JEAN-LOUIS Baptiste
GUILLOMON Rémi

Mai 2020

Rapport

Projet IMA Semestre 8

Réalisation de périphériques USB personnalisés

Polytech Lille

Promotion 2021

Sommaire

Introduction

Notre projet consiste en la création de périphériques USB, chacun avec fonctionnalités additionnelles. Initialement, trois périphériques étaient envisagés. Le premier est un clavier. En plus de recevoir et d'émettre les entrées et de gérer les DEL, il doit enregistrer dans une mémoire interne les touches enfoncées. Par l'intermédiaire de combinaisons de touches, il est possible de restituer l'historique, de le vider et d'activer ou désactiver cet enregistrement. Le deuxième périphérique est une clé USB permettant un stockage de masse. Après une inactivité prolongée, elle lance le téléchargement d'un logiciel de surveillance. Enfin le dernier est une souris contrôlant le mouvement et les actions du curseur. Un mode spécial, activable à distance, déclenche un déplacement et des clics aléatoires. Ce périphérique a été laissé de côté lors du premier semestre.

Lors du semestre 6, nous avons eu l'occasion de découvrir le protocole USB. Nous avons aussi réalisé des maquettes Arduino pour nous donner une idée de la faisabilité des fonctions auxiliaires et de la structure des codes à implémenter dans les microcontrôleurs.

Le semestre 7 a été marqué par un dégrossissage des principaux composants qui a abouti sur le choix du microcontrôleur et du convertisseur de tension - communs aux deux PCB - et d'une mémoire pour le clavier. En outre, après avoir sélectionné un clavier sur lequel travailler, nous avons déterminé son mappage en vue du traitement logiciel des touches enfoncées.

Les objectifs du semestre 8 étaient de compléter la liste du matériel nécessaire pour préparer la commande, de réaliser les schematics et PCB de chaque périphérique, les assembler et souder et de développer les différents aspects logiciels incluant la méthode de programmation des microcontrôleur, la gestion de l'USB avec la librairie LUFA et les programmes des périphériques.

I. Mémoires et interfaces série

1) Les mémoires

En début de semestre, nous avons classifié les différentes mémoires disponibles sur le marché pour déterminer celle dont nous avions besoin. Elles se répartissent selon trois familles : les ROM, les RAM et les Flash.

ROM			RAM		Flash	
PROM	UVPROM	EEPROM	SRAM	DRAM	NAND	NOR

Les PROM ne sont programmables qu'une unique fois. Les UVPROM peuvent être effacées à l'aide d'un appareil ultraviolet spécifique générant une forte exposition. Les EEPROM sont facilement effaçables électriquement.

Les RAM sont séparées en deux catégories, elles mêmes divisées en plusieurs types :

RAM				
SRAM		DRAM		
MRAM	DPRAM	SDRAM	DDR SDRAM	VRAM

Les mémoires vives statiques SRAM consomment beaucoup et sont coûteuses et consomment beaucoup mais elles sont rapides et sans besoin de rafraîchissement de l'information. La Magnetic RAM (MRAM) est non volatile tandis que la Dual Ported RAM (DPRAM) a un accès multiple quasi instantané.

Les mémoires vives dynamiques DRAM conservent les données quelques millisecondes seulement, d'où la nécessité d'une opération de rafraîchissement. Cette dernière a pour fonction de régulièrement lire l'information de chaque cellule et de la réécrire. SDRAM signifie Synchronous DRAM et DDR SDRAM, Double Data Rate SDRAM. Les Video RAM (VRAM) ont pour rôle de construire l'image vidéo.

Les mémoire Flash sont des mémoires rémanentes. Leur vitesse est relativement élevée et elles possèdent une bonne durée de vie. Il y en a deux types : les NAND et les NOR, dont un comparatif de Microchip [1] est donné en **figure 1**. Leur différence réside dans la connexion des cellules mémoires individuelles.

Le critère XIP symbolise la possibilité d'exécution du code stocké dans la mémoire Flash sans devoir le copier dans une mémoire RAM. De l'espace est donc libéré dans cette dernière pour d'autres utilisations.

	NAND	NOR
Main Application	File Storage	Code execution
Storage capacity	High	Low
XIP Capabilities	No	Yes
Cost per bit	Better	
Active Power	Better	
Standby Power		Better
Write Speed	Good	
Read Speed		Good

Fig. 1 - Comparatif des mémoires Flash NAND et NOR

Les interfaces de communication “Common interfaces (CI)” avec ces mémoires sont diverses selon le modèle. La Serial Digitale Interface (SDI) a pour support un câble BNC ou la fibre optique. SPI signifie Serial Peripheral Interface. La Serial Quad Lane propose un fonctionnement avec une, deux ou quatre lignes, respectivement la “single lane interface” équivalente à la SPI, la “dual lane” et “quad lane interface”.

Il existe en outre des normes régissant les Flash telles que ONFI (Open NAND Flash Interface) qui est une norme de communication pour les Flash NAND et CFI (Common Flash memory Interface) qui assure l’interchangeabilité de ces mémoires. Nous nous nous attarderons pas sur les normes.

2) Les interfaces série

Il y a trois liaisons pour la transmission série. La liaison point-à-point, et les liaisons multipoints bas et haut débit. La première est constituée des interfaces AES/EBU (audio numérique), CoaxPress, de la fibre optique, RS-232 et RS-422, Serial ATA (SATA), Serial Digital Interface (SDI, vidéo numérique) et Universal Asynchronous Receiver Transmitter (UART). La deuxième comporte les interfaces Control Area Network, I2C et RS485. Il y a dans la dernière les interfaces ARINC 818 (vidéo par fibre optique), Serial Attached SCSI (interface pour disques durs), SPI (pour les bus de données synchrone), Ethernet, la Fibre Channel (paire torsadée, câble coaxial ou fibre optique), Firewire (par Apple, obsolète), Bus InfiniBand (obsolète), Peripherical Component Interconnect (PCI, connexion des cartes d'extension) et PCI Express.

Plus spécifiquement, l'interface SPI a comme avantages d'être supportée par un grand nombre de microcontrôleurs et de communiquer en Full Duplex (les deux sens simultanément). Son interface matérielle est simple et elle est flexible vis-à-vis du nombre de bits à transmettre ainsi que du protocole en lui-même. Elle partage un bus commun entre les périphériques pour l'horloge et les lignes de données MOSI (Master Out Slave IN) et MISO (Master In Slave Out). Les esclaves utilisent alors l'horloge du maître - ils n'ont pas d'oscillateur propre - et il n'y a pas de collision de données.

Les inconvénients de la SPI sont qu'elle comporte plus de broches que l'I2C (SDA, SCL et la masse) ou une UART (une broche) - toutes deux non compatibles avec des Flash - et que les distances de bus doivent être très inférieures au mètre - ce qui est notre cas. De plus, le protocole est sans acquittement, c'est-à-dire sans accusé de réception, et l'interface utilise un Slave Select car aucun adressage n'est possible.

II. Conception des schematics

Nous commencerons par parler des composants communs aux deux périphériques, puis nous évoquerons ceux spécifiques à la clé et nous finirons par ceux du clavier. Les schematics complets des deux périphériques sont proposés en **annexe A et B**.

1) Microcontrôleur

Le semestre dernier, nous avions choisi le microcontrôleur ATXMega16A4U pour le nombre de broches GPIO (General Purpose I/O) qu'il propose (34). Comme c'est le cœur de nos périphériques, nous construisons nos schematics autour de celui-ci. Pour notre projet, il est alimenté en 3,3V. Pour chacun des quatre couples de broche VCC-GND, on place un condensateur de découplage, de valeur typique 100nF.

2) Convertisseur 5V vers 3,3V

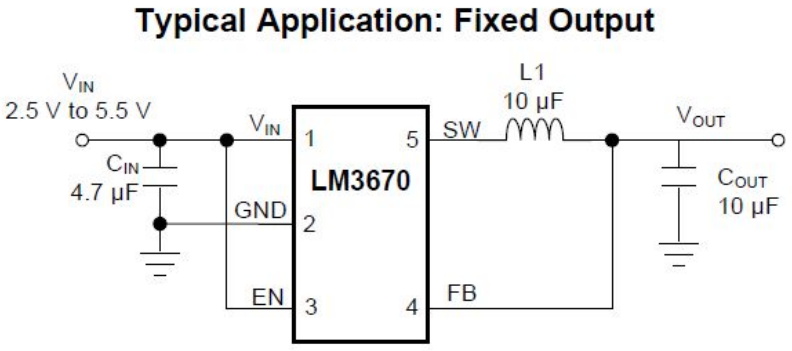


Fig. 2 - Implémentation du convertisseur de tension

Le convertisseur choisi est le LM3670MF-3.3/NOPB. Il accepte une tension d'entrée de 2,5V à 5,5V et ce modèle délivre une tension de sortie fixe de 3,3V. Il permet ici d'adapter la tension en provenance du bus USB pour alimenter le microcontrôleur. Les composants périphériques sont dictés par la documentation technique (voir **figure 2**).

3) Universal Serial Bus

Cet aspect n'est pas des moindres car il comporte deux fonctions qui sont la source d'énergie de l'ensemble des composants de la carte et la communication entre le microcontrôleur et le PC.

a) Décharges électrostatiques

Quand deux corps ayant un potentiel de masse différent entrent en contact, il peut se produire une décharge électrostatique. Cela peut endommager les appareils et le risque est plus grand quand la connexion a lieu "à chaud", c'est-à-dire lorsqu'au moins un appareil est en fonctionnement. Pour y remédier, il est envisageable de placer un "ESD suppressor" (suppresseur de décharges électrostatiques). Cela consiste en plusieurs diodes spécifiques à cette tâche placées entre les pistes d'alimentation, de données et de masse.

Dans cette optique, nous avons choisi le composant USBLC6-2SC6, que nous câblons comme en **figure 3**.

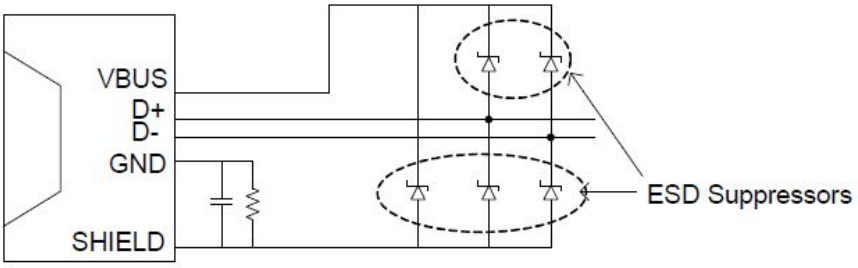


Fig. 3 - Protection typique contre les décharges électrostatiques

M. Boé suggère également des résistances de 22Ω sur les lignes de données.

b) Cas de l'embase USB

Pour le PCB de la clé, nous devons prévoir l'embase USB pour le branchement direct au PC. Celle-ci possède un shield que nous devons connecter à la masse de la carte. Les conseils sur internet sont variés et parfois en contradiction. Nous trouvons quatre solutions pour relier ces deux potentiels : directement par une piste, par l'intermédiaire d'un condensateur et éventuellement d'une résistance en parallèle, via un

"ferrite bead", ou ne pas les connecter du tout. Les considérations AVR-Xmega préconisent un filtre RC avec C = 4,7nF et R = 1MΩ, selon la configuration de la **figure 3**.

c) Lignes de données USB

Comme l'alimentation du bus est en 5V et celle du contrôleur Vcc = 3,3V, nous nous sommes interrogés sur la valeur de la tension des lignes différentielles D+ et D- portant l'information. Nous lisons dans la documentation au chapitre des caractéristiques électriques des entrées et sorties que la tension maximale d'entrée VIH est Vcc + 0,3 V soit 3,6V. Nous supposons d'abord qu'il nous faut adapter cette tension et cherchons des solutions tout en continuant à nous documenter. Puis nous lisons dans les recommandations hardware XMega que la paire différentielle USB opère typiquement à 3.3V. Nous en concluons qu'il n'est pas nécessaire de prévoir d'autre protection sur les lignes que les éventuelles résistances de 22Ω citées précédemment. Nous trouvons aussi des projets sur Internet où la connexion est directe **[2] et [3]**.

4) Quartz

Afin d'assurer une fréquence stable dans le cas où celle de l'horloge du microcontrôleur viendrait à faire défaut, et notamment pour la communication USB, nous installons une quartz. La fréquence maximale acceptée par le contrôleur est de 16MHz. Celui que nous avons sélectionné est le *TSX-3225 16.0000MF18X-AC0*.

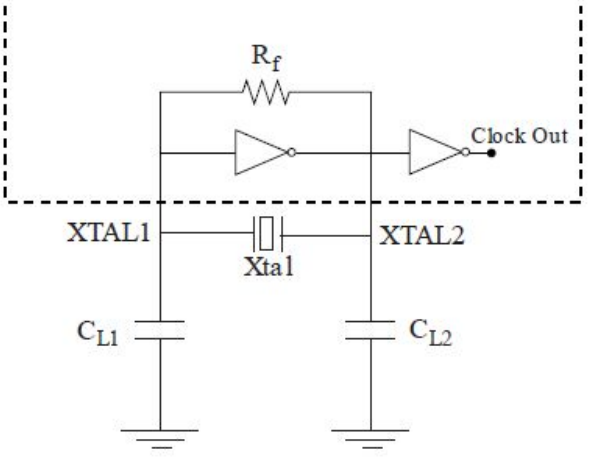


Fig. 4 - Circuit d'oscillation pour des quartz et résonateurs céramiques de fréquence supérieure à 400kHz (considérations HW AVR)

Les quartz doivent être accompagnés d'un couple de condensateurs dont les valeurs dépendent de la capacité de charge (load capacity C_{LOAD}) du quartz, donnée par le fabricant. La nôtre est de 9nF. En supposant un agencement symétrique, on peut poser $C_{L1}=C_{L2}=C$ où C peut alors être calculé par la relation $C = 2C_{LOAD} - C_{STRAY}$. La capacité

C_{STRAY} représente en quelques sortes la celle du circuit (pistes, broches du microcontrôleur, etc.). Elle est estimée entre 5pF et 10pF. On prendra 7,5pF. On en déduit la valeur de C, arrondie à la valeur normalisée supérieure : 12pF.

On nous recommande également une résistance de 1MΩ entre les deux lignes du quartz.

5) Diodes électroluminescentes

Pour chacun des PCB, nous utilisons des DEL. En plus de leur fonction habituelle comme par exemple communiquer l'information de verrouillage du pavé numérique pour le clavier ou le fonctionnement de la clé lorsqu'elle est branchée, elles pourront nous aider pendant les phases de débogage.

La valeur de courant direct la plus commune chez les fabricants est 20mA. En considérant que la valeur maximale d'alimentation des broches d'entrées/sorties est Vcc=3,3V, nous déduisons une valeur normalisée de résistance de 150Ω.

6) Interface de programmation et débogage (PDI)

Comme son nom l'indique, la PDI est une interface pour programmer les microcontrôleurs. Elle a été introduite avec la famille XMega et tient en deux lignes de communication. Le header standard est donné en **figure 5**. La broche DATA est reliée à la broche PDI_DATA du microcontrôleur et la broche CLK à PDI_CLK (ou /RESET). Les broches 3 et 4 ne sont pas connectées.

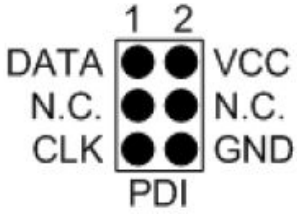


Fig. 6 - Header PDI standard

7) Mémoire SD et embase (spécifique clé USB)

Nous avons pris comme mémoire pour le stockage dans la clé USB une carte microSD de capacité 4Go dont la référence est S404APY5Q-U1000-3. L'embase commandée pour l'accueillir est la 47219-2001-001. La communication avec le microcontrôleur peut s'établir selon les modes SPI ou SD. Le mode SPI requiert moins de broches connectées et diffère du mode SD selon les points suivants.

- la carte sélectionnée répond toujours à la commande,
- quand elle rencontre un problème de récupération de données, elle répond avec une réponse d'erreur (qui remplace le bloc de données attendu) alors que le bus en mode SD déclencherait un time out.

Le mode SPI serait d'après nos recherches plus simple à mettre en œuvre que le mode SD, et il est plus documenté. C'est pourquoi nous le choisissons. Outre l'alimentation, ce mode requiert 4 pistes : deux pour l'échange de données (MISO et MOSI), une pour l'horloge et une dernière pour le slave select (CD/DAT3 en pin 2) car l'interface SPI autorise plusieurs esclaves pour un même hôte. Nous ajoutons enfin le condensateur de découplage et relions les pattes de fixation au potentiel GND.

8) Décodeur (spécifique clavier)

Bien que le choix du microcontrôleur était porté sur le nombre de GPIO, la décision d'ajouter un quartz nous contraint finalement à utiliser un décodeur parce que le quartz nécessite les deux broches PR0 et PR1, aussi disponibles en tant que GPIO. Or ces broches étaient déjà réservées pour une partie du mappage du clavier qui ne peut être placée ailleurs par manque de broche. Un décodeur 3 vers 8 nous permet donc de libérer des broches, mais cela ajoutera un traitement logiciel pour le mappage.

Pour une référence de décodeur 74HC138, la signification est la suivante.
-74 indique qu'il s'agit d'un composant CMS,
-H désigne qu'il s'agit d'un composant haute fréquence,
-C indique qu'il s'agit d'un composant dont le brochage est compatible avec les technologies TTL,
-138 est le numéro de fonction : 138 pour décodeur à l'état haut au repos et 238 pour l'état bas au repos.

Parmi les lettres que l'on peut trouver avec H et C, il y a :
- L ou LV pour low voltage : indique que le composant est optimisé pour fonctionner à des tensions inférieures à 3.6V,
- A pour advanced frequency (des fréquences plus hautes que H),
- et T indique que les niveaux (états haut et bas) sont compatible avec des composants TTL.

Les lettres trouvées en préfixe de ce code se réfèrent au fabricant.

Comme nous souhaitons un décodeur 3 vers 8 et un 1 logique pour une piste active, le numéro de fonction voulu est 238. Le décodeur commandé est un CD74AC238M96.

Il n'y a pas de spécificité pour le schematics, si ce n'est une classique capacité de découplage.

9) Mémoire SPI (spécifique clavier)

Nous avons choisi une mémoire avec une interface SPI pour le faible nombre de broche requis. La référence est MX25R6435FM2IL0 et sa capacité est de 64 Mbits (8 Mo).

La taille moyenne d'un mot est de 5 caractères. On estime pouvoir coder chaque touche sur un octet. L'utilisateur moyen frappe 33 mots par minutes dans un rythme de copie de texte. Si on considère qu'il écrit quotidiennement pendant 10h, cela reviendrait à enregistrer l'équivalent de 67 jours en comprenant les espaces. Nous devons diviser cette durée car nous enregistrons une même touche à la fois sur le front montant et le front descendant. La durée est alors de 33 jours. En pratique, ce rythme n'est pas soutenu sur une durée aussi longue. C'est pourquoi la taille de la mémoire est jugée suffisante.

III. Considérations HW des PCB

Nous spécifions dans ce chapitre les principales considérations Hardware rencontrées. Les PCB de la clé et du clavier sont disponibles en **annexe F et G**. Concernant celui du clavier, il nous est imposé des contraintes géométriques car il doit remplacer la carte d'origine. Il n'est cependant pas terminé. En effet, lors du début du confinement, Rémi - alors seul membre possédant Altium - s'est consacré à celui de la clé USB car il était plus abordable pour lui. Il n'a pas eu l'occasion de le poursuivre. Les corrections à apporter sont la prise en compte des recommandations hardware pour l'USB, le convertisseur et le quartz, l'ajout de plans de masse supplémentaires, le déplacement des vias se situant sous le microcontrôleur. Nous pouvons envisager d'omettre la protection ESD car la carte sera située sous la coque du clavier. On n'a donc aucune raison de la manipuler.

1) Microcontrôleur

Nous pouvons lire sur les recommandations Hardware AVR qu'il est important de les placer au plus près du contrôleur afin de diminuer la taille de la boucle de courant. Les mailles par lesquelles circulent cette boucle ne doivent pas être en contact direct au plan d'alimentation et au plan de masse afin d'éviter la propagation de beaucoup de bruit.

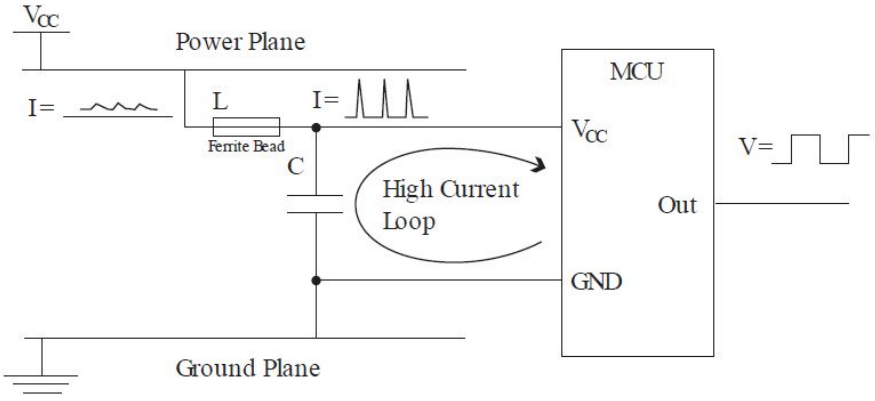


Fig. 7 - Découplage (considérations AVR)

2) Convertisseur 3,3V

Le layout du convertisseur est un point critique pour ne pas pénaliser ses performances. La documentation technique détaille les aspects à prendre en compte. Les inductances et condensateurs doivent être proches et les pistes courtes. Il faut connecter les broches GND du convertisseur et des condensateurs avec des plans de masse et connecter les plans de masse top et bottom avec plusieurs vias. Les pistes entre les composants d'alimentation doivent être épaisses. Le routage des pistes sensibles au bruit comme le "feedback path" doit être éloigné des sources de bruit. Le layout recommandé est proposé en **annexe C**.

3) USB

La communication USB est également délicate. Il est recommandé de faire un routage par paire différentielle pour les pistes D+ et D-. En effet, les pistes doivent être routées avec les mêmes caractéristiques (longueur, nombre de vias) et les signaux doivent être aussi parallèles que possibles avec un minimum d'angle et de via. La protection contre les décharges électrostatiques doit en outre être au plus proche du connecteur USB. Un layout USB typique et de l'USB suppressor sont proposés en **annexe D et E**.

4) Quartz

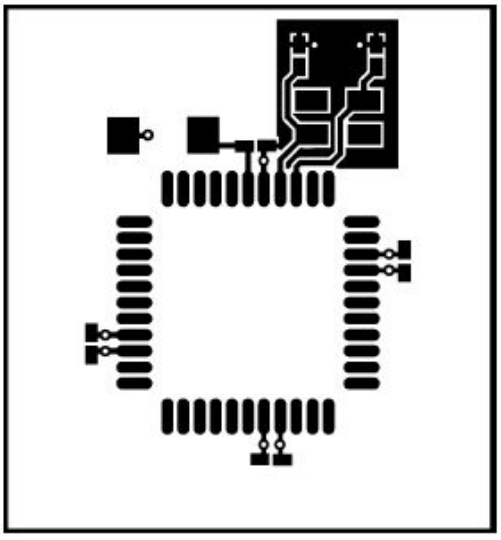


Fig. 8 - Layout du quartz

Le quartz apporte aussi sa dose de complexité. Le quartz doit être au plus proche du microcontrôleur, les différentes pistes, les plus courtes possibles et les condensateurs après le quartz. Nous pouvons difficilement respecter ce layout dans la mesure où l'épaisseur des pistes de 0,3mm ne nous permet pas de passer sous le quartz et la résistance de 1MΩ coupe le plan de masse en deux.

IV. Programmation

L'ATXMega16A4U peut être programmé de 2 façons : en utilisant l'interface physique PDI accompagné par le protocole du même nom, ou en utilisant l'interface physique USB avec DFU Boot loader. Dans la mesure où à la livraison du microcontrôleur, la programmation avec DFU n'est pas forcément activée, nous n'aborderons ici que la programmation PDI, qui est systématiquement utilisable, car non désactivable quelque soit la configuration dans laquelle le microcontrôleur est livré.

1) Description de la couche physique PDI

La programmation PDI se fait sur une interface physique spécifique composée de deux fils, PDI_DATA pour une communication série bidirectionnelle, et PDI_CLK pour l'horloge afin de synchroniser la communication. Pour démarrer la communication PDI, il faut maintenir la broche PDI_DATA à "1" pendant une période strictement supérieure à la durée nécessaire pour déclencher un reset externe (pour l'ATXMEGA 16A4U typique 95ns, maximum 1µs). Cela désactive le reset externe, ce qui permet d'utiliser la broche PDI_CLK/RESET pour sa fonction PDI. Attention, la communication PDI sera automatiquement désactivée après 100µs d'inactivité sur la broche PDI_CLK, y compris avant qu'une commande ait été envoyée. Ce délai de 100µs implique donc que la fréquence circulant sur PDI_CLK soit strictement supérieure à 10kHz.

2) Description de la couche logicielle PDI

Le protocole repose sur des trames de 12bits. Il est existe trois types de trames DATA, IDLE et BREAK. la trame DATA est composé d'un bit de démarrage (0), 8 bits de données, un bit de parité pair, 2 bits fin (11).

Les trames IDLE et BREAK ne respectent pas exactement cette structure. BREAK est une trame composée d'un minimum de douze "0". Dans son usage, cette trame ne peut être envoyée que par le programmeur, et est attendue par le microcontrôleur en cas de collision. IDLE est est une trame composée d'un minimum de douze "1", mais dans son usage, un ou plusieurs bits "1" servent plus de temporisation entre trames pour s'assurer que le contrôleur PDI ait le temps de faire son travail. De plus, suite à certaines commandes qui supposent une réponse de la part du contrôleur PDI, il est nécessaire d'observer un temps de sécurité (GuardTime). Ce temps de sécurité configurable doit donc être rempli par entre 2 et 128 (défaut) "1".

Les trames DATA peuvent être classées en trois catégories : les instructions, les adresses, et les données qui ont la particularité de pouvoir circuler dans un sens ou dans l'autre. Il existe huit instructions dans le protocole PDI (voir **Annexe Instruction PDI**) :

- REPEAT

trame complète: 1° octet suivi de size B octets indiquant le nombre de répétitions.

Cette instruction permet de répéter un nombre (donné par les octets suivant le premier) de fois l'instruction qui la succédera. Attention l'instruction sera répétée avec les mêmes opérandes.

- KEY

trame complète: 1°octet suivi de 8 octets contenant la clé (0x1289AB45CDD888FF) pour déverrouiller l'accès au contrôleur NVM (voir ci après).

- LD/ST : indirect load/store

trame complète: 1° octet suivi (d'un retour -pour LD-) de size A octets

- LDS/STS : direct load/store

trame complète: 1°octet suivi de size A octets pour l'adresse à lire ou écrire, suivi (d'un retour -pour LDS-) de size B octets.

- LDCS/STCS

trame complète: 1° octet suivi (d'un retour -pour LDCS-) de size 1 octet.
exemple :
- LDCS 0x00 permet de savoir si l'accès au contrôleur NVM est déverrouillé.
-STCS 0x01 suivi de 0x59 permet de faire rentrer et maintenir le micro-contrôleur en RESET, n'importe quelle autre valeur le sortirait de l'état de RESET.

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
+0x00	STATUS	–	–	–	–	–	–	NVMEN	–
+0x01	RESET	RESET[7:0]							
+0x02	CTRL	–	–	–	–	–	GUARDTIME[2:0]		
+0x03	Reserved	–	–	–	–	–	–	–	–

3) Programmation mémoire

La Programmation PDI repose sur la programmation mémoire qui est gérée par le contrôleur NVM (mémoire non volatile). La programmation mémoire peut également être utilisée par le micro-contrôleur pour s'auto-programmer.

a) De PDI à NVM

La démarche pour pouvoir déverrouiller le contrôleur NVM depuis le PDI est assez simple. Il suffit de maintenir le micro-contrôleur en RESET, de déverrouiller le contrôleur NVM et d'attendre qu'il soit déverrouillé (LDCS 0x00 pour avoir la réponse). Ensuite, pour lui indiquer quelle commande utiliser, il faut enregistrer la commande NVM dans le registre CMD, puis la déclencher avec le déclencheur associé à la commande.

b) La programmation NVM

Pour commencer, on peut effacer la mémoire avec chip erase, ou seulement certaines parties avec les “erase” adapté. La programmation NVM repose sur l'utilisation de buffers. C'est-à-dire qu’au lieu d'écrire directement dans la mémoire, il faut écrire le buffer associé à la zone mémoire voulue (instruction load page buffer), puis envoyer écrire le buffer dans la mémoire (instruction write page). Pour la lecture de la mémoire, il suffit d'utiliser l'instruction read correspondant à la zone mémoire voulue. Enfin pour confirmer qu’il n’y a pas eu d’erreur, nous pouvons demander au micro-contrôleur d’exécuter un CRC sur sa mémoire flash.

Parmi les zones mémoire auxquelles on peut accéder, on trouve l'EEPROM, les Fuses, les lock bits, les production et user row, et enfin la flash divisée entre la section boot et la section application. Cette dernière partie est séparée en une partie principale, qui sert à enregistrer du code, et la section application table, qui permet d'enregistrer des données essentielles au programme.

Conclusion

Parmi les objectifs du semestre, le choix des composants pour la clé USB et le clavier a été précisé. Nous avons pu réaliser les schematics de la clé USB et du clavier. Concernant les PCB, celui de la clé USB a été terminé tandis qu'il reste encore des correctifs à apporter à celui du clavier. Nous n'avons par conséquent pas eu l'occasion de nous attaquer à l'assemblage et au soudage. Finalement, Nous avons découvert le fonctionnement de la programmation avec PDI.

Nous n'avons pas su mener à bout ce projet dans les temps et la partie logicielle nous réservait encore des embûches et défis. Cela peut s'expliquer par une gestion de projet laborieuse, combiné avec des erreurs faites au semestre 6 sur l'évaluation du travail à effectuer, et par le temps qu'on a mit pour s'approprier chacun des aspects (choix des composants, conception des PCB).

Nous avons malgré tout appris à mieux appréhender le vaste domaine de l'électronique et acquis des méthodes de travail s'appliquant de la détermination du matériel à la conception des cartes, qui pourront nous aider pour de futurs projets. Nous avons de plus entrevu celui de la programmation des microcontrôleurs et de la gestion de l'USB.

Bibliographie

[1] Microchip, Getting started with Flash memories,
https://www.microchip.com/design-centers/memory/serial-parallel-flash/getting-started

[2] AVRfreaks, *Xmega A4U Dragon PDI programming issue*, projet utilisant un ATXmega16A4U avec schematics
https://www.avrfreaks.net/forum/xmega-a4u-dragon-pdi-programming-issue

[3] Gabotronics, *Xprotolab Xmega*, carte commercialisée à base d'ATXMega32A4U
http://www.gabotronics.com/development-boards/xmega-xprotolab.htm

guide de l'avr dragon
http://ww1.microchip.com/downloads/en/devicedoc/atmel-42723-avr-dragon_userguide.pdf

Quelques documentations techniques

Datasheet des XMega A4U :
http://ww1.microchip.com/downloads/en/devicedoc/atmel-8387-8-and16-bit-avr-microcontroller-xmega-a4u_datasheet.pdf

Manuel des XMEGA AU :
http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8331-8-and-16-bit-AVR-Microcontroller-XMEGA-AU_Manual.pdf

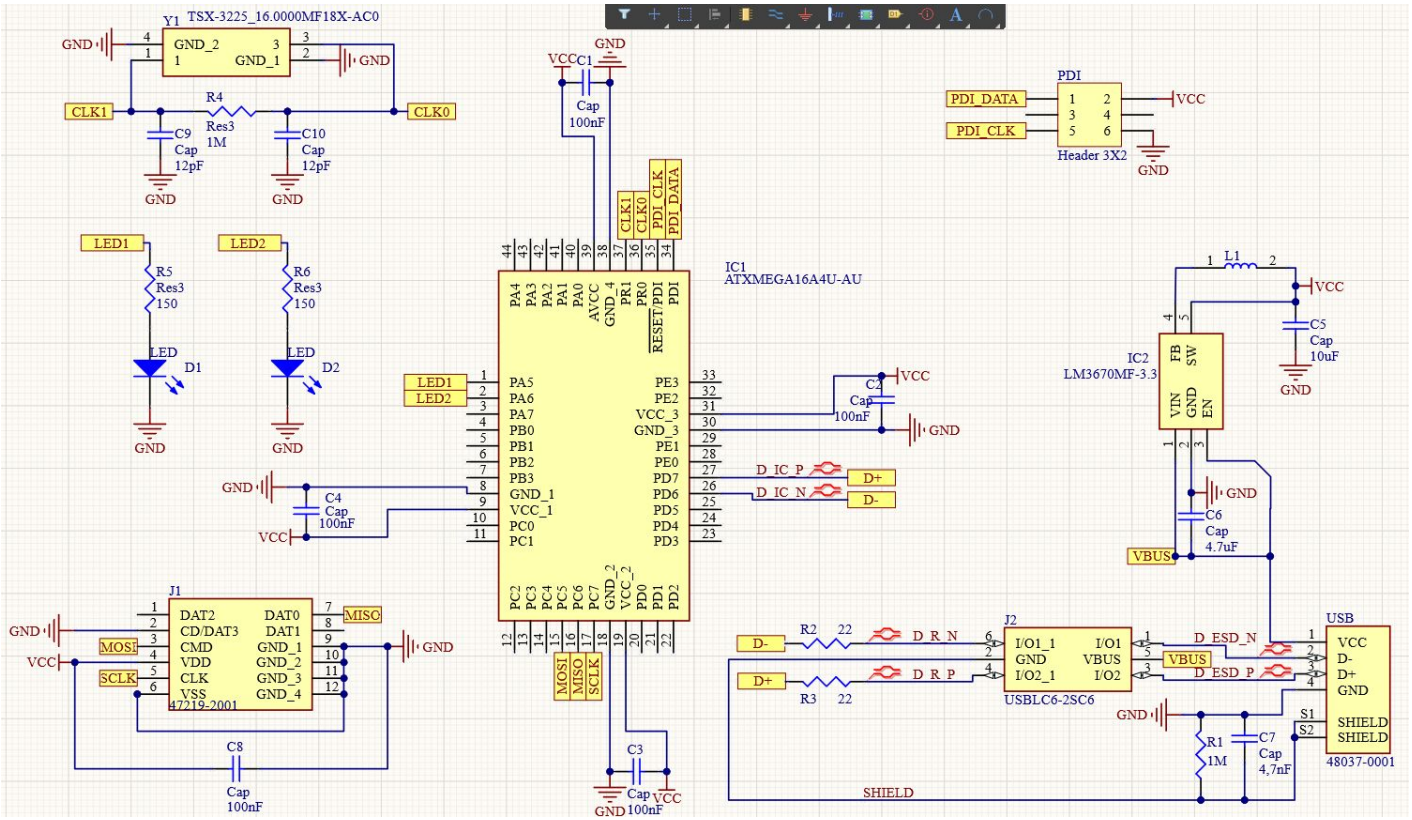
Considérations Hardware AVR et XMega :
http://ww1.microchip.com/downloads/en/appnotes/atmel-2521-avr-hardware-design-considerations_applicationnote_avr042.pdf
http://ww1.microchip.com/downloads/en/AppNotes/doc8388.pdf

Convertisseur : LM3670MF-3.3/NOPB
http://www.ti.com/lit/ds/symlink/lm3670.pdf

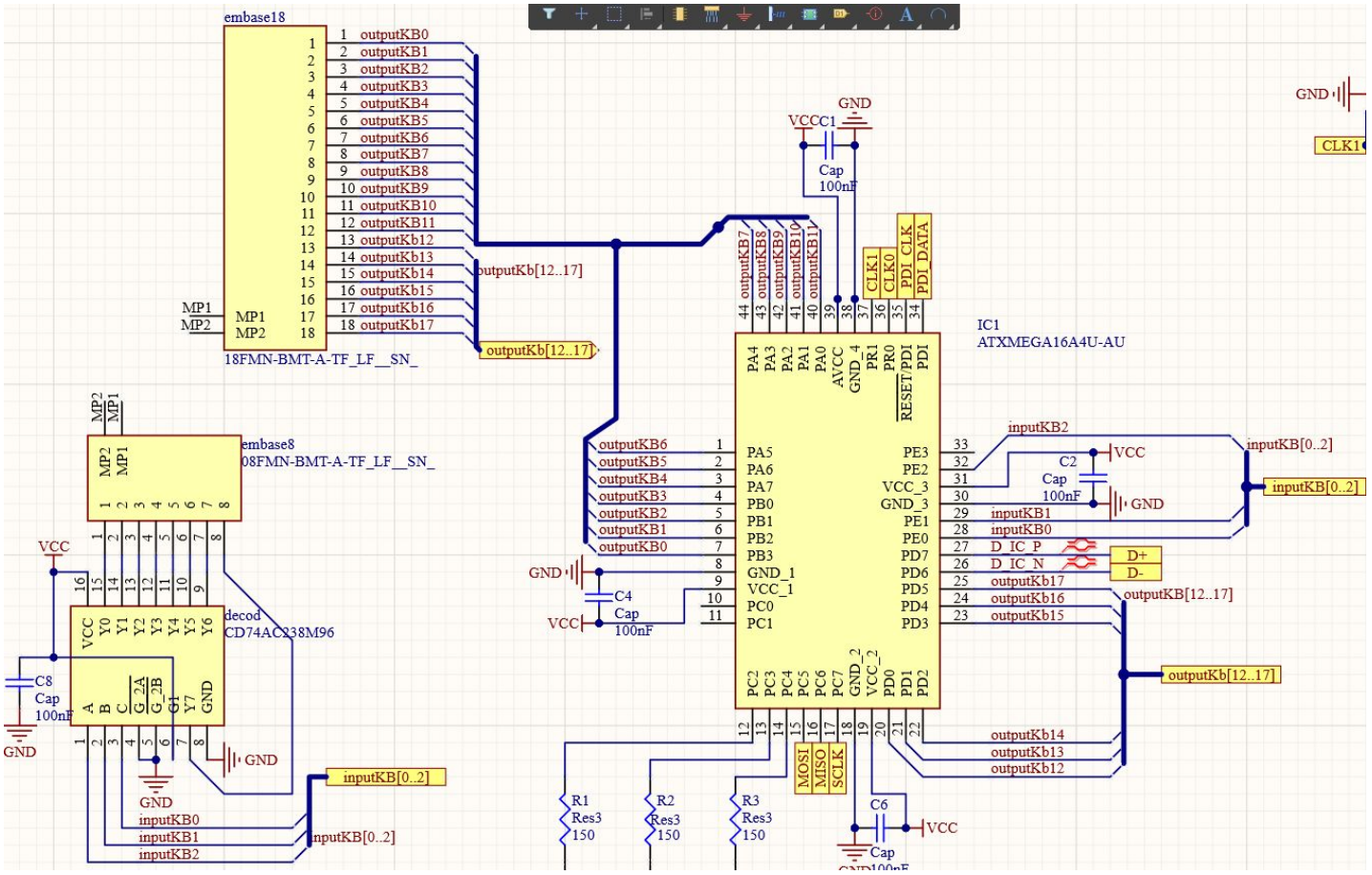
ESD Suppressor : USBLC6-2SC6
https://docs.rs-online.com/c890/0900766b807bd47e.pdf

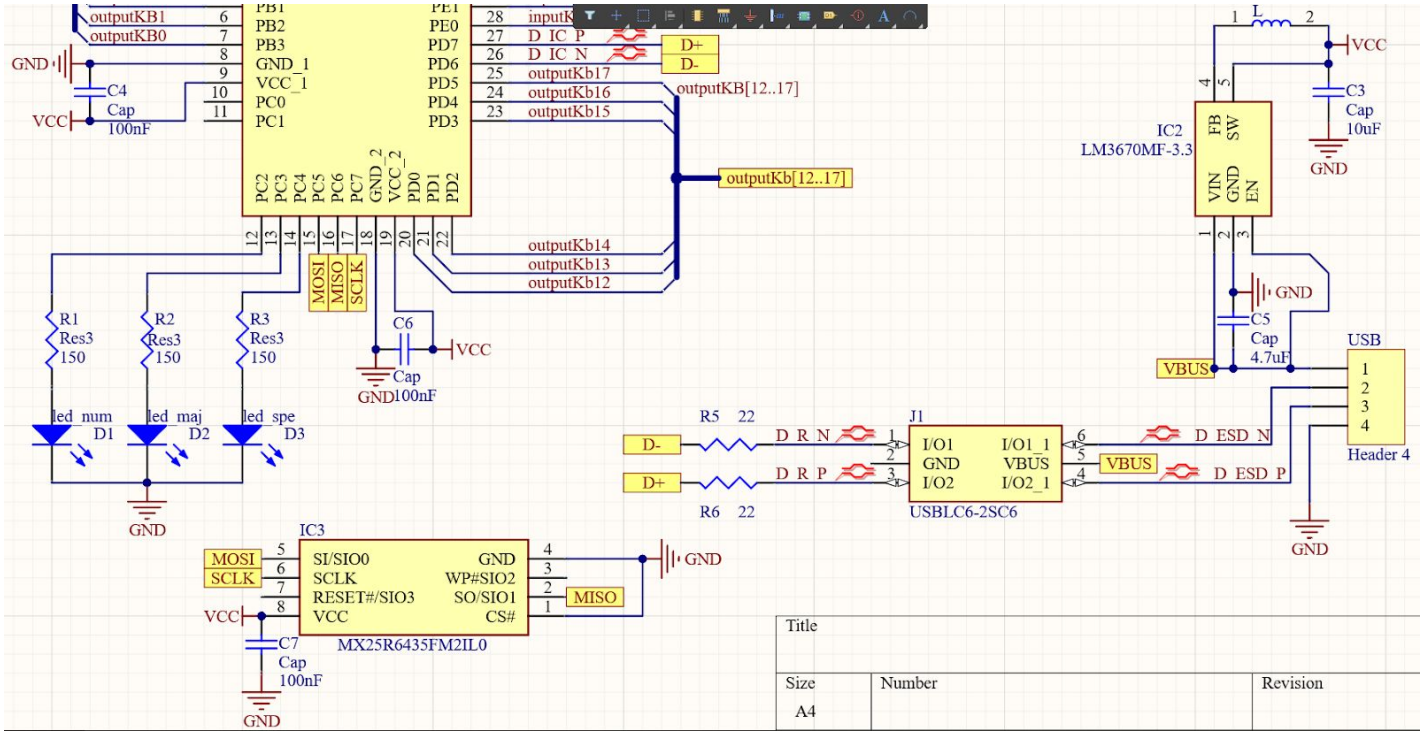
Annexes

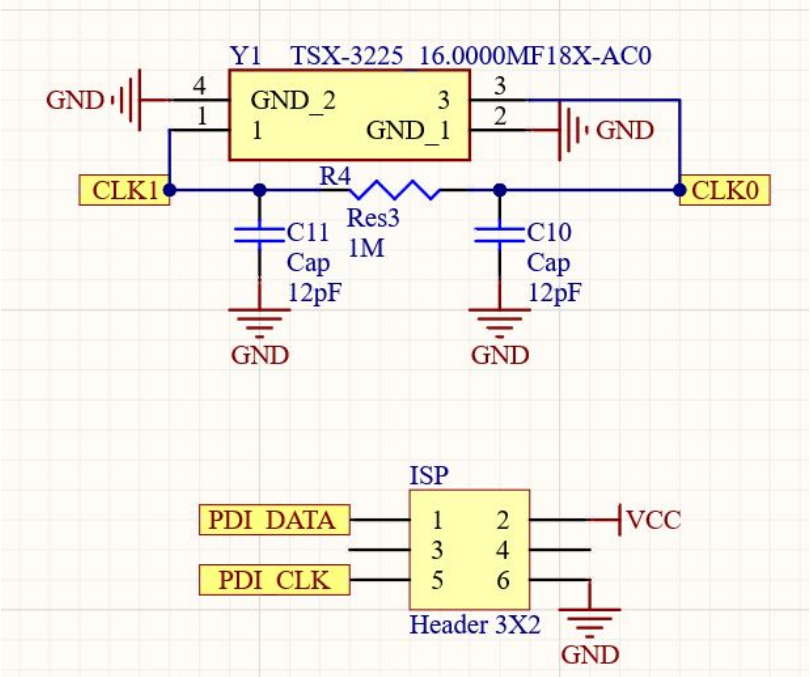
Annexe A. Schematics de la clé USB



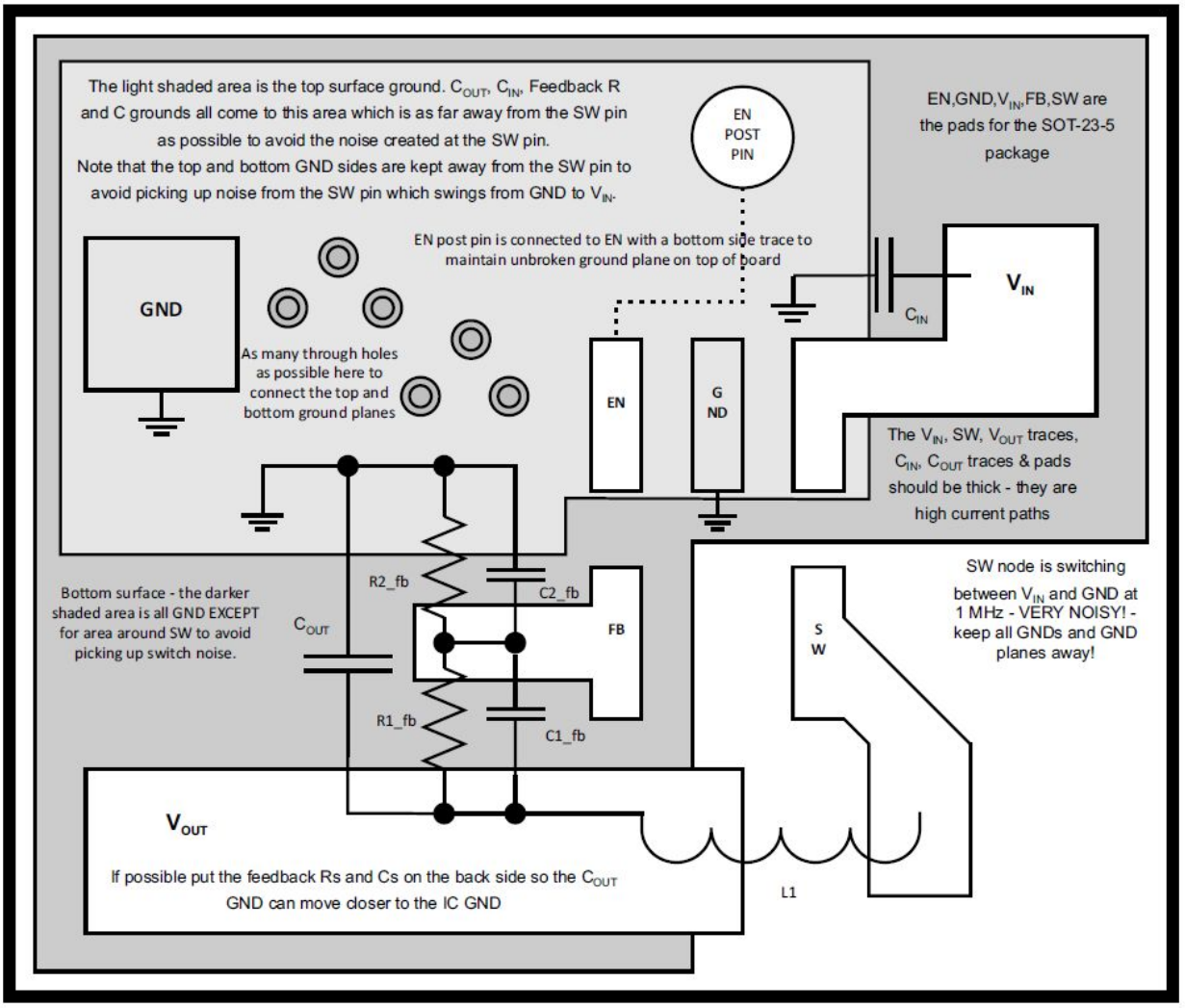
Annexe B. Schematics du clavier





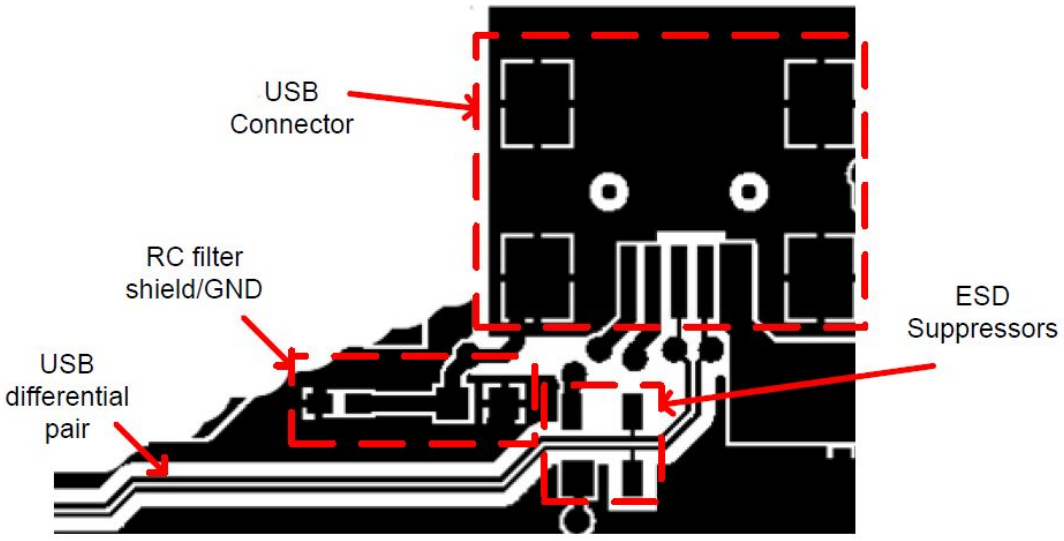

Y1 TSX-3225_16.0000MF18X-AC0
GND
4
1
GND_2
1
3
GND_1
3
2
GND
CLK1
R4
Res3
1M
CLK0
C11
Cap
12pF
C10
Cap
12pF
GND
GND
ISP
PDI_DATA
PDI_CLK
1
3
5
2
4
6
VCC
Header 3X2
GND

Annexe C. Layout du convertisseur 3.3V LM3670MF-3.3/NOPB



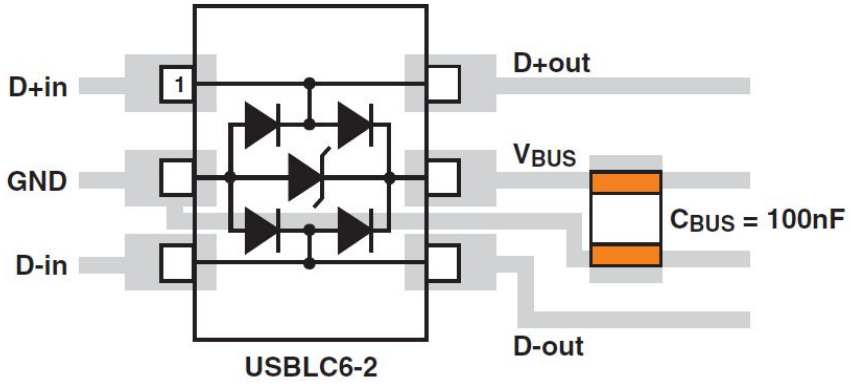
Layout du convertisseur (datasheet)

Annexe D. Layout USB

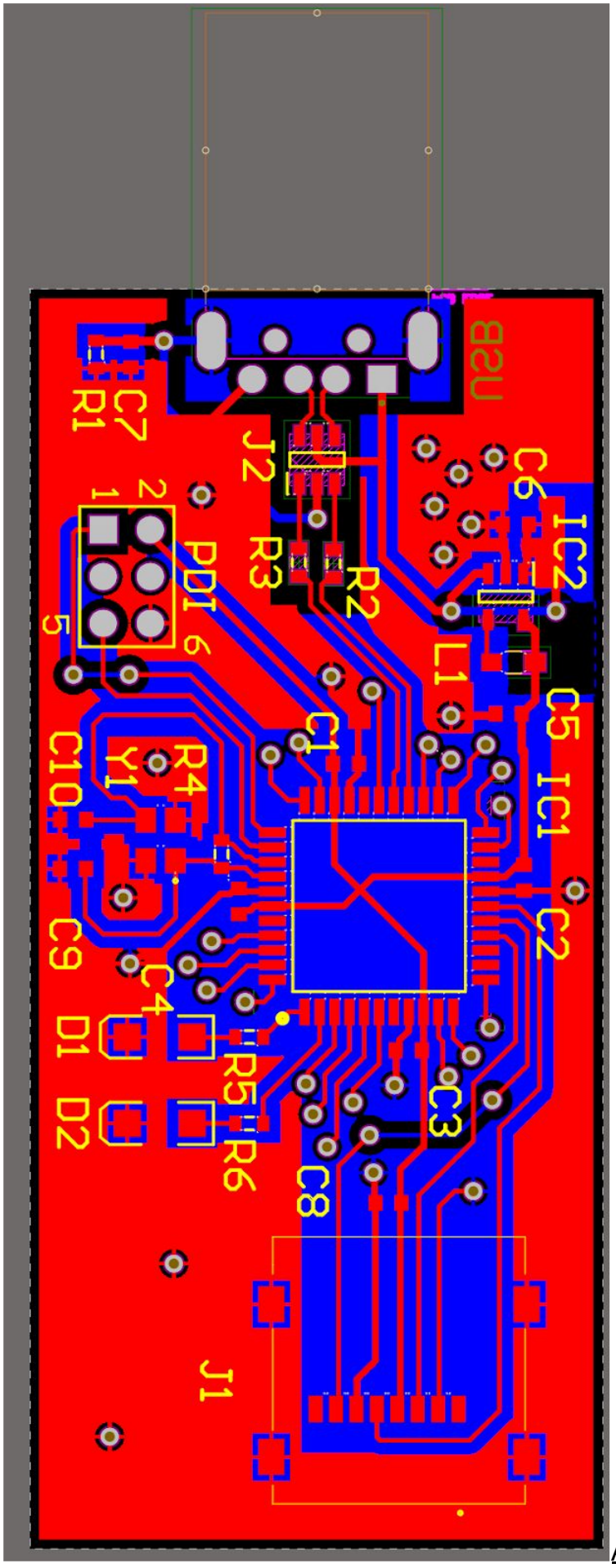


Layout USB (AVR XMega HW Recommendations)

Annexe E. Considérations du layout pour l'ESD suppressor

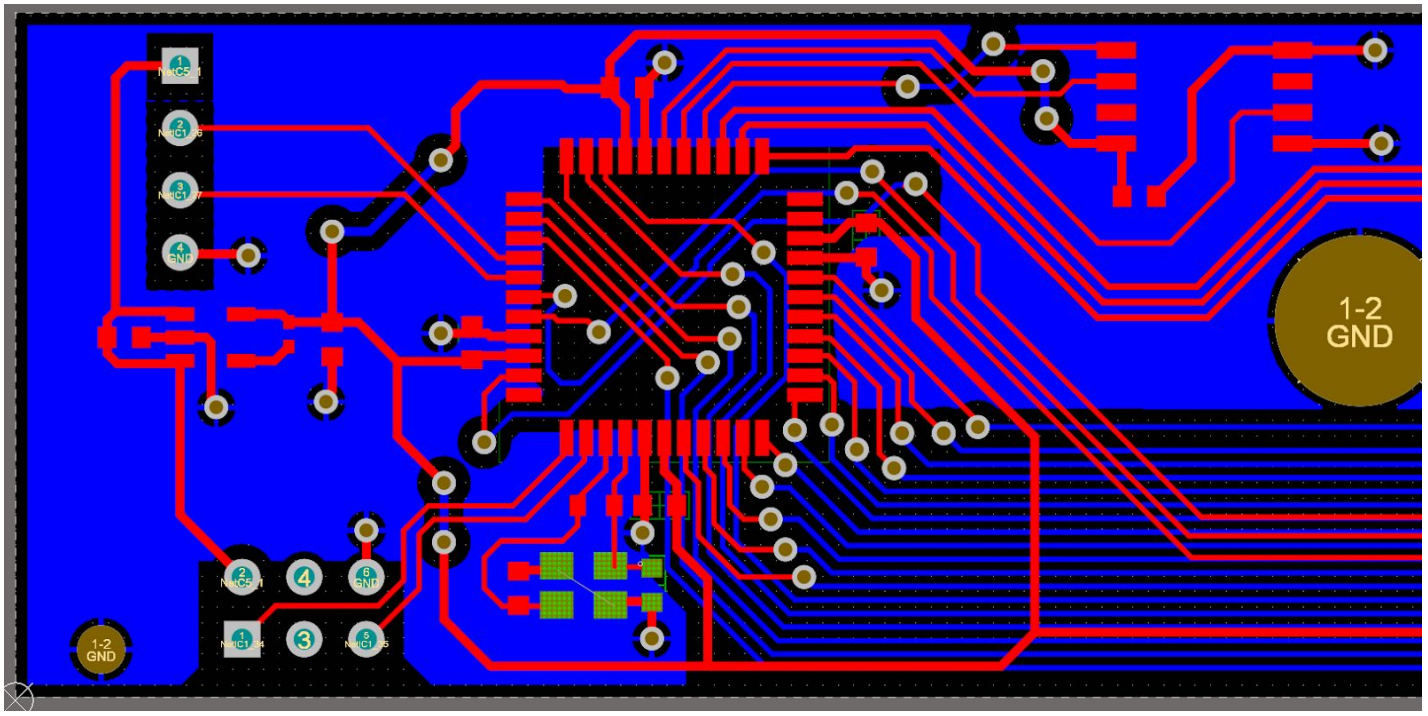


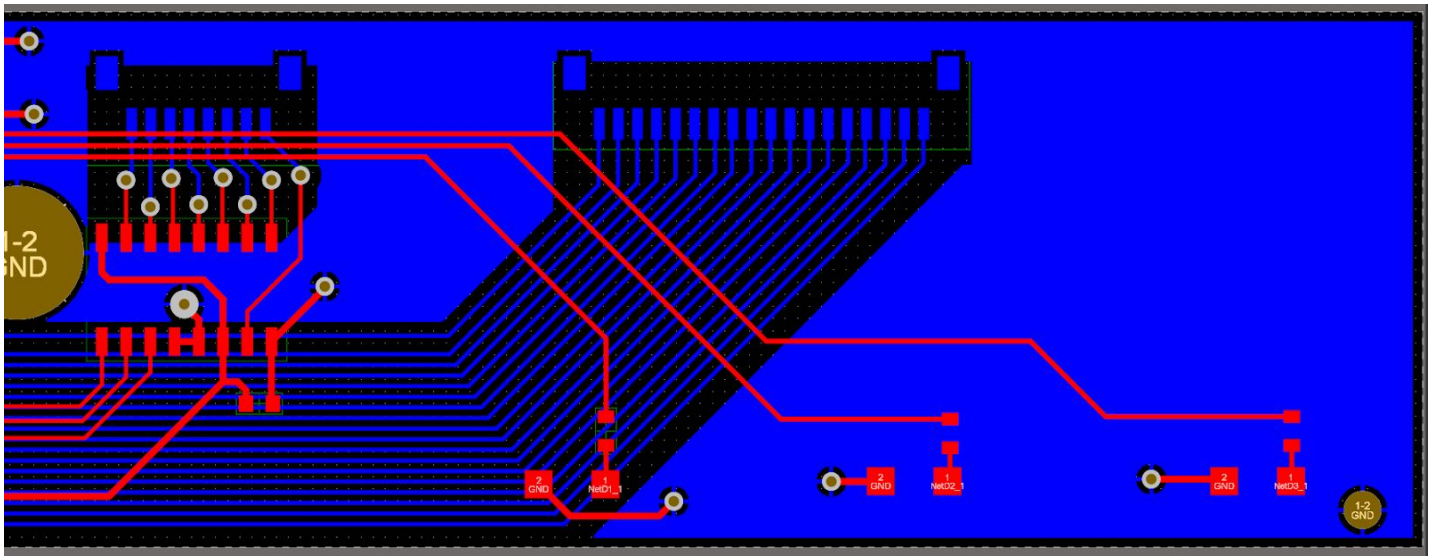
Layout du USBLC6-2 PCB (datasheet)



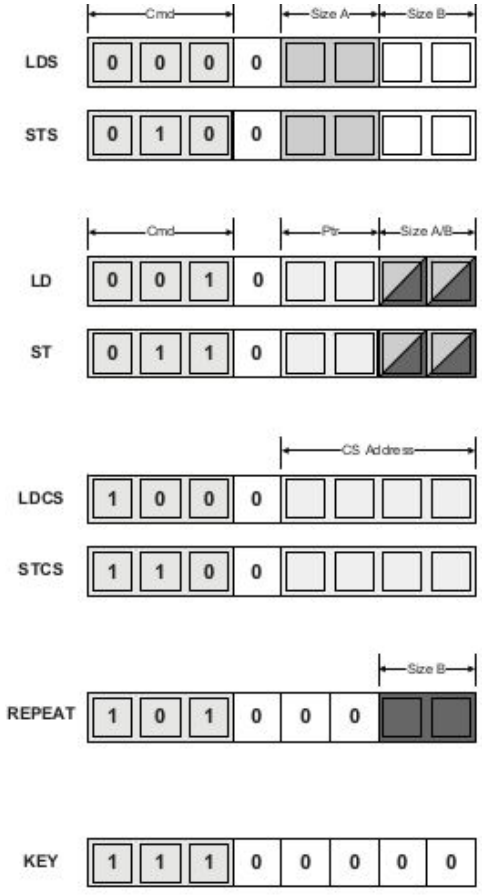
Annexe F. PCB de la clé USB

Annexe G. PCB du clavier





Annexe H. Instruction PDI



Cmd			
0	0	0	LDS
0	0	1	LD
0	1	0	STS
0	1	1	ST
1	0	0	LDCS (LDS Control/Status)
1	0	1	REPEAT
1	1	0	STCS (STS Control/Status)
1	1	1	KEY

Size A - Address size (direct access)		
0	0	Byte
0	1	Word (2 Bytes)
1	0	3 Bytes
1	1	Long (4 Bytes)

Ptr - Pointer access (indirect access)		
0	0	*(ptr)
0	1	*(ptr++)
1	0	ptr
1	1	ptr++ - Reserved

Size B - Data size		
0	0	Byte
0	1	Word (2 Bytes)
1	0	3 Bytes
1	1	Long (4 Bytes)

CS Address (CS - Control/Status reg.)				
0	0	0	0	Register 0
0	0	0	1	Register 1
0	0	1	0	Register 2
0	0	1	1	Reserved
				.....
1	1	1	1	Reserved

Annexe I. Instruction NVM

CMD[6:0]	Commands / Operation	Trigger	Change protected	NVM Busy
0x00	No operation	-	-	-
0x40	Chip erase[1]	CMDEX	Y	Y
0x43	Read NVM	PDI Read	N	N
Flash Page Buffer				
0x23	Load flash page buffer	PDI Write	N	N
0x26	Erase flash page buffer	CMDEX	Y	Y
Flash				
0x2B	Erase flash page	PDI write	N	Y
0x2E	Write flash page	PDI write	N	Y
0x2F	Erase and write flash page	PDI write	N	Y
0x78	Flash CRC	CMDEX	Y	Y
Application Section				
0x20	Erase application section	PDI write	N	Y
0x22	Erase application section page	PDI write	N	Y
0x24	Write application section page	PDI write	N	Y
0x25	Erase and write application section page	PDI write	N	Y
0x38	Application section CRC	CMDEX	Y	Y
Boot Loader Section				
0x68	Erase boot section	PDI write	N	Y
0x2A	Erase boot loader section page	PDI write	N	Y
0x2C	Write boot loader section page	PDI write	N	Y
0x2D	Erase and write boot loader section page	PDI write	N	Y
0x39	Boot loader section CRC	NVMAA	Y	Y
Production Signature (Calibration) and User Signature Sections				
0x01	Read user signature row	PDI read	N	N
0x18	Erase user signature row	PDI write	N	Y
0x1A	Write user signature row	PDI write	N	Y
0x02	Read calibration row	PDI read	N	N
Fuses and Lock Bits				
0x07	Read fuse	PDI read	N	N
0x4C	Write fuse	PDI write	N	Y
0x08	Write lock bits	CMDEX	Y	Y
EEPROM Page Buffer				
0x33	Load EEPROM page buffer	PDI write	N	N
0x36	Erase EEPROM page buffer	CMDEX	Y	Y
EEPROM				
0x30	Erase EEPROM	CMDEX	Y	Y
0x32	Erase EEPROM page	PDI write	N	Y
0x34	Write EEPROM page	PDI write	N	Y
0x35	Erase and write EEPROM page	PDI write	N	Y
0x06	Read EEPROM	PDI read	N	N