# VEHICULE AUTONOME ET INTERACTIF

Rapport

Responsables pédagogiques :

Mr Sagonéro Cyril et Mr Aubry Pierre

Intervenants :

Mr Oukrat Rémi et Mr Nazim Zakari Saibi

COUTANT THOMAS

2016/2017

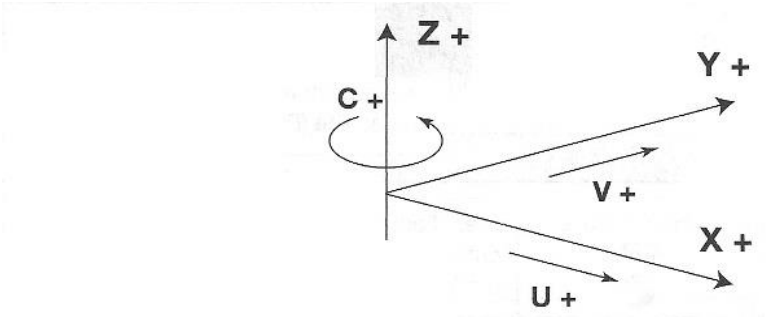
estei BORDEAUX

# Table des matières :

# I Présentation du projet

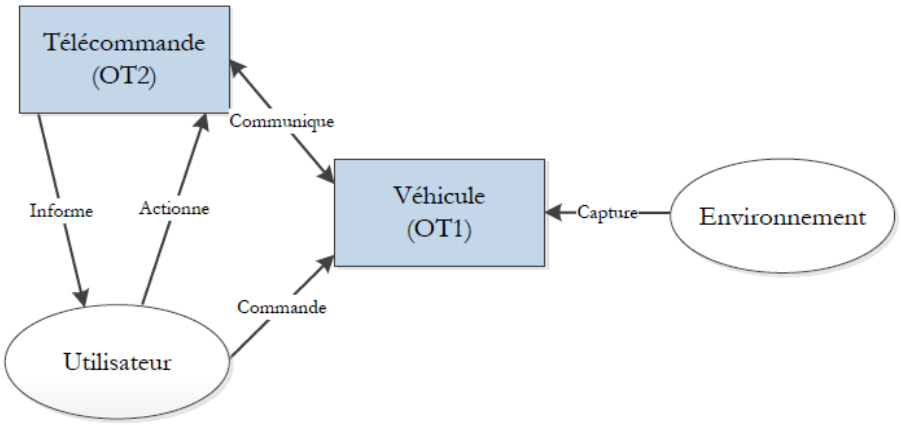
Lors de cette deuxième année nous avons un projet en robotique.

Ce robot que l'on peut assimiler à un rover, doit répondre à différentes tâches : il doit pouvoir suivre une ligne noire sur fond blanc, éviter les objets, obstacles qui barre sa route et tout cela dans un plan à trois axes.




Le robot doit également pouvoir être repris en main à l'aide d'une télécommande. Il a ainsi fallu apprendre à travailler en groupe. La classe a donc été divisée en deux groupes distinct : télécommande, rover. Chacun disposant d'une partie précise, qui une fois finie seront rassembler pour aboutir au projet final.

Diagramme sagittal :




Une partie sur le rover (OT1) qui regroupe différentes fonctions, dans lesquelles on retrouve :

L'alimentation et l'information dirigés par Eric, le déplacement dirigé par Jean, la détection d'obstacle dirigée par Samuel, la communication entre le véhicule et la télécommande dirigée par Yassine, le suivi de ligne dirigé par Axel et Clément qui s'occupe de gérer l'ensemble du rover avec toute ses fonctionnalités.

Une partie sur la télécommande (OT2) regroupée elle aussi en différentes fonctions :

L'alimentation est dirigée par Pierre-Louis, le contrôle du véhicule et les informations de la télécommande dirigé par Joris et enfin moi-même qui m'occupe de gérer l'ensemble de de la télécommande pour faire cohabiter les différentes fonctions.

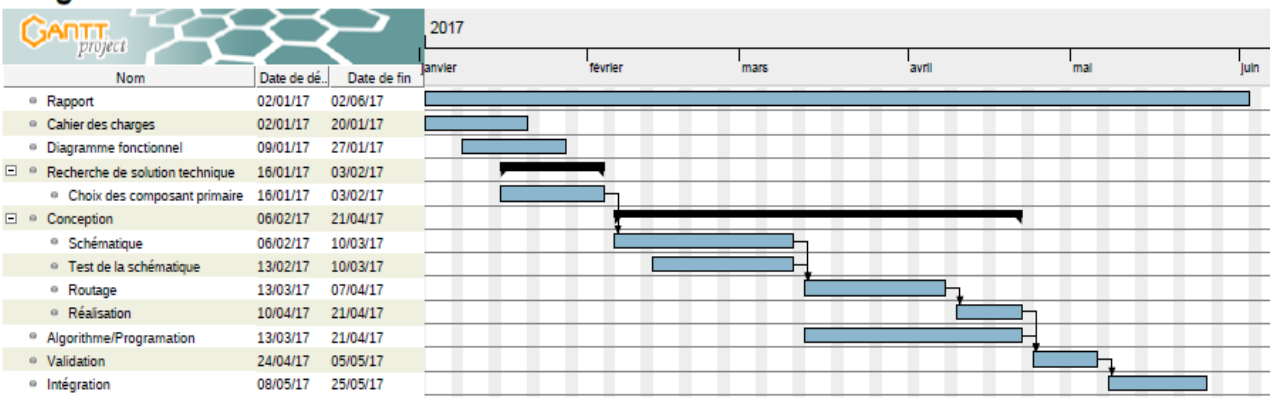
## 1) Cahier des Charges :

| Critères | Initial | Personnel |
|---|---|---|
| Température | NC | -40°C à +60°C |
| Microcontrôleur | CISC | CISC |
| Interface | NC | SCI, SPI, CAN, GPIO |
| Fréquence | NC | >40MHz($F_{bus}$=20MHz) |
| Alimentation | NC | 3,3V(Batterie), 5V(câble) |
| Courant | NC | <120mA |
| Taille carte | NC | <12cmL, <8cml |
| Prix | <300€ | <25€ |
| Taille mémoire | NC | >1KB(RAM) |

## 2) Diagramme prévisionnel :

| Nom | Date de début | Date de fin |
|---|---|---|
| Rapport | 02/01/17 | 02/06/17 |
| Cahier des charges | 02/01/17 | 20/01/17 |
| Diagramme fonctionnel | 09/01/17 | 27/01/17 |
| Recherche de solution technique | 16/01/17 | 03/02/17 |
| Choix des composant primaire | 16/01/17 | 03/02/17 |
| Conception | 06/02/17 | 21/04/17 |
| Schématique | 06/02/17 | 10/03/17 |
| Test de la schématique | 13/02/17 | 10/03/17 |
| Routage | 13/03/17 | 07/04/17 |
| Réalisation | 10/04/17 | 21/04/17 |
| Algorithme/Programation | 13/03/17 | 21/04/17 |
| Validation | 24/04/17 | 05/05/17 |
| Intégration | 08/05/17 | 25/05/17 |

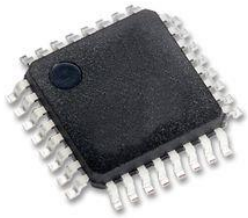
**Diagramme de Gantt**



# II Recherche des solutions techniques :

## 1) Recherche :

Lors de mes premières recherches, je me suis penché sur la famille DZ des microcontrôleurs (MC9S08DZ32ACLC, MC9S08DZ16CLC) puisque cette famille disposait selon moi d'un assez grand nombre de fonctions pour répondre aux demandes potentielles telle que : SCI, SPI, I2C



7,30 €
Prix pour : **Pièce**
Multiple: **1** Minimum: **1**

| Quantité | Prix |
|---|---|
| 1 + | 7,30 € |
| 10 + | 6,37 € |
| 25 + | 6,00 € |
| 50 + | 5,25 € |
| 100 + | 4,50 € |

| | |
|---|---|
| Tension, alimentation max..: | 5.5V |
| Type de packaging: | Pièce |
| Famille / Série de contrôleur: | S08D |
| Gamme de produit: | S08 Microcontrollers |
| Nombre d'E/S: | 26E/S |
| Normes Qualification Automobile: | - |
| Taille de mémoire programme: | 32KB |
| Vitesse de processeur: | 40MHz |
| Type de boîtier MCU: | LQFP |
| Type d'interface embarquée: | CAN, I2C, SCI, SPI |
| Tension, alimentation min.: | 2.7V |
| Nombre de broches: | 32Broche(s) |
| Taille mémoire, RAM: | 2KB |
| MSL: | MSL 3 - 168 heures |

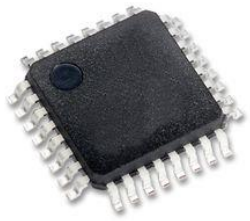
Cependant après avoir reçu les différentes spécificités de mes collègues, je me suis rendu compte que je n'avais pas besoin de beaucoup de port, j'ai donc également fait des recherches sur des micro-contrôleurs à 28 broches (MC9S08SH16CWL/CTL).

3,51 €
Prix pour : **Pièce**
Multiple: **1** Minimum: **1**

| Quantité | Prix |
|---|---|
| 1 + | 3,51 € |
| 10 + | 2,88 € |
| 25 + | 2,79 € |
| 50 + | 2,44 € |
| 100 + | 2,09 € |

| | |
|---|---|
| **Tension, alimentation max..:** | 5.5V |
| **Type de packaging:** | Pièce |
| **Famille / Série de contrôleur:** | S08SH |
| **Gamme de produit:** | S08 Microcontrollers |
| **Nombre d'E/S:** | 23E/S |
| **Normes Qualification Automobile:** | - |
| **Taille de mémoire programme:** | 16KB |
| **Vitesse de processeur:** | 40MHz |
| **Type de boîtier MCU:** | SOIC |
| **Type d'interface embarquée:** | I2C, SCI, SPI |
| **Tension, alimentation min.:** | 2.7V |
| **Nombre de broches:** | 28Broche(s) |
| **Taille mémoire, RAM:** | 1KB |
| **MSL:** | MSL 3 - 168 heures |

Toutefois ceux-ci n'ont pas un espace mémoire assez grand en RAM et en FLASH 1kB et 16kB, ce qui est insuffisant pour Joris qui s'occupe de l'écran de la télécommande. J'ai ensuite appris que je pouvais me passer de certaines fonctions qui étaient en trop sur la famille que j'avais choisi au préalable comme de l'I2C, je me suis donc redirigé vers la famille QE, avec des prix plus abordables puisque l'on est aux environs de 3/4€ pour 5/7€ avec la famille DZ. Je suis donc parti sur le microcontrôleur MC9S08QE32CLC



**4,06 €**
Prix pour : **Pièce**
Multiple: **1** Minimum: **1**

| Quantité | Prix |
|---|---|
| 1 + | **4,06 €** |
| 10 + | **3,44 €** |
| 25 + | **3,24 €** |
| 50 + | **2,84 €** |
| 100 + | **2,43 €** |

| | |
|---|---|
| **Tension, alimentation max..:** | 3.6V |
| **Type de packaging:** | Pièce |
| **Famille / Série de contrôleur:** | S08QE |
| **Gamme de produit:** | S08 Microcontrollers |
| **Nombre d'E/S:** | 26E/S |
| **Normes Qualification Automobile:** | - |
| **Taille de mémoire programme:** | 32KB |
| **Vitesse de processeur:** | 50.33MHz |
| **Type de boîtier MCU:** | LQFP |
| **Type d'interface embarquée:** | I2C, SCI, SPI |
| **Tension, alimentation min.:** | 1.8V |
| **Nombre de broches:** | 32Broche(s) |
| **Taille mémoire, RAM:** | 2KB |
| **MSL:** | MSL 3 - 168 heures |

Ainsi ce microcontrôleur est le plus aptes parmi ceux recherchés, puisqu'il répond au cahier des charges disposant :

D'une vitesse de processeur supérieure à 40MHz (50.33MHz).

D'une tension d'alimentation maximum de 3.6V (requis 3.3v).

D'une taille de mémoire (RAM) supérieure au 1KB demandé.

Et d'un faible prix.

## 2) Solutions techniques :

### a) Choix du mode d'oscillation

On choisira de préférence un oscillateur externe, puisque l'oscillateur interne du microcontrôleur n'est composé que d'une résistance et d'un condensateur, ce qui nous donnera une fréquence de bus très imprécise. Cependant pour réaliser la communication entre la télécommande et le rover, il faut avoir une fréquence précise, ce qui est aussi préférable pour l'affichage de l'écran, écran qui nécessitera une haute fréquence de bus (supérieure a 10MHz). Ainsi on utilise une oscillation externe avec un quartz qui permettra d'obtenir une fréquence bien plus précise. On choisira également de travailler à une consommation dite « faible », et dans une gamme de fréquence basse. Pour obtenir des résultats cohérent on utilise la PLL (Phase-locked loop) du microcontrôleur, qui permet de multiplier la fréquence externe préalablement choisie, par un facteur spécifique.

Il faut donc sélectionner un quartz en adéquation avec ce qui est ci-dessus. Pour parer à toute éventualité on cherchera à avoir la fréquence la plus élevée possible, on choisira de prendre un quartz externe avec une fréquence de 32.768Khz, puisque à cette fréquence on atteindra le maximum possible. Soit 50,33MHz qui est la limite possible pour le CPU, ce qui nous donne une fréquence de bus $f_{bus}$=25,165MHz et un temps de cycles de $T_{cycles}$=39,73ns.

Pour obtenir ces résultats et en comprendre le fonctionnement il est nécessaire de faire une étude :

**Table 11-7. DCO frequency range[1]**

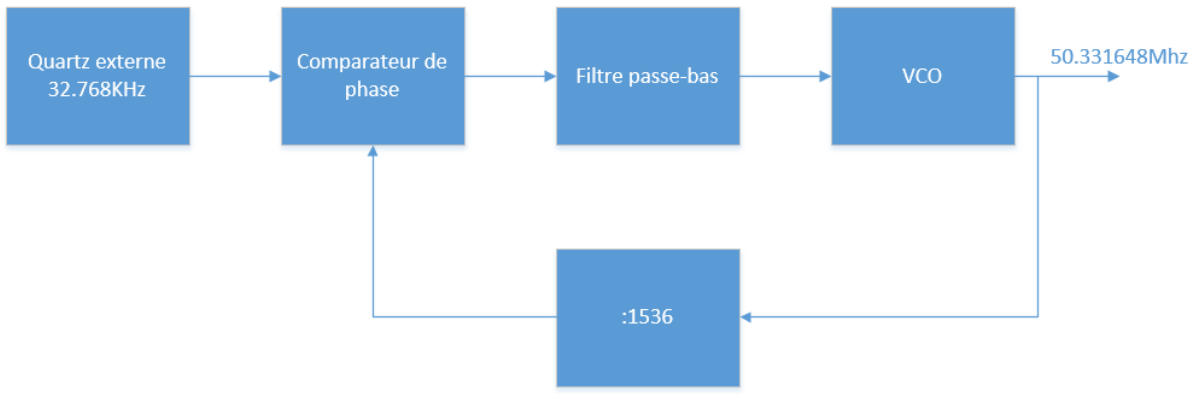
| DRS | DMX32 | Reference range | FLL factor | DCO range |
|---|---|---|---|---|
| 00 | 0 | 31.25 - 39.0625 kHz | 512 | 16 - 20 MHz |
| | 1 | 32.768 kHz | 608 | 19.92 MHz |
| 01 | 0 | 31.25 - 39.0625 kHz | 1024 | 32 - 40 MHz |
| | 1 | 32.768 kHz | 1216 | 39.85 MHz |
| 10 | 0 | 31.25 - 39.0625 kHz | 1536 | 48 - 60 MHz |
| | 1 | 32.768 kHz | 1824 | 59.77 MHz |
| 11 | Reserved | | | |

[1] The resulting bus clock frequency should not exceed the maximum specified bus clock frequency of the device.

Il convient donc de faire une division de 1536, pour obtenir une fréquence en sortie de PLL comprise entre 48 et 60MHz. Ce qui revient en réalité à multiplier le quartz externe par 1536. On a donc :

$F_H$=32768*1536=50331648=50.331648MHz

Schéma fonctionnel de la PLL :



Ces réglages sont réalisés lors de la programmation, il suffit de rentrer la fréquence souhaitée dans le CPU, la fréquence du quartz externe utilisée et d'activer la PLL souhaitée. Dans notre cas nous avons choisi une FEE, soit une PLL engagée sur le quartz externe :

| | | |
|---|---|---|
| CPU type | MC9S08QE32CLC | |
| **Clock settings** | | |
| **Internal clock** | | |
| Internal oscillator frequency [kHz] | 32.768 | 32.768 kHz |
| **External clock** | Enabled | |
| **Clock source** | External crystal | |
| Clock frequency [MHz] | 0.032768 | 0.032768 MHz |
| Clock range | Low frequency | <32 kHz, 38.4 kHz> |
| Oscillator operating mode | Low power | |
| **Low-power modes settings** | | |
| Initialization interrupt priority | interrupts enabled | 1 |
| **CPU interrupts** | | |
| **Enabled speed modes** | | |
| **High speed mode** | Enabled | |
| High speed clock | External Clock | 32.768 kHz |
| Internal bus clock | 25.165824 | 25.165824 MHz; (50.331648/1/2) |
| Fixed frequency clock [MHz] | 0.016384 | |
| **FLL mode** | Engaged | FEE |
| **Ref. clock source** | External Clock | |
| Ref. clock source freq. [MH | 0.032768 | 0.032768 MHz |
| Ref. clock freq. [MHz] | 0.032768 | 0.032768 MHz; (0.032768/1) |
| **DCO mode** | Auto select | Default (1536) |

## b) Choix USBDM

Le microcontrôleur a besoin d'être programmé pour cela on utilise une interface USBDM pour cette partie j'ai fait des recherches, qui m'ont amené sur le site http://usbdm.sourceforge.net . On y trouve différentes interface USBDM qui répondent à plusieurs cas spécifiques, pour mon cas j'ai choisi l'USBDM_JS16CWJ qui peut programmer mon microcontrôleur puisque celui-ci fait partie de la famille HCS08. N'ayant que pour seule besoin la programmation du microcontrôleur cette USBDM semble être le plus adéquate car il ne se surcharge pas de toutes autres fonctions, support**.** Ce qui minimise le prix et l'espace pris lors de la réalisation.

Ci-contre une partie du tableau qui informe sur les différents supports disponibles en fonction de l'USBDM choisi. Dans notre cas c'est l'USBDM_JS16CWJ qui sera sélectionné.

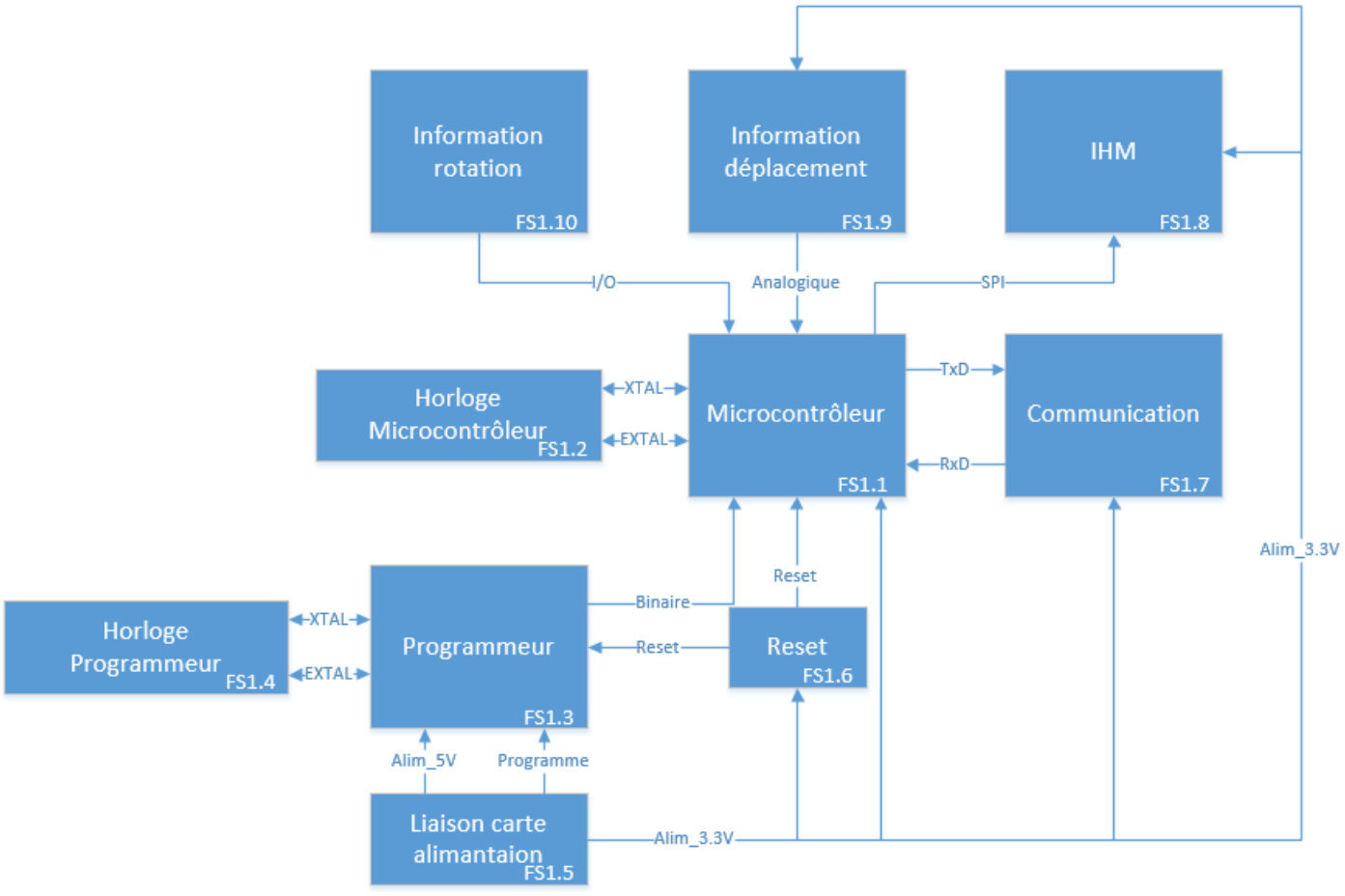
| Description | Current Design | Device Support | | | | | Features | |
|---|---|---|---|---|---|---|---|---|
| | | RS08 | HCS08, HCS12, CFV1 | Kinetis | CFV2, CFV3, CFV4* | DSC* | Target Power | Serial Port |
| **USBDM_JB16 (Unsupported)** | No | | X | | | | | |
| USBDM_CF_JMxxCLD | No | X | X | JTAG | X | X | X | |
| USBDM_SER_JS16CWJ | No | | X | | | | | X |
| USBDM_CF_SER_JS16CWJ | No | | X | JTAG | X | X | | |
| USBDM_JMxxCLC | No | X | X | | | | X | |
| USBDM_JMxxCLD | No | X | X | | | | X | |
| **USBDM_JS16CWJ** | No | | X | | | | | |

# III Analyse fonctionnelle

## 1) Schéma fonctionnel :

Niveau 1


Niveau 2



## 2) Analyse fonctionnelle :

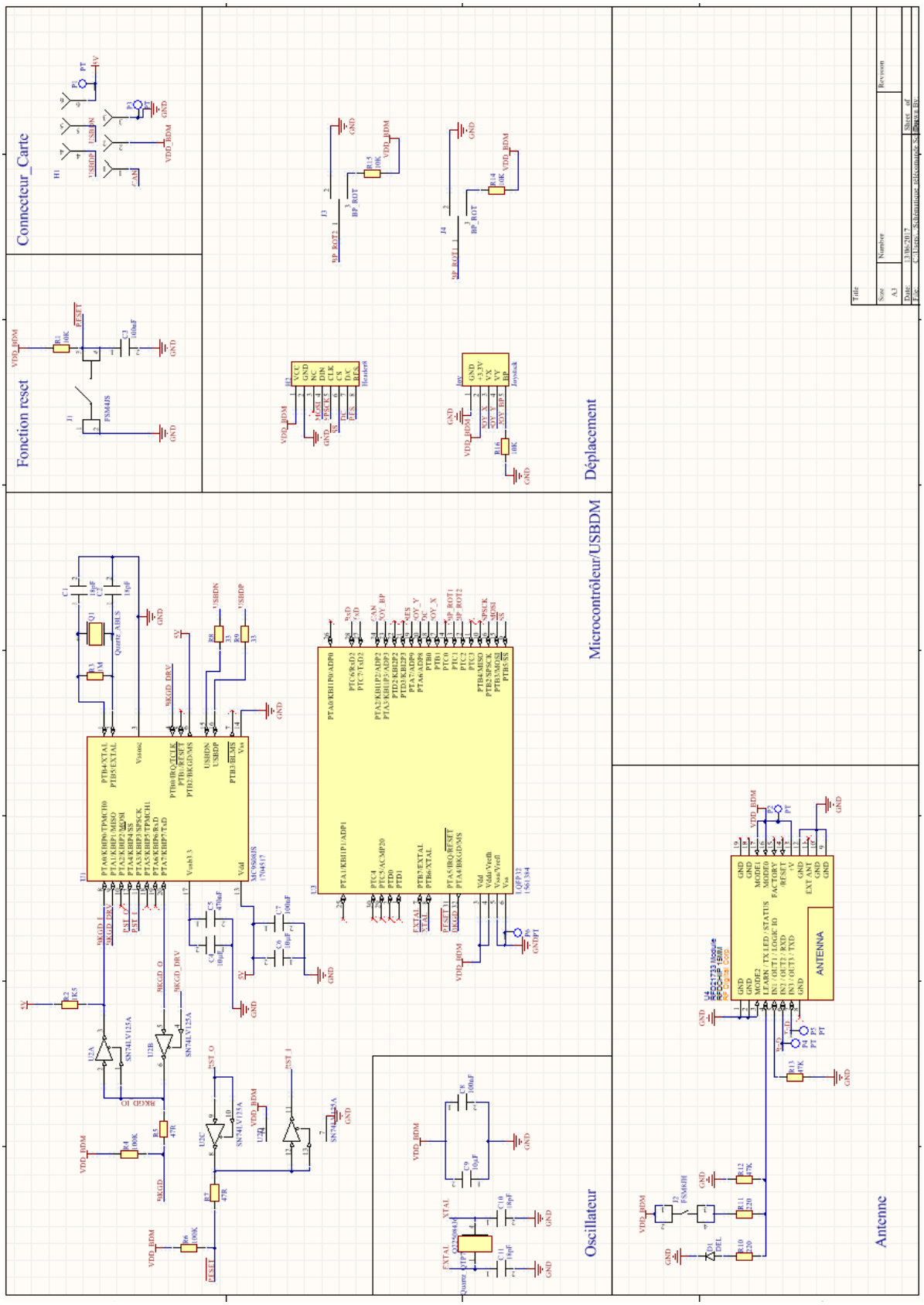
| | |
|---|---|
| FS1.1 | Microcontrôleur exécutant les demandes. |
| FS1.2 | Fonction horloge permettant de cadencer le microcontrôleur. |
| FS1.3 | Transforme le programme envoyé par USB en binaire pour que le microcontrôleur puisse exécuter le programme. |
| FS1.4 | Fonction horloge permettant de cadencer le programmeur. |
| FS1.5 | Nappe permettant la liaison entre la carte microcontrôleur et la carte d'alimentation de la télécommande. Délivrant l'alimentation 5V, 3.3V et le programme. |
| FS1.6 | Fonction reset du microcontrôleur et du programmeur à l'aide d'un bouton poussoir. |
| FS1.7 | Module radio faisant la liaison entre la télécommande et le rover. |
| FS1.8 | Ecran jouant le rôle d'interface homme machine, affichant les états de la télécommande et du rover. |
| FS1.9 | Joystick permettant le déplacement du rover. |
| FS1.10 | Boutons poussoirs permettant la rotation du rover. |

## 3) Description des signaux :

| | |
|---|---|
| EXTAL<br>XTAL | Signal sinusoïdal et Signal carré permettant de cadencer le micro et le programmeur. |
| Pro-gramme | Envoi du programme codé par la carte d'alimentation, via la nappe en direction du programmeur. |
| Alim_5V | Alimentation fournie par la nappe (5V). |
| Binaire | Traduction du code (c ) en code binaire. |
| Reset | Redémarre le microcontrôleur et l'interface de programmation. |
| I/O | Boutons poussoirs interagissant avec le microcontrôleur, pour les rotations du rover. |
| Analogique | Envoi d'un signal analogique de la direction à prendre (joystick), vers le microcontrôleur |
| SPI | Envoi des données à l'aide de la fonction SPI, permettant l'affichage de l'écran. |
| TxD | Transmission des données de communication vers le module radio, envoyé au rover. |
| RxD | Réception des données de communication (Rover/Télécommande) vers le microcontrôleur. |
| Alim 3.3V | Alimentation en 3.3V |

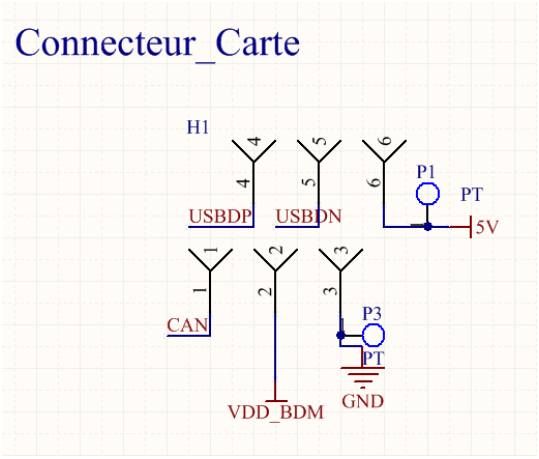
# IV Analyse structurelle

## 1) Schéma structurel



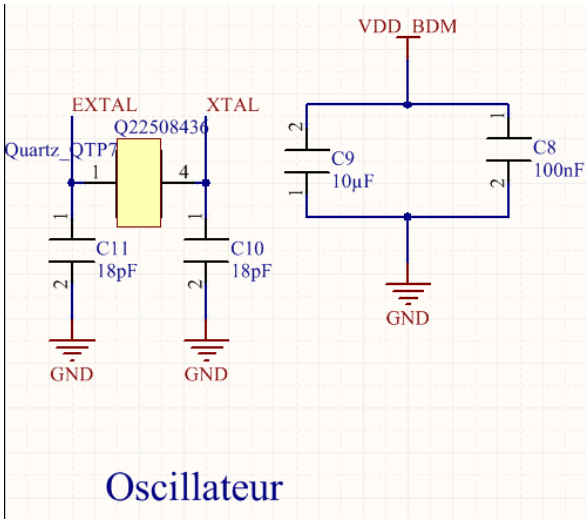
## 2) Analyse structurelle

Connecteur six broche reliant les deux cartes de la télécommande (carte d'alimentation et carte microcontrôleur). On y retrouve les alimentations en 5v (seulement fournie par câble) et le 3.3V(VDD_BDM), ainsi que les données envoyées par USB (USBDP, USBDN) et un CAN pour informer la charge de la batterie.




QTP7 : Quartz de 32.768kHz donnant la fréquence d'horloge du microcontrôleur. Associé à la PLL du microcontrôleur on obtiendra une fréquence de 50.33MHz.
C11 et C12 sont des capacités de charge indiqué par le constructeur pour l'oscillation du quartz.

C9,C8 : Capacités de découplage réalisant un filtre RC (passe bas) avec la résistance d'entrée du microcontrôleur. C9 filtre une grande partie du bruit cependant au-delà d'une certaine fréquence le filtre ne fait plus d'effet, on place donc un autre condensateur (C8) qui filtre à partir d'une fréquence plus élevée pour avoir un filtrage plus important et plus propre.




Calcule de la fréquence de coupure des filtre RC :

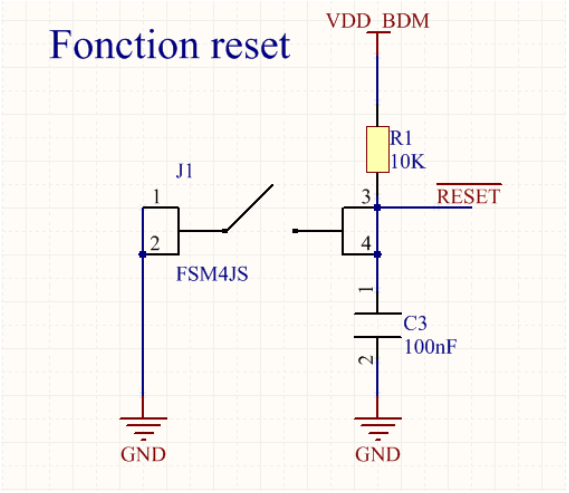
$F_c=\frac{1}{2\pi RC}$ ; R est la résistance d'entrée du microcontrôleur soit $R=\frac{U}{I}$ avec U= 3.3V et I=120mA(au maximum)

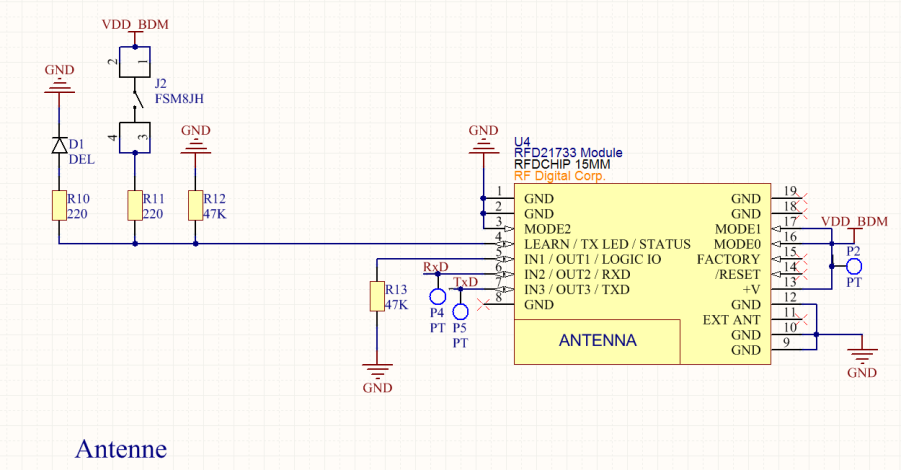
$R=\frac{3.3}{120\times10^{-3}}=27.5\Omega$

$F_{C9}=\frac{1}{2\pi\times27.5\times10\times10^{-6}}=578.7Hz$

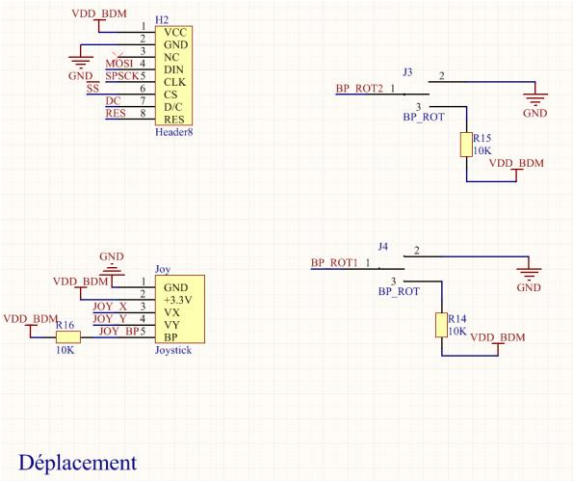
$F_{C8}=\frac{1}{2\pi\times27.5\times0.1\times10^{-6}}=57874.5Hz$

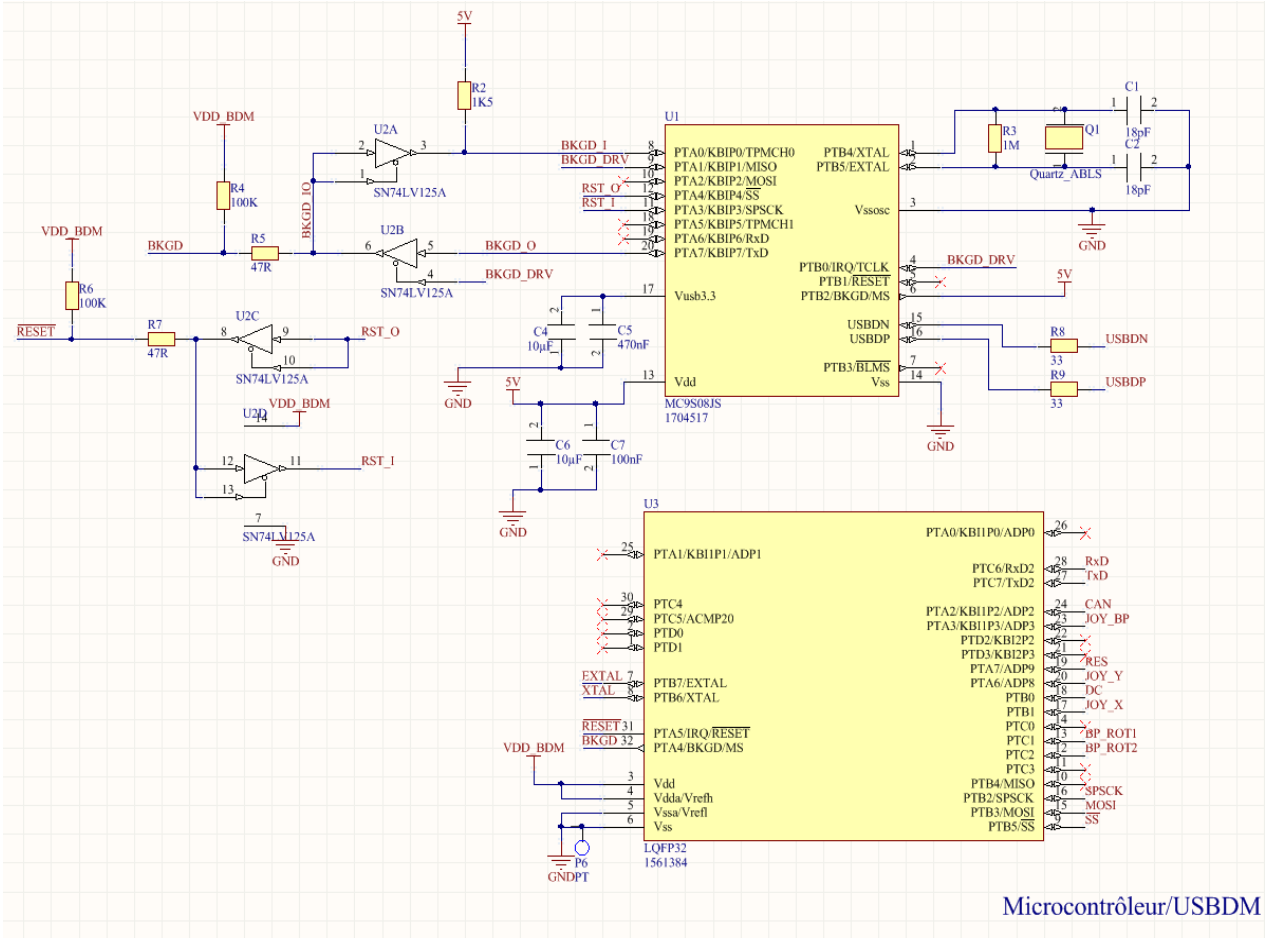
Fonction Reset, redémarrant le microcontrôleur et le programmeur par appuie sur le bouton poussoir**.** Il faut cependant d'abord activer la fonction reset sur le microcontrôleur en mettant la broche du RESET\ à 1 (soit PTA5).
Son activation doit respecter une période minimum à l'état bas de 100ns.
Vérification de la période : $\tau = R1 \times C3$
$\tau = 10 \times 10^{3} \times 0.1 \times 10^{-6} = 100ns$




La partie antenne (Yassine) a directement été intégrée sur la carte microcontrôleur. Ainsi nous avons limité l'épaisseur éventuelle, qu'aurait engendré l'association de plusieurs cartes empilées les unes sur les autres. De plus cette partie comporte peu de composant il n'a donc pas été difficile de l'ajouter directement sur la carte microcontrôleur. Il en va de même pour la partie déplacement (Joris) qui a également été intégrée sur la carte principale. On retrouve donc l'accroche pour l'écran H2 qui est directement câblé au microcontrôleur, deux boutons poussoirs (J3, J4) pour la rotation auxquelles on a rajouté une résistance de tirage (R15, R14). Et le joystick directement relié au microcontrôleur, possédant également une résistance de tirage (R16). L'antenne est aussi soudée à la carte, cependant une disposition spécifique doit être prise en compte. Puisqu'il ne faut pas cacher, encombrer la partie antenne (ANTENNA) du module radio pour qu'il puisse émettre et recevoir. Il est donc préférable de mettre le module dans un des coins de la carte.






Dans cette partie on retrouve le microcontrôleur (LQFP32) et la partie USBDM qui est utilisé pour la programmation de celui-ci :

Pour le LQFP32 la partie concernant l'oscillation et le filtrage de l'alimentation en 3.3V a été traitée précédemment.

Il faut ensuite choisir, faire la sélection des broches du microcontrôleur dont on a besoin. Une sélection est préalablement faite, cependant c'est lors de la conception que la disposition se préciser réellement. Puisque on facilitera celle-ci en choisissant quelles broches utiliser avec les composants qui l'entoure, il y aura donc moins de via et le routage sera plus court.

La partie USBDM comporte le MC9S08JS (JS16) qui est utilisé lors de la programmation puisqu'il enverra au microcontrôleur un code lisible pour lui. Et un buffer (SN74LV125A) qui fait la liaison entre le microcontrôleur et le programmeur (JS16), il permet de transmettre l'information et de protéger les deux composants entre eux dans le cas où un des deux viendrait à griller.

Le programmeur fonctionne comme un microcontrôleur, il a donc besoin d'une horloge pour être cadencer. On prendra ici également un quartz externe pour avoir une précision plus importante. On retrouve donc un quartz (Q1) avec deux condensateurs de charge (C1, C2) et une résistance (R3).

La transmission entre l'USB et le programmeur est directe on rajoutera seulement une résistance de 33Ω pour les deux données (USBDN, USBDP).

On retrouve également des capacités de découplage, qui jouent le même rôle que sur le microcontrôleur. Ce sont des filtres RC (passe bas).

Calcule de la fréquence de coupure des filtre RC :

$F_c=\frac{1}{2\pi RC}$ ; R est la résistance d'entrée du programmeur soit $R=\frac{U}{I}$ avec U= 5V et I=120mA (au maximum)

$R=\frac{5}{120\times10^{-3}}=41.6\Omega$

$F_{C6}=\frac{1}{2\pi\times41.6\times10\times10^{-6}}=382.5Hz$ = $F_{C4}$

$F_{C7}=\frac{1}{2\pi\times41.6\times0.1\times10^{-6}}=38258.4Hz$

$F_{C5}=\frac{1}{2\pi\times41.6\times0.47\times10^{-6}}=8140.1Hz$

Le buffer ne fait que relier les informations, on rajoutera seulement des résistances de tirage (R6, R4, R2)

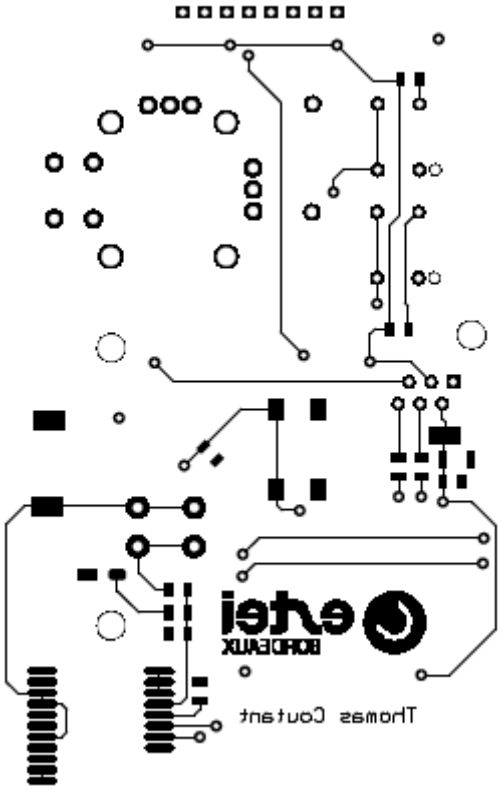
# V Fabrication

La réalisation de la carte est réalisée sur un logiciel de conception assisté par ordinateur (CAO), dans notre cas il s'agit de Altium. Lors du routage j'ai compris que le placement des composant était primordiale, car c'est ce placement qui donnera forme à la télécommande. Je me suis donc repris à plusieurs fois pour obtenir un dissigne convenable, qui apporterait lors de la fabrication à une télécommande esthétique et accessible.

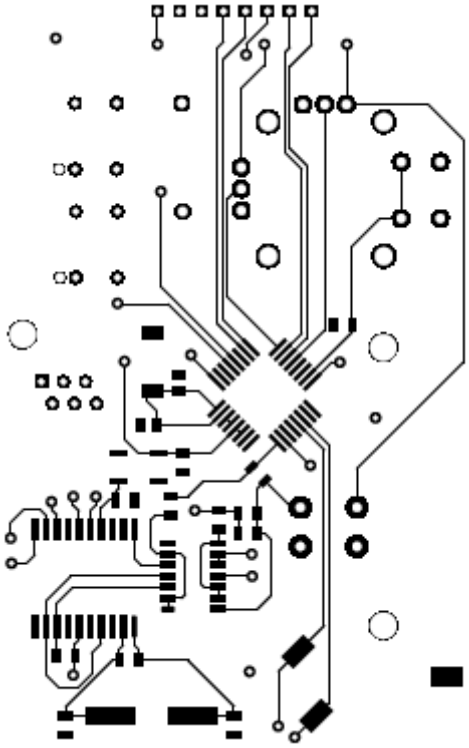
## 1) Plan :

### a) Typon (échelle 1)

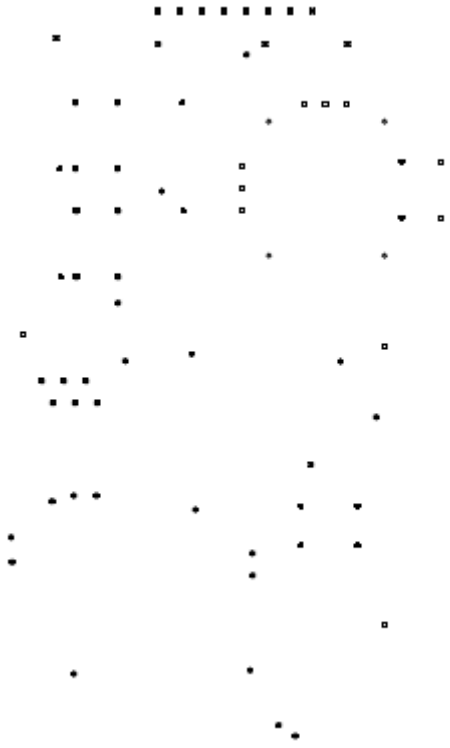
Top :



Bottom :

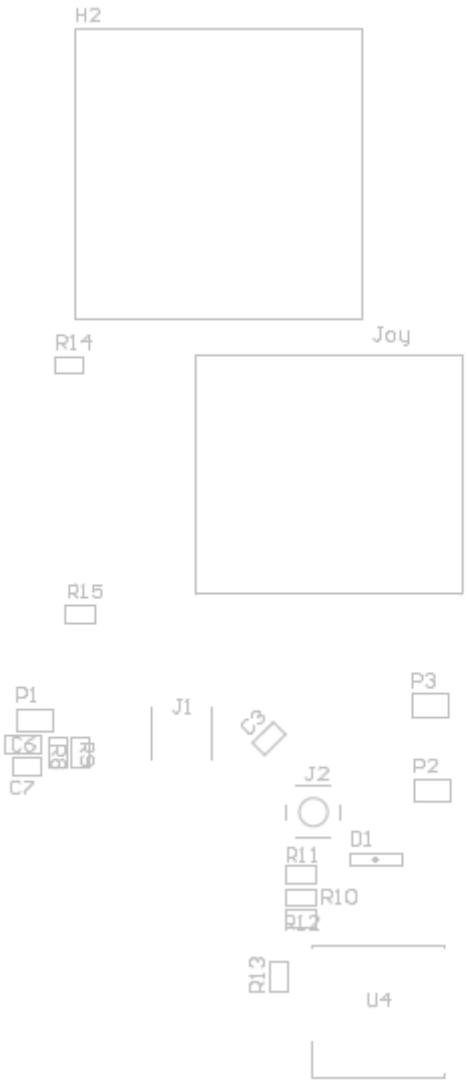


b) Plan de perçage (échelle 1)

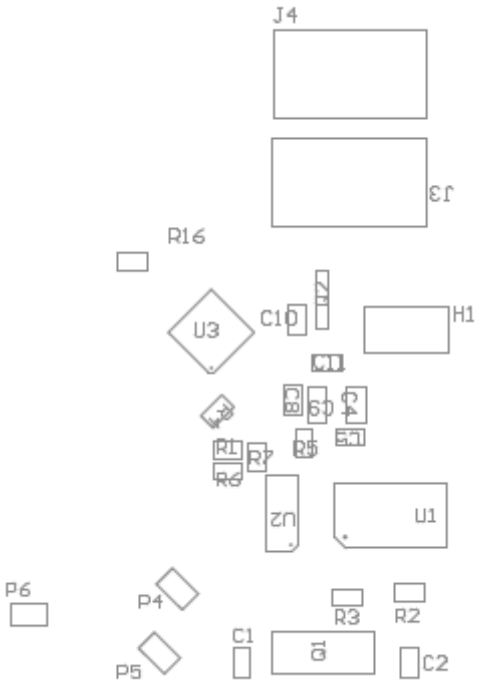


### c) Schéma d'implantation

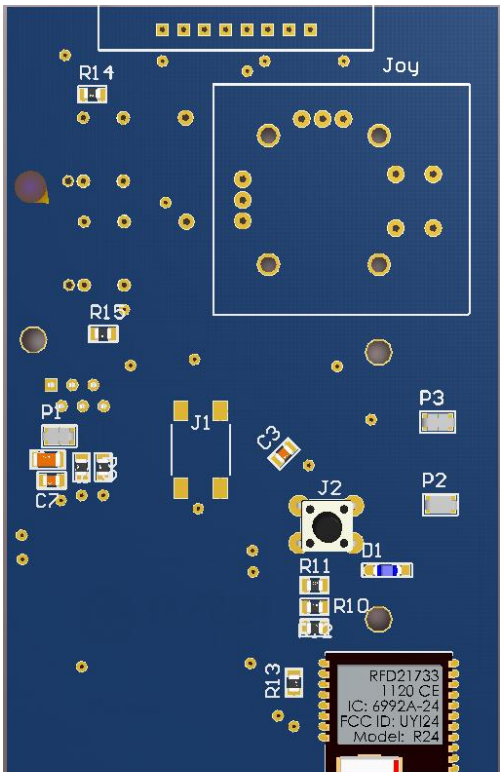
Top :



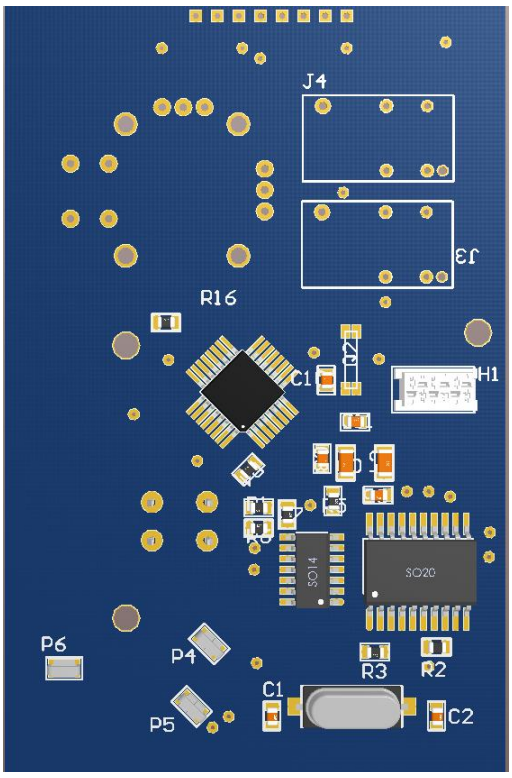
## Bottom :




## 2) Visualisation 3D :

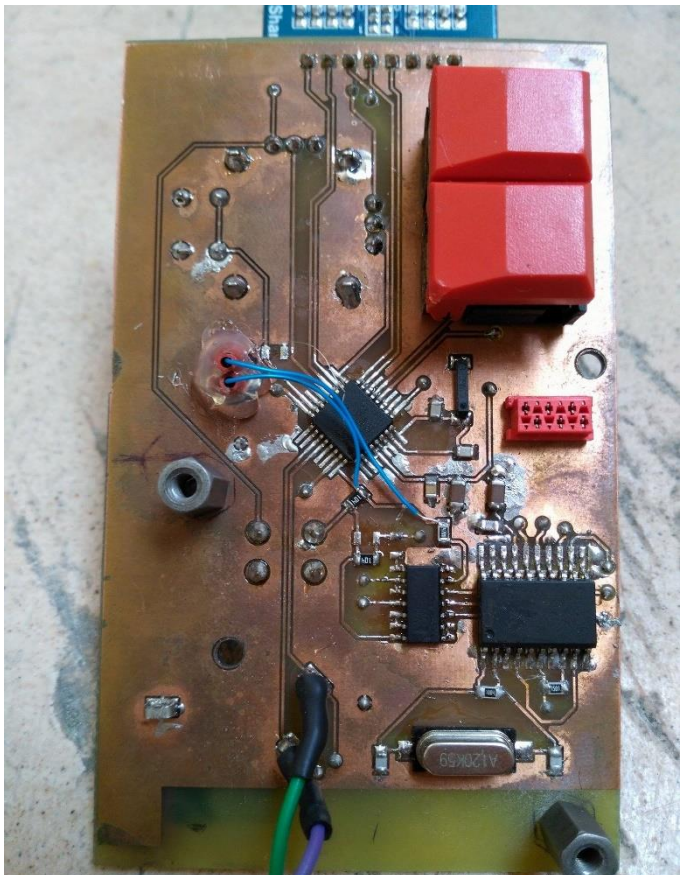



Top




*Bot*

## 3) Réalisation :



Bot


Top

## 4) Nomenclature :

| Désignation | Description | Caractéristique | Temperature de fonctionnement | Fournisseur | Code commande | Référence | Quantité | Prix UHT/U | Lot | Prix TTC |
|---|---|---|---|---|---|---|---|---|---|---|
| C8,C7,C3 | C8,C7,C3 | 0,1µF;50V;±10% | -54°C à +125°C | Farnell | 2496944 | 0805B104K500CT(lot10) | 1 | 0,01720 | 10 | 0,2064 |
| C9,C6,C4 | C9,C6,C4 | 10µF;16V;±10% | -55°C à +125°C | Farnell | 2320922 | MC1206X106K160CT(lot5) | 1 | 0,09360 | 5 | 0,56160 |
| C5 | C5 | 470nF | -30°C à +85°C | Farnell | 1759175 | MC0805F474Z250CT (lot10) | 1 | 0,0238 | 10 | 0,2856 |
| C1,C2 | C1,C2 | 18pF | -55°C à +125°C | Farnell | 1759194 | MC0805N180J500CT(lot10) | 1 | 0,0196 | 10 | 0,2352 |
| R1 | R1 | 10KΩ;125mW;±1% | -55°C à +155°C | Farnell | 2447553 | MCWR08X1002FTL(lot10) | 1 | 0,0058 | 10 | 0,0696 |
| R2 | R2 | 1,5KΩ;125mW;±1% | -55°C à +155°C | Farnell | 2447592 | MCWR08X1501FTL(lot10) | 1 | 0,0059 | 10 | 0,0708 |
| R3 | R3 | 1MΩ;125mW;±1% | -55°C à +155°C | Farnell | 2447596 | MCWR08X1004FTL(lot10) | 1 | 0,0058 | 10 | 0,0696 |
| R4,R6 | R4,R6 | 100KΩ;125mW;±5% | -55°C à +155°C | Farnell | 9234250 | RC0805JR-07100KL(lot10) | 1 | 0,0121 | 10 | 0,1452 |
| R5,R7 | R5,R7 | 47Ω;125mW;±1% | -55°C à +155°C | Farnell | 2447666 | MCWR08X47R0FTL(lot10) | 1 | 0,007 | 10 | 0,084 |
| R8,R9 | R8,R9 | 33Ω;125mW;±1% | / | Farnell | 2447640 | MCWR08X33R0FTL(lot10) | 1 | 0,0075 | 10 | 0,09 |
| Q1 | Quartz_USBDM | 12MHz | -20°C à +70°C | Farnell | 1652551 | ABLS-12.000MHZ-B2-T | 1 | 0,314 | 1 | 0,3768 |
| Q2 | Quartz_Micro | 32,768KHz | -40°C à +85°C | Farnell | 2508436 | QTP732.768128Z0R | 1 | 0,254 | 1 | 0,3048 |
| U1 | U1 | microcontrolleur | -55°C à +150°C | Farnell | 1704517 | MC9S08JS16CWJ | 1 | 2,52 | 1 | 3,024 |
| U2 | U2 | Buffer | -40°C à +85°C | Farnell | 9592504 | SN74LV125AD | 1 | 0,35 | 1 | 0,42 |
| U3 | U3 | microcontrolleur | -55°C à +150°C | Farnell | 2313245 | MC9S08QE32CLC | 1 | 2,88 | 1 | 3,456 |
| J1 | J1 | Bouton poussoir | / | Farnell | 2292960 | FSM4JSMATR | 1 | 0,245 | 1 | 0,294 |
| J2 | GPIO | 1,27mm,6 contact | / | Farnell | 148519 | 7-2150794-6 | 1 | 0,769 | 1 | 0,9228 |
| T | Test | | | Farnell | 2293786 | S1751-46R | 1 | 0.149 | 100 | 14,9 |
| | | | | | | | | | Total TTC | 25,5164 |

## 5) Problème et résolution :

Pour la vérification de la carte j'ai d'abord commencé par un test de continuité suivi d'un teste d'ohmique. Le test ohmique se déroule en deux principales étapes à commencer par vérifier que les résistances on les bonnes valeurs, il faut ensuite vérifier qu'il y ait une grande résistance entre la masse et les circuits ou passe l'alimentions cette résistance doit être en général de l'ordre du méga-ohm. Les tests semblaient cohérents j'ai donc décidé de brancher la carte. Toute fois la carte microcontrôleur ne dispose pas de port USB puisque cette partie et rattachée à Pierre-Louis. Il a donc fallu que rattache à la carte un port USB, pour cela je me suis servis de la nappe qui relis les deux cartes et j'y ai soudé un port USB. De ce fait je n'ai d'abord branché que la partie USBDM de la carte puisque c'est la seule partie qui est alimentée en 5V. L'USBDM a bien été reconnu par l'ordinateur et nous avons pu le flasher. Cependant lorsque j'ai voulu ramener le 3.3V pour alimenter le microcontrôleur à l'aide d'un grippe fil, j'ai accidentellement touché la masse. Cela a eu pour effet de griller le JS16. J'ai donc enlevé se composant et j'ai procédé à une vérification de toute la carte pour vérifier que aucun n'autre composant ne soit HS. J'ai donc changé le JS16, suite aux vérifications j'ai de nouveau procédé au branchement. Cependant cette fois-ci j'ai ramené port USB_B le 3.3V afin de ne toucher à rien lors du branchement, pour éviter de griller un composant. J'ai donc procédé à un deuxième essai, et là encore le JS16 à bien été reconnu mais il ne trouvait pas le microcontrôleur. J'ai donc cherché à savoir d'où venait le problème, j'ai donc analysé ma schématique puis je l'ai comparée avec celle des troisièmes années. Malgré tout rien ne semblait mauvais tout était en ordre. C'est alors qu'on s'est aperçue que toute les cartes qui ont été flashé par l'ordinateur d'Aloïs ne fonctionnaient pas et présentais les mêmes problèmes. J'ai donc encore changé de JS16 puisqu'il n'était plus fonctionnel. Après de nouvelles vérifications j'ai décidé d'enlever le microcontrôleur, puisque j'avais des doutes sur son bon fonctionnement j'ai donc préféré le changer. Une fois les dernières vérifications de faites, j'ai flashé ma carte avec l'ordinateur de Marc-Antoine et cette fois-ci le microcontrôleur a bien été reconnu. J'ai donc pu procéder aux tests des différentes fonctions utilisées par mes collègues.

En effectuant ces premiers tests j'ai rencontré un problème sur le bouton poussoir du joystick. Puisque son état ne changeait jamais, il restait à zéro. Le problème venait de la schématique, car la résistance de tirage était reliée à la masse et non pas à l'alimentation (3.3V). Il a donc fallu que je retire cette résistance et afin de palier à ce problème il faudra utiliser une résistance de tirage logiciel du microcontrôleur.

Lors de l'intégration avec la carte d'alimentation de Pierre-Louis, j'ai rencontré un petit problème. La programmation de la carte fonctionnait très bien, cependant si on alimentait directement la carte avec le 3.3V de la batterie le programme qui est présent dans le microcontrôleur ne se lançait pas. Il en résultait que en 3.3V le JS16 n'est pas allumé et que de se fait il y a une perturbation sur la broche BKGD du microcontrôleur. Ainsi celui-ci se met en état de reset via cette broche, qui d'origine sert à la programmation du microcontrôleur lorsque le JS16 est en marche. J'ai donc coupé la piste qui relie le microcontrôleur au JS16, et j'ai fait ne reprise filaire avec un cavalier. Ainsi lorsque le cavalier est en place on programme et lorsque qu'on enlève le cavalier c'est pour utiliser la télécommande (en mode batterie).

J'ai également rajouté des fils, sur la sortie de la SCI pour pouvoir contrôler le rover en filaire dans le cas où les modules radios ne fonctionneraient pas.

# VI Test et programmation

## 1) Etudes des fonctions utilisées

La télécommande utilise différentes interfaces de communication, on y retrouve la SPI (Serial Peripheral Interface) et la SCI (Serial Communications Interface) ainsi que l'utilisation de CAN (Convertisseur Analogique Numérique) et de boutons poussoirs qui utilisent des GPIO.

De ce fait, j'ai réalisé un programme qui mettait en relation ces différentes partie. Pour cela il faut d'abord commencer par faire une étude de registre :

### a) Etude de registre :

#### SPI :

SPIC1 : #$5E

| | SPE | | MSTR | CPOL | CPHA | SSOE | LSBFE |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 1 | 0 |

SPE : 1 active la fonction SPI.

MSTR : 1 configure la fonction SPI en maître.

CPOL : 1 le signal d'horloge est à l'état bas au repos.

CPHA : 1 l'échantillon s'effectue sur le second front.

SSOE : 1 préconfigure la gestion de la sortie SS(barre) en mode automatique.

LSBFE : 0 les données seront envoyées en commençant par le MSB.

SPIC2 : #$10

| | | | MODFEN | | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |

MODFEN : 1 finalise la configuration du SS(barre) en mode automatique.

SPIBR : #$00

| | SPPR | | | SPR | | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

.

SPPR[2;0] : 000
SPR[3;0] : 0000 c'est deux champs permettent de configurer la division de la fréquence du bus de clock, dans ce cas la division est de 2.

SPIS :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | SPTEF | | | | | |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |

SPTEF : est a 1 quand le buffer d'émission est vide.
Remise à zéro par lecture du registre SPIS suivant d'une écriture dans le registre SPID.

SPID :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

SPID est le registre d'émission des données.

Tous les bits non traité sont laissé dans leur état par défaut.

## SCI :

SCIBDH/L : #$00A4

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | SBR | | | | |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| SBR | | | | | | | |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 0 |

SBR[12;0] : c'est 13 bits permettent de configurer le baudrate que l'on n'a déjà prédéfinie à 9600 pour cela il faut faire un calcul.

$$9600 = \frac{BusCLK}{16 \times Baudrate}$$

$$9600 = \frac{25165824}{16 \times x}$$

X=163.84 On prendra donc 164

$$9590{,}63 = \frac{25165824}{16 \times 164}$$

On peut donc calcule le taux d'erreur :

$$1 - \frac{9590.63}{9600} = 9.76 \times 10^{-4}$$

Soit 0.097%

SCI2C1 : #$00

| LOOPS | | | M | | | PE | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

LOOPS : 0 la transition s'effectue sur deux lignes séparées.

M : 0 on choisit un format de 8 bit de donné avec 1 bit de start et un bit de stop.

PE : 0 on ne prend pas en compte la parité dans l'information de la donnée.

SCI2C2 : #$0C

| | | | | TE | RE | | |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 |

TE : 1 active la transmission de la fonction SCI.

RE : 1 active la réception de la fonction SCI.

SCI2S1 :

| | TC | | RDRF | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

TC : à 1 quand la transmission de la donnée est terminée.
Remise à zéro par lecture du registre SCIS1 suivit d'une écriture dans SCID.

SCID :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

Registre d'émission de la donnée sur le bus SCI.

Tous les bits non traité sont laissé dans leur état par défaut.

## CAN :

ADCSC1 :

| COCO | | ADCO | ADCH | | | | |
|---|---|---|---|---|---|---|---|
| | | 1 | x | x | x | x | x |

ADCH[0;4] : Champ de cinq bits permettant de sélectionner la voie de conversion dans notre cas on utilisera plusieurs channels de CAN ( deux pour le joystick (axe y et axe x) et un pour la batterie)

ADCO : 1 le convertisseur réalisera des conversions sur la voie présélectionnée (activation ADC)

COCO : Bit accessible en lecture seul, est à 1 quand la conversions est terminée.

Remise à zéro lors d'une écriture dans le registre ADCSC1 ou d'une lecture du registre ADCRL.

ADCCFG : #$60

| | ADIV | | ADLSMP | MODE | | ADICLK | |
|---|---|---|---|---|---|---|---|
| 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |

ADIV : 11 Division de la fréquence de bus par 8.

ADLSMP : 0 active l'échantillonnage long.

MODE : 00 le résultat de conversion est sur 8 bits.

ADICLK : 00 sélectionne l'horloge du bus clock du microcontrôleur comme source d'horloge pour la fonction CAN.

ADCRH/L :

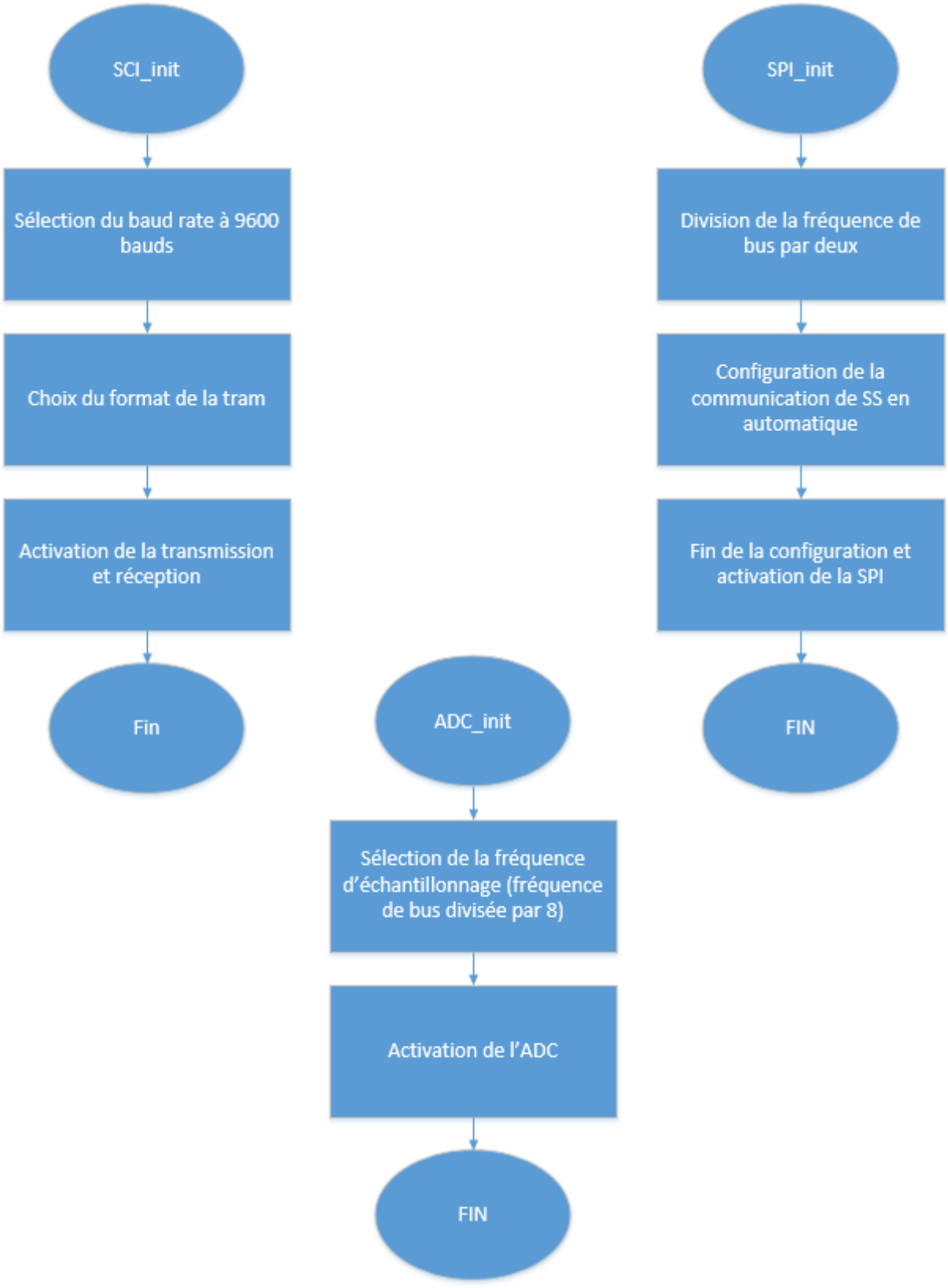
| | | | | | | | |
|---|---|---|---|---|---|---|---|
| | | | | | | | |

ADCRH/ADCRH : double registre 8 bits accessible en lecture seule contenant la valeur du résultat convertie

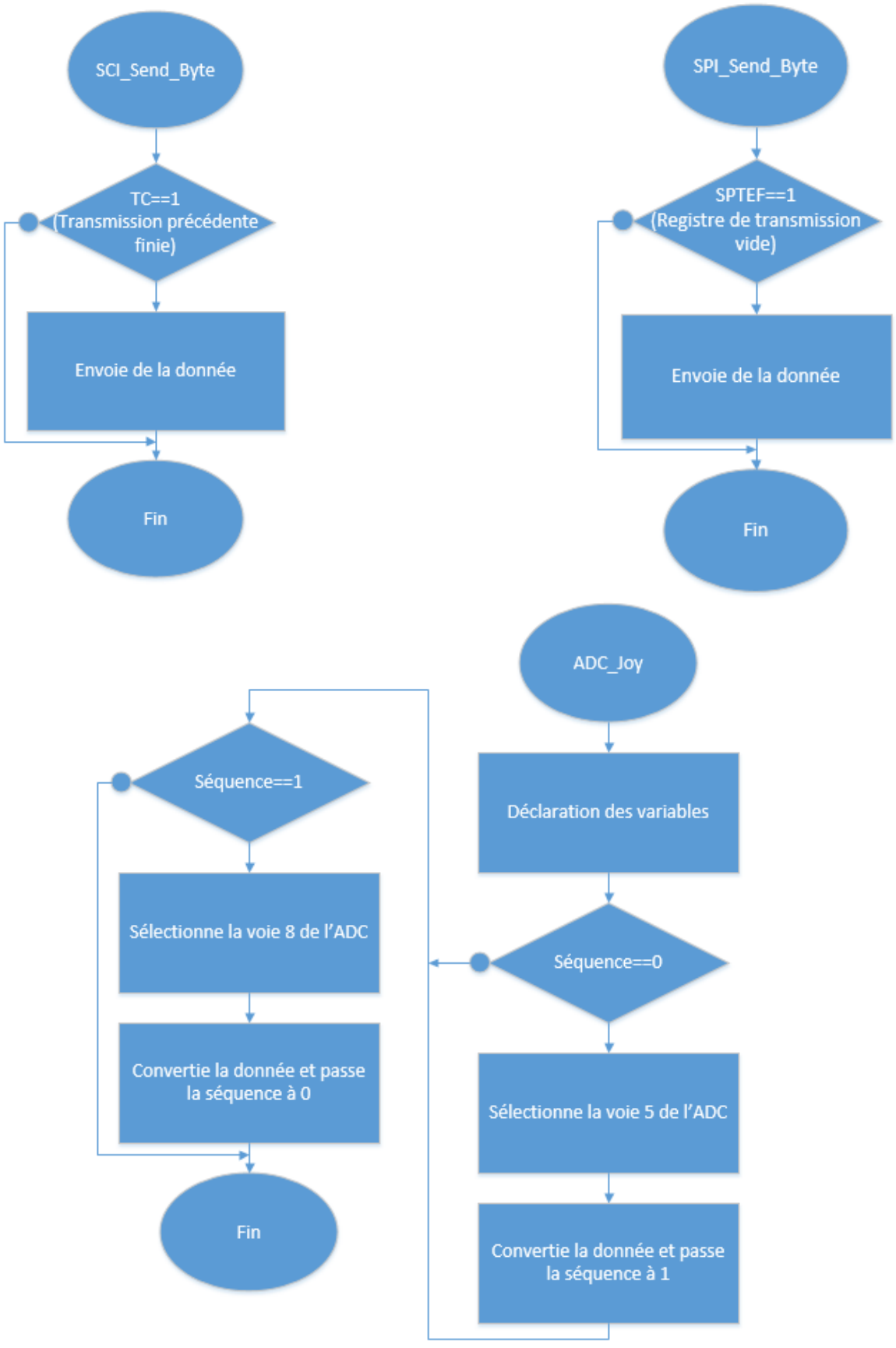
Tous les bits non traité sont laissé dans leur état par défaut.

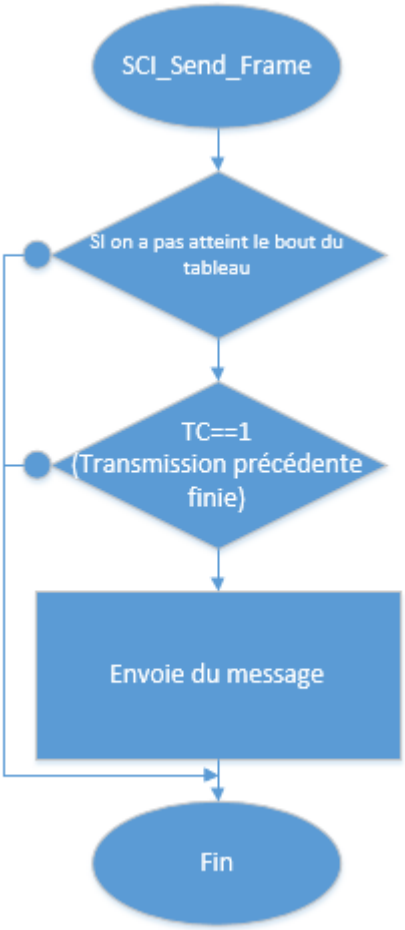
## 2) Algorigramme :

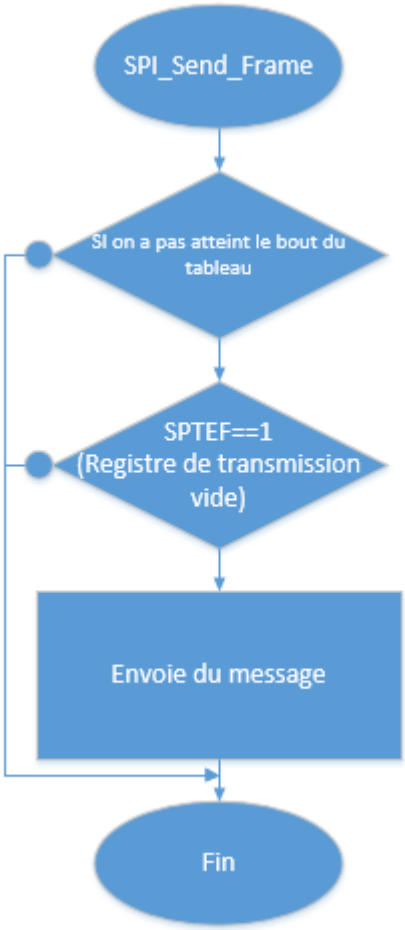
### a) Initialisation



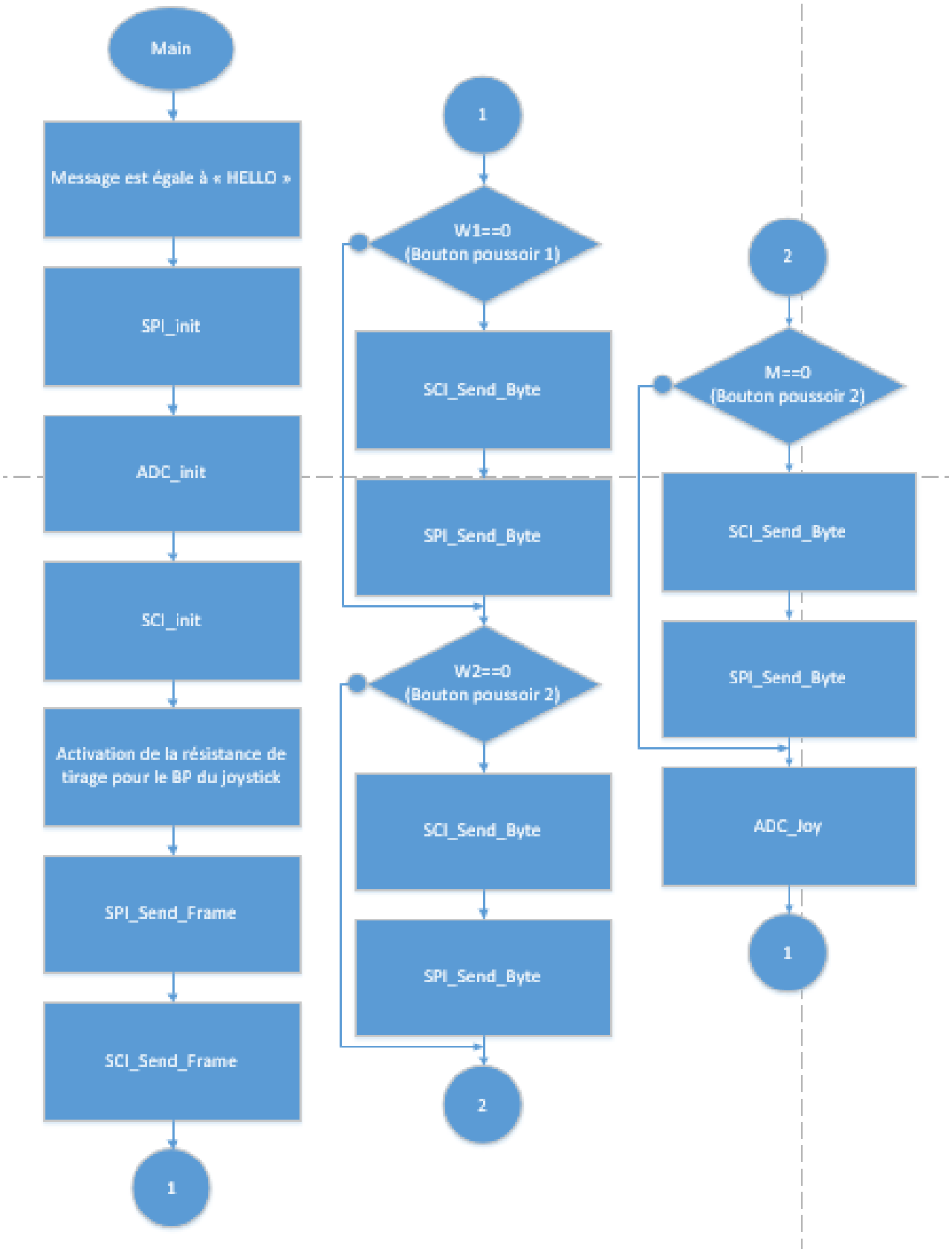
## b) Fonctions


SCI_Send_Byte
TC==1 (Transmission précédente finie)
Envoie de la donnée
Fin
SPI_Send_Byte
SPTEF==1 (Registre de transmission vide)
Envoie de la donnée
Fin
ADC_Joy
Déclaration des variables
Séquence==0
Sélectionne la voie 5 de l'ADC
Convertie la donnée et passe la séquence à 1
Séquence==1
Sélectionne la voie 8 de l'ADC
Convertie la donnée et passe la séquence à 0
Fin


SCI_Send_Frame
SI on a pas atteint le bout du tableau
TC==1 (Transmission précédente finie)
Envoie du message
Fin

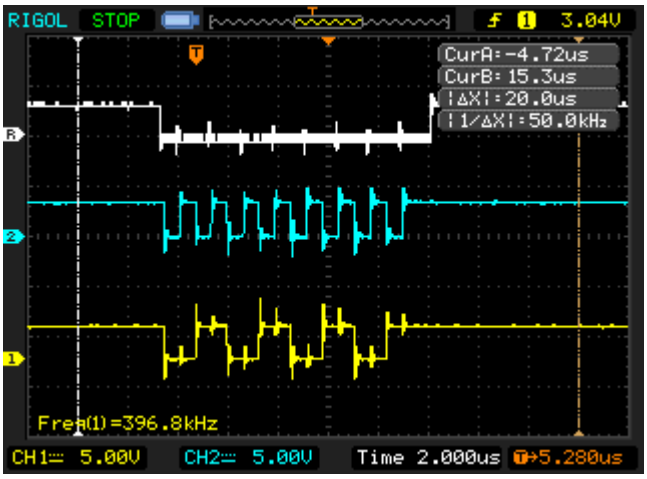

SPI_Send_Frame
SI on a pas atteint le bout du tableau
SPTEF==1 (Registre de transmission vide)
Envoie du message
Fin

## c) Programme principal

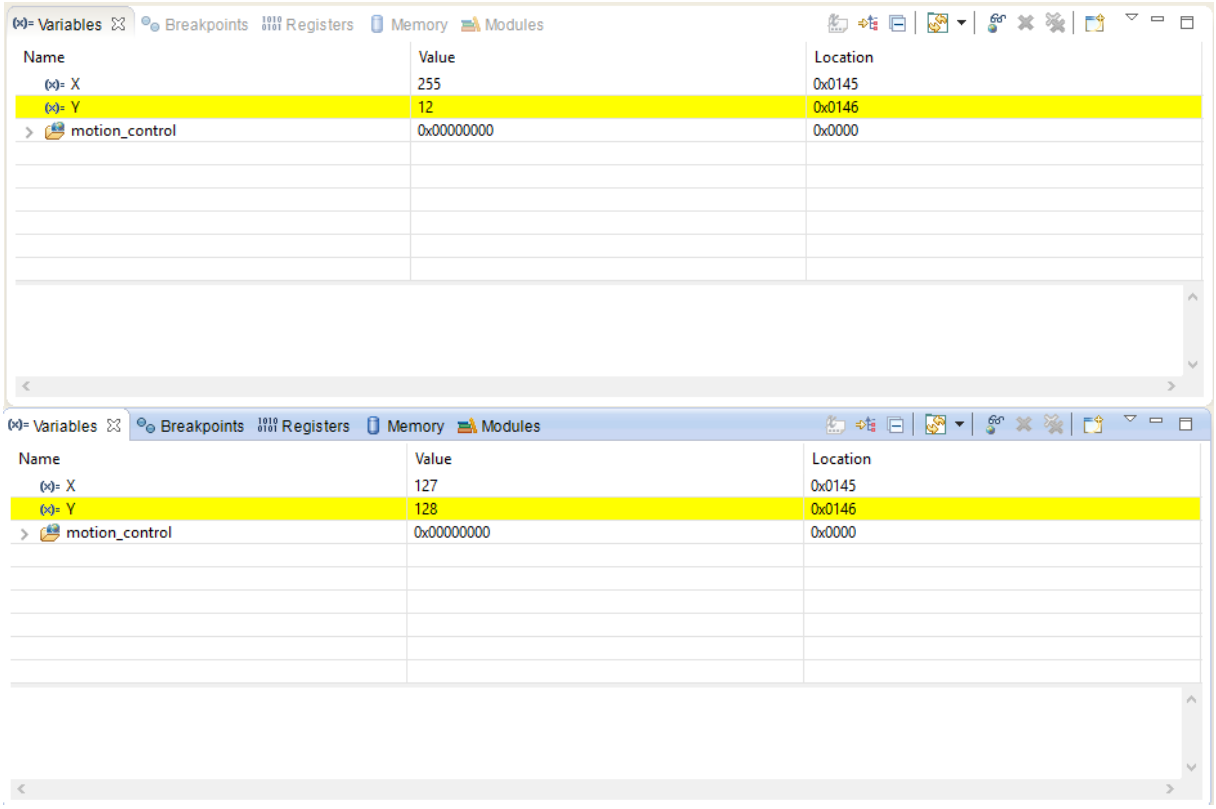


## 3) Résultat des tests :

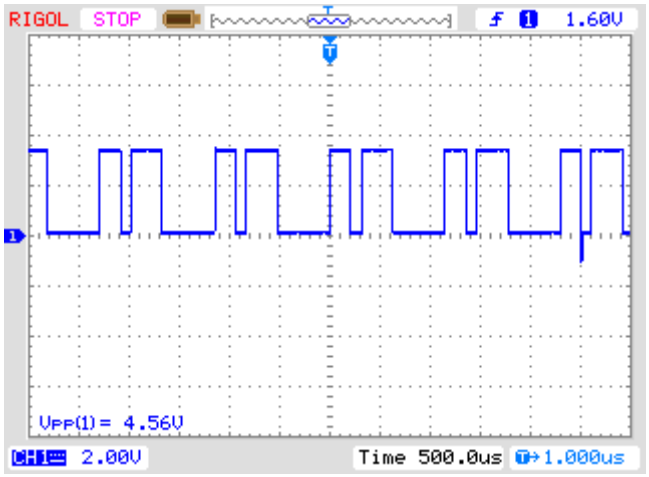
Une fois le programme réalisé j'ai procédé aux tests, pour vérifier que tout fonctionné correctement :




Test de la SPI




Tests de l'ADC




Test des boutons poussoirs avec utilisation de la SCI

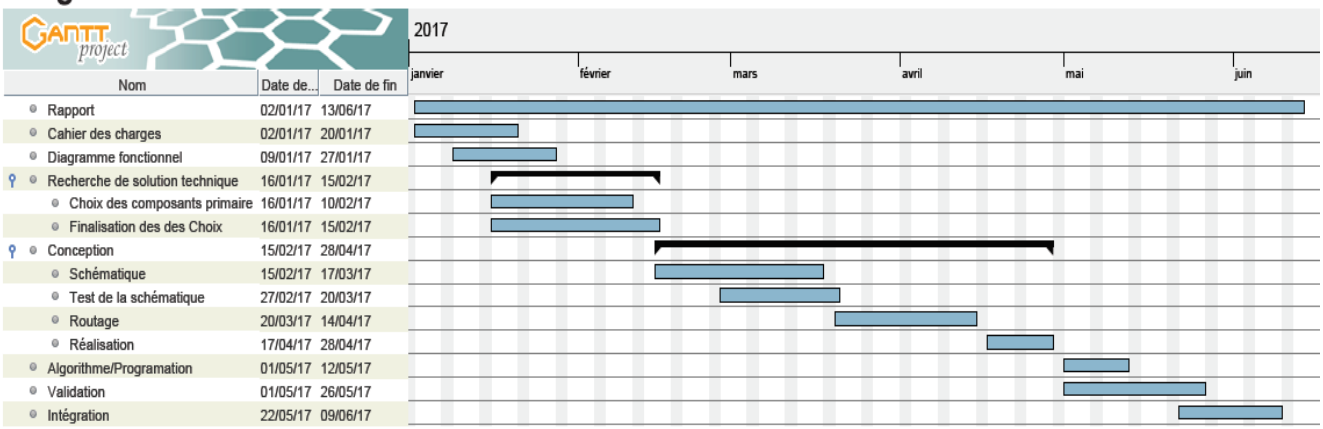
Après avoir réalisé les tests, en vérifiant que tout se déroulaient comme il faut. J'ai passé la main à Joris qui devait s'occuper de sa partie, puisque celle-ci est directement sur ma carte. Ainsi que celle de Yassine.

# VII Conclusion

## 4) Diagramme réel :

| Nom | Date de début | Date de fin |
|---|---|---|
| Rapport | 02/01/17 | 13/06/17 |
| Cahier des charges | 02/01/17 | 20/01/17 |
| Diagramme fonctionnel | 09/01/17 | 27/01/17 |
| | | |
| Recherche de solution technique | 16/01/17 | 15/02/17 |
| Choix des composants primaire | 16/01/17 | 10/02/17 |
| Finalisation des des Choix | 16/01/17 | 15/02/17 |
| | | |
| Conception | 15/02/17 | 28/04/17 |
| Schématique | 15/02/17 | 17/03/17 |
| Test de la schématique | 27/02/17 | 20/03/17 |
| Routage | 20/03/17 | 14/04/17 |
| Réalisation | 17/04/17 | 28/04/17 |
| | | |
| Algorithme/Programation | 01/05/17 | 12/05/17 |
| Validation | 01/05/17 | 26/05/17 |
| Intégration | 22/05/17 | 09/06/17 |

Diagramme de Gantt




On peut constater un retard sur le diagramme réel par rapport au diagramme prévisionnel. Ce retard est dû au fait que j'ai sous-estimé le temps que me prendrais chaque partie. De plus j'ai rencontré des problèmes lors de l'initialisation du microcontrôleur avec le programmeur, ce qui a pris plus de temps qu'escompté.

## 5) Conclusion.

Pour ce premier projet nous étions livrés à nous même, les encadrants ne sont là que pour aider en cas de besoin ou valider les différentes étapes réalisées. Il a donc fallu commencer par faire un planning prévisionnel, pour répertorier les différentes étapes que l'on doit réaliser. Ce planning est très important, puisqu'il donne un ordre d'idée sur ce qu'il y a à faire et sur l'avancement du projet. Travailler de concert n'a pas été très facile au début, car dans mon cas j'ai besoin de connaître les fonctions qu'utiliseront mes camarades, de la fréquence dont ils ont besoin pour travailler et ainsi de suite. Qui sont des critères principaux pour mes recherche technique, et tout le monde travailler à son rythme. Il faut donc savoir être patient, et étendre ses recherches en envisageant les cas de figures les plus défavorables. Une fois ces recherches finie tout le monde est plongé dans sa partie pour aboutir à des résultats concrets. Pour ma part j'ai trouvé que c'était la partie la plus intéressante mais aussi rude, car on n'avance pas forcement sur quelque chose de concret il faut parfois tout reprendre. Mais au bout d'un moment en se rend compte du travail qu'on a fait et de son aboutissement, surtout lors de la fabrication au labo une fois la carte tiré et soudé. Malgré tout lors des tests, j'ai pu rencontrer quelques problèmes, parfois épuisant quand on n'arrive pas à les résoudre. Toute fois grâce à l'aide de Mr Nazim Zakari Saibi j'ai réussi à faire correctement fonctionner la carte.

Ce projet a donc été quelque chose de nouveau, en grande partie parce qu'il a fallu travailler en groupe et que parfois il est difficile de se mettre d'accord. Mais aussi parce que l'on est livré à nous-même pour la réalisation de nos parties, en restant tout de même encadré en cas de problème. Ainsi nous aurons appris à nous gérer et à être autonome durant ce projet.

# Annexe :

Main.c :

```
/* MODULE main */
#include "Cpu.h"
#include "Events.h"
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"
#include "Test.h"
#define W1 PTCD_PTCD1
#define W2 PTCD_PTCD2
#define M  PTAD_PTAD3

void main(void)
{
    /* Write your local variable definition here */
    const unsigned char *pointeur="HELLO";
    /*** Processor Expert internal initialization. DON'T REMOVE THIS CODE!!! ***/
    PE_low_level_init();
    /*** End of Processor Expert internal initialization.                    ***/

    /* Write your code here */
    /* For example: for(;;) { } */
    SPI_init();
    ADC_init();
    SCI_init();
    PTAPE_PTAPE3=1; // Pull Up sur la broche du  Joystick
    for(;;){
            SPI_Send_Frame(&pointeur);
            SCI_Send_Frame(&pointeur);
        if (W1==0) {
            SPI_Send_Byte(0x55);
            SCI_Send_Byte(0x55);
        }
        if (W2==0) {
            SPI_Send_Byte(0x80);
            SCI_Send_Byte(0x80);
        }
        if (M==0) {
            SPI_Send_Byte(0x01);
            SCI_Send_Byte(0x01);
        }
        ADC_Joy();
```

Test.c :

```
#include "Test.h"
#define Fbus  25165824.0
#define Baudrate 9600
void ADC_init(void){
    ADCCFG_ADIV=0x03; // Fréquence d'échantillonge = Fbus/8
    ADCSC1_ADCO=0x01; // Activation ADC
    return;
}

void SCI_init(void){
    SCI2BD=(unsigned int)((Fbus/(16.0*(float)Baudrate))+0.5);   // Baud rate 9600 bauds
    SCI2C1=0x00;    // format trame 8N1
    SCI2C2=0x0C;    //  Activation de la Reception et de l'Emission
    return;
}

void SPI_init(void){
    SPIBR=0x00; //Division de l'horloge par 2
    SPIC2=0x10; // Configuration de la commutation de ss automatique
    SPIC1=0x5E; // Fin de la configuration et activation de la fonction SPI
    return;
}


void ADC_Joy(void){
    static unsigned char joyX,joyY;
    static unsigned char sequence=0;
    switch (sequence) {
    case 0:
        ADCSC1_ADCH=0x05; // channe 5 de ADC correspond à JoyX
        if(ADCSC1_COCO==1){
            joyX=ADCRL;
            sequence=1;
        }
    case 1:
        ADCSC1_ADCH=0x08; // channe 8 de ADC correspond à JoyY
        if(ADCSC1_COCO==1){
            joyY=ADCRL;
            sequence=0;
        }
        break;
    default:
        break;
    }
}
void SCI_Send_Byte(char data){
    if(SCI2S1_TC){   // Transmission précédente finie ?
        SCI2D=data;
        return;
    }
}
```

```
void SPI_Send_Byte(char data){
    if(SPIS_SPTEF){ // le registre de transmission est-il vide ?
        SPID=data;
        return;
    }
}

void SPI_Send_Frame(unsigned char *message){
    if(*message!='\0'){
        if(SPIS_SPTEF){ // le registre de transmission est-il vide ?
            SPID=*message;
            ++message;
        }
    }
    return;
}

void SCI_Send_Frame(unsigned char *message){
    if (*message!='\0'){
        if (SCI2S1_TC) { // Transmission précédente finie ?
            SCI2D=*message;
            ++message;
        }
    }
    return;
}
```

Test.h

```
#ifndef TEST_H_
#define TEST_H_
#include "Cpu.h"
#include "Events.h"
/* Include shared modules, which are used for whole project */
#include "PE_Types.h"
#include "PE_Error.h"
#include "PE_Const.h"
#include "IO_Map.h"

void SPI_init(void);
void SPI_Send_Frame(unsigned char *);
void SPI_Send_Byte(char );
void SPI_Receive(void);
void ADC_init(void);
void SCI_init(void);
void SCI_Send_Byte(char octet);
void ADC_Joy(void);
void SCI_Send_Frame(unsigned char *);

#endif /* TEST_H_ */
```