

|  |  |  |
| --- | --- | --- |
| **Auteur :** | **Encadrant :** | **Année:** |
| Aymeric Maurice Vincent Desnos | Pierre Gaucher | DII3 |

Rapport du projet de développement

Ce document caractérise le déroulement de notre projet de développement

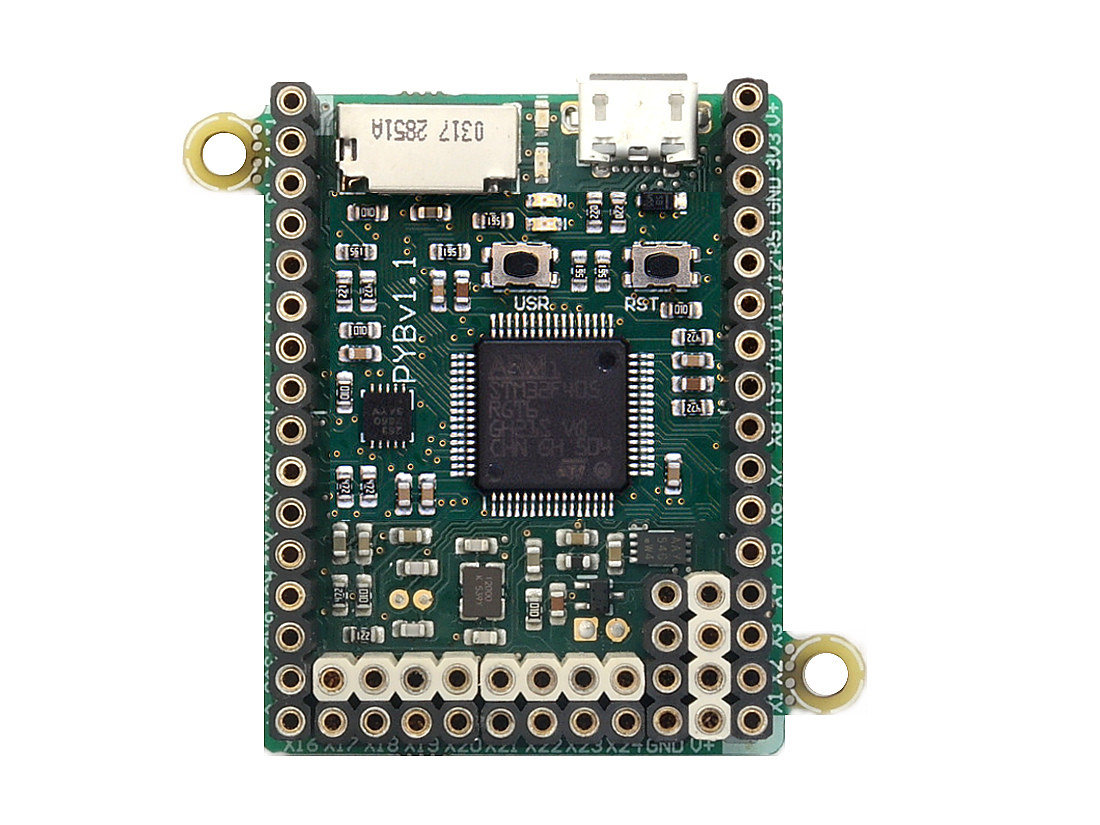


Table des matières

[Introduction 4](#_Toc514519672)

[Contexte 5](#_Toc514519673)

[Cahier des charges 5](#_Toc514519674)

[Faisabilité 6](#_Toc514519675)

[Comparatif et description des outils de développement 8](#_Toc514519676)

[Format du livrable 9](#_Toc514519677)

[Choix du support 9](#_Toc514519678)

[Forme du tutoriel 9](#_Toc514519679)

[Contenu du livrable 11](#_Toc514519680)

[Description et utilisation des périphériques reçus 11](#_Toc514519681)

[LEDs 11](#_Toc514519682)

[Accéléromètre 11](#_Toc514519683)

[Moteur CC 11](#_Toc514519684)

[Exemple: 12](#_Toc514519685)

[Servomoteur 13](#_Toc514519686)

[Périphériques I2C 13](#_Toc514519687)

[Ressources internes Pyboard 13](#_Toc514519688)

[Mise à jour Firmware/Drivers sous Windows 14](#_Toc514519689)

[Mise à jour Firmware/Drivers sous Linux 15](#_Toc514519690)

[Gestion de projet 16](#_Toc514519691)

[Plan de réalisation 16](#_Toc514519692)

[Gestion du temps 16](#_Toc514519693)

[Planning 16](#_Toc514519694)

[Aspect financier 18](#_Toc514519695)

[Problématique 18](#_Toc514519696)

[Bilan 19](#_Toc514519697)

# Introduction

Nous avons choisi ce projet dans le but de découvrir un nouveau microcontrôleur, cela présente une opportunité d’élargir notre spectre de connaissance. Le python est un langage en pleine croissance, par sa simplicité et son indentation bien structuré. Il permet d’acquérir de très bon automatisme. Pour nous se former sur du python présente un atout majeur dans notre formation, car ce langage est apprécié par de plus en plus en entreprise et montre une certaine polyvalence.

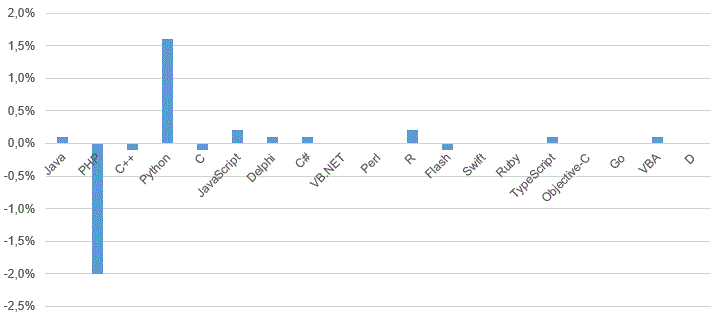


Figure 1:taux de fréquentation des rubriques associées aux différents langages de programmation en 2015 et 2016

L’intention de ce projet est aussi de nous faire découvrir le “Micropython” : il s’agit une implémentation logicielle du langage de programmation Python 3 , écrit en C , optimisé pour fonctionner sur un microcontrôleur. MicroPython est un compilateur Python complet et une exécution qui s'exécute sur le matériel du micro-contrôleur. Avec ceci, nous verrons aussi les différentes librairies du micro-contrôleur (I2C, ADC, GPIO, Accéléromètre ...), ainsi que les ressources internes disponibles sur ce type de carte. Le micro-contrôleur que l’on va utiliser est la “Pyboard”.

# Contexte

Ce robot aura pour objectif de faire découvrir à un public jeune, avec un langage simple et intuitif, la robotique d’un point de vue hardware et software. Aussi le partage d’une passion, et permettre de faire découvrir ce que l’on peut réaliser dans des filières comme en DII ou DI à Polytech Tours. Pour nous, ce projet s’inscrit dans notre cursus et représente une partie conséquente de notre UE Informatique. Il constitue aussi un moyen de travailler dans un domaine qui nous correspond plus que sur les matières classiques. Ce premier projet est en quasi total autonomie pour nous en apprendre un peu plus sur la gestion de projet et nous faire comprendre l’importance des outils et compétences nécessaires à mettre en œuvre pour réussir le projet. Notre client, Pierre Gaucher, nous laisse une grande liberté pour le coté rédactionnel du projet. Cependant, ceci doit suivre une démarche ingénieur comme on l’a vu en cours (mais aussi ce que nous n’avons pas encore vu).

# Cahier des charges

Le cahier des charges définit avec notre responsable de projet M. GAUCHER est définit avec les objectifs et les contraintes suivantes :

Il sera possible de :

* Pouvoir exploiter le tutoriel pour qu’un non initié puisse utiliser les ressources internes de la carte PyBoard
* Pouvoir exploiter le tutoriel pour qu’un non initié puisse utiliser des périphériques externes avec la carte PyBoard

Contraintes :

* Au moins un périphérique I2C
* Rendre compte de toutes les ressources disponibles de la carte
* Au moins un moteur

# Faisabilité

Pour étudier la faisabilité, nous avons eu plusieurs documents à notre disposition sur internet et plusieurs sources :

* D’abord celle du **constructeur** : <https://micropython.org/>

Il nous permet de réaliser plusieurs test sans aucune ressource électroniques. On peut utiliser différent périphériques comme on peut voir ci-dessous :

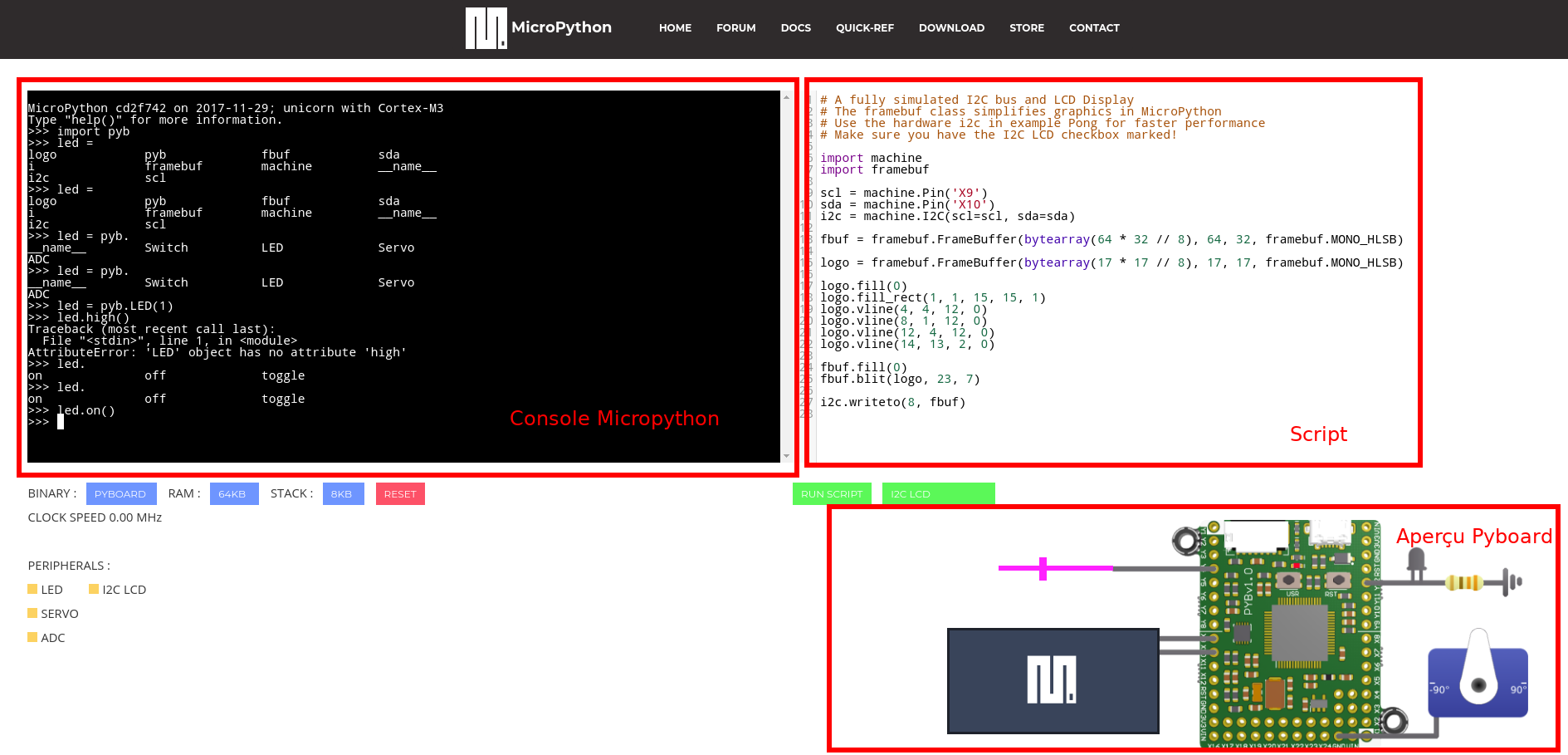


Figure 2: Aperçu Micropython Online

Ce site nous à été utile car au début du projet nous n’avions pas encore la PyBoard à notre disposition. On dispose d’un interface intuitive dans laquel on a:

1. Une console REPL pour voir les librairies embarquées et tester des commandes simple (comme allumer une LED)
2. Un éditeur de texte pour réaliser ses propres scripts, (des exemples sont mis à notre disposition), et les lancer.
3. Un aperçu (Fritzing like), de la Pyboard avec les composants que l’on choisi de connecter.

* Puis la **documentation** : <http://docs.micropython.org/en/latest/pyboard/>

Elle nous a permis de valider beaucoup de chose sur la faisabilité du projet : des librairies complète qui ouvrent pas mal de possibilité sur la carte en terme de ressources logiciels.

* Comme c’est open source, le **dépot Github** : <https://github.com/micropython/micropython>

Ce dépôt permet d’avoir accès au code source en C de Micropython, des drivers, des librairies diverses, des détails sur les ports, etc. En cas de problèmes spécifiques c’est vers ce lien que doit se tourner

* Un **livre** : “Python for Microcontrollers: Getting Started with MicroPython” de Donald Norris

Voici une liste de ce que peut fournir la carte MicroPython en réponse aux attentes du cahier des charges:

* Gestion des bus I2C, SPI, UART, CAN
* Conversion analogiques numériques (CAN et CNA)
* Timer, et interruptions
* Possibilité de stockage de données dans une carte micro SD
* Gestion de PWM
* 1024KiB flash ROM
* GPIO

Avec tous ceci, on peut valider la faisabilité du projet, car les ressources disponibles sur la carte sont suffisantes pour utiliser plusieurs périphériques. De plus, on ne veut pas voir si le microcontrôleur *Pyboard* est compatible, mais on veut voir les ressources que l’on peut exploiter avec ce type de cartes.

# Comparatif et description des outils de développement

Pour trouver l’outil de développement le plus adapter, la première chose qu’on a fait est de regarder la documentation MicroPython. Le fonctionnement de la Pyboard pour implémenter un code à l’intérieur diffère des autres microcontrôleurs. D’abord il n’y a pas d’IDE, ni aucuns logiciels spécifiques. Cependant la Pyboard garde la même logique que lorsqu’on veut exécuter un code python classique :

* On peut utiliser le fameux *main.py* directement disponible parmi les fichiers sur la Pyboard. Pour cela on peut choisir n’importe quel éditeur que l’on veut.
* Ou alors on peut choisir le MicroPython REPL(Read-Eval-Print Loop) prompt: il s’agit d’un environnement de programmation interactif sur un terminal de commandes. Ce terminal est indispensable pour determiner un script.

Problèmes rencontrés: on a recherché un IDE, pour pouvoir exécuter un code et voir directement ce qui ce produit avec un terminal comme en C. Cependant en python cela ne fonctionne pas pareil. Le script et le REPL sont deux choses différentes. Seul, le REPL nous permet de voir le comportement de notre script.

Solution: Nous avons finalement choisi d’utiliser Atom, un éditeur de texte qui dispose de nombreuses extensions. C’est un éditeur de texte très utilisé est très intuitif. DE plus, il est possible d’intégrer le REPL directement dans l’éditeur, ce qui est pratique. Le langage MicroPython est encore très jeune et de nouveaux outils de développement vont vite arriver.

L’éditeur se rapproche donc très fortement de MicroPython Online (cf annexe).

# Format du livrable

## Choix du support

Il existe des centaines de façon de faire un tutoriel. Parmi les plus connus, le format vidéo et le format écrit:

Le format vidéo est efficace quand il s’agit de faire découvrir à une personne qui ne connait vraiment rien sur le sujet. C’est beaucoup plus captivant que le format écrit donc on a toujours tendance à préféré ce format. Or dans notre cas se tutoriel s’adresse plutôt à nous même ou à d’autre étudiant qui suivent le même cursus. On réalise donc un livrable pour des personnes qui ont déjà quelques notion de base (ex: connaissance sur la PWM, un servomoteur, un bus I2C ou SPI, etc.). Sachant cela, le format écrit est plus intéressant car à partir d’une table de matière ou un sommaire, on peut directement cibler une ressource de la carte qui va nous intéresser. Par exemple si on veut configurer l’adresse d’un composant I2C, il suffit juste de rechercher le mot clé “adresse”, et on va très vite trouver l’information. Un dernière avantage du support écrit, est qu’on va aller beaucoup plus vite pour la rédaction du livrable, ce qui nous laissera plus de temps pour prendre en main la carte MicroPython.

## Forme du tutoriel

Le tutoriel suit un ordre chronologique, pour découvrir la Pyboard:

* D’abord les caractéristiques de la carte sont détaillées pour comprendre ce qu’on a notre disposition en matériel.
* On explique ensuite comment installer son environnement de travaille, et comment l’utiliser.
* La partie la plus importantes du livrable se découpe en 2 sous-parties :
  + La gestion des ressources matérielles : c’est à dire tous ce qui est déjà présent sur la carte.
  + La gestion des ressources périphériques: c’est à dire, comment gérer des périphériques connecté à la carte.

Ces deux parties ont la forme suivante :

1. On cite le constructeur (une classe) qui permet de gérer la ressource du microcontrôleur.
2. Les méthodes (fonctions) principales de ce constructeur sont chacune citée et détaillée
3. Les paramètres de chaque méthodes sont aussi cité et expliqué (quand nécessaire).
4. Un exemple de code est fourni pour mieux comprendre l’utilisation des fonctions. Afin de bien différencier le code du reste, un certain format leur sont appliqués.

Exemple:

1. **import** pyb
2. **import** os
3. a=a+1
4. Si besoin, des conseils, ou détails important sont ajouté avec la forme suivante :

|  |  |
| --- | --- |
|  | Ceci est une information importante |

1. Dans le cas des périphériques externes à la carte on a choisi d’utiliser le logiciel Fritzing pour réaliser des schémas.
   1. Pourquoi avoir choisis Fritzing ?

Parce que c’est le plus couramment utiliser par le tutoriel sur internet. Donc l’utilisateur ne sera pas perturbé par ce format. D’un simple cou d’œil, on comprend directement ce que c’est. On peut facilement reproduire le schéma. Contrairement aux schémas électriques classiques qui seront pas aussi facile à comprendre donc moins efficace. Surtout que les schémas sont très simples donc les schémas restent clairs et propre.

# Contenu du livrable

## Description et utilisation des périphériques reçus

### LEDs

Les LEDs de la PyBoard sont la toute première tâche effectuée dans notre projet. La PyBoard possède quatre LEDs. Pour les allumer nous nous sommes renseigner sur le site <https://docs.micropython.org> afin de connaitre le constructeur et la méthode correspondant à l’utilisation des LEDs.

1. Recherche sur <https://docs.micropython.org> d’un constructeur (classe) ou d’une méthode (fonction) pour utiliser les LEDs
2. Mise en place d’un programme test
3. Validation de la fonction et du périphérique.

### Accéléromètre

L’accéléromètre est l’une des premières ressources internes que nous avons explorée. Nous avons donc cherché dans les libraires de la cartes, la documentation ainsi que sur internet le moyen de mettre ce périphérique en service. Voici les différentes étapes suivi lors de sa mise en route :

1. Recherche sur <https://docs.micropython.org> d’une méthode pour utiliser l’accéléromètre
2. Recherche dans la librairie de la carte un constructeur (classe) ou une méthode (fonction) correspondant à l’accéléromètre.
3. Mise en place d’un programme test permettant d’allumer trois LEDs de la carte sur les axe x, y et z de l’accéléromètre
4. Validation de la fonction et du périphérique.

### Moteur CC

Un des premiers périphériques que nous avons reçu était un driver de moteur. On a donc récupéré des équipements au service informatique pour élaborer notre environnement de test. Voici les étapes qui ont été suivi :

1. Récupération des header pour souder sur le driver de moteur( il faut savoir que les composants et le PCB nous a été livré mais c’est a nous de souder)
2. Soudage de touts les composants
3. Connexion des moteurs DC
4. Import de la librairie Arduino et des codes exemples fourni avec le driver puis flashage du microcontrôleur
5. Connexion de la carte Arduino au driver et de l’alimentation
6. Test du bon fonctionnement du driver et validation
7. Passage sous python avec la Pyboard

En mettant des traces dans le code, on a pu retrouver les fonctions par lesquels passe le code pour faire avancer un moteur. On a passé du temps à comprendre comment fonctionnait le driver car il n’est pas fait comme la plupart des driver : en effet, celui-ci dispose d’un registre à décalage de 8bits (le composant au milieu de la carte, voir annexe). L’intérêt de ce circuit est d’utiliser moins de GPIO sur le µC.

#### Exemple:

Si on souhaite contrôler 8 leds, avec un registre à décalage, on a seulement besoin d’utiliser 3 GPIO sur le µC.

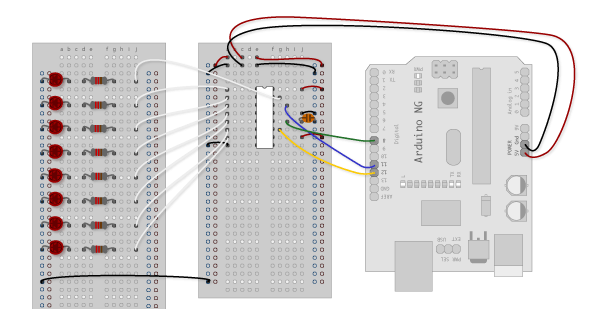


Figure 3 : 8 Bits shift register

Après avoir retranscris la librairie codée en C en python (voir annexe), on a testé la librairie mais rien n’a marché comme prévu. Deux solutions se sont offertes à nous:

* Continuer de faire des tests jusqu’a ce que la librairie soit utilisable
* Changer de driver de moteur

Nous choisissons de changer de driver, ce choix sera expliqué dans les problèmes rencontrés, plus bas dans le rapport. Le nouveau driver est cette fois plus répandu dans le commerce, donc plus commun. Ce critère est important et nous influencera pour le projet électronique.

C’est le driver Ardumoto de sparkfun (voir tutoriel). Après avoir déplacé notre environnement de test on décide cette fois d’alimenter le module avec une alimentation stabilisée de l’école pour être sur de ne pas avoir de problème d’alimentation du driver. Un premier test est effectué une nouvelle fois avec une carte Arduino, puis on passe sur la Pyboard pour faire les mêmes tests. Cette fois ce dernier est concluant, on arrive à contrôler les moteurs en Python. Ce périphérique nous aura appris plusieurs chose:

* Vérifier chaque élément de son environnement de test
* Comprendre le comportement d’un code
* Gérer son temps

### Servomoteur

Ce périphérique nous à été fourni dans l’optique de tester la librairie pour les servomoteurs, en Micropython. L’utilisation de ce type de moteur est vraiment simple dans le cas d’un Pyboard, car elle fourni assez de courant pour observer le comportement du servomoteur en le connectant directement sur le µC. Cependant en réalisant un test avec le servomoteur, on s’est rendu compte qu’il était à rotation continue, c’est à dire qu’il n’est pas commandable en angle mais bien en vitesse de rotation. Pour confirmer, notre constatation nous avons testé avec un µC Arduino et sa librairie “Servo.h” dédiée, ce qui a mené au même résultat.

### Périphériques I2C

Pour expérimenter de l’I2C sur la PyBoard, nous avons reçu un baromètre BMP085. Utiliser un périphérique I2C avec Python est vraiment simplifié. En effet après plusieurs recherches sur internet et dans la documentation de la carte nous avons trouvé une librairie spécifique à l’I2C déjà développé par le constructeur. Ces librairies permettent par exemple de renseigner directement l’adresse du périphérique I2C sur 7 bits afin de laisser une fonction (appartenant à la librairie) changer le 8ème bit en 0 ou en 1 pour pouvoir se configurer en écriture ou en lecture. L’étape d’après était de se renseigner dans la documentation du baromètre sur les trames et les adresses des registres à envoyer dans le baromètre. Après avoir écrit l’algorithme, nous avons écrit un programme de test pour essayer de récupérer les données du baromètre. Parallèlement nous avons utilisé un analyseur logique pour vérifier que nous envoyons les trames voulu.

### Ressources internes Pyboard

Exploiter les ressources d’un µC en micropython est vraiment plus facile que sur d’autre µC. On dispose d’une console REPL, où on peut tester des instructions, et comprendre rapidement le fonctionnement, on se rend compte que le langage Python reste très intuitif à défaut des performances qui s’en voient affectée. Dans le cas où une entreprise souhaite réaliser rapidement un prototype, la Pyboard va devenir un sérieux concurrent, car malgré sont prix élevé, le temps qu’elle fait gagner n’est pas négligeable et rembourse largement son prix. Les ressources internes sont, comme on peut le voir sur le tutoriel, très rapide à prendre en main.

De plus, les documentations sont bien fournies et aussi il y a une communauté conséquente

### Mise à jour Firmware/Drivers sous Windows

Pour connaître la méthode à utiliser afin de télécharger de nouveaux drivers sur la carte, nous avons tout d’abord effectué nos recherches dans la documentation de Python et sur internet. En effet la manipulation à effectuer est simple, il faut tout simplement copier/coller le fichier du driver dans la carte quand celle-co agit comme une clé USB sur votre ordinateur.

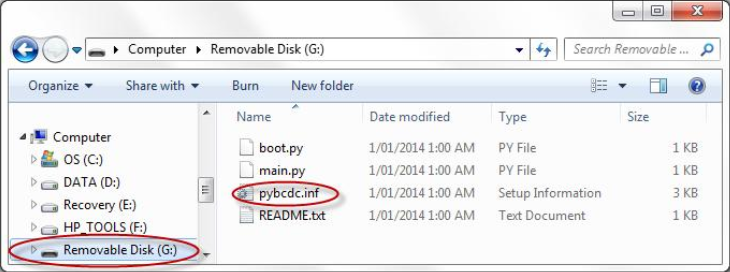


Figure 4 1er méthode

Une autre méthode consiste à effectuer une recherche en ligne par le biais de l’interface Windows « gestionnaire de périphérique ».

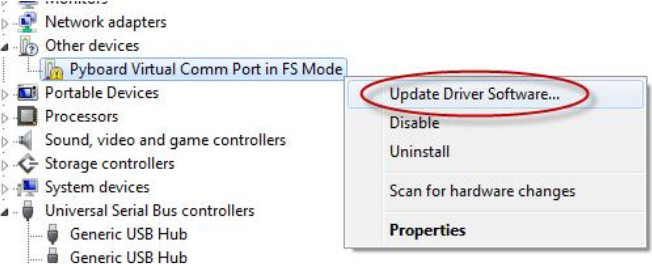


Figure 5 2ème méthode

Pour effectuer une mise à jour du Firmware, la tâche est plus complexe. En effet il faut tout d’abord réaliser un pontage sur les pins DFU et 3.3V.

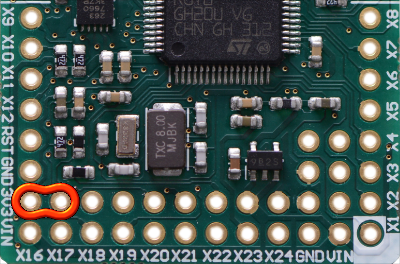


Figure 6 DFU Mode

Une fois le pontage réalisé, la carte se trouve en DFU mode. La première fois que la PyBoard se connectera au PC, Windows essayera d’installer les nouveaux drivers. Jusqu'à ce que les nouveaux drivers soient installés, la PyBoard apparaitra avec une icône « Attention » dans le gestionnaire de périphérique.

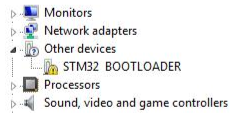


Figure 7 Warning

### Mise à jour Firmware/Drivers sous Linux

Pour réaliser une mise à jour du Firmware sous Linux, la manipulation est bien plus simple que sur Widows. Il faut dans un premier temps utiliser le gestionnaire de paquets pour installer « dfu-util ». Ensuite il faut mettre la PyBoard en DFU mode en réalisant un pontage entre les Pins DFU et 3.3V. Une fois la PyBoard en DFU mode nous pouvons téléverser le nouveau Firmware grâce à la commande « sudo dfu-util –alt 0 –D NomDuFicher.dfu ».Pour finir déconnecter la PyBoard et enlever le pontage.

# Gestion de projet

Suite à l’affectation des projets fin novembre, nous avons très rapidement pris un rendez-vous avec notre responsable de projet, Monsieur GAUCHER, afin d’avoir des informations plus détaillées sur le contenu final qui était attendu mais aussi afin de borner notre projet.

Tout au long du déroulement de notre projet, nous avons essayé de nous tenir au mieux à notre planning mais cette problématique du temps c’est avéré difficile à respecter.

Une autre difficulté majeure qui est apparu au cours du projet est l’éloignement géographique. En effet pendant les périodes entreprises mon binôme et moi n’avions pas moyen de nous rencontrer pour travailler le projet ensemble. Il a fallut trouver des méthodes de travailles à distance comme des « To do list » ou faire des points régulier sur l’avancement du projet.

## Plan de réalisation

Afin de répondre correctement au cahier des charges nous avons dans un premier temps nous avons décidé d’explorer la carte PyBoard et son langage de manière autonome. Pour pouvoir réaliser un tutoriel sur la carte PyBoard il nous fallait bien la connaitre. Nous avons donc exploré la carte et effectué des recherches sur internet pour enfin mettre en commun ce que nous avions apprit. Il y avait des notions à assimiler comme par exemple la notion de constructeur (classe) et de méthodes (fonctions) ou encore apprendre à ce servir de l’accéléromètre de la carte PyBoard grâce à sa bibliothèque intégré.

## Gestion du temps

Le temps est la plus grosse problématique du projet. En effet la difficulté était de travailler sur le projet en période d’école et en période d’entreprise. De plus en période d’entreprise nous ne pouvions nous rencontrer à cause de l’éloignement géographique. Pour pallier à cela, nous avons réalisé un GANTT afin de déterminer les tâches à réaliser chacun de nôtre coté lors de ces périodes. Nous avions aussi mis en place une zone de dépôt en ligne afin de déposer notre travail une fois celui-ci mis à jour, ce qui nous permet de prendre en temps réelle les dernières modifications du projet.

## Planning

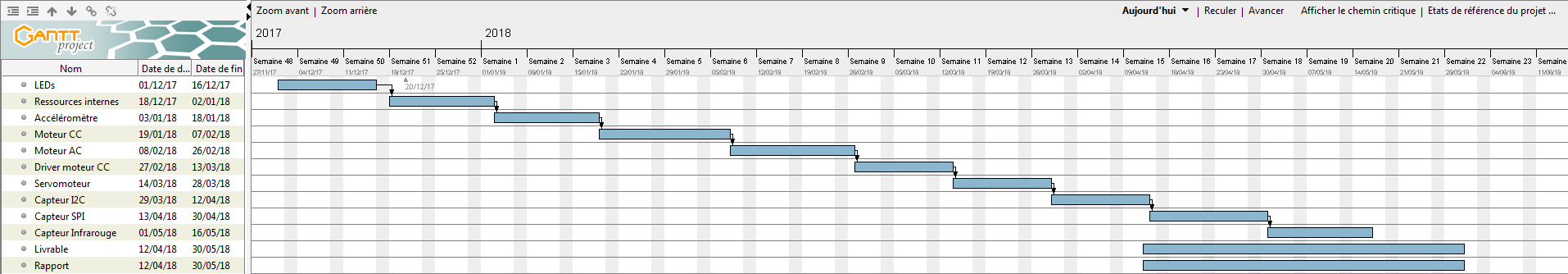


Figure 8 GANTT début du projet

Voici le planning réalisé lors du début de notre projet. Nous pouvons voir des tâches organisé avec un temps déterminé. Toutes les notions que nous pensions aborder dans notre tutoriel y figure. Nous avions pour objectif de travailler sur les notions suivantes :



Figure 9 Tâches réalisées à la fin du projet

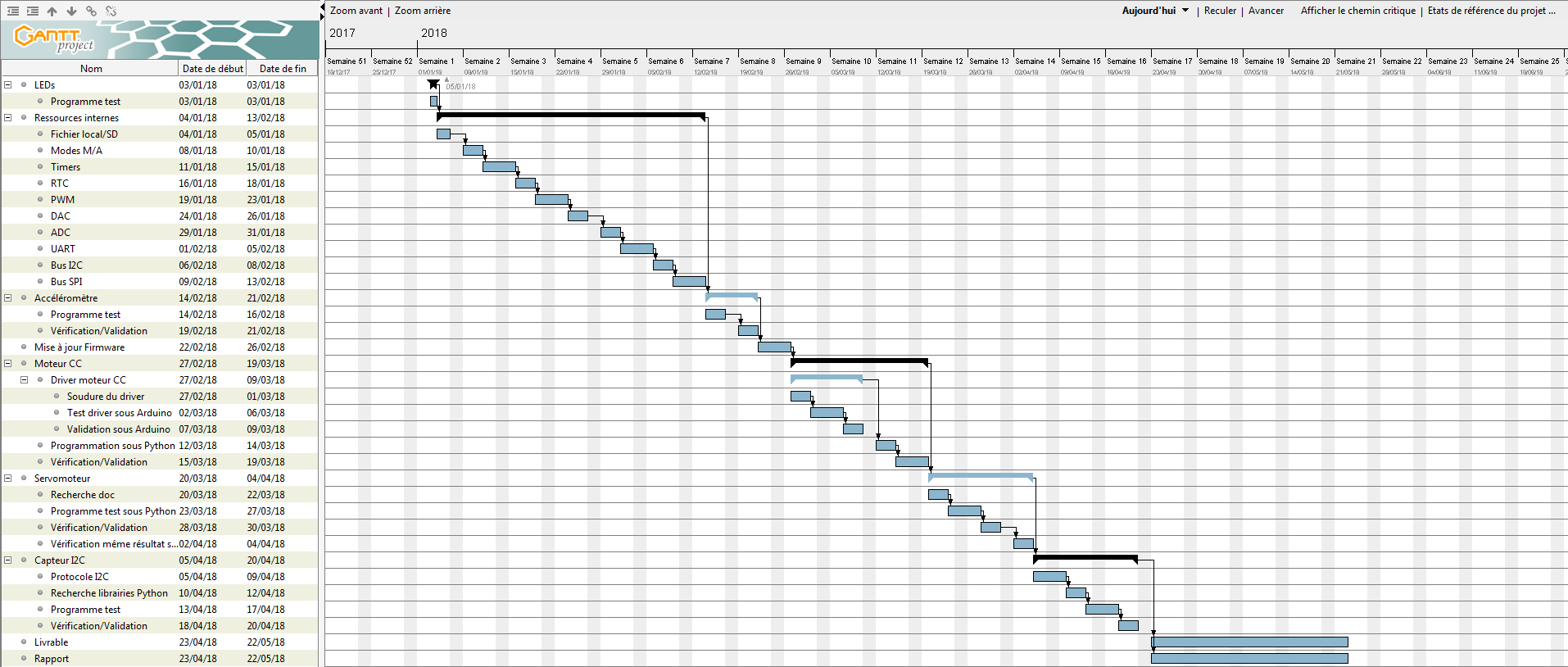


Figure 10 GANTT réel

En réalité voici le planning réel. Nous pouvons voir un GANTT plus conséquent avec plus de tâches que prévu lors du GANTT réalisé au début du projet. Au fur et à mesure de l’avancement du projet, des problèmes et des imprévus nous ont fait prendre du retard sur le planning. C’est pour quoi nous n’avons pas pu explorer l’utilisation des capteurs SPI, Infrarouge ainsi que les moteurs AC sur la carte PyBoard. Ces trois tâches n’apparaîtront pas dans le tutoriel. Ce projet nous à permit de réaliser la difficulté sur la gestion du temps, nous n’avions pas assez anticipé les imprévu, la charges de travail et la durée des tâches.

# Aspect financier

Les ressources allouées à ce projet sont essentiellement d’ordre financier et temporel.

En effet, aucun bon de commande ni achat de matériel n’a été réalisé, en dehors du prix de la carte, de l’ordre d’une quarantaine d’euros.

Le calcul des ressources allouées se base sur un salaire brut de 31 000€/an, soit le salaire moyen d’un ingénieur informatique industriel. En net, cela représente 28 600€, soit 10€ de l’heure.

Pour mener à bien ce projet, XXX heures ont été allouées à ce projet et XXX € dépensés/débloqués.

# Problématique

La problématique du projet concerne l’éducation. En effet le but du projet est de réaliser un tutoriel permettant aux non-initiés de pouvoir apprendre le fonctionnement et le langage d’une carte python (ici la PyBoard V1.1) de manière simple et ludique sur une base robotique. En effet lorsque les utilisateurs devront changer un capteur ou un moteur sur le robot, il sera possible de suivre le tutoriel afin de pouvoir mettre le capteur en service rapidement. Il permettra aussi aux utilisateurs d’être informé et d’utiliser les ressources internes de la PyBoard

# Bilan

Au cours de ce projet, les plus grandes difficultés rencontrées sont avérée être le temps et l’anticipation des charges de travail. En effet nous n’avons pas su anticiper le temps de travail de certaines tâches ainsi que les imprévus apparu durant le projet.

TABLE DES ILLUSTRATIONS

[Figure 1:taux de fréquentation des rubriques associées aux différents langages de programmation en 2015 et 2016 4](#_Toc514519660)

[Figure 2: Aperçu Micropython Online 6](#_Toc514519661)

[Figure 3 : 8 Bits shift register 12](#_Toc514519662)

[Figure 4 1er méthode 14](#_Toc514519663)

[Figure 5 2ème méthode 14](#_Toc514519664)

[Figure 6 DFU Mode 14](#_Toc514519665)

[Figure 7 Warning 15](#_Toc514519666)

[Figure 8 GANTT début du projet 16](#_Toc514519667)

[Figure 9 Tâches réalisées à la fin du projet 17](#_Toc514519668)

[Figure 10: Editeur Atom et REPL 21](#_Toc514519669)

[Figure 11: 8 bits shift register 25](#_Toc514519670)

ANNEXE

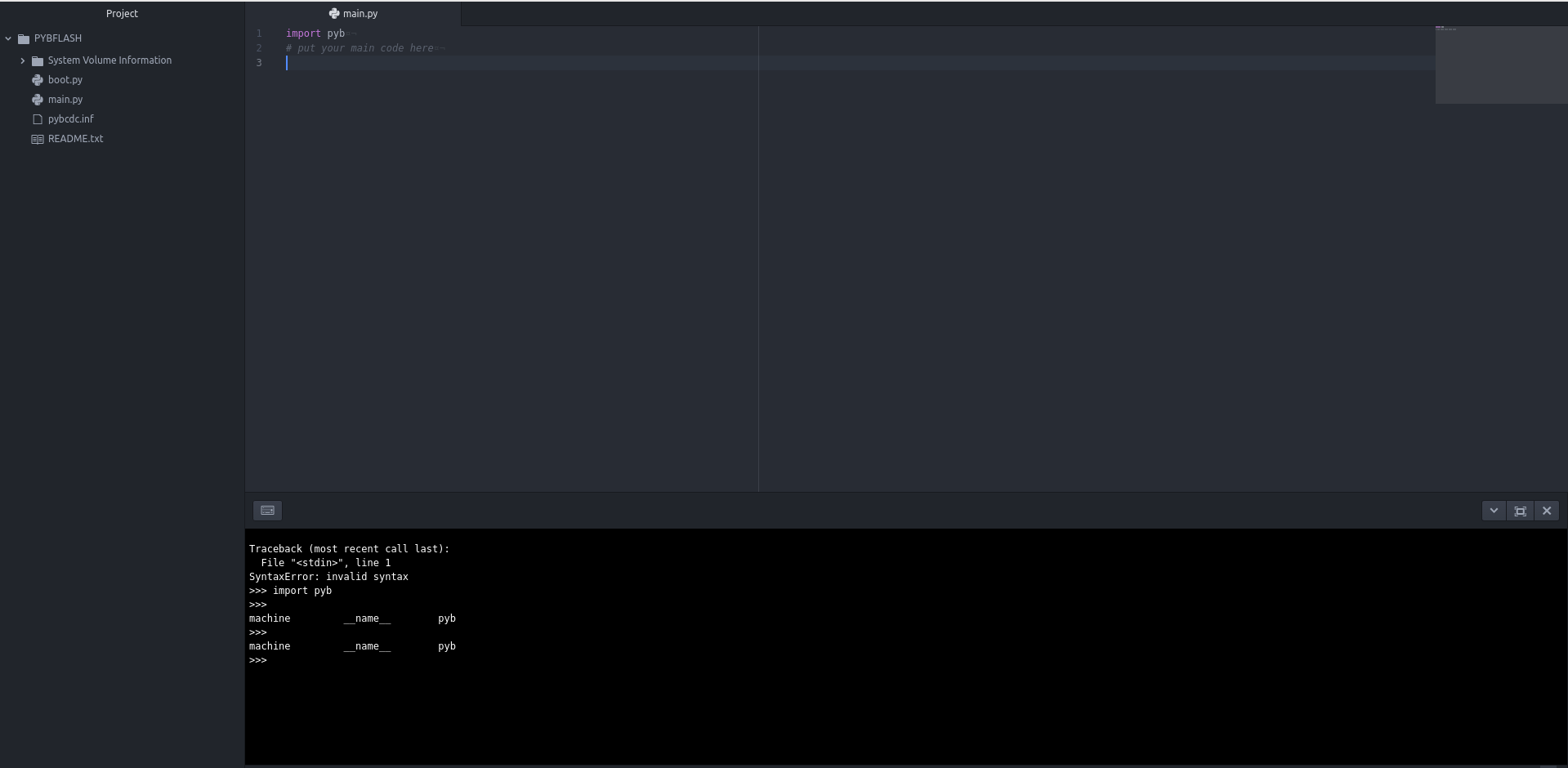


Figure 10: Editeur Atom et REPL

Librairie python driver de moteur avec registre à décalage

1. **import** pyb
2. **import** time
3. # Bit positions in the 74HCT595 shift register output
4. MOTOR1\_A = 2
5. MOTOR1\_B = 3
6. MOTOR2\_A = 1
7. MOTOR2\_B = 4
8. MOTOR4\_A = 0
9. MOTOR4\_B = 6
10. MOTOR3\_A = 5
11. MOTOR3\_B = 7
13. # Constants that the user passes in to the motor calls
14. FORWARD = 1
15. BACKWARD = 2
16. BRAKE = 3
17. RELEASE = 4
19. # Constants that the user passes in to the stepper calls
20. SINGLE = 1
21. DOUBLE = 2
22. INTERLEAVE = 3
23. MICROSTEP = 4
25. DC\_MOTOR\_PWM\_RATE = 8000
27. numOfRegisterPins = 8
28. registers = [1,1,1,1,1,1,1,1]
30. **def** MotorEnable ():
31. #setup des entrées/sorties
32. clearRegisters() # mettre à O le tableau register
33. writeRegisters()# appliquer les changement de register
34. MOTORENABLE.low() # Logique inverser je crois
35. MOTORLATCH = Pin('X1', Pin.OUT\_PP)# Pin 12 sur le shield (RCLK)
36. MOTORCLK = Pin('X2', Pin.OUT\_PP)# Pin 4 sur le shield (SRCLK)
37. MOTORENABLE = Pin('X3', Pin.OUT\_PP) # Pin 7 sur le shield
38. MOTORDATA = Pin('X4', Pin.OUT\_PP)# Pin 8 sur le shield (SER\_Pin)
40. **def** clearRegisters():
41. **for** i **in** range(numOfRegisterPins):
42. registers[i] = 0
44. **def** writeRegisters():
45. #Met et affiche les registre sur les sortie du 74HCT595
46. #Appeler cette fonction aprés avoir mis les valeurs voulues
47. MOTORLATCH.low()
48. **for** i **in** range(numOfRegisterPins):
49. MOTORCLK.low()
50. val = registers[i]
51. **if** val == 1:
52. MOTORDATA.high()
53. **elif** val == 0:
54. MOTORDATA.low()
55. **else**:
56. **print**("Error in variable registers")
57. MOTORCLK.high()
59. **def** setRegisterPin(index, value):
60. registers[index] = value
62. **def** initPWM1 ( f):
63. #generer la PWM sur la pin 11 du shield (PB3)
64. p1 = Pin('X9')
65. tim1 = Timer(4,freq = f)
67. **def** setPWM1(speed):
68. ch1 = tim.channel(1, Timer.PWM, pin=p1)
69. ch1.pulse\_width\_percent(speed)
71. **def** initPWM2 ( f):
72. #generer la PWM sur la pin 3 du shield (PE5)
73. p2 = Pin('Y8')
74. tim2 = Timer(12,freq = f)
76. **def** setPWM2(speed):
77. ch2 = tim.channel(2, Timer.PWM, pin=p2)
78. ch2.pulse\_width\_percent(speed)

81. **def** initPWM3 ( f):
82. #generer la PWM sur la pin 6 du shield (PD6)
83. p3 = Pin('Y4')
84. tim3 = Timer( 4, freq = f)
86. **def** setPWM3(speed):
87. ch3 = tim.channel(4, Timer.PWM, pin=p3)
88. ch3.pulse\_width\_percent(speed)
90. **def** initPWM4 ( f):
91. #generer la PWM sur la pin 5 du shield (PE3)
92. p4 = Pin('Y6')
93. tim4 = Timer( 4, freq = f)
95. **def** setPWM4(speed):
96. ch4 = tim.channel(3, Timer.PWM, pin=p4)
97. ch4.pulse\_width\_percent(speed)
99. **class** DcMotors ():
100. **def** \_\_init\_\_(self, num):
101. self.motornum = num
102. self.pwmfreq = DC\_MOTOR\_PWM\_RATE
103. init()
105. **def** init():
106. MotorEnable()
107. **if** self.motornum == 1:
108. setRegisterPin(MOTOR1\_A, 0)
109. setRegisterPin(MOTOR1\_B, 0)
110. writeRegisters()
111. initPWM1(self.pwmfreq)
112. a = MOTOR1\_A
113. b = MOTOR1\_B
114. **elif** self.motornum == 2:
115. setRegisterPin(MOTOR2\_A, 0)
116. setRegisterPin(MOTOR2\_B, 0)
117. writeRegisters()
118. initPWM2(self.pwmfreq)
119. a = MOTOR2\_A
120. b = MOTOR2\_B
121. **elif** self.motornum == 3:
122. setRegisterPin(MOTOR3\_A, 0)
123. setRegisterPin(MOTOR3\_B, 0)
124. writeRegisters()
125. initPWM3(self.pwmfreq)
126. a = MOTOR3\_A
127. b = MOTOR3\_B
128. **elif** self.motornum == 4:
129. setRegisterPin(MOTOR4\_A, 0)
130. setRegisterPin(MOTOR4\_B, 0)
131. writeRegisters()
132. initPWM4(self.pwmfreq)
133. a = MOTOR4\_A
134. b = MOTOR4\_B
135. **def** run(cmd):
136. **if** cmd == "FORWARD":
137. setRegisterPin(a, 0)
138. setRegisterPin(b, 1)
139. writeRegisters()
140. **elif** cmd == "BACKWARD":
141. setRegisterPin(a, 0)
142. setRegisterPin(b, 1)
143. writeRegisters()
144. **elif** cmd == "RELEASE":
145. setRegisterPin(a, 0)
146. setRegisterPin(b, 0)
147. writeRegisters()
148. **def** setSpeed(speed):
149. **if** self.motornum == 1:
150. setPWM1(speed)
151. **elif** self.motornum == 2:
152. setPWM2(speed)
153. **elif** self.motornum == 3:
154. setPWM3(speed)
155. **elif** self.motornum == 4:
156. setPWM4(speed)

Figure 4: Librairie python driver de moteur avec registre à décalage

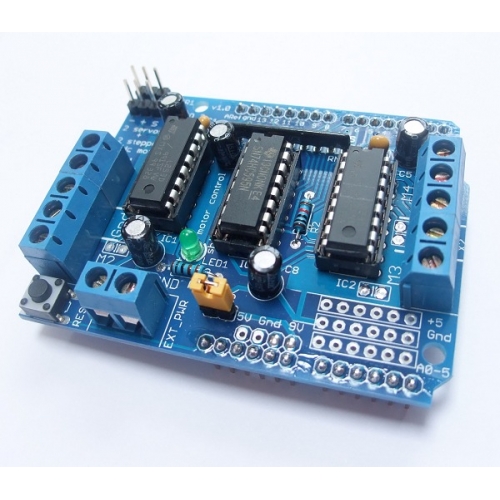


Figure 11: 8 bits shift register