Pour provoquer cette erreur, il faut perturber la lecture dun signal. En effet, lors de linitialisation des périphériques, **U-Boot** communique avec eux. S'il narrive pas à établir de communication, **U-Boot** lève une erreur et propose alors un *shell*.

La communication avec la *flash SPI* est assez facile à corrompre. En effet, la ligne **MOSI** sera interroger par le micro-contrôleur pour vérifier si la *flash* est bien présente. Si cette ligne est mise à 0 V alors le **U-Boot** ne comprendra pas. La figure 7.21 présente la mise à la masse de la patte **MISO** de la *flash*.

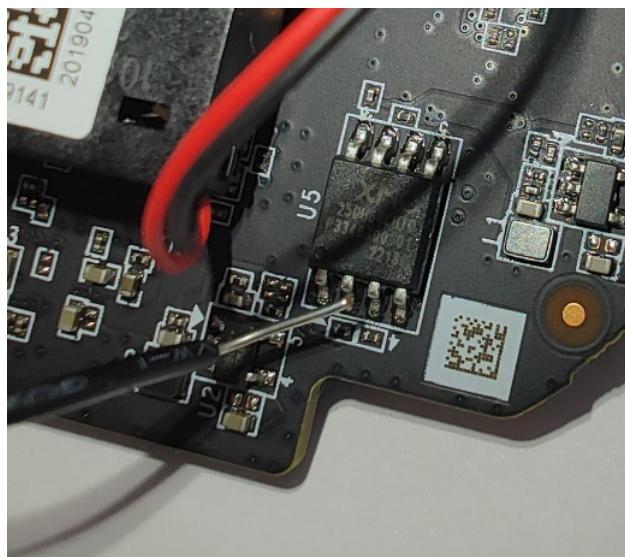


FIGURE 7.21 – Mise à la masse de la patte **MISO** de la *flash*.

Cette technique peut être vue comme une technique d'injection de faute, car l'idée est de provoquer une faute dans l'exécution du *bootloader*. Bien que facile à mettre en place, il faut néanmoins trouver le bon timing. En effet, mettre à la masse un peu trop tôt conduit à lerreur présentée sur la figure 7.22 et, trop tard, à lerreur présentée sur la figure 7.23. Le bon timing est lorsque **U-Boot** tente d'accéder à la *flash*. C'est un jeu d'essai-erreur.

```
[19590.08h.00m.47s.049] ROM:    Use nor flash.
ERR:    Flash is corrupted          [19590.08h.00m.56s.596]
[19590.08h.00m.56s.597]
```

FIGURE 7.22 – Erreur lorsque la patte **MISO** de la *flash* est mise à la masse trop tôt.

Une fois la vulnérabilité exploitée, on obtient la figure 7.24. On voit en effet lerreur en bas de la figure qui indique *can't get kernel image*, ce qui est normal vu qu'on a empêché la *flash* de communiquer. Le *shell* est alors disponible : **HKVS**.

A partir de là, il est possible d'afficher l'aide (**help**) mais aussi de lister la liste des variables d'environnement (**printenv**) comme cela est présenté sur la

```
[19590.08h.00m.18s.236] [<0xa08e8724>] 00000001 00000005 00000000 a07e76c
[19590.08h.00m.18s.242] [<0xa08e8704>] a0471e7c a046c878 a08e8708 a08e876
[19590.08h.00m.18s.251] [<0xa08e86e4>] a15a64a5 a097d124 a15a64a5 a097d12
[19590.08h.00m.18s.265] [<0xa08e86c4>] a046c7bc a08e1524 a08e879c a08e870
[19590.08h.00m.18s.265] [<0xa08e86a4>] a08e8a60 00000003 00000004 a097d12
[19590.08h.00m.18s.273] [<0xa08e8684>] a04728b8 a04ac210 a08e8688 a08e86b
[19590.08h.00m.18s.280] [<0xa08e8664>] 00000000 00000000 00000000 00000000
[19590.08h.00m.18s.298] [<0xa08e8644>] 00000000 00000000 00000000 00000000
[19590.08h.00m.18s.298] [<0xa08e8624>] 00000000 00000000 a0474f2c a047293
[19590.08h.00m.18s.305] [<0xa08e8604>] a097d124 a097d024 a15e9254 0000003
[19590.08h.00m.18s.312] [<0xa08e85e4>] 00000000 00000000 00000000 00000000
[19590.08h.00m.18s.328] [<0xa08e85c4>] 00000000 00004145 00000001 0000001
[19590.08h.00m.18s.328] [<0xa08e85a4>] 00000000 00000044 00000000 0000000
[19590.08h.00m.18s.335] [<0xa08e8584>] 00000000 00000000 00000000 a08e85a
[19590.08h.00m.18s.343] [<0xa08e8564>] a097d024 00000000 a002ad98 a002b20
[19590.08h.00m.18s.349] [<0xa08e8544>] a08db7e4 60000013 a07e9408 a07e6b8
[19590.08h.00m.18s.358] [<0xa08e8524>] a0004f64 a002addc a08e8538 a08e855
[19590.08h.00m.18s.365] [<0xa08e8504>] a001125c a0004fe4 a08e8508 a08e852
[19590.08h.00m.18s.374] [<0xa08e84e4>] a07a0ffc 0000014b a0027c90 a001130
[19590.08h.00m.18s.380] [<0xa08e84c4>] a0798428 a07a1008
[19590.08h.00m.18s.385] 1:[<0xa0474f20>]
[19590.08h.00m.18s.385] 2:[<0xa04728ac>]
[19590.08h.00m.18s.386] 3:[<0xa04ac128>]
[19590.08h.00m.18s.388] 4:[<0xa04715bc>]
[19590.08h.00m.18s.389] 5:[<0xa0471e70>]
[19590.08h.00m.18s.391] 6:[<0xa046c7bc>]
[19590.08h.00m.18s.393] 7:[<0xa045fd2c>]
[19590.08h.00m.18s.395] 8:[<0xa04616ac>]
[19590.08h.00m.18s.397] 9:[<0xa01d0840>]
[19590.08h.00m.18s.397] 10:[<0xa01d0dbc>]
[19590.08h.00m.18s.399] EntryPoint:[<0xa0017190>]
[19590.08h.00m.18s.406] shutdown...
```

FIGURE 7.23 – Erreur lorsque la patte MISO de la *flash* est mise à la masse trop tard.

```
ROM:      Use nor flash.
ROM:      Init DDR..Training done.
R

U-Boot 2010.06-svn172490 (Sep 20 2022 - 15:20:42)

DRAM:   64 MiB
MMC:   FH_MMC: 0
master [ctl : mem] = [0 : 0]
SF: Got idcode 20 40 17 20 40
product name:CS_XP1
Using SZ_8M TYPE_RT flash partition choice.
MMC FLASH INIT: No card on slot!
No mmc storage device found!
load_update_file fail
Net:   fh_gmac_initialize
FH EMAC
Hit Ctrl+u to stop autoboot: 0
load rt app

header_data.u32Magic is 0x00000000
header_data.u32header_len  is 0x0
header_data.u32RawDataLen is 0x0
magic err! it should be 0xa7b4c9f8,but header_data.u32Magic=0
* kernel: cmdline image address = 0xa0007fc0
Wrong Image Format for bootm command
ERROR: can't get kernel image!
HKVS # help
```

FIGURE 7.24 – Exploitation de la vulnérabilité pin2pwn.

figure 7.25. Il est également possible d'extraire le *firmware*, de modifier la configuration, etc. Il s'agit là d'exploitation, ce qui sort du cadre de l'analyse offensive

de PCB.

```
HKVS # printenv
bootargs=console=ttyS0,115200 root=/dev/ram0 mem=32M
bootcmd=loadss;bootm
bootdelay=2
baudrate=115200
ipaddr=192.0.0.64
serverip=192.0.0.128
gatewayip=192.0.0.1
netmask=255.255.255.0
phymode=RMII
update_source=net
boot_from=PART_MAIN
update_flag=update_
rst=0
stdin=serial
stdout=serial
stderr=serial
ethaddr=78:c1:ae:e3:db:a8
ethact=FH EMAC

Environment size: 343/65532 bytes
HKVS #
```

FIGURE 7.25 – Affichage des variables d'environnement de U-Boot.

Chapitre 8

Analyse d'un téléphone

Les smartphones ont évolué bien au-delà de simples dispositifs de communication. De par leur polyvalence et leur connectivité, ils ont redéfini la manière dont nous interagissons avec le monde qui nous entoure. Le cœur même de tout smartphone réside dans sa composante matérielle, qui englobe une diversité d'éléments électroniques. Comprendre le fonctionnement de ces composants est essentiel non seulement pour les ingénieurs et les développeurs, mais aussi pour les experts en sécurité, les enquêteurs criminels et les chercheurs. L'analyse approfondie du hardware des smartphones offre une perspective unique permettant de comprendre les mécanismes internes de ces appareils, ouvrant ainsi la voie à une meilleure compréhension de leur fonctionnement, de leurs vulnérabilités potentielles et des implications liées à la confidentialité et à la sécurité des données.

Dans cette optique, il est impératif de disséquer les différents éléments électroniques qui composent un smartphone. Du processeur aux capteurs spécialisés, en passant par la mémoire, la batterie et les modules de communication, chaque composant joue un rôle crucial dans l'expérience globale de l'utilisateur. Les composants suivants sont retrouvés dans les smartphones modernes :

- **Processeur** : Il est le cœur du smartphone. Il est responsable de l'exécution du système d'opération et de l'exécution des différentes applications. Il s'agit le plus souvent d'un **SoC** (*System on Chip*) qui combine un **CPU** à un ou plusieurs coeurs, une mémoire statique ou dynamique, des capteurs ou co-processeurs, ...
- **BBP** (*BaseBand Processor*) : Il permet de gérer les fonctions radio du smartphone (modulation, encodage, décodage, ...) qui tournent sur un système d'exploitation en temps réel (**RTOS** : *Real Time Operating System*).
- **Étage d'alimentation** : Permet d'adapter les tensions aux différents modules, mais aussi d'optimiser l'utilisation de la batterie.
- **Module d'affichage et contrôleur LCD** : Il est responsable du rendu visuel sur l'écran du smartphone et permettre ainsi à l'utilisateur d'interagir avec.
- **Audio** : L'interface audio permet de gérer les entrées (microphone) et sor-

ties audio (casque) du smartphone.

- **Autres modules** : Il peut y avoir des modules supplémentaires comme par exemple un module GPS ou un module Bluetooth.

Deux types de téléphones seront analysés dans cette section. Tout d'abord un téléphone classique, parfois appelé dumbphone, pour lequel une attaque par glitching de l'horloge sera effectuée afin de contourner l'authentification par code PIN. Puis un second téléphone plus moderne sera étudié afin d'appréhender les techniques utilisées pour récupérer le *firmware* depuis une eMMC (boîtier **BGA**).

8.1 Analyse statique du PCB

8.1.1 Dumbphone

La figure 8.1a présente la face avant du dumbphone qui sera analysé tandis que la figure 8.1b en présente la face arrière. Cette dernière permet de constater la présence de certains éléments présentés ci-dessus.



FIGURE 8.1 – Sur la figure 8.1a, la face avant présentant l'interface d'authentification par code PIN. Sur la figure 8.1b, la face arrière présentant les divers composants électroniques.

On peut observer sur la face arrière une partie des différents composants électroniques composants le **PCB**. En rouge la batterie et son connecteurs à

trois broches, en violet, le connecteur de carte **SIM**, le micro, le haut parleur, le vibreur et la caméra. D'autres composants sont disposés sous le blindage entouré en bleu. Afin de retirer ce blindage, une mini meuleuse droite Dremel a été utilisée (voir figure 8.2). De nouveaux composants sont alors visibles. Parmi ceux-ci, on peut observer sur la gauche ce qui semble être un amplificateur radiofréquence **RTM7292**, sur la droite un micro-contrôleur Spreadtrum Communication **SC6531E** embarquant un transceiver (*Transmitter-Receiver*) RF, une **flash PSRAM** pour les applications GSM/GPRS/BT/FM ainsi que la partie traitement du signal RF et permettant d'implémenter le traitement du signal de la caméra, de l'interface LCD, de la vidéo, de l'audio et interpréter les retours du clavier. Sous ce circuit intégré, on peut observer un cristal permettant de délivrer un signal d'horloge de 26 MHz.



FIGURE 8.2 – Blindage retiré sur le dumbphone laissant apparaître de nouveaux composants.

8.1.2 Smartphone

8.2 Analyse dynamique du PCB

8.3 Attaques

8.3.1 Glitch sur CLK

Le téléphone utilisé ne peut s'allumer que si la batterie est correctement positionnée. Ainsi il n'est pas possible d'allumer le téléphone avec le chargeur tout en enlevant la batterie. Une fois le téléphone démonté, il n'est plus possible de maintenir correctement la batterie, il faut alors la souder. Comme cela peut se voir sur la figure 8.1b, la batterie a trois pins (voir le connecteur). Il est nécessaire de souder les trois sans quoi le téléphone ne démarre pas. De gauche à droite, les pins sont :

1. VCC
2. un pin qui peut avoir plusieurs utilités comme par exemple répartir la charge de la batterie dans les différentes cellules ou bien pour mesurer la

température interne de la batterie.

3. GND

La figure 8.3 permet de constater le soudage de la batterie sur son connecteur. On aperçoit également un fil noir souder directement sur le **PCB**. Ce fil est relié à la masse et permettra d'y relier l'entrée du cristal par l'intermédiaire de l'autre extrémité. Un tel *glitch* effectué à la main est sommaire et très peu précis mais suffisant pour contourner le mécanisme d'authentification par code PIN.

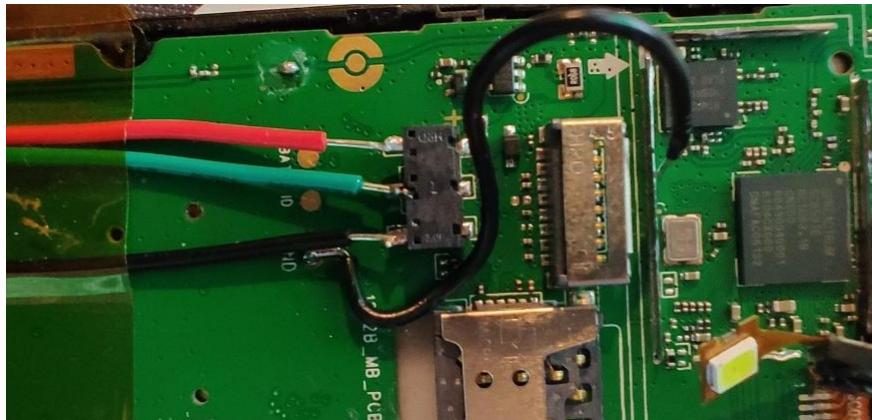


FIGURE 8.3 – Montage permettant de mettre un *glitch* sur **CLK**.

Le *glitch* doit durer quelques dizaines de millisecondes afin de perturber l'exécution de l'OS et alors de provoquer un redémarrage de celui-ci. Cependant la routine d'un *soft reboot* ne semble pas inclure le mécanisme d'authentification. Effectuer un *soft reboot* permet alors de contourner la protection par code PIN.

8.3.2 Extraction d'une eMMC

Chapitre 9

Analyse d'une tablette

9.1 Analyse statique du PCB

9.2 Analyse dynamique du PCB

9.3 Attaques

9.3.1 Attaque DMA

Annexe A

Annexes Électronique

A.1 Les différents types de boitiers SMD

Les composants **SMD** (*Surface Mounted Device*) sont disponibles sous diverses formes que l'on appelle des boîtiers. À la différence des composants **THT** (*Through Hole Technology*), les **SMD** ne traversent pas le **PCB**. De plus, les composants **SMD** sont des composants miniaturisés qui sont faits pour être positionnés et souder sur le **PCB** par des machines. Cependant, certains boîtiers sont suffisamment gros pour être positionnés et soudés à la main.

Les composants passifs tels que les résistances, les condensateurs, bobines et diodes (certains sont actifs) ont des boîtiers rectangulaires avec deux pattes (une à chaque extrémité). Le tableau suivant retranscrit les dimensions ainsi que les noms associés aux boîtiers.

Nom du boitier	taille (mm × mm)
01005	0,4 × 0,2 mm ²
015015	0,38 × 0,38 mm ²
0201	0,6 × 0,3 mm ²
0202	0,5 × 0,5 mm ²
02404	0,6 × 1,0 mm ²
0303	0,8 × 0,8 mm ²
0402	1,0 × 0,5 mm ²
0603	1,5 × 0,8 mm ²
0805	2,0 × 1,3 mm ²
1008	2,5 × 2,0 mm ²
1111	2,8 × 2,8 mm ²
1206	3,0 × 1,5 mm ²
1210	3,0 × 2,5 mm ²
1806	4,5 × 1,6 mm ²
1808	4,5 × 2,0 mm ²
1812	4,5 × 3,0 mm ²
1825	4,5 × 6,4 mm ²
2010	5,0 × 2,5 mm ²
2512	6,3 × 3,2 mm ²
2725	6,9 × 6,3 mm ²
2920	7,4 × 5,1 mm ²

TABLE A.1 – Présentation de différents boîtiers de composants **SMD** passifs ainsi que les tailles associées.

Les circuits intégrés ont d'autres types de boîtiers. Le nombre de pattes de ces composants étant de manière générale plus important que celui des composants passifs, il apparaît alors une nomenclature différente. On peut citer les boîtiers suivants :

- **SOIC/SOP/DIP** (*Small Outline Integrated Circuit*) : boîtier standard pour les circuits intégrés logiques (voir figure A.1a).
- **TSSOP** (*Thin Shrink Small Outline Package*) : boîtier utilisé notamment pour les amplificateurs analogiques et les mémoires (voir figure A.1b).
- **QFP** (*Quad Flat Package*) : boîtier utilisé notamment pour des micro-contrôleurs (voir figure A.1c).
- **QFN** (*Quad Flat No-lead*) : boîtier utilisé notamment pour des micro-contrôleurs (voir figure A.1d).
- **PLCC** (*Plastic Leaded Chip Carrier*) : boîtier utilisé notamment pour du prototypage (voir figure A.1e).
- **SOT** (*Small Outline Transistor*) : boîtier utilisé notamment pour les transistors, les diodes et les régulateurs de tension (voir figure A.1f).
- **BGA** (*Ball Grid Array*) : boîtier dans lequel les pins sont situés sous le composant et ne sont donc plus visibles une fois le composant soudé. Ce type de boîtiers est notamment utilisé pour des micro-processeurs, des **FPGA**, mais aussi des mémoires (voir figure A.1g).

Ces noms de boîtiers sont généralement affublés du nombre de pattes. Ainsi,

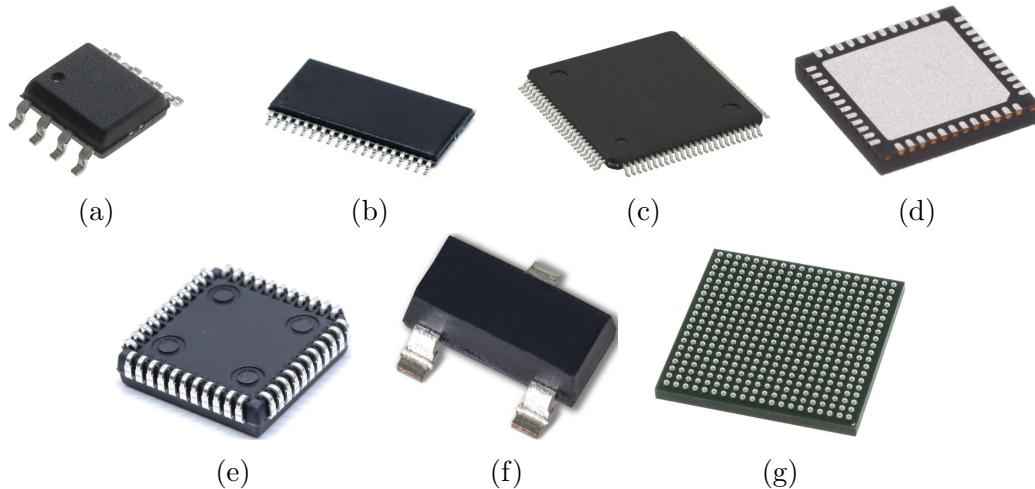


FIGURE A.1 – Présentation des différents types de boitiers pour les circuits intégrés.

une **EEPROM** à huit pattes dans un boîtier **SOIC** sera appelée **SOIC-8**. Un micro-contrôleur à 64 pattes dans un boîtier **QFP** sera appelé **QFP-64**.

A.2 Les différents types de vias

<https://www.protoexpress.com/kb/blind-and-buried-vias/>

A.3 Annexe soudure

blabla

A.4 Nomenclature des labels associés aux composants

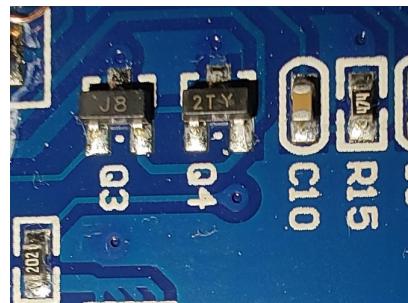
Lorsqu'un circuit électronique est dessiné, l'auteur y ajoute des labels afin de pouvoir s'y retrouver lors de la réalisation schématiques. Les outils de **CAO électronique (EDA)** en anglais : *Electronic Design Automation*) suivent une nomenclature afin d'associer un type de composant à une lettre. Ainsi, à chaque composant peut être associé un label :

Composant	label
résistance	R
condensateur	C
bobine	L
diode	D
cristal	Y ou X
led	LED
transistor	Q
circuit intégré	U

TABLE A.2 – Nomenclature des différents composants.

La ligne "circuit intégré" comprend tout ce qui est **LDO**, **flash**, **micro-contrôleur**, etc.

Ainsi il devient facile d'identifier les composants présents sur un **PCB** lorsque les labels y sont inscrits. Il en est de même lors de la compréhension de schématiques. La figure A.2 présente une portion de **PCB** sur laquelle sont identifiés certains composants par leurs labels. Bien sûr ce n'est pas toujours le cas.

FIGURE A.2 – Identification des composants grâce à leur label sur le **PCB**.

Annexe B

Annexes Bootloader

B.1 Fonctionnement d'un bootloader

Un *bootloader* est un programme spécialisé qui permet de charger et d'exécuter un autre programme (comme un système d'exploitation ou une application) sur un micro-contrôleur ou un système embarqué. Le principe général de fonctionnement d'un *bootloader* implique plusieurs étapes clefs :

1. **Activation** : Lorsque le dispositif démarre, il passe d'abord par le code du *bootloader*, qui est généralement stocké dans une zone spécifique de la mémoire, telle qu'une puce de mémoire *flash* ou une **ROM** interne au micro-contrôleur. Le *bootloader* peut être intégré en permanence dans le micro-contrôleur ou être temporairement chargé lors du démarrage.
2. **Initialisation** : Le *bootloader* initialise les composants matériels nécessaires à la communication, comme l'interface série (**UART**, **USB**, etc.) ou d'autres méthodes de communication telles que le réseau.
3. **Attente des commandes** : Le *bootloader* attend des commandes ou des instructions de l'extérieur. Ces commandes sont généralement envoyées par l'intermédiaire de l'interface de communication et peuvent inclure des opérations comme la mise à jour du *firmware*, l'effacement de la mémoire, la vérification de l'intégrité du code, etc.
4. **Réception des données** : Lorsqu'une commande est reçue, le *bootloader* est prêt à recevoir les données correspondantes, comme un nouveau *firmware*. Les données sont généralement envoyées en petits paquets pour éviter les problèmes de mémoire tampon ou de surcharge.
5. **Écriture en mémoire** : Les données reçues sont écrites dans la mémoire du dispositif. Cela peut impliquer l'écriture directe dans la mémoire *flash*, la mise à jour de certaines zones réservées, etc.
6. **Validation et Intégrité** : Après avoir reçu toutes les données, le *bootloader* peut vérifier l'intégrité du *firmware* en effectuant des contrôles de somme, des signatures numériques ou d'autres méthodes de validation pour s'assurer que les données sont correctes.
7. **Changement de contexte** : Une fois le nouveau *firmware* écrit et validé, le **bootloader** peut effectuer les opérations nécessaires pour passer du *bootloader* au programme principal (par exemple, en modifiant les pointeurs de démarrage).

8. **Lancement du programme principal :** Enfin, le *bootloader* transfère le contrôle au programme principal, qui peut être le système d'exploitation ou l'application. Le nouveau *firmware* est maintenant exécuté.

Les *bootloaders* sont utilisés pour faciliter la mise à jour du *firmware* ou bien son chargement en mémoire sans avoir besoin de dispositifs matériels externes (comme des programmeurs) et permettent également la récupération en cas de corruption du *firmware*. Ils sont couramment utilisés dans les systèmes embarqués, les micro-contrôleurs, les cartes de développement, etc.

Il est possible d'avoir plusieurs *bootloaders* qui se succèdent. Le premier permettant de charger le deuxième qui se chargera le troisième pour enfin mettre en mémoire le programme principale. Lorsque plusieurs *bootloaders* se succèdent, comme un **SPL** (*Secondary Program Loader*) suivi d'un **TPL** (*Tertiary Program Loader*), le processus suit généralement un schéma similaire à celui d'un seul *bootloader*, mais avec quelques variations pour gérer les différentes étapes et les différents niveaux de chargement. Voici comment cela pourrait se passer :

1. **PPL (Primary Program Loader) :**

- Le **PPL** est le premier *bootloader* qui est généralement exécuté lors du démarrage du dispositif.
- Il peut être responsable de l'initialisation de la pile et des registres du processeur, mais aussi de la configuration de la mémoire **RAM**.

2. **SPL (Secondary Program Loader) :**

- Le **SPL** est le second *bootloader* qui est généralement exécuté après le **PPL**.
- Il peut être responsable de tâches de bas niveau telles que l'initialisation des périphériques matériels, la configuration de la mémoire, etc.
- Le **SPL** peut ensuite passer le contrôle au **TPL** si nécessaire.

3. **TPL (Tertiary Program Loader) :**

- Le **TPL** est le troisième niveau de *bootloader*, généralement exécuté après le **SPL**.
- Il peut être responsable de tâches plus avancées, comme la gestion des communications, la validation des signatures numériques, etc.
- Le **TPL** peut avoir la possibilité de charger différents types de programmes, tels que des systèmes d'exploitation ou des applications.
- Il peut également être capable de gérer des mises à jour plus complexes et sécurisées du *firmware*.

4. **Changement du programme principal :**

- Une fois que le **TPL** a effectué toutes les tâches nécessaires, il peut transférer le contrôle au programme principal, tel qu'un système d'exploitation ou une application.
- Le programme principal est ensuite exécuté.

Chaque *bootloader* successif peut avoir des responsabilités plus avancées et spécialisées que le précédent. Par exemple, le **SPL** peut être principalement responsable de l'initialisation matérielle, tandis que le **TPL** peut gérer des tâches de communication, de sécurité et de chargement de programmes plus complexes.

Il n'y a pas d'actions spécifiques pour chaque *bootloader*, cela dépend de ce que souhaite faire le concepteur. Ainsi les étapes présentées ci-dessus ne sont que des exemples d'utilisation du **PPL**, du **SPL** et du **TPL**.

Lorsque plusieurs *bootloaders* sont impliqués, il est important de gérer soigneusement les transitions entre eux pour assurer une communication fluide et la continuité de l'exécution. Chaque *bootloader* doit être conçu pour pouvoir transmettre les informations nécessaires au suivant, comme les données de configuration, les informations de validation, etc.

En fin de compte, le schéma précis et la manière dont les *bootloaders* se succèdent dépendent de la conception du système, de l'architecture matérielle et des besoins spécifiques du projet.

B.2 Les différents bootloaders

Il existe plusieurs *bootloaders* propriétaires sur le marché, développés par différentes entreprises pour répondre aux besoins spécifiques de leurs produits et plateformes. Voici quelques-uns des *bootloaders* propriétaires bien connus :

- **FastBoot** : Développé par **Google**, **FastBoot** est un *bootloader* utilisé principalement sur les appareils **Android**. Il permet le chargement du système d'exploitation **Android**, des noyaux personnalisés et d'autres images système.
- **EDK2 (Extensible Firmware Interface Development Kit 2)** : Également connu sous le nom de **UEFI** (*Unified Extensible Firmware Interface*), **EDK2** est un *firmware* moderne qui prend en charge l'initialisation matérielle et le démarrage sur les plates-formes **x86** et **ARM**. Bien que certains aspects de l'**EDK2** soient *opensource*, il en existe des implémentations propriétaires.
- **iBoot** : Développé par **Apple**, **iBoot** est un *bootloader* utilisé sur les appareils **iOS**, notamment les **iPhones** et les **iPads**. Il est responsable du chargement du système d'exploitation **iOS** et de la vérification de l'intégrité du système.
- **SecureROM** : Également développé par **Apple**, **SecureROM** est un *bootloader* bas niveau utilisé sur les appareils **iOS**. Il est chargé directement à partir de la mémoire **ROM** du processeur et est responsable de l'amorçage initial du système.
- **QNX Boot Loader** : Utilisé sur les systèmes embarqués et en temps réel, le *bootloader* **QNX** permet de charger le système d'exploitation **QNX Neutrino RTOS** et d'autres logiciels associés.
- **Texas Instruments Bootloader** : **Texas Instrument** propose différents *bootloaders* propriétaires pour ses gammes de micro-contrôleurs, tels que **U-Boot based bootloaders** et d'autres solutions spécifiques.
- **Microchip MPLAB Bootloader** : **Microchip** propose des *bootloaders* propriétaires pour ses micro-contrôleurs **PIC** et **AVR**, qui permettent la mise à jour du *firmware* via différentes interfaces de communication.
- **STMicroelectronics ST Bootloader** : **STMicroelectronics** offre des *bootloaders* propriétaires pour ses micro-contrôleurs **STM32**, permettant

la mise à jour du firmware via des interfaces comme **UART**, **USB**, etc.

Il existe également de nombreux *bootloaders opensource* disponibles sur le marché, développés par des communautés de contributeurs et d'entreprises pour offrir des solutions d'amorçage flexibles et personnalisables. Voici quelques-uns des *bootloaders opensource* les plus connus :

- **U-Boot** : **U-Boot** est l'un des *bootloaders opensource* les plus populaires. Il prend en charge un large éventail d'architectures matérielles et de fonctionnalités. **U-Boot** est extensible et personnalisable, et il est souvent utilisé dans les systèmes embarqués pour charger des systèmes d'exploitation et des applications.
- **Das U-Boot (DENX Software Engineering)** : Cette version spécifique d'**U-Boot** est largement utilisée et maintenue par **DENX Software Engineering**. Elle inclut de nombreuses fonctionnalités et des améliorations pour différents matériels.
- **Coreboot** : Anciennement connu sous le nom de **LinuxBIOS**, **Coreboot** est un projet *opensource* visant à remplacer les **BIOS/UEFI** propriétaires. Il est conçu pour une utilisation sur des plates-formes **x86** et prend en charge un grand nombre de cartes mères et de puces.
- **AT91Bootstrap** : C'est un *bootloader opensource* pour les micro-contrôleurs **ARM** de la famille **AT91** (maintenant **Microchip**). Il est souvent utilisé avec **U-Boot** pour amorcer des systèmes basés sur ces micro-contrôleurs.
- **Libreboot** : **Libreboot** est un projet *opensource* visant à fournir une alternative entièrement libre et dépourvue de logiciels propriétaires au *firmware* des systèmes basés sur les puces **x86**. Il est basé sur **Coreboot** et vise à éliminer les composants non libres du **BIOS/UEFI** pour offrir un environnement de démarrage entièrement open source.
- **RedBoot** : **RedBoot** est un *bootloader opensource* développé par la société **Wind River**. Il prend en charge plusieurs architectures matérielles et peut être utilisé pour le développement, la mise à jour du firmware et la gestion des systèmes embarqués.
- **Oreboot** : **Oreboot** est un *bootloader opensource* pour l'architecture **RISC-V**. Il est conçu pour être simple, modulaire et facile à personnaliser.

B.3 U-boot bootloader

U-Boot est l'un des *bootloaders opensource* les plus populaires et largement utilisés dans le domaine des systèmes embarqués. Il s'agit d'un programme de démarrage capable de charger et d'exécuter diverses images, notamment des systèmes d'exploitation, des noyaux de noyaux, des applications et des images de *firmware* sur une large gamme de plates-formes matérielles. Les points essentiels sont les suivants :

1. **Rôle et Fonctionnalités** : **U-Boot** joue un rôle crucial dans la séquence de démarrage d'un système embarqué. Ses principales fonctionnalités incluent
 - **Initialisation matérielle** : Configuration des composants matériels,

- des registres du processeur, de la mémoire et des interfaces de communication.
- **Chargement d'images** : Chargement en mémoire des images, telles que les systèmes d'exploitation, les noyaux de noyaux, les applications, etc.
 - **Gestion des périphériques** : Prise en charge des périphériques matériels tels que l'UART, l'USB, l'Ethernet, les cartes SD, etc.
 - **Gestion de la mémoire** : Allocation de la mémoire pour les images chargées et gestion des zones-mémoires réservées.
 - **Validation d'images** : Vérification de l'intégrité des images chargées pour s'assurer qu'elles n'ont pas été corrompues.
 - **Interaction utilisateur** : Fourniture d'un *shell* de commande interactif pour la configuration, le débogage et la manipulation des images.
2. **Adaptabilité** : U-Boot est hautement adaptable et peut être utilisé sur une variété de plates-formes matérielles, telles que les processeurs **ARM**, **x86**, **MIPS**, **PowerPC**, **RISC-V**, et bien d'autres. Il est souvent utilisé sur des cartes de développement, des systèmes embarqués personnalisés, des routeurs, des **NAS** (*Network Attached Storage*) et d'autres appareils.
 3. **Personnalisation** : U-Boot est conçu pour être hautement personnalisable. Il peut être configuré avec différentes fonctionnalités en fonction des besoins spécifiques de chaque projet. Les développeurs peuvent personnaliser les options de compilation, les commandes disponibles, les pilotes matériels pris en charge et plus encore.
 4. **Image Types** : U-Boot prend en charge différents types d'images, notamment les images de noyau **Linux**, les images de système de fichiers, les images d'applications, les images de *firmware*, etc. Ces images peuvent être stockées sur divers supports, tels que la mémoire *flash*, les cartes SD, les disques USB, etc.
 5. **Modes de Fonctionnement** : U-Boot peut fonctionner de différentes manières, notamment en mode interactif où l'utilisateur peut interagir avec le *shell* de commande, et en mode automatique où il exécute des scripts de démarrage prédéfinis.
 6. **Communauté et Documentation** : U-Boot est soutenu par une communauté active de développeurs *opensource*. Il dispose de documentation approfondie, de guides d'utilisation, de références de commandes et de ressources en ligne pour aider les utilisateurs et les développeurs à démarrer, configurer et personnaliser le *bootloader*.

Annexe C

Annexes Carte à puce

C.1 Liste des codes de status

La présente liste des codes de status concerne la norme **EMV** relative aux cartes bancaires (source : [8]).

Table C.1: Exemples de code d'erreur (source : [8]).

SW1	SW2	Type	Description
6		Error	Class not supported.
61	—	Info	Response bytes still available
61	XX	Info	Command successfully executed; 'XX'bytes of data are available and can be requested using GET RESPONSE.
62	—	Warning	State of non-volatile memory unchanged
62	00	Warning	No information given (NV-Ram not changed)
62	01	Warning	NV-Ram not changed 1.
62	81	Warning	Part of returned data may be corrupted
62	82	Warning	End of file/record reached before reading Le bytes
62	83	Warning	Selected file invalidated
62	84	Warning	Selected file is not valid. FCI not formated according to ISO
62	85	Warning	No input data available from a sensor on the card. No Purse Engine enslaved for R3bc
62	A2	Warning	Wrong R-MAC
62	A4	Warning	Card locked (during reset())

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
62	CX	Warning	Counter with value x (command dependent)
62	F1	Warning	Wrong C-MAC
62	F3	Warning	Internal reset
62	F5	Warning	Default agent locked
62	F7	Warning	Cardholder locked
62	F8	Warning	Basement is current agent
62	F9	Warning	CALC Key Set not unblocked
62	FX	Warning	—
62	XX	Warning	RFU
63	—	Warning	State of non-volatile memory changed
63	00	Warning	No information given (NV-Ram changed)
63	81	Warning	File filled up by the last write. Loading/updating is not allowed.
63	82	Warning	Card key not supported.
63	83	Warning	Reader key not supported.
63	84	Warning	Plaintext transmission not supported.
63	85	Warning	Secured transmission not supported.
63	86	Warning	Volatile memory is not available.
63	87	Warning	Non-volatile memory is not available.
63	88	Warning	Key number not valid.
63	89	Warning	Key length is not correct.
63	C0	Warning	Verify fail, no try left.
63	C1	Warning	Verify fail, 1 try left.
63	C2	Warning	Verify fail, 2 tries left.
63	C3	Warning	Verify fail, 3 tries left.
63	CX	Warning	The counter has reached the value 'x'(0 = x = 15) (command dependent).
63	F1	Warning	More data expected.
63	F2	Warning	More data expected and proactive command pending.
63	FX	Warning	—
63	XX	Warning	RFU
64	—	Error	State of non-volatile memory unchanged

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
64	00	Error	No information given (NV-Ram not changed)
64	01	Error	Command timeout. Immediate response required by the card.
64	XX	Error	RFU
65	—	Error	State of non-volatile memory changed
65	00	Error	No information given
65	01	Error	Write error. Memory failure.
65	81	Error	Memory failure
65	FX	Error	—
65	XX	Error	RFU
66	—	Security	
66	00	Security	Error while receiving (timeout)
66	01	Security	Error while receiving (character parity error)
66	02	Security	Wrong checksum
66	03	Security	The current DF file without FCI
66	04	Security	No SF or KF under the current DF
66	69	Security	Incorrect Encryption/Decryption Padding
66	XX	Security	—
67	—	Error	
67	00	Error	Wrong length
67	XX	Error	length incorrect (procedure)(ISO 7816-3)
68	—	Error	Functions in CLA not supported
68	00	Error	No information given (The request function is not supported by the card)
68	81	Error	Logical channel not supported
68	82	Error	Secure messaging not supported
68	83	Error	Last command of the chain expected
68	84	Error	Command chaining not supported
68	FX	Error	—
68	XX	Error	RFU
69	—	Error	Command not allowed
69	00	Error	No information given (Command not allowed)

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
69	01	Error	Command not accepted (inactive state)
69	81	Error	Command incompatible with file structure
69	82	Error	Security condition not satisfied.
69	83	Error	Authentication method blocked
69	84	Error	Referenced data reversibly blocked (invalidated)
69	85	Error	Conditions of use not satisfied.
69	86	Error	Command not allowed (no current EF)
69	87	Error	Expected secure messaging (SM) object missing
69	88	Error	Incorrect secure messaging (SM) data object
69	8D		Reserved
69	96	Error	Data must be updated again
69	E1	Error	POL1 of the currently Enabled Profile prevents this action.
69	F0	Error	Permission Denied
69	F1	Error	Permission Denied – Missing Privilege
69	FX	Error	–
69	XX	Error	RFU
6A	—	Error	Wrong parameter(s) P1-P2
6A	00	Error	No information given (Bytes P1 and/or P2 are incorrect)
6A	80	Error	The parameters in the data field are incorrect.
6A	81	Error	Function not supported
6A	82	Error	File not found
6A	83	Error	Record not found
6A	84	Error	There is insufficient memory space in record or file
6A	85	Error	Lc inconsistent with TLV structure
6A	86	Error	Incorrect P1 or P2 parameter.
6A	87	Error	Lc inconsistent with P1-P2
6A	88	Error	Referenced data not found
6A	89	Error	File already exists
6A	8A	Error	DF name already exists.

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
6A	F0	Error	Wrong parameter value
6A	FX	Error	—
6A	XX	Error	RFU
6B	—	Error	
6B	00	Error	Wrong parameter(s) P1-P2
6B	XX	Error	Reference incorrect (procedure byte), (ISO 7816-3)
6C	—	Error	Wrong length Le
6C	00	Error	Incorrect P3 length.
6C	XX	Error	Bad length value in Le ; ‘xx’ is the correct exact Le
6D	—	Error	
6D	00	Error	Instruction code not supported or invalid
6D	XX	Error	Instruction code not programmed or invalid (procedure byte), (ISO 7816-3)
6E	—	Error	
6E	00	Error	Class not supported
6E	XX	Error	Instruction class not supported (procedure byte), (ISO 7816-3)
6F	—	Error	Internal exception
6F	00	Error	Command aborted – more exact diagnosis not possible (e.g., operating system error).
6F	FF	Error	Card dead (overuse, ...)
6F	XX	Error	No precise diagnosis (procedure byte), (ISO 7816-3)
9-	—		
90	00	Info	Command successfully executed (OK).
90	04	Warning	PIN not successfully verified, 3 or more PIN tries left
90	08		Key/file not found
90	80	Warning	Unblock Try Counter has reached zero
91	00		OK
91	01		States.activity, States.lock Status or States.lockable has wrong value
91	02		Transaction number reached its limit
91	0C		No changes

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
91	0E		Insufficient NV-Memory to complete command
91	1C		Command code not supported
91	1E		CRC or MAC does not match data
91	40		Invalid key number specified
91	7E		Length of command string invalid
91	9D		Not allow the requested command
91	9E		Value of the parameter invalid
91	A0		Requested AID not present on PICC
91	A1		Unrecoverable error within application
91	AE		Authentication status does not allow the requested command
91	AF		Additional data frame is expected to be sent
91	BE		Out of boundary
91	C1		Unrecoverable error within PICC
91	CA		Previous Command was not fully completed
91	CD		PICC was disabled by an unrecoverable error
91	CE		Number of Applications limited to 28
91	DE		File or application already exists
91	EE		Could not complete NV-write operation due to loss of power
91	F0		Specified file number does not exist
91	F1		Unrecoverable error within file
92	0x	Info	Writing to EEPROM successful after 'x' attempts.
92	10	Error	Insufficient memory. No more storage available.
92	40	Error	Writing to EEPROM not successful.
93	01		Integrity error
93	02		Candidate S2 invalid
93	03	Error	Application is permanently locked
94	00	Error	No EF selected.
94	01		Candidate currency code does not match purse currency

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
94	02		Candidate amount too high
94	02	Error	Address range exceeded.
94	03		Candidate amount too low
94	04	Error	FID not found, record not found or comparison pattern not found.
94	05		Problems in the data field
94	06	Error	Required MAC unavailable
94	07		Bad currency : purse engine has no slot with R3bc currency
94	08		R3bc currency not supported in purse engine
94	08	Error	Selected file type does not match command.
95	80		Bad sequence
96	81		Slave not found
97	00		PIN blocked and Unblock Try Counter is 1 or 2
97	02		Main keys are blocked
97	04		PIN not successfully verified, 3 or more PIN tries left
97	84		Base key
97	85		Limit exceeded – C-MAC key
97	86		SM error – Limit exceeded – R-MAC key
97	87		Limit exceeded – sequence counter
97	88		Limit exceeded – R-MAC length
97	89		Service not available
98	02	Error	No PIN defined.
98	04	Error	Access conditions not satisfied, authentication failed.
98	35	Error	ASK RANDOM or GIVE RANDOM not executed.
98	40	Error	PIN verification not successful.
98	50	Error	INCREASE or DECREASE could not be executed because a limit has been reached.
98	62	Error	Authentication Error, application specific (incorrect MAC)

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
99	00		1 PIN try left
99	04		PIN not successfully verified, 1 PIN try left
99	85		Wrong status – Cardholder lock
99	86	Error	Missing privilege
99	87		PIN is not installed
99	88		Wrong status – R-MAC state
9A	00		2 PIN try left
9A	04		PIN not successfully verified, 2 PIN try left
9A	71		Wrong parameter value – Double agent AID
9A	72		Wrong parameter value – Double agent Type
9D	05	Error	Incorrect certificate type
9D	07	Error	Incorrect session data size
9D	08	Error	Incorrect DIR file record size
9D	09	Error	Incorrect FCI record size
9D	0A	Error	Incorrect code size
9D	10	Error	Insufficient memory to load application
9D	11	Error	Invalid AID
9D	12	Error	Duplicate AID
9D	13	Error	Application previously loaded
9D	14	Error	Application history list full
9D	15	Error	Application not open
9D	17	Error	Invalid offset
9D	18	Error	Application already loaded
9D	19	Error	Invalid certificate
9D	1A	Error	Invalid signature
9D	1B	Error	Invalid KTU
9D	1D	Error	MSM controls not set
9D	1E	Error	Application signature does not exist
9D	1F	Error	KTU does not exist
9D	20	Error	Application not loaded
9D	21	Error	Invalid Open command data length

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
9D	30	Error	Check data parameter is incorrect (invalid start address)
9D	31	Error	Check data parameter is incorrect (invalid length)
9D	32	Error	Check data parameter is incorrect (illegal memory check area)
9D	40	Error	Invalid MSM Controls ciphertext
9D	41	Error	MSM controls already set
9D	42	Error	Set MSM Controls data length less than 2 bytes
9D	43	Error	Invalid MSM Controls data length
9D	44	Error	Excess MSM Controls ciphertext
9D	45	Error	Verification of MSM Controls data failed
9D	50	Error	Invalid MCD Issuer production ID
9D	51	Error	Invalid MCD Issuer ID
9D	52	Error	Invalid set MSM controls data date
9D	53	Error	Invalid MCD number
9D	54	Error	Reserved field error
9D	55	Error	Reserved field error
9D	56	Error	Reserved field error
9D	57	Error	Reserved field error
9D	60	Error	MAC verification failed
9D	61	Error	Maximum number of unblocks reached
9D	62	Error	Card was not blocked
9D	63	Error	Crypto functions not available
9D	64	Error	No application loaded
9E	00		PIN not installed
9E	04		PIN not successfully verified, PIN not installed
9F	00		PIN blocked and Unblock Try Counter is 3
9F	04		PIN not successfully verified, PIN blocked and Unblock Try Counter is 3
9F	XX		Command successfully executed; 'xx'bytes of data are available and can be requested using GET RESPONSE.

Continued on next page

Table C.1: Exemples de code d'erreur (source : [8]). (Continued)

SW1	SW2	Type	Description
9x	XX		Application related status, (ISO 7816-3)

C.2 Liste d'instructions connues

Les instructions (**INS**) sont définies par la norme **ISO 7816** et sont donc connues [?]. Pour pouvoir les utiliser il faut leur ajouter en préfixe la classe (**CLA**) (voir tableau C.2). Ces instructions sont présentées dans le tableau C.3.

Table C.2: Liste des classes connues.

CLA	norme
0X	ISO 7816
10-7F	Reservé
8x	ISO 7816
AX	Instructions spécifiques applications/vendeurs
B0-CF	ISO 7816
D0-FE	Instructions spécifiques applications/vendeurs
FF	Réservé pour la sélection du protocole

Table C.3: Liste des instructions connues.

INS	Nom
0E	Erase Binary
20	Verify
70	Manage Channel
82	External Authenticate
84	Get Challenge
88	Internal Authenticate
A4	Select File
B0	Read Binary
B2	Read Record(s)
C0	Get Response
C2	Envelope
CA	Get Data
D0	Write Binary
D2	Write Record

Continued on next page

Table C.3: Liste des instructions connues.
(Continued)

INS	Nom
D6	Update Binary
DA	Put Data
DC	Update Record
E2	Append Record

1. Select File

- Cette commande établit un pointeur logique vers un fichier particulier dans le système de fichiers de la carte à puce.
- Ce pointeur est requis pour toute opération de manipulation de fichier.
- L'identification du dossier peut être fournie des manières suivantes :
 - Identifiant de fichier (valeur sur 2 octets)
 - Nom **DF** (chaîne d'octets)
 - Chemin (concaténation des identifiants de fichiers)
 - ID court

2. Read Binary

- Cette commande est utilisée par l'application côté lecteur pour récupérer une partie d'un **EF**.
- Cependant, l'**EF** doit être un fichier transparent (non orienté enregistrement).
- La commande *Read Binary* prend deux paramètres :
 - Un pointeur de décalage depuis le début du fichier vers l'octet initial à lire
 - Le nombre d'octets à lire et à renvoyer au lecteur.

3. Write Binary

- Cette commande est utilisée pour insérer des données dans un **EF** transparent sur la carte.

4. Update Binary

- Une application côté lecteur peut utiliser cette commande pour effacer et stocker directement une séquence d'octets dans un **EF** transparent sur la carte.
- Cette commande fonctionne comme une commande d'écriture, c'est-à-dire qu'une chaîne d'octets fournie dans la commande est écrite dans l'**EF**.
- Les paramètres sont constitués d'un pointeur de décalage depuis le début du fichier ainsi que du nombre d'octets à écrire.

5. Erase Binary

- La commande **Erase Binary** est utilisée pour effacer les octets dans un **EF** transparent.
- Les paramètres d'entrée comprennent un décalage du début de l'**EF** à effacer ainsi que le nombre d'octets à effacer.

6. Read Record

- Cette commande est utilisée pour lire et renvoyer le contenu d'un ou plusieurs enregistrements d'un **EF**.
- Si la est appliquée à un **EF** transparent, la commande sera abandonnée et une erreur sera renvoyée au lecteur.
- Les éléments suivants peuvent être renvoyés par cette commande, en fonction des paramètres d'entrée :
 - Un enregistrement spécifié.
 - Tous les enregistrements depuis le début du fichier jusqu'à un enregistrement spécifique.
 - Tous les enregistrements depuis un enregistrement spécifique jusqu'à la fin du fichier.

7. Write Record

- Cette commande est utilisée pour écrire un enregistrement dans un **EF** orienté enregistrement.
- Comme avec la commande *Write Binary*, cette commande peut être utilisée pour écrire un enregistrement dans un **EF**.

8. Append Record

- Cette commande est utilisée pour ajouter un enregistrement à la fin d'un **EF** linéaire orienté enregistrement ou pour écrire le premier enregistrement dans un EF cyclique orienté enregistrement sur une carte.

9. Update Record

- Cette commande écrit un enregistrement spécifique dans un **EF** orienté enregistrement sur une carte.
- Comme pour la commande *Update Binary*, l'ancien enregistrement est effacé et le nouveau est écrit dans l'**EF**.

10. Get Data

- Cette commande lit et renvoie le contenu d'un objet de données stocké dans le système de fichiers sur la carte.
- La commande *Get Data* est spécifique à la carte car la définition d'un objet de données varie entre différentes cartes.

11. Put Data

- Cette commande place des informations dans un objet de données sur la carte. Comme avec la commande *Get Data*, il s'agit d'une commande spécifique à la carte.

12. Verify

- Cette commande est envoyée par l'application côté lecteur au système de sécurité de la carte.
- Son but est de convaincre la carte que le lecteur connaît un mot de passe conservé par la carte afin de restreindre l'accès aux informations sensibles stockées sur la carte.
- Les informations peuvent être associées à un fichier spécifique ou à tout ou partie de la hiérarchie des fichiers.

- Si la commande de vérification échoue, c'est-à-dire que le lecteur fournit un mot de passe incorrect, une erreur est renvoyée au lecteur.

13. Internal Authenticate

- Cette commande permet à la carte de s'authentifier auprès du lecteur en prouvant qu'elle possède une clef secrète partagée avec le lecteur.
- Le logiciel d'application du lecteur génère d'abord un nombre aléatoire et le chiffre avec un algorithme connu à la fois de la carte et du lecteur. Ceci constitue un *challenge* à la carte.
- La carte déchiffre ensuite ce *challenge* avec la clef secrète (qui est stockée sur la carte) et renvoie les données résultantes au lecteur.
- Si les données reçues par le lecteur correspondent au nombre aléatoire qu'il a généré, le logiciel d'application du lecteur est assuré de l'identité de la carte.

14. External Authenticate

- Cette commande est utilisée conjointement avec la commande *Get Challenge* pour activer le logiciel d'application du lecteur pour s'authentifier auprès de la carte.
- Le lecteur reçoit un *challenge* (un nombre aléatoire) de la carte et le chiffre avec une clef secrète. Ceci est ensuite envoyé à la carte à l'aide de la commande *External Authenticate*.
- La carte déchiffre les données et les compare au nombre aléatoire qu'il a généré avec la commande *Get Challenge* précédente.
- S'il y a une correspondance, la carte est alors assurée de l'identité de l'application lecteur.

15. Get Challenge

- Cette commande est envoyée par le lecteur à la carte.
- Son but est de fournir au lecteur un nombre aléatoire généré par la carte à puce.
- Comme décrit précédemment, ce numéro est utilisé avec la commande *External Authenticate*.

16. Manage Channel

- Cette commande est utilisée par le lecteur pour ouvrir et fermer les canaux de communication entre celui-ci et la carte.
- Initialement, la carte ouvre un canal de communication en établissant un protocole au niveau de l'application avec l'application de lecture grâce à l'achèvement d'une séquence **ATR**.
- Ce canal est ensuite utilisé pour ouvrir ou fermer des canaux logiques supplémentaires via la commande *Manage Channel*.

17. Envelope

- Cette commande prend en charge l'utilisation de la messagerie sécurisée utilisant le protocole T=0.
- Elle autorise un **APDU** à chiffrer puis incorporer dans la section de données de la commande **Envelope** (de son **APDU**).
- Le processeur **APDU** de la carte peut alors extraire et exécuter la commande.

- Elle est utilisée par les ordres entrants/sortants lorsque le protocole de transport ne les supportent pas (**T=0**).

18. Get Response

- Comme pour la commande *Envelope*, la commande *Get Response* permet d'utiliser le protocole T=0 pour transférer des **APDU**.
- La commande *Get Response* est ensuite utilisée pour récupérer ces données depuis la carte lorsque le protocole de transport ne le permet pas directement (**T=0**).

C.3 Liste d'AIDs connus

Certains **AIDs** sont connus (source : [8]). Une liste non exhaustive est présentée dans le tableau C.4.

Table C.4: Liste d'AIDs connus (source : [8]).

AID	Vendeur	Pays	Nom	Type
315041592E535953 2E4444463031	Visa International	United States	Visa Payment System Environment – PSE	
325041592E535953 2E4444463031	Visa International	United States	Visa Proximity Payment System Environment – PPSE	
44464D46412E4446 6172653234313031	DeviceFidelity	United States	DeviceFidelity In2Pay DFare applet	
504F4B455231			GP Poker-Demo	
534B542E4D454D42 45523031			SKT	
534C424352595054 4F			SLB PKI	
6D6966617265			MiFare	
A00000000101	PBS Danmønt A/S	Denmark	MUSCLE Card Applet	
A00000000201	Mondex International Ltd.	United Kingdom	MONDEX	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000030000	Visa International	United States	Global Platform CardManager	
A000000003000000	Visa International	United States	Card Manager	GP
A00000000300037561	Visa International	United States	Bonuscard	
A00000000305076010	Visa International	United States	VISA ELO Credit	EMV
A0000000031010	Visa International	United States	VISA Debit/Credit	EMV
A000000003101001	Visa International	United States	VISA Credit	EMV
A000000003101002	Visa International	United States	VISA Debit	EMV
A0000000031111	Visa International	United States	Visa ? ? ? ?	
A0000000032010	Visa International	United States	VISA Electron	EMV
A0000000032020	Visa International	United States	VISA	EMV
A0000000033010	Visa International	United States	VISA Interlink	EMV
A0000000033060	Visa International	United States	Visa PreAuthorized ?	
A0000000034010	Visa International	United States	VISA Specific	EMV
A0000000035010	Visa International	United States	VISA Specific	EMV
A000000003534441	Visa International	United States	Schlumberger Security Domain	GP
A0000000035350	Visa International	United States	Security Domain	GP
A000000003535041	Visa International	United States	Security Domain	GP
A0000000036000	Visa International	United States	ATM card ?	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000036010	Visa International	United States	Domestic Visa Cash Stored Value	EMV
A0000000036020	Visa International	United States	International Visa Cash Stored Value	EMV
A0000000038002	Visa International	United States	VISA Auth, VisaRemAuth EMV-CAP	EMV
A0000000038010	Visa International	United States	VISA Plus	EMV
A0000000039010	Visa International	United States	VISA Loyalty	EMV
A000000003999910	Visa International	United States	VISA Proprietary ATM	EMV
A0000000040000	Mastercard International	United States	MasterCard Card Manager	GP
A00000000401	Mastercard International	United States	MasterCard PayPass	EMV
A0000000041010	Mastercard International	United States	MasterCard Credit/Debit	EMV
A000000004101012 13	Mastercard International	United States	MasterCard Credit	EMV
A000000004101012 15	Mastercard International	United States	MasterCard Credit	EMV
A0000000041010BB 5449435301	Mastercard International	United States	[UNKNOWN]	
A0000000041010C0 000301	Mastercard International	United States	Comptant	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000041010C0 000302	Mastercard International	United States	Credit Pass	
A0000000042010	Mastercard International	United States	MasterCard Specific	EMV
A0000000042203	Mastercard International	United States	MasterCard Specific	
A0000000043010	Mastercard International	United States	MasterCard Specific	EMV
A0000000043060	Mastercard International	United States	Maestro	EMV
A000000004306001	Mastercard International	United States	Maestro	EMV
A0000000044010	Mastercard International	United States	MasterCard Specific	EMV
A0000000045000	Mastercard International	United States	MODS	
A0000000045010	Mastercard International	United States	MasterCard Specific	EMV
A0000000045555	Mastercard International	United States	APDULogger	
A0000000046000	Mastercard International	United States	Cirrus	EMV
A0000000046010	Mastercard International	United States	Cirrus	
A0000000048002	Mastercard International	United States	SecureCode Auth EMV-CAP	EMV
A0000000049999	Mastercard International	United States	MasterCard Pay-Pass ??	EMV

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000050001	Switch Card Services Ltd.	United Kingdom	Maestro UK	EMV
A0000000050002	Switch Card Services Ltd.	United Kingdom	Solo	EMV
A0000000090001FF 44FF1289	ETSI	France	Orange	
A0000000101030	Europay International	Belgium	Maestro-CH	
A00000001800	GEMPLUS	France	Gemplus ?	
A0000000180F0000 0167	GEMPLUS	France	Gemclub Loyalty	
A0000000181001	GEMPLUS	France		
A000000018434D	GEMPLUS	France	Gemplus card manager	GP
A000000018434D00	GEMPLUS	France	Gemplus Security Domain	GP
A00000002401	Midland Bank Plc	United Kingdom	Self Service	EMV
A000000025	American Express	United Kingdom	American Express	EMV
A0000000250000	American Express	United Kingdom	American Express	EMV
A00000002501	American Express	United Kingdom	American Express	EMV
A000000025010104	American Express	United Kingdom	American Express	
A000000025010402	American Express	United Kingdom	American Express	EMV
A000000025010701	American Express	United Kingdom	ExpressPay	EMV
A000000025010801	American Express	United Kingdom	American Express	EMV

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000291010	LINK Interchange Network Ltd	United Kingdom		EMV
A000000029450875 10100000	LINK Interchange Network Ltd	United Kingdom	CO-OP	
A000000029490340 10100001	LINK Interchange Network Ltd	United Kingdom	HSBC	
A000000029492820 10100000	LINK Interchange Network Ltd	United Kingdom	Barclay	
A000000029564182	LINK Interchange Network Ltd	United Kingdom	HAFX	
A000000030000090 078141100000	Schlumberger Industries Identif d'En- carteur PR050	France	SLB PIN Manager	
A000000030290570 00AD13100101FF	Schlumberger Industries Identif d'En- carteur PR050	France		
A000000030800000 0000280101	Schlumberger Industries Identif d'En- carteur PR050	France	Gemalto .NET Card AID	
A0000000421010	Groupement des Cartes Bancaires "CB"	France	Cartes Bancaire EMV Card	EMV

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000000422010	Groupement des Cartes Bancaires "CB"	France		EMV
A0000000423010	Groupement des Cartes Bancaires "CB"	France		EMV
A0000000424010	Groupement des Cartes Bancaires "CB"	France		EMV
A0000000425010	Groupement des Cartes Bancaires "CB"	France		EMV
A0000000426010	Groupement des Cartes Bancaires "CB"	France	Contactless payment using Apple Pay	EMV
A00000005945430 100	Zentraler Kreditausschuss	Germany	Girocard Electronic Cash	
A00000005950414 3450100	Zentraler Kreditausschuss	Germany	Geldkarte	
A0000000634B4559	RSA Laboratories	United States	KPKI	
A000000063504B4 3532D3135	RSA Laboratories	United States	PKCS-15	
A00000006357415 02D57494D	RSA Laboratories	United States	WAP-WIM	
A00000006510	JCB CO., LTD.	Japan	JCB	EMV
A0000000651010	JCB CO., LTD.	Japan	JCB J Smart Credit	EMV

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A00000006900	Société Eu-ropéenne de Monnaie Elec-tronique SEME	France	Moneo	EMV
A000000077010000 021000000000003B	Oberthur Technologies	France	Visa AEPN	EMV
A0000000790100	Activcard Europe S.A.	France	CACv2 PKI ID	
A0000000790101	Activcard Europe S.A.	France	CACv2 PKI Sign	
A0000000790102	Activcard Europe S.A.	France	CACv2 PKI Enc	
A00000007901F0	Activcard Europe S.A.	France	CACv1 PKI Iden-tity Key	
A00000007901F1	Activcard Europe S.A.	France	CACv1 PKI Di-gital Signature Key	
A00000007901F2	Activcard Europe S.A.	France	CACv1 PKI Key Mana-gement Key	
A0000000790200	Activcard Europe S.A.	France	CACv2 DoD Per-son	
A0000000790201	Activcard Europe S.A.	France	CACv2 DoD Per-sonnel	
A00000007902FB	Activcard Europe S.A.	France	CACv1 BC	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A00000007902FD	Activcard Europe S.A.	France	CACv1 BC	
A00000007902FE	Activcard Europe S.A.	France	CACv1 BC	
A0000000790300	Activcard Europe S.A.	France	CACv2 Access Control Applet	
A0000000791201	Activcard Europe S.A.	France	CAC JDM	
A0000000791202	Activcard Europe S.A.	France	CAC JDM	
A0000000871002F F49FF0589	Third Generation Partnership Project	France	Telenor USIM	USIM
A00000008810200 105C100	Buypass AS	Norway	BuyPass BIDA	BuyPass
A00000008810220 1034221	Buypass AS	Norway	BuyPass BEID	BuyPass
A00000008810220 1034321	Buypass AS	Norway	BuyPass BEID	BuyPass
A0000000960200	Sa Proton World International N.V.	Belgium	Proton World International Security Domain	GP
A000000098	Visa USA	United States	Debit Card	EMV
A0000000980840	Visa USA	United States	Visa Common Debit	
A0000000980848	Visa USA	United States	Debit Card	EMV
A0000001110101	Die Post Postfinance	Switzerland	Postcard	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000001110201	Die Post Postfinance	Switzerland	Postcard	
A0000001160300	GSA – TFCS	United States	PIV CHUID	
A0000001166010	GSA – TFCS	United States	PIV Fingerprints	
A0000001166030	GSA – TFCS	United States	PIV Facial Image	
A0000001169000	GSA – TFCS	United States	PIV Security Object	
A000000116A001	GSA – TFCS	United States	PIV Authentication Key	
A000000116DB00	GSA – TFCS	United States	CCC	
A000000118010000	Austria Card	Austria	DF Verkehr	
A000000118020000	Austria Card	Austria	DF Partner	
A000000118030000	Austria Card	Austria	DF Schülerdaten	
A000000118040000	Austria Card	Austria	DF KEP SIG	
A0000001184543	Austria Card	Austria	Digital Signature	
A000000118454E	Austria Card	Austria	Encryption Application	
A0000001211010	PBS Danmark A/S	Denmark	Dankort	EMV
A0000001320001	Java Card Forum	United States		
A0000001408001	TDS TODOS DATA SYSTEM AB	Sweden	eCode	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000001410001	Associazione Bancaria Italiana	Italy	PagoBANCO	EMV
A0000001510000	GlobalPlatform Inc.	United States	Global Platform Security Domain AID	GP
A000000151535043 41534400	GlobalPlatform Inc.	United States	CASD AID	GP
A0000001523010	Diners Club International Ltd.	United States	Discover, Pulse D Pas	EMV
A0000001524010	Diners Club International Ltd.	United States	Discover	EMV
A0000001544442	Banrisul – Banco do Estado do Rio Grande do SUL – S.A.	Banricompra Debito	Brazil	EMV
A0000001570010	Zühlke Engineering AG	Switzerland	AMEX	
A0000001570020	Zühlke Engineering AG	Switzerland	MasterCard	
A0000001570021	Zühlke Engineering AG	Switzerland	Maestro	
A0000001570022	Zühlke Engineering AG	Switzerland	Maestro	
A0000001570023	Zühlke Engineering AG	Switzerland	CASH	
A0000001570030	Zühlke Engineering AG	Switzerland	VISA	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000001570031	Zühlke Engineering AG	Switzerland	VISA	
A0000001570040	Zühlke Engineering AG	Switzerland	JCB	
A0000001570050	Zühlke Engineering AG	Switzerland	Postcard	
A0000001570051	Zühlke Engineering AG	Switzerland	Postcard	
A0000001570100	Zühlke Engineering AG	Switzerland	MCard	
A0000001570104	Zühlke Engineering AG	Switzerland	MyOne	
A0000001570109	Zühlke Engineering AG	Switzerland	Mediamarkt Card	
A000000157010A	Zühlke Engineering AG	Switzerland	Gift Card	
A000000157010B	Zühlke Engineering AG	Switzerland	Bonuscard	
A000000157010C	Zühlke Engineering AG	Switzerland	WIRCard	
A000000157010D	Zühlke Engineering AG	Switzerland	Power Card	
A0000001574443	Zühlke Engineering AG	Switzerland	DINERS CLUB	
A0000001574444	Zühlke Engineering AG	Switzerland	Supercard Plus	
A000000167413000 FF	IBM	Germany	JCOP Identify Applet	JCOP

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A000000167413001	IBM	Germany	FIPS 140-2	
A000000167455349 474E	IBM	Germany	eGK- ESIGN	
A000000172950001	Financial Infor- mation Service Co. Ltd.	Taiwan	BAROC Financial Appli- cation Taiwan	EMV
A000000177504B43 532D3135	Ministère de L'Intérieur	Belgium	BelPIC	
A0000001850002	Post Office Limited	United Kingdom	UK Post Office Account card	EMV
A0000001884443	Diners Club Swit- zerland Ltd	Switzerland	DINERS CLUB	
A0000002040000	Association for Pay- ment Clearing Services	United Kingdom		
A0000002281010	Saudi Arabian Monetary Agency	Kingdom of Saudi Arabia	SPAN	EMV
A0000002282010	Saudi Arabian Monetary Agency	Kingdom of Saudi Arabia	SPAN	EMV
A00000022820101010	Saudi Arabian Monetary Agency	Kingdom of Saudi Arabia	SPAN	
A0000002471001	ISO JTC1/SC17	United Kingdom	Machine Readable Travel Doc- uments	MRTD

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000002472001	ISO JTC1/SC17	United Kingdom	Machine Readable Travel Documents	MRTD
A0000002771010	Interac Association	Canada	INTERAC	EMV
A000000306000000 00000000	PS/SC Work-group	United States	PC/SC Initial access data AID	
A000000308000010 000100	National Institute of Standards and Technology	United States	Personal Identity Verification/ ID-ONE PIV BIO	
A00000031510100528	Currence Holding/-PIN BV	The Netherlands	Currence PuC	EMV
A0000003156020	Currence Holding/-PIN BV	The Netherlands	Chipknip	EMV
A00000032301	Identity Alliance	United States	MUSCLE Applet Package	
A0000003230101	Identity Alliance	United States	MUSCLE Applet Instance	
A0000003241010	Discover Financial Services LLC	United States	Discover Zip	
A000000333010101	China Unionpay Co. Ltd	China	UnionPay Debit	
A000000333010102	China Unionpay Co. Ltd	China	UnionPay Credit	
A000000333010103	China Unionpay Co. Ltd	China	UnionPay Quasi Credit	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A000000333010106	China Unionpay Co. Ltd	China	UnionPay Electronic Cash	
A000000333010108	China Unionpay Co. Ltd	China	U.S. UnionPay Common Debit AID	
A000000337101000	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Standard	
A000000337101001	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Prepaye Online	
A000000337102000	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Classic	
A000000337102001	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Prepaye Possibile Offline	
A000000337301000	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Retrait	
A000000337601001	Groupement Interban- caire Monétique de l'UE- MOA	Senegal	Porte Monnaie Electro- nique	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000003591010	Euro Alliance of Payment Schemes s.c.r.l. – EAPS	Belgium		
A000000359101002 8001	Euro Alliance of Payment Schemes s.c.r.l. – EAPS	Belgium	Girocard EAPS	EMV
A000000359101003 80	Euro Alliance of Payment Schemes s.c.r.l. – EAPS	Belgium		
A0000003660001	Poste Italienne S.P.A	Italy	Postamat	
A0000003660002	Poste Italienne S.P.A	Italy	Postamat VISA	
A0000003710001	Verve	Nigeria	InterSwitch Verve Card	EMV
A00000038410	eftpos, Australian Payments Clearing Association Ltd	Australia	Savings	
A00000038420	eftpos, Australian Payments Clearing Association Ltd	Australia	Cheque	
A0000003964D66344D 0002	NXP Semiconductors Germany GmbH	Germany	MIFARE4M	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A00000039742544659	Microsoft Corporation	United States	Microsoft IDMP AID	
A0000003974349445F 0100	Microsoft Corporation	United States	Microsoft PNP AID	
A0000004271010	Unibanco	Brazil	Hiperchip	
A0000004320001		Russia	Universal Electronic Card	
A0000004360100	Edenred	Belgium	Ticket Restaurant	
A0000004391010	ACCEL/Ex	United States	Exchange ATM card	
A0000004540010	eTranzact	Nigeria	Etranzact Genesis Card	EMV
A0000004540011	eTranzact	Nigeria	Etranzact Genesis Card 2	EMV
A0000004762010	Google	United States	GOOGLE CONTROL-LER AID	
A0000004763030	Google	United States	GOOGLE MIFARE MANA-GER AID	
A0000004766C	Google	United States	GOOGLE PAY-MENT AID	EMV
A000000476A010	Google	United States	GSD MA-NAGER AID	GP
A000000476A110	Google	United States	GSD MA-NAGER AID	GP
A000000485	JVL Ventures, LLC	United States	Softcard SmartTap	
A0000005241010	RuPay	India	RuPay	EMV

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000005271002	Yubico	Sweden	Yubikey NEO U2F Demo applet	YKNEO
A000000527200101	Yubico	Sweden	Yubikey NEO Yu-bikey2 applet interface	YKNEO
A000000527210101	Yubico	Sweden	Yubikey NEO OATH Applet	YKNEO
A0000005591010FFF FFFFF8900000100	GSMA	United Kingdom	ISD-R Application. Used as TAR.	
A0000005591010FFF FFFFF8900000200	GSMA	United Kingdom	ECASD Application. Used as TAR.	
A0000005591010FFF FFFFF8900000D00	GSMA	United Kingdom	ISD-P Executable Load File.	
A0000005591010FFF FFFFF8900000E00	GSMA	United Kingdom	ISD-P Executable Module.	
A0000005591010FFF FFFFF8900000F00	GSMA	United Kingdom	Reserved value for the Profile's ISD-P	
A0000005591010FFF FFFFF8900001000	GSMA	United Kingdom		
A00000061700	Fidesmo	Sweden	Fidesmo javacard	
A0000006200620	Debit Network Alliance	United States	Debit Network Alliance	
A0000006260101010001	SwissPass	Switzerland	SwissPass/SwissPass Plus	
A0000006351010	Bancnet	Philippines		

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
A0000006581010	MIR	Russia	MIR Credit	
A0000006581011	MIR	Russia	MIR Credit	
A0000006582010	MIR	Russia	MIR Debit	
A0000006723010	TROY	Turkey	TROY chip credit card	EMV
A0000006723020	TROY	Turkey	TROY chip debit card	EMV
A000000743CC843 413925E20C59B0100	SALTO JustIN Mobile		SALTO JustIN Mobile	
A0000007705850	Indian Oil Cor- poration Limited	India	XTRA POWER Fleet Card Program	EMV
A0000007790000	Zimswitch	Zimbabwe		
A000000A031010	VSDC RF		VSDC RF	
A00102030400000002	OPBytes		OPBytes	
B012345678	MasterCard Internatio- nal	United States	Maestro TEST	EMV
D040000001000002	Paylife	Austria		
D040000002000002	Austria Card	Austria	RFU	
D040000003000002	Austria Card	Austria	POS	
D040000004000002	Austria Card	Austria	ATM	
D04000000B000002	Austria Card	Austria	Retail	
D04000000C000002	Austria Card	Austria	Bank Data	
D04000000D000002	Austria Card	Austria	Shopping	
D040000013000001	Austria Card	Austria	DF Schüler1	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
D04000001300001	Austria Card	Austria	DF UNI Kepler1	
D04000001300002	Austria Card	Austria	DF Schüler2	
D04000001300002	Austria Card	Austria	DF UNI Kepler2	
D04000001400001	Austria Card	Austria	DF Mensa	
D04000001500001	Austria Card	Austria	DF Ausweis	
D04000001500001	Austria Card	Austria	DF UNI Ausweis	
D0400000190001	Austria Card	Austria	EMV ATM Maestro	
D0400000190002	Austria Card	Austria	EMV POS Maestro	
D0400000190003	Austria Card	Austria	EMV ATM MasterCard	
D0400000190004	Austria Card	Austria	EMV POS MasterCard	
D0400000190010	Austria Card	Austria	Digital ID	
D250000024D465F564 954414C45		France	Carte vitale MF	
D25000002564954414C45		France	Carte vitale	
D250000044164E86C650101		France	Carte vitale ?	
D26800001	Ministry of Finance of Georgia	Georgia	Fiscal module application	
D2760000101		Germany	KVK	
D2760000102		Germany	eGK-HCA	
D276000015445535442		Germany	eGK-HBA/SMC	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
D276000005	Giesecke and De- vrient	Germany		
D276000005AA040360010410	Giesecke and De- vrient	Germany	G D App Nokia 6212	
D276000005AA0503E00401	Giesecke and De- vrient	Germany	G D App Nokia 6212	
D276000005AA0503E00501	Giesecke and De- vrient	Germany	G D App Nokia 6212	
D276000005AA0503E0050101	Giesecke and De- vrient	Germany	G D App Nokia 6212	
D276000005AB0503E0040101	Giesecke and De- vrient	Germany	G D App Nokia 6212	
D276000002		Germany	ZKA domestic	
D27600002200000001	IBM Labo- ratories	Germany	SCT LOYALTY	
D27600002200000060	IBM Labo- ratories	Germany	PKCS-11 Token	
D276000025	ZKA	Germany	Girocard	
D27600002545410100	ZKA	Germany		
D27600002545430200	ZKA	Germany	electronic cash	
D27600002545500100	ZKA	Germany	Girocard	EMV
D27600002545500200	ZKA	Germany	GeldKarte	
D27600002546530200	ZKA	Germany	Fahrschein	
D27600002547410100	ZKA	Germany	Girocard ATM	
D27600002548420200	ZKA	Germany	ZKA HBCI	
D2760000254D010200	ZKA	Germany	ZKA Markt- platz	
D2760000254E500100	ZKA	Germany	ZKA Note- pad	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
D27600002554430100	ZKA	Germany	ZKA TAN Credit	
D27600002554430200	ZKA	Germany	ZKA TAN Credit alternative AID	
D27600002554440100	ZKA	Germany	ZKA TAN Debit	
D27600002554440200	ZKA	Germany	ZKA TAN Debit alternative AID	
D2760000255A410200	ZKA	Germany	ZKA-Zusatzanwendungen	
D276000060	Wolfgang Rankl	Germany		
D27600006601	BAPT/BSI	Germany	QES	
D276000085	NXP Semiconductors / NFC Forum	Germany	NFC Forum	
D2760000850100	NXP Semiconductors / NFC Forum	Germany	NDEF Tag Application / Mifare DESFire Tag Application	
D2760000850101	NXP Semiconductors / NFC Forum	Germany	NDEF Tag Application	
D276000118	Giesecke and Devrient Java Card Telekommunikation	Germany		

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
D2760001180101	Giesecke and De- vrient Java Card Tele- communi- cation	Germany	Giesecke	
D27600012401	fsfEurope	Germany	OpenPGP Card	OpenPGP
D276000124010101FFFF000 000010000	fsfEurope	Germany	OpenPGP Card	OpenPGP
D27600012401020000000000 000010000	fsfEurope	Germany	OpenPGP Card	OpenPGP
D27600012402	fsfEurope	Germany	SmartChess	SmartChess
D2760001240200010000000 000000000	fsfEurope	Germany	SmartChess	SmartChess
D27600014401	gematik	Germany	eGK-root	
D2760001448000	gematik	Germany	eGK – el.Gesundheitskarte	
D28000000101	KVK	Germany	KVK	
D4100000011010		Republic of Korea		
D4106470004B5410000001	KAuth		KT GPIN	
D4106470004B5410000002	KAuth		KT PIMS	
D4106470004B5410000004	KAuth		KAuth	
D4106470004B5410000005	KAuth		KTicket	
D4106509900030	KCert		KCert	
D5280050218002		The Nether- lands		EMV
D578000002	Bankaxeft	Norway	Bankaxeft Norway	
D5780000021010	Bankaxeft	Norway	Bankaxeft	EMV
D7560000010101	Swiss Travel Fund	Switzerland		
D7560000300101	Migros	Switzerland	M Budget	
D8040000013010	PROSTIR	Ukraine	PROSTIR Card	
E80704007F00070302	nPA, German eID	Germany	nPA	

Continued on next page

Table C.4: Liste d'AIDs connus (source : [8]). (Continued)

AID	Vendeur	Pays	Nom	Type
E82881C11702	AlphaCard application		AlphaCard application	
E828BD080F	Carte vitale		ISO-7816-15 EF.DIR	
E828BD080FA00000016 7455349474E	Carte vitale		eGK-CIA.ESIGN	
E828BD080FD25000000 44164E86C65	Carte vitale		Carte vitale	
F0000000030001	BRADESCO	Brazilian Bank Banco Bradesco	BRADESCO	EMV
F0004B5446303031	KTF		KTF	

Bibliographie

- [1] 3GPP. Universal mobile telecommunications system. https://www.etsi.org/deliver/etsi_ts/131100_131199/131102/13.05.00_60/ts_131102v130500p.pdf. [Online ; accessed 28-August-2023].
- [2] Altera. Altera 5m160ze64c5n datasheet. <https://www.alldatasheet.com/datasheet-pdf/pdf/530671/ALTERA/5M160ZE64C5N.html>, 2011. [Online ; accessed 05-July-2023].
- [3] ARM. Arm debug interface v5 - architecture specification. <https://developer.arm.com/documentation/ihi0031/a/The-Serial-Wire-Debug-Port--SW-DP-/Introduction-to-the-DAP-Serial-Wire-Debug-Port?lang=en>, 2006. [Online ; accessed 04-July-2023].
- [4] ARM. Esp32 wroom 32d datasheet. https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf, 2023. [Online ; accessed 22-August-2023].
- [5] ARM. Esp8266ex datasheet. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf, 2023. [Online ; accessed 05-July-2023].
- [6] Riccardo Focardi Claudio Bozzaturo and Francesco Palmarini. Shaping the glitch, optimizing voltage fault injection attacks. *Transactions on Cryptographic Hardware and Embedded Systems*, 2019.
- [7] Wolfgang Denk. Das u-boot. <https://github.com/u-boot/u-boot>. [Online ; accessed 20-August-2023].
- [8] EFTlab. liste d'aids et de réponse d'apdu. <https://www.eftlab.com>. [Online ; accessed 27-August-2023].
- [9] Espressif. Documentation esp8266. <https://docs.espressif.com/projects/esptool/en/latest/esp8266/esptool/basic-commands.html>. [Online ; accessed 22-July-2023].
- [10] Espressif. esptool. <https://github.com/espressif/esptool>. [Online ; accessed 03-August-2023].
- [11] exploitee.rs. Hacking hardware with a \$10 sg card reader. [https://www.blackhat.com/docs/us-17/wednesday/us-17-Etemadieh-Hacking-Hardware-With-A-\\$10-SD-Card-Reader.pdf](https://www.blackhat.com/docs/us-17/wednesday/us-17-Etemadieh-Hacking-Hardware-With-A-$10-SD-Card-Reader.pdf), 2017. [Online ; accessed 06-July-2023].
- [12] Patrick Gueulle. *Cartes à puce et PC*. ETSF, 2008.
- [13] Red Hat. Jffs2 : The journalling flash file system, version 2. <https://www.sourceryware.org/jffs2/>. [Online ; accessed 20-August-2023].
- [14] Hex-rays. Ida pro disassembler. <https://hex-rays.com/ida-pro/ida-disassembler/>. [Online ; accessed 03-August-2023].

- [15] Carlos Moreno. Side channel analysis. <https://ece.uwaterloo.ca/~vganesh/TEACHING/S2014/ECE458/Lecture-9.pdf>. [Online ; accessed 07-August-2023].
- [16] NewAE. Chipwhisperer lite. <https://rtfm.newae.com/Capture/ChipWhisperer-Lite/>. [Online ; accessed 16-August-2023].
- [17] NSA. Ghidra disassembler. <https://ghidra-sre.org/>. [Online ; accessed 03-August-2023].
- [18] Ludovic Rousseau. Smartcard list. https://pcsc-tools.apdu.fr/smartcard_list.txt. [Online ; accessed 26-August-2023].
- [19] RT-Thread. Rt-thread operating system. <https://www.rt-thread.io/>. [Online ; accessed 20-August-2023].
- [20] Craig Smith. *The Car Hacker's Handbook - A guide for the penetration tester*. No Starch Press.
- [21] Pascal Urien. Carte à puce. https://perso.telecom-paristech.fr/uriен/cours_cartes_urien-2008.pdf. [Online ; accessed 28-August-2023].
- [22] Benjamin Vernoux. Hydrafw binary spi mode guide. <https://github.com/hydrabus/hydrafw/wiki/HydraFW-Binary-SPI-mode-guide>. [Online ; accessed 02-August-2023].
- [23] Winbond. Puya p24c512 datasheet. <https://www.puyasemi.com/uploadfiles/2018/08/201808062051155115.pdf>. [Online ; accessed 08-July-2023].
- [24] Winbond. Stc stc10f12xe datasheet. <https://datasheetspdf.com/datasheet/STC10F12XE.html>, 2011. [Online ; accessed 07-July-2023].
- [25] Winbond. Winbond w25q128fv datasheet. <https://www.alldatasheet.com/datasheet-pdf/pdf/506494/WINBOND/W25Q128FV.html>, 2012. [Online ; accessed 06-July-2023].
- [26] Jasper Van Woudenberg and Colin O'Flynn. *The Hardware Hacking Handbook*. No Starch Press, 2022.

Index

B

Bruit

- Capacitif, 13
- Conductif, 13
- Inductif, 13
- Radiatif, 13

C

Collecteur ouvert, 15

Composant

- Condensateur, 15
- Résistance, 14
- Résistance de rappel, 14
- Résistance de tirage, 14

D

Drain ouvert, *voir aussi* Collecteur ouvert

M

Mesure

- Bruit, 12

O

Ohm (loi), 8

P

Protocole

- 1-Wire, 34
 - CAN, 30
 - eMMC, 32
 - Ethernet, 37
 - FireWire, 37
 - I2C, 27
 - JTAG, 38
 - PCIe, 36
 - RS-232, 35
 - SATA, 35
 - SPI, 25
 - SWD, 39
 - UART, 23
 - USB, 36
- Pull-down (résistance), 14
- Pull-up (résistance), 14