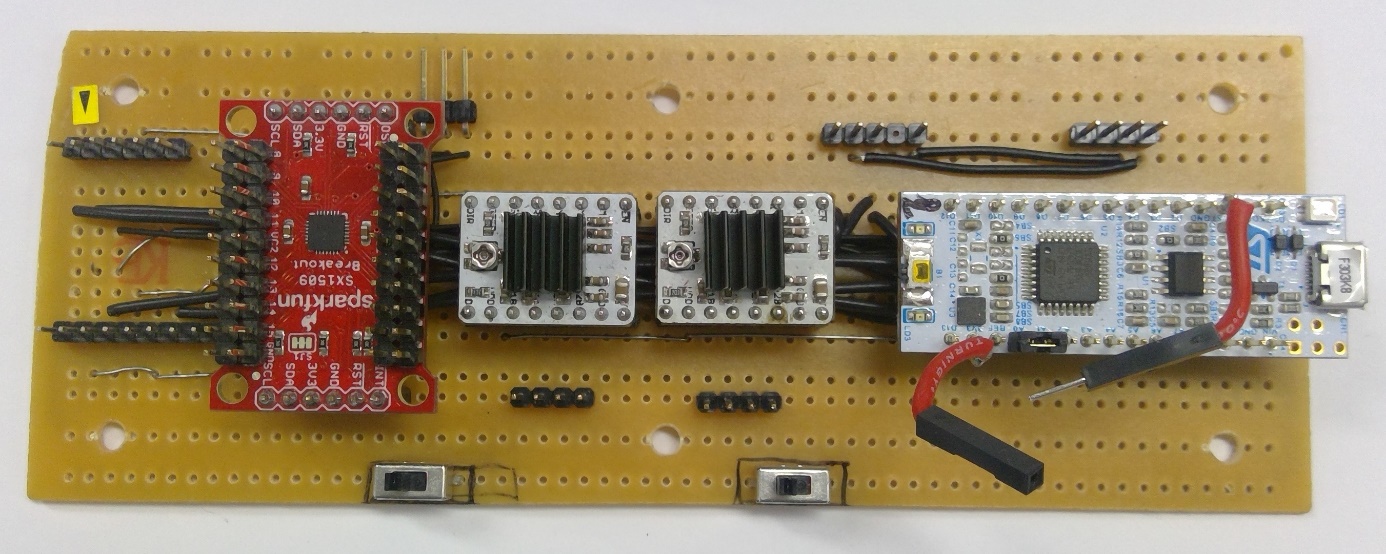
Carte Expérience

|  |  |
| --- | --- |
| Coupe de Robotique 2019 | 27/05/2019 |

La carte principale de “l’expérience” est basée sur une platine de prototypage orange comportant un module Nucleo-F303K8.

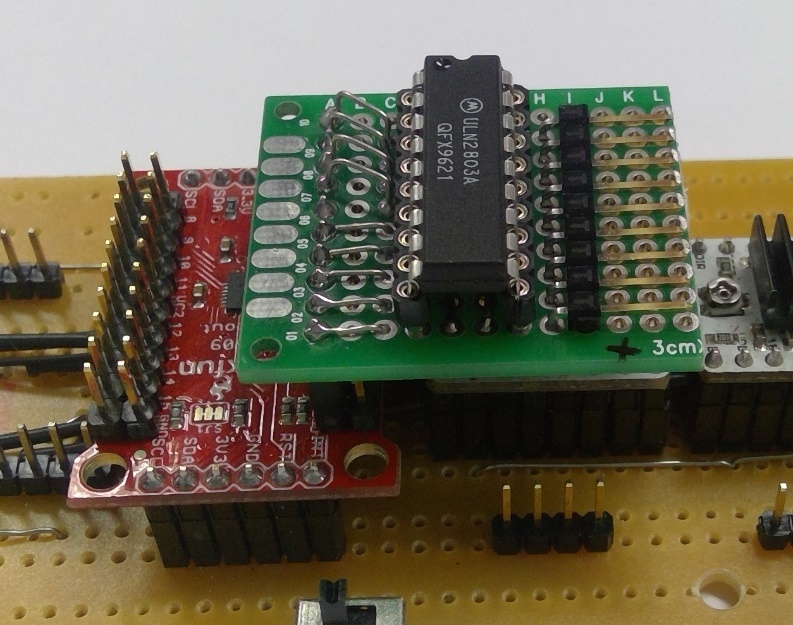
# Matériel

La carte comporte plusieurs modules sur supports. Se référer à cette photo pour les rebrancher si nécessaire. **Une inversion causera des dégâts**.



La carte est câblée spécifiquement pour des modules Stepstick A4988, **ne pas utiliser d’autres modules**.

Pour le pilotage des bandeaux de LED, un module de puissance ULN2803A additionnel vient se brancher sur le module rouge SX1509 :



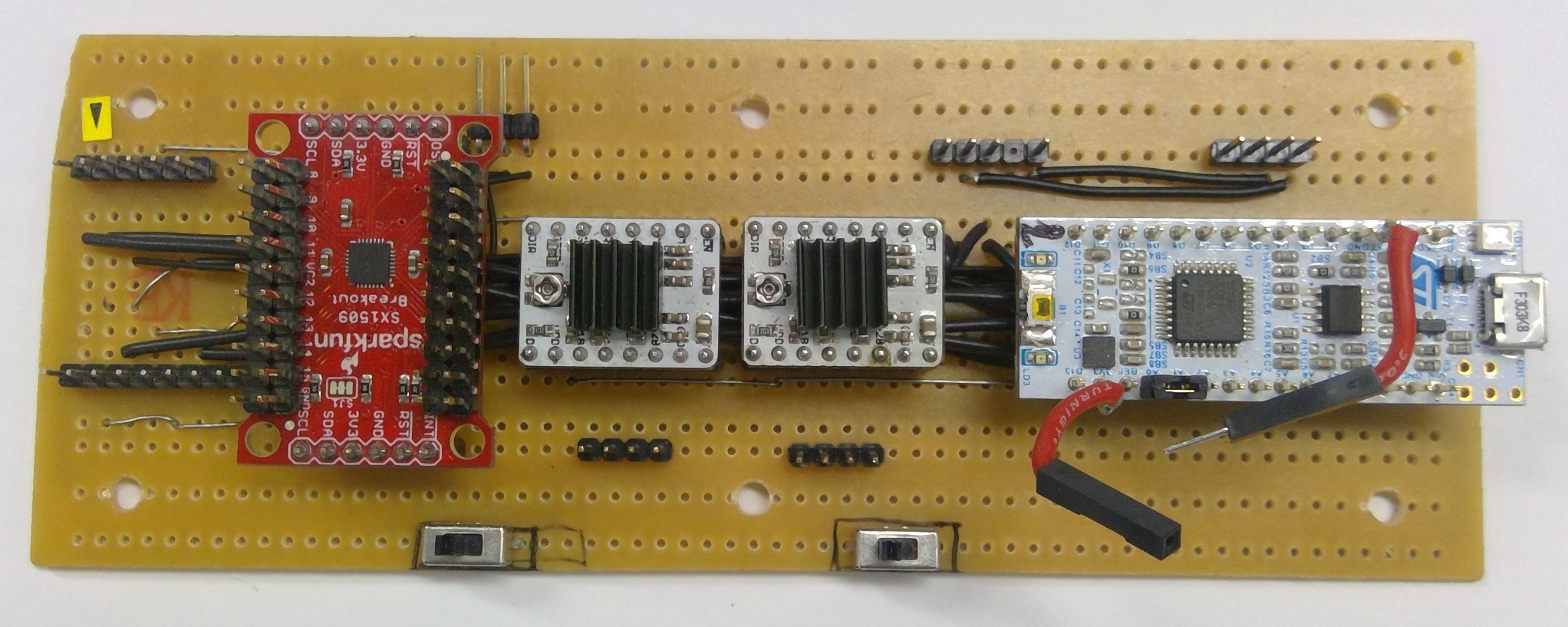
Le module se branche sur la rangée de broches **intérieure**. La rangée extérieure est à la masse.

## Connections

### Alimentation

La carte est alimentée par une carte puissance Goldorak qui lui fournit 12 V pour les moteurs pas à pas, 5 V « puissance » pour les servos et 5 V « logique » pour le module Nucleo. A noter : ce dernière génère un rail 3.3 V qui alimente les autres modules de la carte.

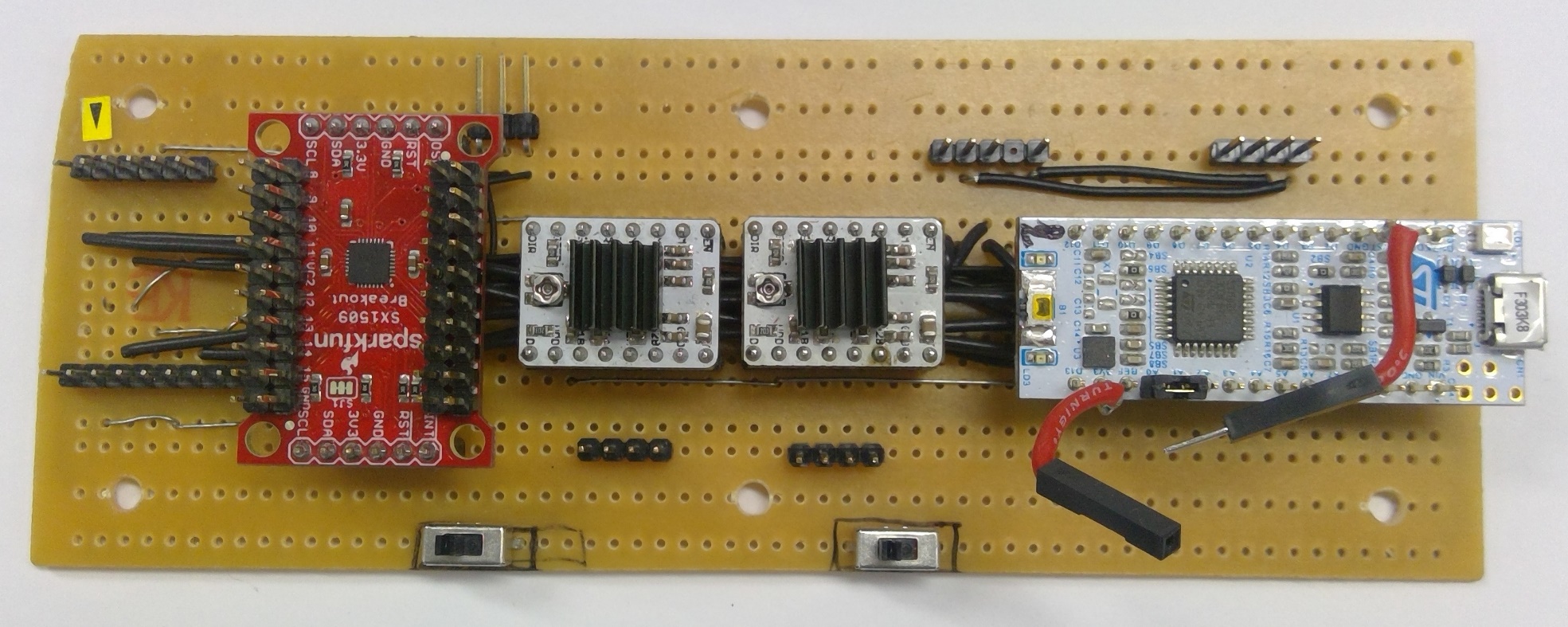
L’alimentation se branche sur un header 4 broches détrompé :



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 12 V | 5 V Servos | 5 V Logique |  | GND |

### Sorties Figurine

La figurine Goldorak emporte un module I²C PCA9685 qui pilote les servos de la figurine. Celle-ci est connectée à un header 4 broches :



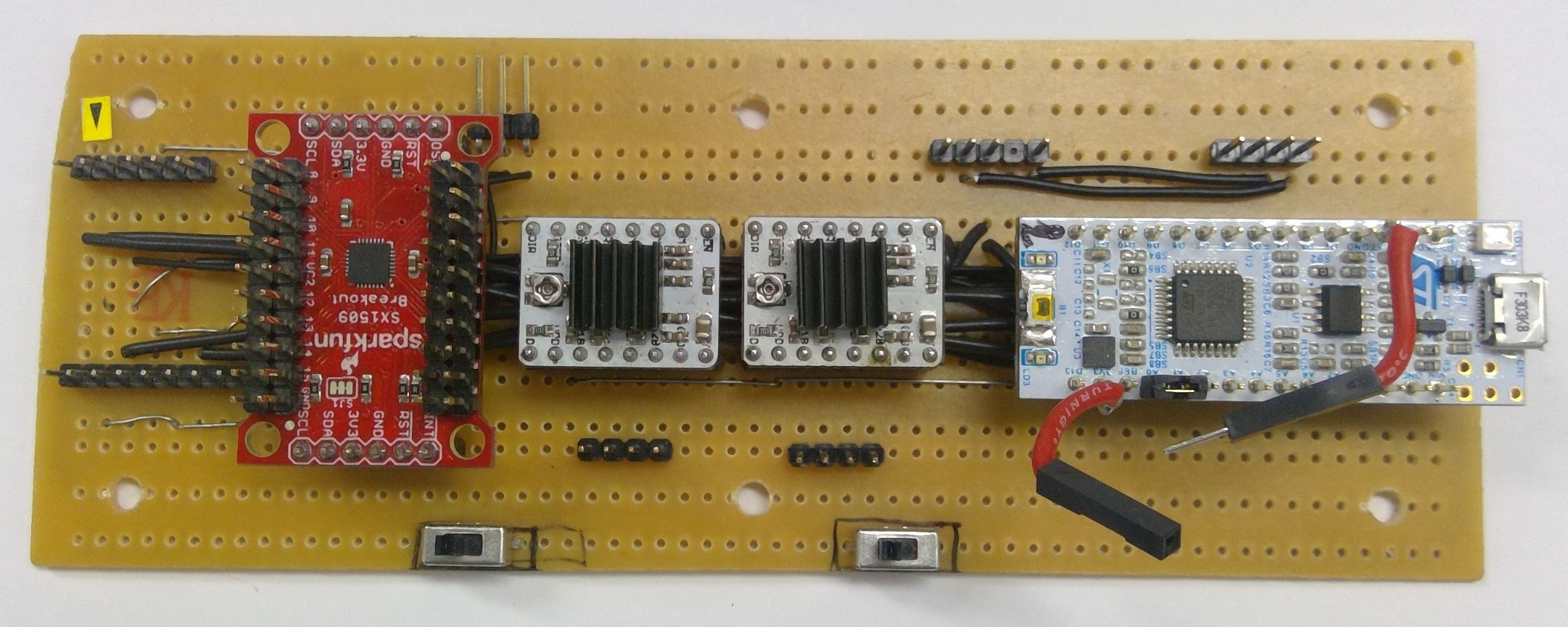
|  |  |  |  |
| --- | --- | --- | --- |
| SCL | SDA | GND | 5 V Servos |

Noter que le bus I²C est en 3.3 V : ne **jamais** monter de pull-ups vers le 5 V Servos !

Ce bus I²C est également utilisé par le module SX1509 présent sur la carte, et sous lequel se trouvent deux pull-up fortes : à prendre en considération si d’autres esclaves I²C doivent être ajoutés dans le futur.

### Moteurs Pas à Pas

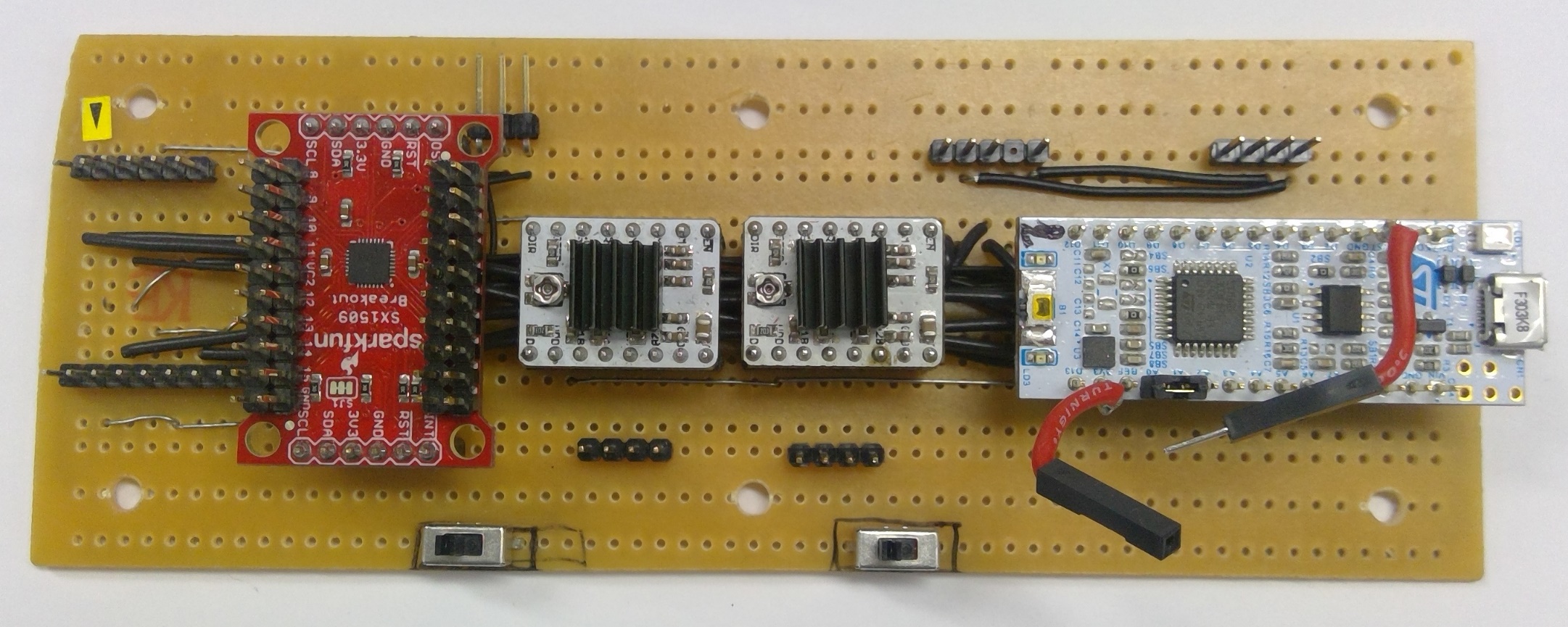
Chaque moteur vient se brancher sur un header à 4 broches situé près du module « Stepperstick » correspondant :



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Moteur Horizontal (Y) | | | |  | Moteur Vertical (Z) | | | |
| A1 | A2 | B1 | B2 |  | A1 | A2 | B1 | B2 |

### Entrées Capteurs TOR

Un header à 6 broches comporte trois GPIO du STM32 :



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| PA10 | PA9 | PA8 | 3.3 V | GND | 5 V (Servos ?) |

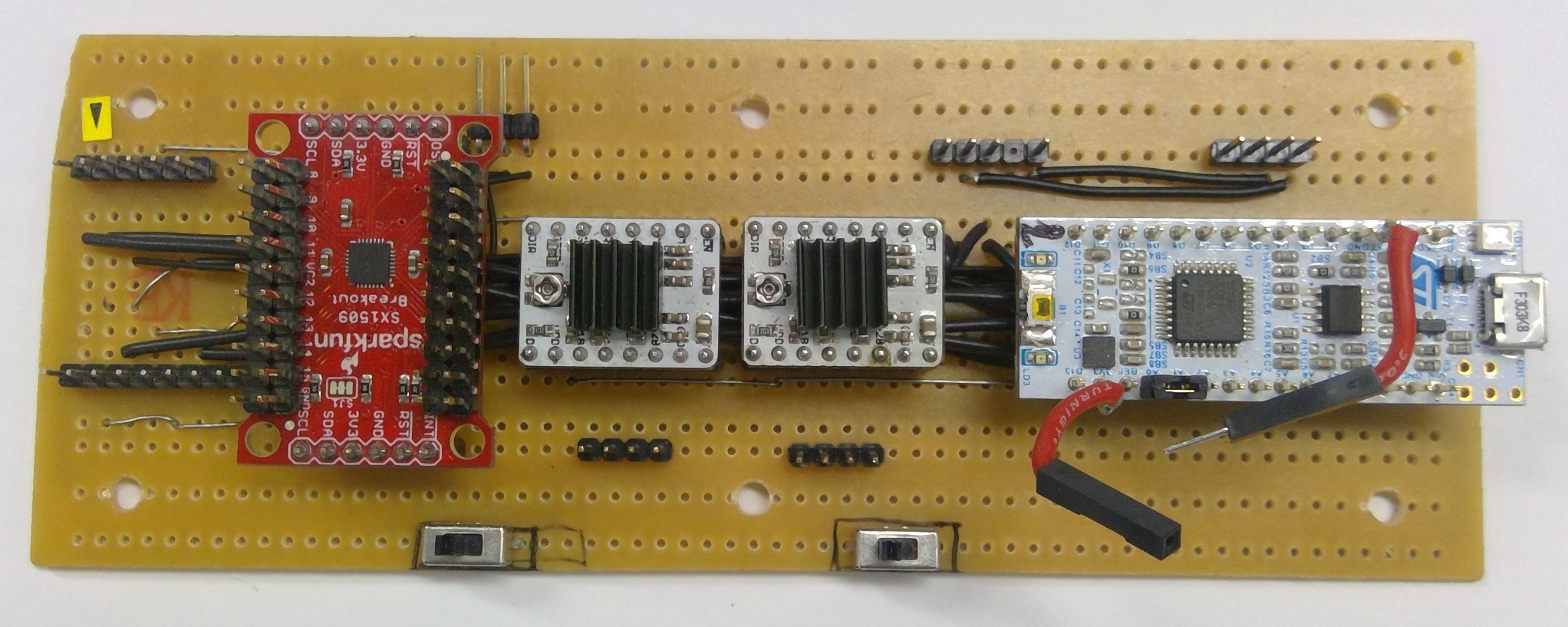
Ces trois GPIO sont utilisées comme entrées pour les capteurs suivants :

* PA8 : Capteur à ultrason de déclenchement de l’expérience
* PA9 : Switch de limite, axe horizontal
* PA10 : Switch de limite, axe vertical

Note : ce connecteur était initialement prévu pour relier une caméra JeVois A33. Les broches PA9 et PA10 peuvent être multiplexées vers l’UART1 du STM32.

### Sorties Servomoteurs

Outre les servos de la figurine, pilotés via le module PCA9685, le STM32 peut piloter directement trois servomoteurs via un header à 9 broches :



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Servo 3 | | | Servo 2 | | | Servo 1 | | |
| GND | 5 V | PWM | GND | 5 V | PWM | GND | 5 V | PWM |

Source des PWM :

|  |  |  |
| --- | --- | --- |
| Servo | Broche STM32 | Multiplexage |
| Servo 1 (Electron) | PA6 | TIM3\_CH1 |
| Servo 2 (Toit) | PA7 | TIM3\_CH2 |
| Servo 3 | PB0 | TIM3\_CH3 |

## Contrôles

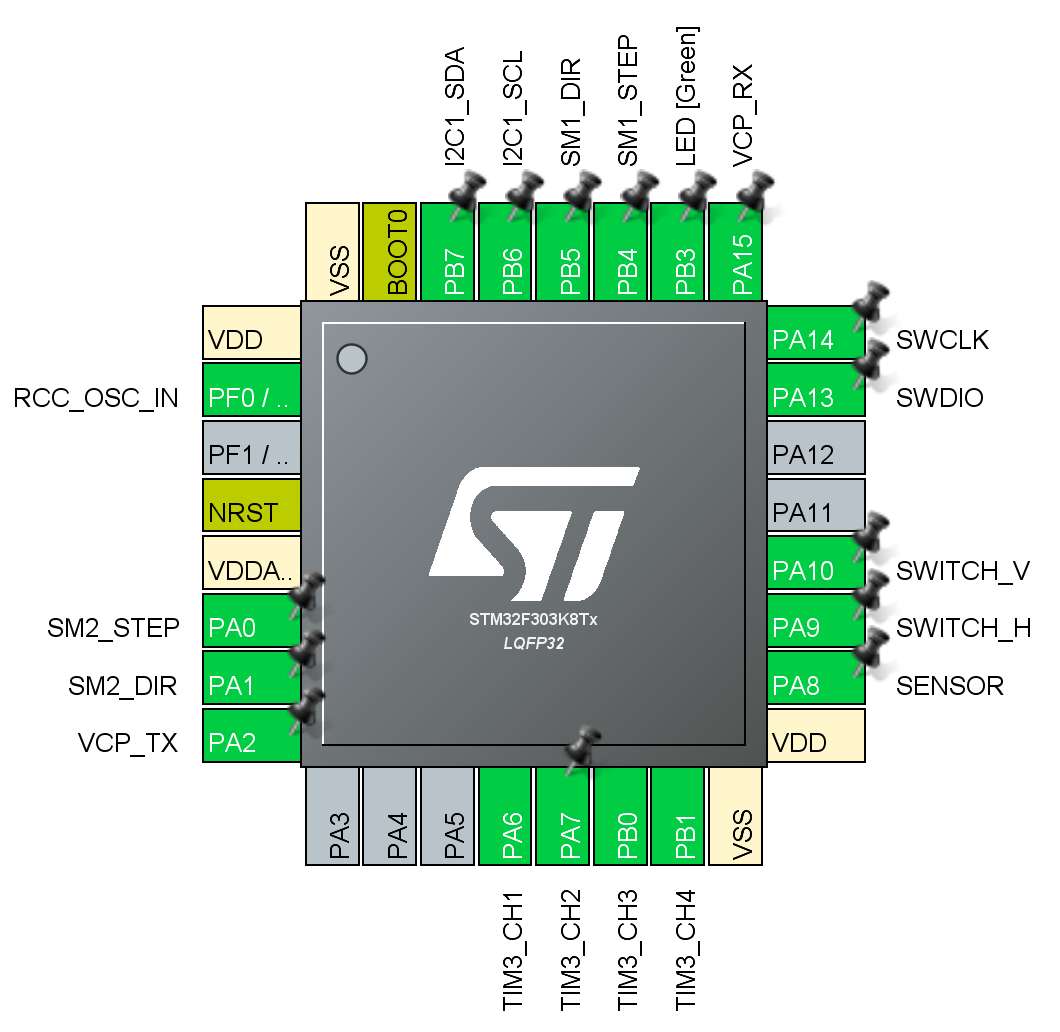
La carte comporte deux interrupteurs bistables connectés chacun entre la masse et une broche GPIO du STM32.

### Sens Electron

### Mise en Sécurité (Rangement)

## Multiplexage du STM32

Tel que défini dans le fichier projet CubeMX « Experience.ioc » dans la racine de ce projet :



# Logiciel

## Commande de l’Enceinte Bluetooth

L’enceinte Bluetooth est commandée par application de sa tension d’alimentation, qui est générée à partir du rail 12 V par un convertisseur DC-DC. Ce convertisseur peut être allumé et éteint depuis « main.c » grâce à deux macros :

* BUCKON
* BUCKOFF

## Commande de la LED du Module Nucléo

Pour aider au test, la LED verte du module Nucléo peut être allumée et éteinte depuis « main.c » grâce à deux macros :

* LEDON
* LEDOFF

## Contrôle des Servos de Expérience

L’expérience comporte deux servos :

* Démarreur de l’électron
* Déploiement du décors (le « toit »)

Ces deux servos sont pilotés directement pas le STM32 en utilisant son timer TIM3 en mode PWM. Une troisième sortie servo est disponible, et moyennant l’ajout d’un connecteur, une quatrième sortie peut être rajoutée.

### Configuration du Timer

TIM3 utilise une horloge 72 MHz et doit générer un signal 50 Hz pour les servos. Etant donné le peu de précision des servos RC, afin de simplifier la programmation on choisit arbitrairement une gamme 8 bits pour indiquer la position du servo, qui doit correspondre à un délai de 0 à 1 ms. Le créneau PWM à générer doit durer de 1 à 2 ms, et donc la valeur entrée dans la PWM pour 2 ms sera de 512. La période de PWM devra donc durer 5120 coups d’horloge pour atteindre 20 ms.

72 MHz / 5120 / 50 Hz = 281.25, arrondi à 280 : cette valeur est utilisée comme prescaler pour TIM3. La valeur de période est 5120. Chaque canal PWM est configuré avec une valeur par défaut médiane (384) qui doit correspondre à la moitié de la course du servo.

Au reset, les PWM ne sont pas activées : elles doivent être déclenchées par un appel à la HAL, qui est exécuté dans la fonction « main » juste avant la première boucle infinie.

### Servo « Electron »

L’électron peut être situé à droite ou à gauche, par conséquent le servo chargé de le démarrer droit effectuer une petite séquence de mouvement « centre-droite-gauche-centre »

### Servo « Toit »

## Contrôle des Servos et LED de la Figurine

<PCA9685>

## Contrôle des LED du Décors

<SX1509>

## Contrôle des Moteurs Pas à Pas

### La Fonction « move »

La fonction « move » dans « main.c » est une fonction bloquante de coordination des mouvements sur des moteurs pas à pas de la figurine.

### Les Timers

Les moteurs pas à pas sont pilotés chacun par un timer (pas) et une GPIO (direction) :

* SM1\_STEP : TIM16\_CH1
* SM1\_DIR : PB5
* SM2\_STEP : TIM2\_CH1
* SM2\_DIR : PA1

Les canaux 1 des timers sont configurés en mode « output compare », la sortie est pilotée en mode « toggle » afin de générer des signaux carrés. Le mode toggle implique que la fréquence générée sera deux fois moindre que celle commandée au timer.

L’interruption de chaque timer est activée. L’ISR de chaque timer sert à décompter le nombre de pas effectués par chaque moteur.

Les ISR sont dans le fichier « src/stm32f3xx\_it.c » :

* void TIM1\_UP\_TIM16\_IRQHandler(void)
* void TIM2\_IRQHandler(void)

Leur code est similaire.

Les variables globales suivantes sont aussi déclarées (et exportées) dans ce fichier source :

uint8\_t sm\_busy = 0;

uint16\_t sm1\_steps;

uint16\_t sm2\_steps;

Comme on utilise des timers et des interruptions, une variable d’état est nécessaire pour permettre au code d’attendre la fin d’un mouvement. « sm\_busy » est chargée à 2 au début d’un mouvement et est décrémentée par chaque ISR quand un moteur arrive au bout de ses pas. Quand « sm\_busy » vaut 0, un nouveau mouvement peut commencer.

« sm1\_steps » est le nombre de pas que le timer TIM16 doit générer. Il est décrémenté à chaque exécution de l’ISR et comme on utilise le mode toggle, « sm1\_steps » doit être préchargé avec le **double** du nombre de pas désiré. Quand cette valeur atteint zero, l’ISR désactive le timer et ainsi met fin à la génération de signaux. Même chose pour « sm2\_steps » et TIM2.

Cette architecture logicielle permet d’implémenter facilement un contrôle des deux moteurs pas à pas qui ne bloque pas l’exécution du code. La complexité réside dans la configuration et le lancement des timers pour accomplir un mouvement désiré. C’est l’un des rôles de la fonction « move » dans « main.c »

# WIP

Je souhaite créer une fonction de déplacement synchronisé de tous les actionneurs. Les actionneurs sont :

* Deux moteurs PaP pilotables par GPIO ou timer.
* Des servos pilotés via bus I²C (PCA9685).

Les servos pilotés directement par le STM32 ne sont pas inclus dans cette fonction.

L’idée est de passer un vecteur de positions à atteindre sur chaque axe dans un temps donné. La fonction sera bloquante jusqu’à exécution du mouvement commandé.

La complexité vient des différences de vitesse sur chaque axe, ainsi que du parallélisme entre les moteurs et les servos.

Une grande simplification serait le pilotage des moteurs par timer. SM2\_STEP peut être multiplexée à TIM2\_CH1, et SM1\_STEP à TIM16\_CH1. Ces deux sorties supportent le mode « output compare ».

Mon but est de les utiliser pour générer un train d’impulsion avec un nombre d’impulsion et une fréquence programmables et indépendents. En mode « output compare » je dois pouvoir générer une fréquence spécifique contrôlée par la période du timer.

Idée : je devrais pouvoir utiliser un timer pour implémenter la durée du mouvement, plutôt que de fixer le nombre d’impulsions pour chaque axe. Moins précis mais plus simple à implémenter.

Il se trouve que le F303K8 n’a qu’un seul timer « full-feature », TIM1, qui n’est pas encore utilisé. Celui-ci inclut un compteur de répétitions et la génération de signal trigger.

L’approche la plus pratique pour piloter les PaP va être l’algo de Bressenham, cadencé par une interruption de timer basique (**TIM6**) dont la fréquence sera choisie en fonction de la plus grande distance à couvrir (H ou V) et du temps imparti pour la couvrir. L’ISR de ce timer gardera le compte du nombre de pas effectué dans une variable globale.

La première étape va donc consister à vérifier que je peux faire fonctionner cette IRQ comme je l’entends. Essai rapide : mes timers sont clockés à 72 MHz, mes PaP font 200 pas/tour et 8mm par tour. Un mouvement de 1mm correspond donc à 25 pas. La gamme de vitesses est fixée arbitrairement de 1 mm/s à 100 mm/s, soit une fréquence de pas de 25 à 2500 Hz. Mettons 24 à 3600 pour simplifier les calculs.

Pour produire 24 Hz à partir de 72 MHz j’ai besoin d’un rapport de 3 millions, et pour 3600 Hz un rapport de 20000. Le prescaler de TIM6 est sur 16 bits, ainsi que la période. Je veux la gamme dynamique la plus large sur la période, disons de 0 à 60000, où 60000 correspondrait au rapport de 3 million. 3M/60K = 50, ce qui sera la valeur du prescaler.

Donc, si je met une période de 1, j’aurai une fréquence de 72 MHz / 50 = 1.44 MHz. Pour une période de 60000 j’aurai une fréquence de 24 Hz. Pour obtenir 3600 Hz il me faudra une période de 400.

Je compte passer comme arguments une distance (en mm) et une vitesse (en mm/s). Je dois pouvoir facilement convertir la vitesse en période de timer. Je sais que mon timer aura une fréquence de comptage de 1.44 MHz et que 1 mm = 25 pas. Donc :

V = 1.44 MHz / 25 / P = 57600 / P

Donc :

P = 57600 / V.

Vérifions :

Si V = 1 mm/s, P = 57600, ce qui produira 72 MHz / 50 / 57600 = 25 Hz, effectivement 1 mm/s

Si V = 100 mm/s, P = 576, ce qui produira 72 MHz / 50 / 576 = 2500 Hz, effectivement 100 mm/s

En parallèle, le pilotage des servos présente un challenge, il est nécessaire d’écrire 4 octets par servo (ou LED) et avec un total de 8 canaux, on a donc 32 octets à envoyer pour mettre à jour la position des servos. Cela va nécessiter 8 ms environ.

Je vais tenter d’utiliser le DMA.

### Deuxième Jour

Idée : plutôt que d’utiliser le DMA, je pourrais envoyer un (ou plusieurs) octet I²C à chaque interruption sur TIM6.

Un problème de jumper sur la Nucléo la faisait tourner trop lentement. En pratique, l’envoi des données en mode burst pour 8 servos (PCA9685) prend moins d’une milliseconde.

Cela peut donc se faire à une fréquence de plus de 1 KHz, hélas pas assez rapide pour être fait en même temps que l’ISR TIM6.

Pour les PaP, je pense que le plus simple est d’utiliser deux timers (TIM2 et TIM16) pour générer les impulsions de commande de pas. Les ISR de ces timers compteront les pas à effectuer.

La procédure de commande sera donc :

* Attente de retombée du bit « busy »
* Détermination des directions en Y et Z => signaux SM1\_DIR et SM2\_DIR
* Stockage du nombre de pas dans des variables globales
* Calcul des périodes de chaque timer pour obtenir la vitesse désirée
* Réglage des timers
* Démarrage des timers
  + Allumage du bit « busy »
* Extinction des timers par leurs ISR respectives, qui décomptent le nombre de pas
  + Extinction du bit « busy »