

**DOCUMENTATION DU PROJET BUS DE COM**

**Département SEI**

**Février-Mars 2018**

***Dominant : ISE-OC***

|  |  |
| --- | --- |
| ***Rapport d’équipe*** |  |
|  | ***Li Zhengxi***  ***Jiang Zhuohang*** |

# Introduction :

# Plan :

Table des matières

[Introduction : 2](#_Toc509498598)

[Plan : 3](#_Toc509498599)

[1) Conception général : 5](#_Toc509498600)

[1-1) Matériels utilisés : 5](#_Toc509498601)

[1-2) Explication générale du projet : 8](#_Toc509498602)

[1-3) Présentation de différent Pins : 9](#_Toc509498603)

[2 ) Conception spécifiée : 10](#_Toc509498604)

[2-1) Liste de fichier : 10](#_Toc509498605)

[2-2) Organigramme : 11](#_Toc509498606)

[2-2-a) Master : 11](#_Toc509498607)

[2-2-b) Slave : 13](#_Toc509498608)

[2-3 ) Présentation de différent module : 14](#_Toc509498609)

[2-3-a) UART : 14](#_Toc509498610)

[2-3-b) Partie IR 17](#_Toc509498611)

[2-3-c) SPI : 19](#_Toc509498612)

[2-3-d) Partie servomoteur : 21](#_Toc509498613)

[2-3-e) Partie SAMBot(robot) : 25](#_Toc509498614)

[3) Test unitaire : 28](#_Toc509498615)

[3-a) Communication entre Bluetooth et MSP430G2553 : 28](#_Toc509498616)

[3-b) L’infrarouge donne l’information à MSP430G2553(Master) 29](#_Toc509498617)

[3-c) Communication entre MSP430G2553 et MSP430G2231(SPI) 30](#_Toc509498618)

[3-c-1) Master : 31](#_Toc509498619)

[3-c-2) Slave : 32](#_Toc509498620)

[3-d) MSP430G2231 dirige le mouvement de servomoteur. 33](#_Toc509498621)

[3-e) MSP430G2553 dirige le mouvement du robot. 34](#_Toc509498622)

[4) Gestion de configuration : 37](#_Toc509498623)

[Conclusion : 38](#_Toc509498624)

[Annex : 39](#_Toc509498625)

[Les schémas utilisées : 39](#_Toc509498626)

# Conception général :

# 

## Matériels utilisés :

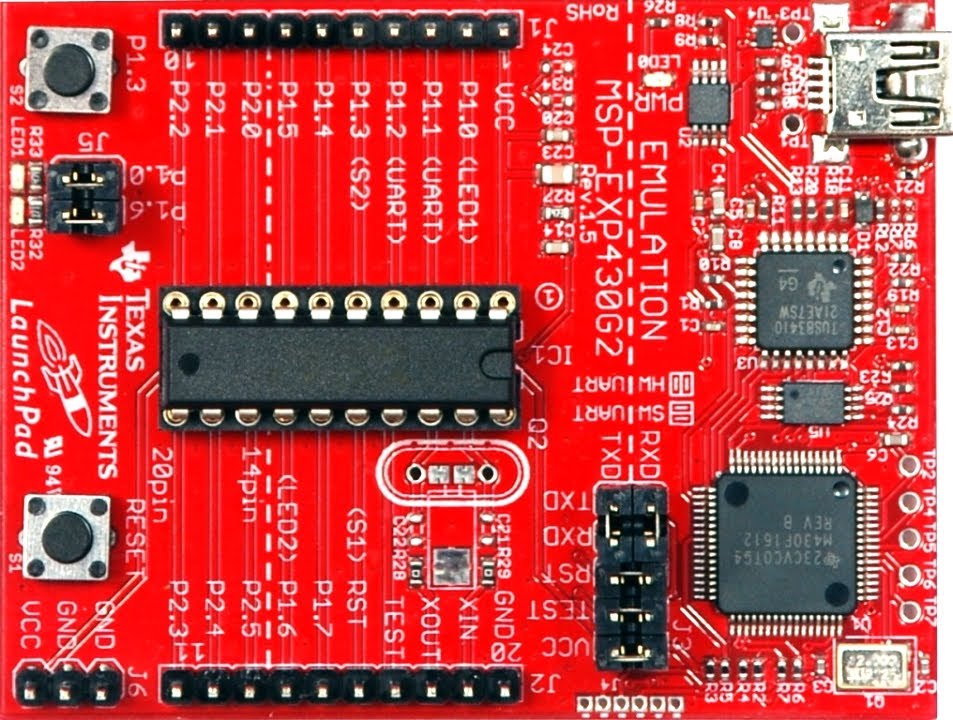


Figure : Lauchpad



Figure : MSP430G2231

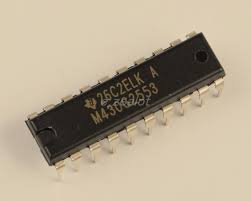


Figure : MSP430G2553



Figure : Servomoteur



Figure : module Bluetooth (installé sur le master)



Figure : L'infrarouge

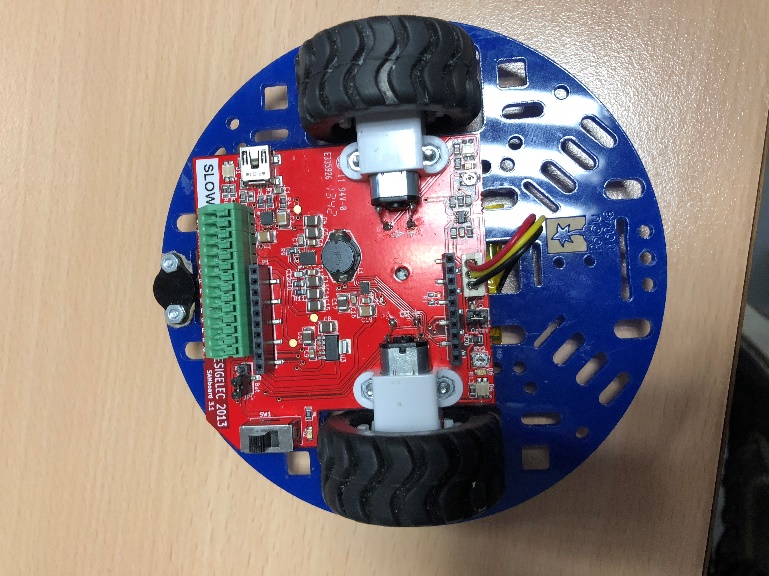


Figure : SAMbot

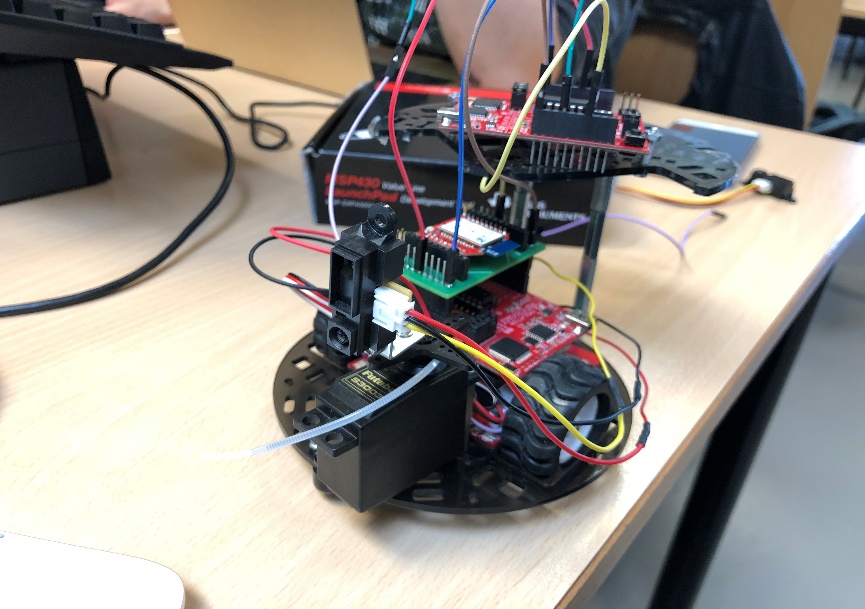
****

Figure : Construction finale

## Explication générale du projet :

Globalement, ce projet nous demande de réaliser un SAMbot (robot rouge) en deux modes :

Mode manuel :

Pour le mode manuel, il faut utiliser un module Bluetooth qui permet de faire la communication avec une téléphone portable ou ordinateur par UART. Et dirige le robot avancer, reculer, tourner à gauche ou droite. Il faut principalement programmer sur le microprocesseur MSP430G2553.

Mode autonome :

Pour le mode autonome, il faut faire la communication entre MSP430G2253(Master) et MSP430G2231(Slave) par SPI. Le slave commander le servomoteur de tourner un certain angle. Et le master envoie les différents caractères vers slave pour tourner à différent angle. Une fois cela est réalisé. Il faut aussi installer l’infrarouge sur le servomoteur pour qu’il peut détecter les objets en différents sens (devant, gauche, droite). Et l’infrarouge est connecté avec le master. Une fois, l’infrarouge détecte un objet devant, il va détecteur à une autre direction pour savoir s’il y a des objets dans les autres directions. Une fois c’est fait, le master va diriger le robot d’avancer sur le chemin où qu’il n’y a pas d’objet devant. Installation des deux modules est montré par l’image suivante.

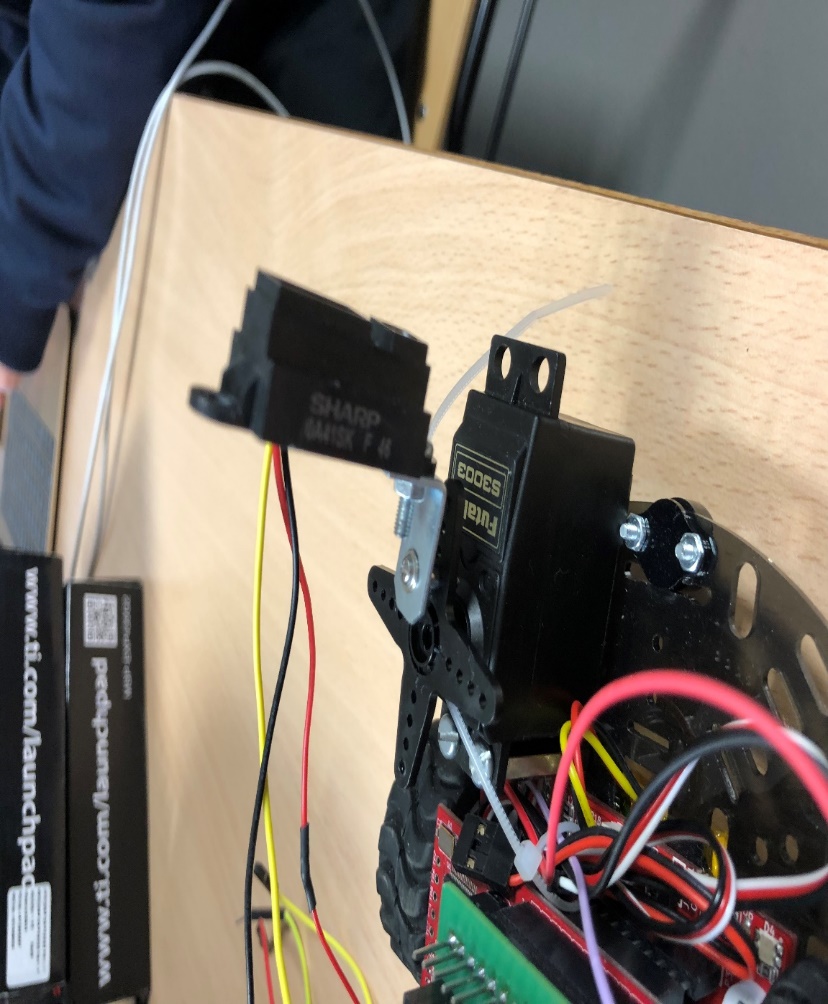
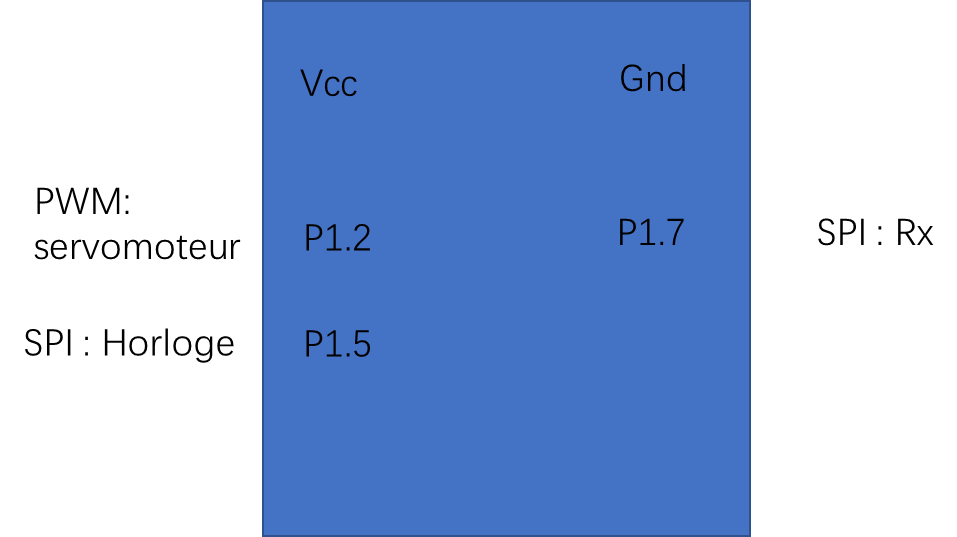


Figure : installation d'infrarouge et servomoteur

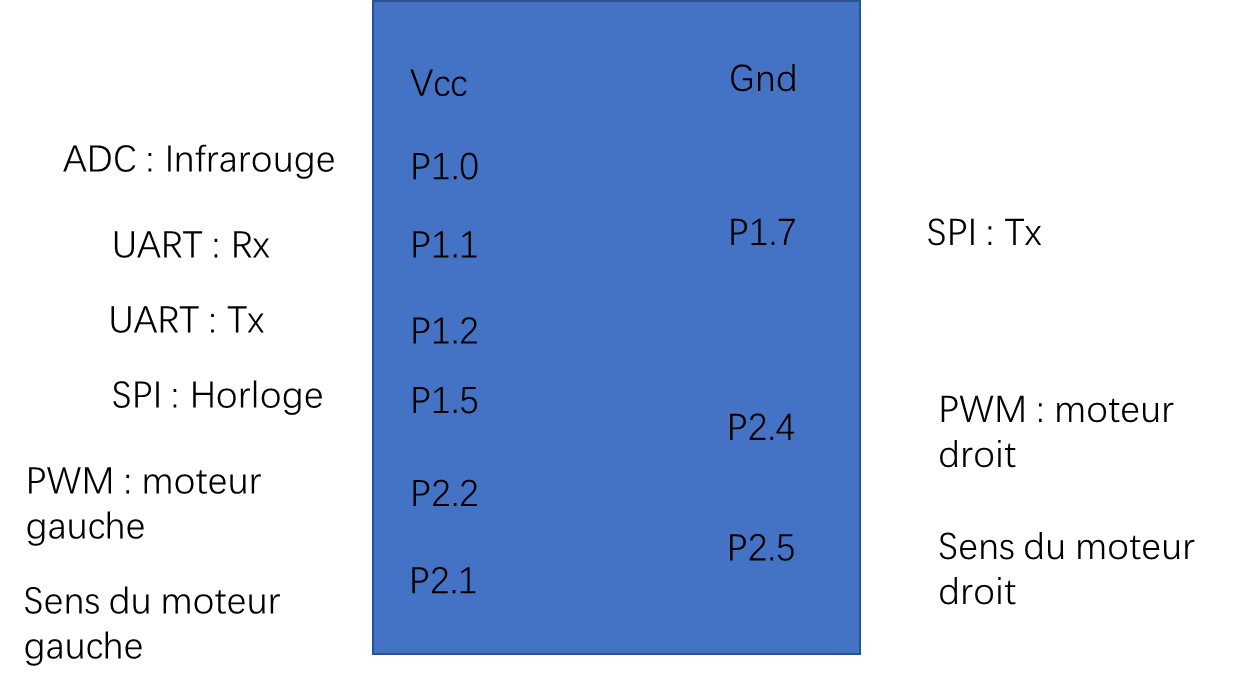
Les parties les plus important sont transporter les informations entre deux cartes de microprocesseurs de types différents par SPI et aussi faire communication entre une carte de microprocesseur et le Bluetooth par UART.

## Présentation de différent Pins :

**Slave :**



**Master :**



**Commentaire :**

Il faut connecter physiquement les fils entre master P1.5 et slave P1.5 et P1.7 de même façon. Et le master connecter avec le SAMBot par les connecteurs et le module Bluetooth de même façon. (Plus de détails voir sur la photo de construction finale (figure 8)

# 2 ) Conception spécifiée :

Pour réaliser ce projet, nous allons séparer en 2 grandes parties (master et slave) :

Et en-dessous nous avons séparé en 5 sous parties :

a) Communication entre Bluetooth et MSP430G2553

b) L’infrarouge donne l’information à MSP430G2553(Master)

c) Communication entre MSP430G2553 et MSP430G223(SPI)

d) MSP430G2231 dirige le mouvement de servomoteur.

e) MSP430G2553 dirige le mouvement du robot.

## 2-1) Liste de fichier :

**Master :**

|  |  |
| --- | --- |
| Fichier | Description |
| Partie ADC | |
| ADC.c | Toutes les fonctions qui concernent infrarouge |
| ADC.h | Bibliothèque |
| Partie robot | |
| robotv2.c | Toutes les fonctions qui concernent le robot |
| robotv2.h | Bibliothèque |
| Partie SPI | |
| SPI\_Init.c | Toutes les fonctions pour initialiser SPI |
| SPI\_Init.h | Bibliothèque |
| Partie UART | |
| Uart\_Init.c | Toutes les fonctions qui concernent UART |
| Uart\_Init.h | Bibliothèque |
| Partie main | |
| main.c | Plus de détaille sur la partie organigramme |

**Slave :**

|  |  |
| --- | --- |
| Fichier | Description |
| Partie servomoteur | |
| Driver\_Motor\_IR.c | Toutes les fonctions qui concernent servomoteur |
| Driver\_Motor\_IR.h | Bibliothèque |
| Partie SPI | |
| SPI\_Init.c | Toutes les fonctions qui concernent SPI |
| SPI\_Init.h | Bibliothèque |
| Partie main | |
| main.c | Plus de détaille sur la partie organigramme |

## 2-2) Organigramme :

### 2-2-a) Master :

**Partie commune :**

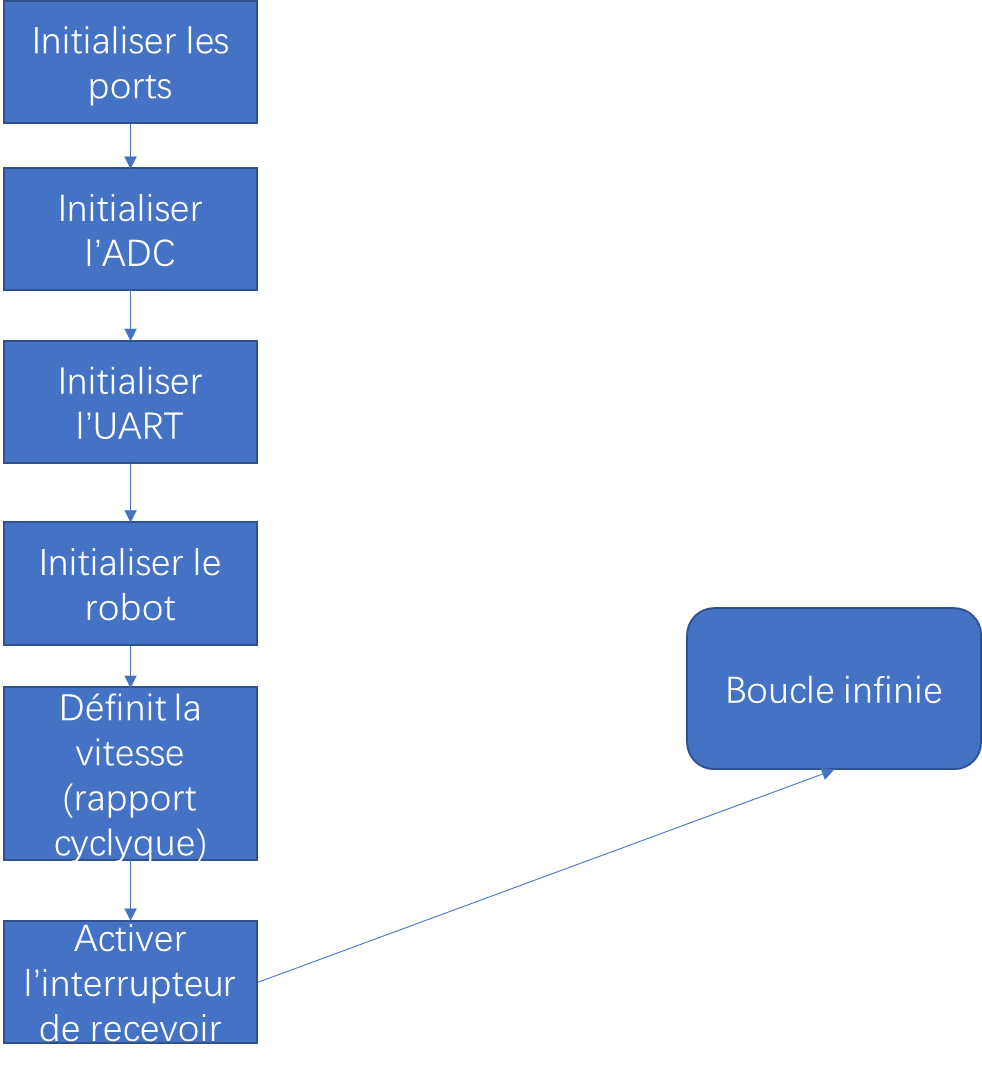


Figure : Organigramme d’initialisation

**Mode autonome :**

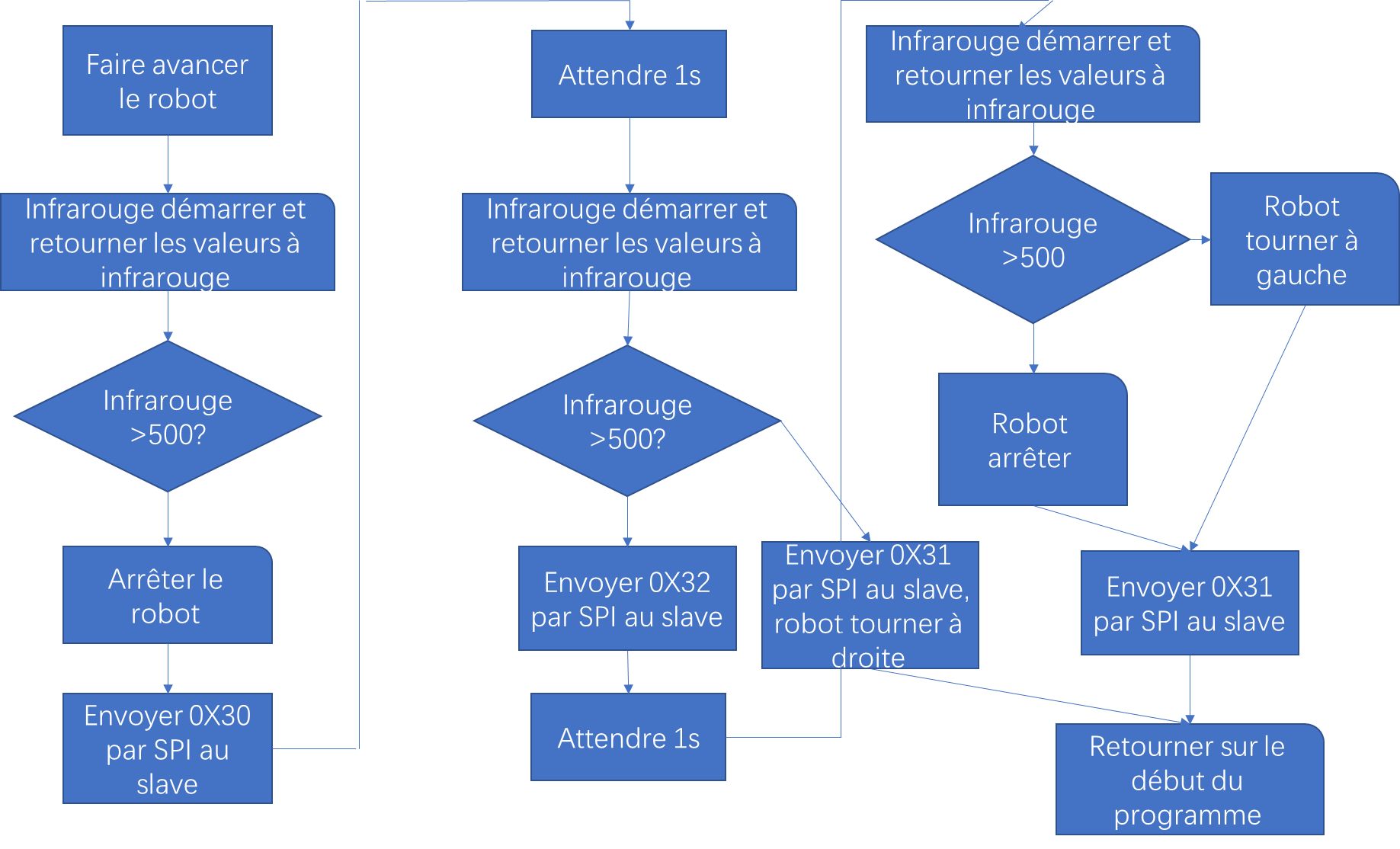


Figure : Organigramme de mode automatique

Tips : 0X30 commander le servomoteur tourner à droit (180°)

0X31 commander le servomoteur tourner au milieu (90°)

0X30 commander le servomoteur tourner à gauche (0°)

**Mode manuel :**

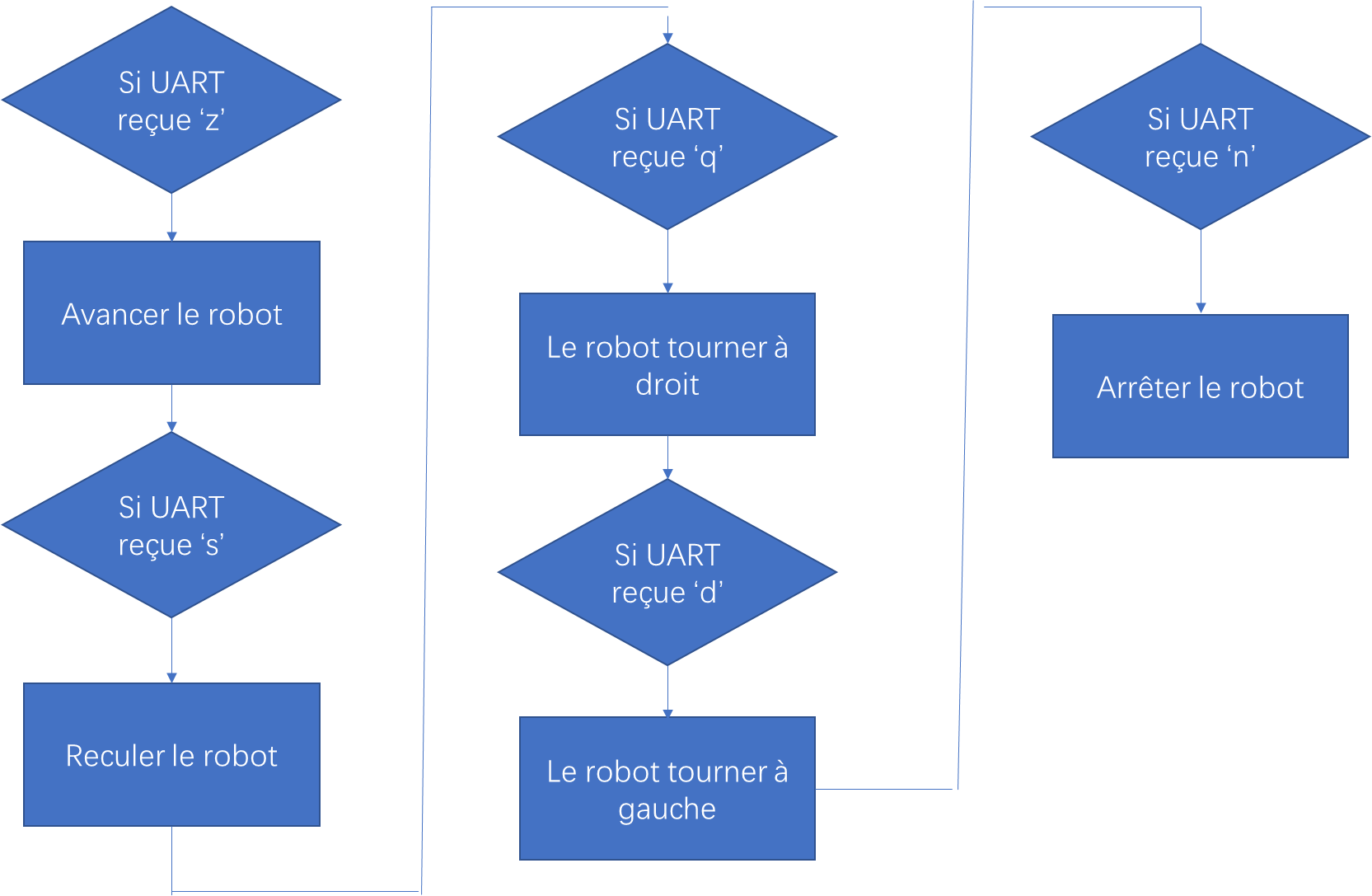


Figure : mode manuel

Tip : En fin de chaque action, il va retourner dans la boucle infinie

**Menu :**

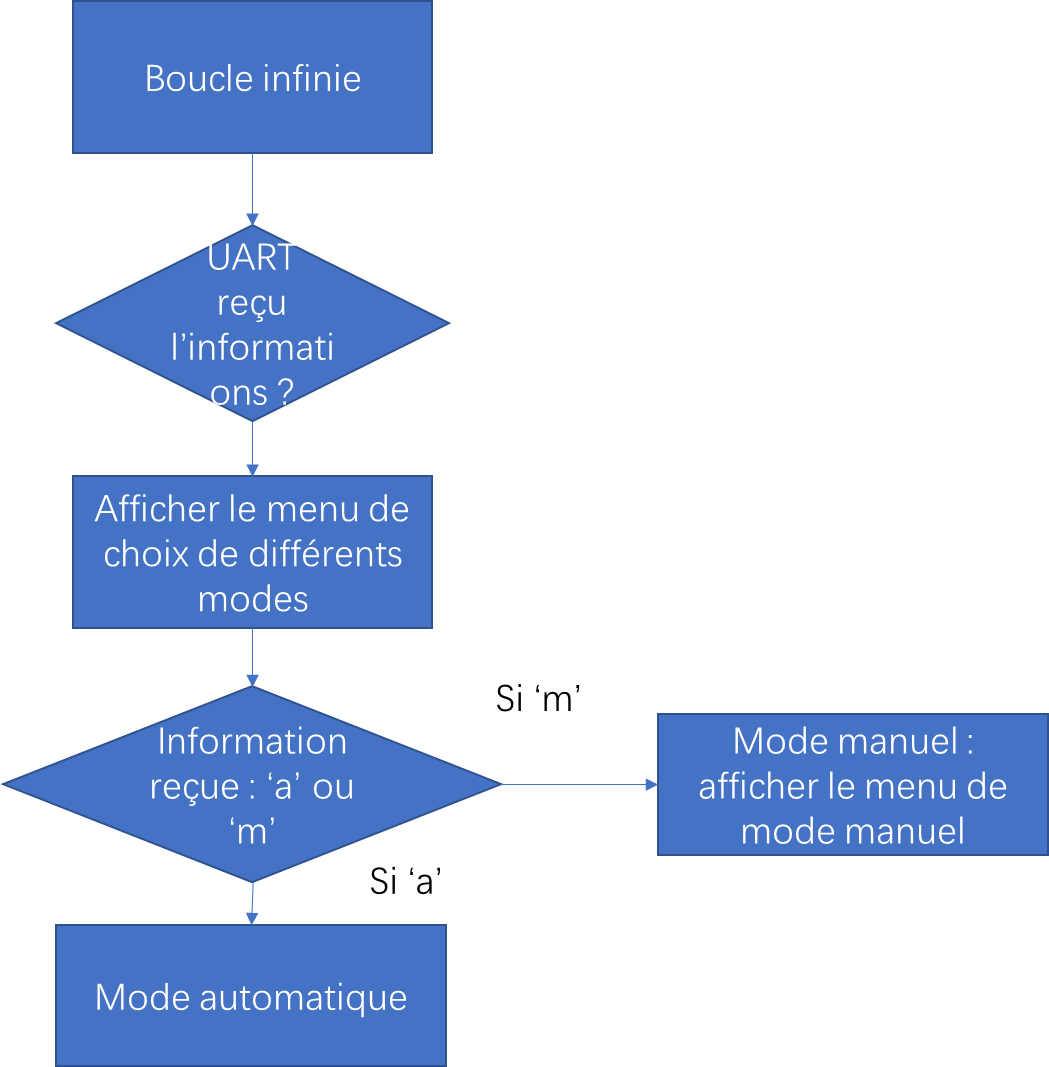


Figure : Organigramme menu

### 2-2-b) Slave :

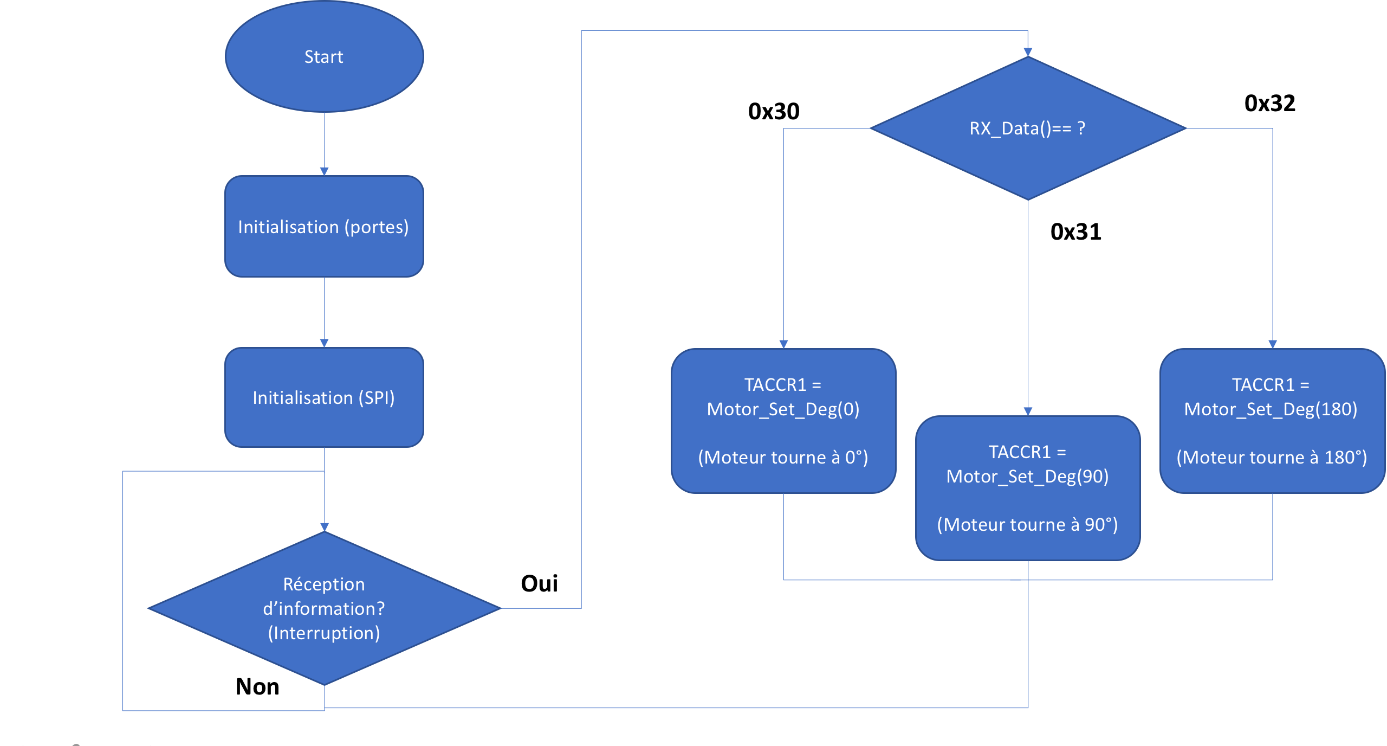


Figure : organigramme de slave

## 2-3 ) Présentation de différent module :

### 2-3-a) UART :

1. **Présentation générale :**

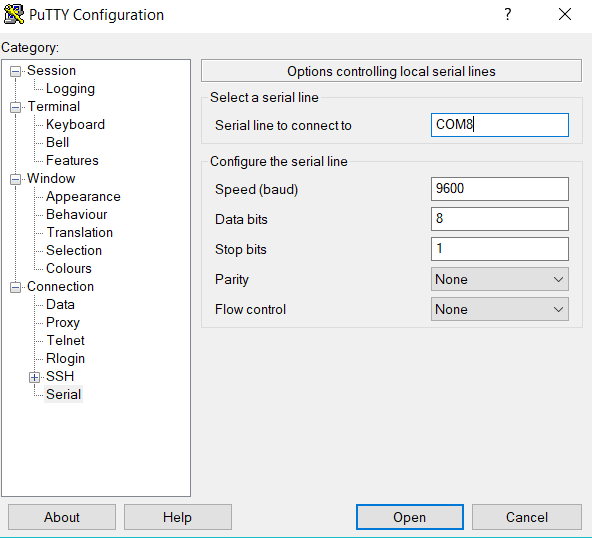
Pour la partie Bluetooth, nous avons décidé d’utiliser le mode UART pour faire la communication entre MSP430G2553 et l’appareil Bluetooth (téléphone portable ou ordinateur avec fonctionnalité Bluetooth). Pour réaliser ce but, nous avons décidons d’utiliser une module Bluetooth. Avec une carte de connecteur qui permet de connecter avec le microprocesseur MSP430G2553.

1. **Spécification détaillée :**

Pour faire la communication par Bluetooth, il faudra d’abord configurer la communication UART. En gros, nous avons configurer UART comme le schéma suivant :



Nous pouvons clairement voir que nous avons configuré le baud rate en 9600. Ce n’est pas super vite. Mais c’est largement suffisant pour notre projet. Et nous avons mettre les bits de donné en 8(Comme il est affiché en ASCII). Et le bit d’arrêt est 1. Et la parité est aucune. De plus, P1.1 est Rx et P1.2 est Tx. C’est le seul choix pour le mode UART.



Pour faire la communication, il faut aussi paramétrer votre HyperTerminal en même paramètres. Sinon, la communication est en échec.

1. **Fonction :**

Pour les codes du programme, nous avons créé un fichier .c pour enregistrer toutes les fonctions qu’on peut utiliser pour UART. Il existe probablement 3 fonctions.

InitUART :

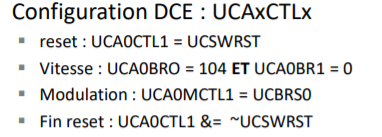
|  |  |
| --- | --- |
| Nom de la fonction : | InitUART |
| Description : | Cette fonction permet d’initialiser tous les registers d’uart. |
| Input : | None |
| Output : | None |

Comme le baud rate doit être 9600, donc nous avons calculé la valeur pour le registre(USCA0BR0).

Nous considérons la vitesse du microprocesseur est 1MHz.



Et aussi la configuration pour les autres termes de registre UCAxCTLx :



Et aussi il faut activer l’interrupteur USCI\_A0\_RX pour que le microprocesseur peut faire différent chose une fois qu’il a reçue des informations.

Print\_str :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Print\_str | |
| Description : | Cette fonction permet d’envoyer et afficher une chaine caractère sur les hyperterminale | |
|  | Type : | Paramètre : |
| Input : | unsigned | Char\*c |
| Output : | non | non |

Cette fonction est développée à partir de fonction TXdata. Comme le registre UCA0TXBUF (registre d’envoie) ne peut qu’accepter une seule caractère chaque fois. Pour envoyer une chaine caractère, nous avons utilisé le pointeur pour qu’il peut envoyer une chaine directement.

TXdata :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | TXdata | |
| Description : | Cette fonction permet d’envoyer et afficher une seule caractère sur les hyperterminale | |
|  | Type : | Paramètre : |
| Input : | unsigned | Char c |
| Output : | non | non |

Cette fonction est la fonction basic pour envoyer les caractères. Chaque fois qu’il faut envoyer un caractère, le programme va vérifier que le buffer Tx est bien disponible.

### 2-3-b) Partie IR

1. **Objectif**

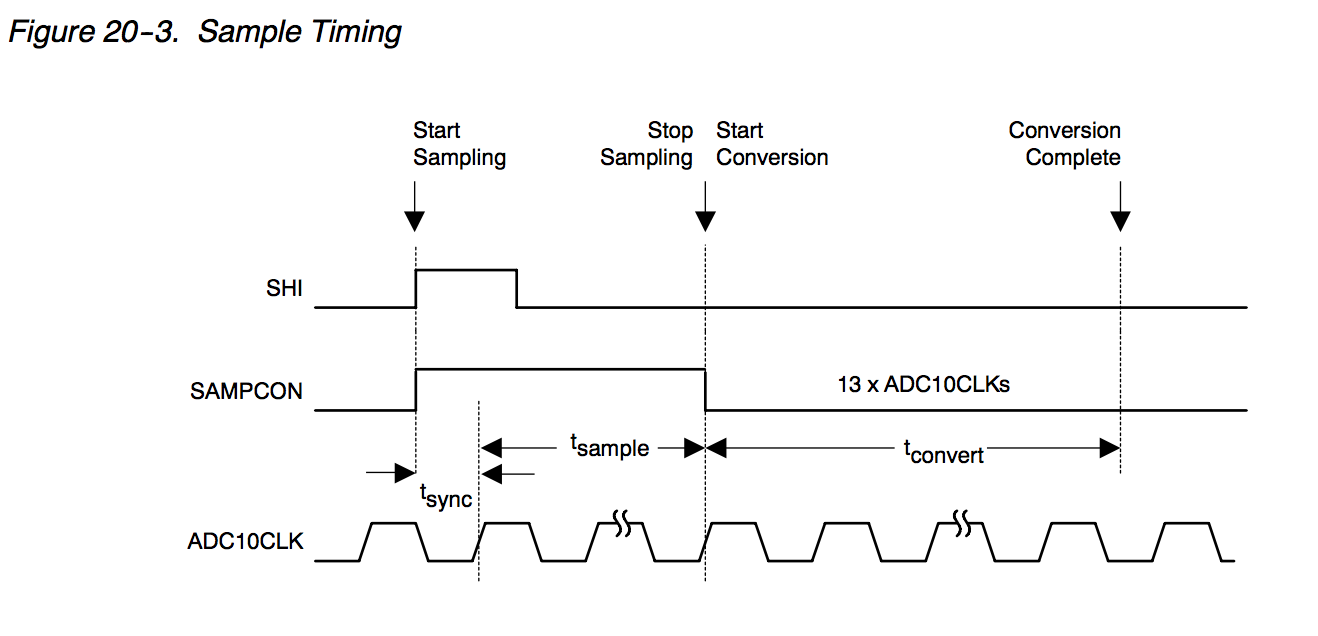
Capturer les obstacles, convertir le signal analogue au signal numérique par la module ADC et ensuite renvoyer des instructions à la carte MSP430g2553.

1. **Spécifications de cette infrarouge**

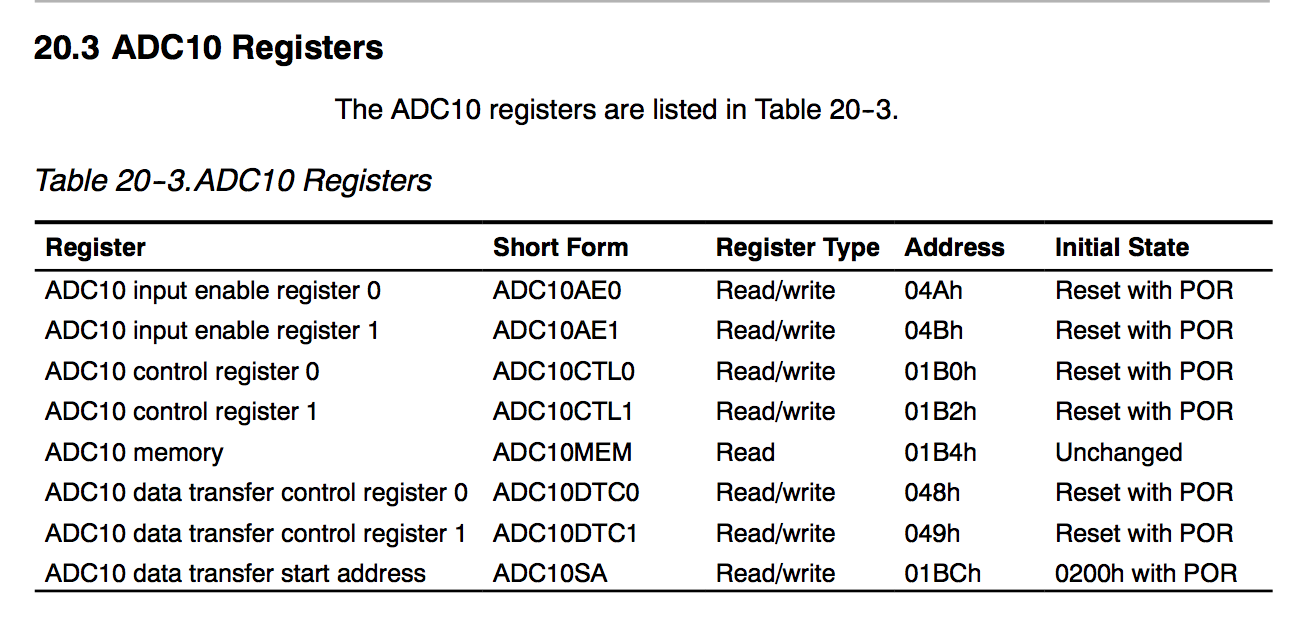
**2-a) Module ADC**

Le cœur ADC convertit une entrée analogique en sa représentation numérique de 10 bits et stocke le résultat dans le registre ADC10MEM.

Ce module est divisé par 2 parties, la partie d’échantillonnage et de conversion. Et on pourrait récupérer le résultat de conversion terminé dans ADC10MEM.

**`**

**2-b) Registres concernés :**

****

Le cœur ADC10 est configuré par deux registres de contrôle, ADC10CTL0 et ADC10CTL1. Le noyau est activé avec le bit ADC10ON.

1. **Fonctions :**

void ADC\_init(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | ADC\_init | |
| Description : | Cette fonction permet de configurer le cœur ADC10. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

void initline(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | initline | |
| Description : | Cette fonction permet d’initialiser les entrées de ADC. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

void ADC\_Demarrer\_conversion(unsigned char voie) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | ADC\_Demarrer\_conversion | |
| Description : | Cette fonction permet de définir l’horloge et la mode d’échantillonnage par registre ADC10TCL1. Aussi, elle permet d’activer la conversion par registre ADC10TCL0. | |
|  | Type : | Paramètre : |
| Input : | unsigned | voie |
| Output : | non | non |

int ADC\_Lire\_Resultat(void) :

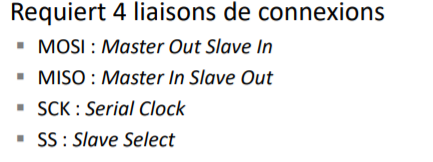
|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | ADC\_Lire\_Resultat | |
| Description : | Cette fonction permet de rendre le résultat dans registre ADC10MEM. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | int | ADC10MEM |

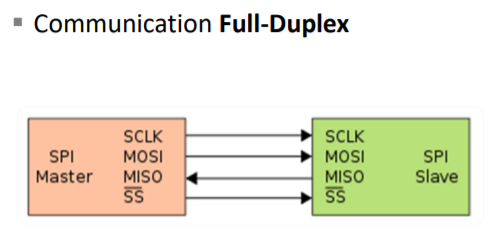
### 2-3-c) SPI :

1. **Présentation générale :**

Le mode de communication SPI permet de faire la communication entre deux microprocesseurs. Dans notre projet, ce sera un master (MSP430G2553) et un slave(MSP430G2231). La communication spi est Full-Duplex. Et normalement le mode spi demande 3 ou 4 fils de liaisons.

Comme le schéma présenté suivant :



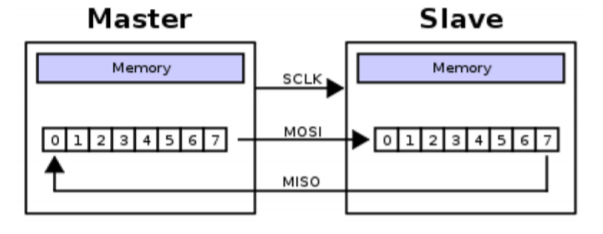


Mais notre projet n’a qu’une seule slave. Donc il n’a pas besoin de slave select. De plus, nous n’avons pas besoin de retour d’information. Donc nous avons supprimé le fil MISO. Donc il ne reste que le SCK(Clock) et le MOSI(Tx).





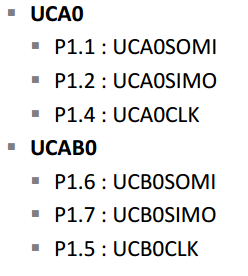
Et le SPI faire l’échange en chaque front montant ou descendant de clock d’après la configuration. Et présenter par le schéma suivant :



Pour la communication SPI, il possède deux parties : Master et Slave. Donc nous décidons de faire la conception séparément.

**SPI Master :**

Il y a deux groupes de registre pour faire la communication SPI.



Mais on sait bien que les pin P1.1 et P1.2 sont déjà occupé par le Tx et Rx d’UART. Donc nous n’avons pas de choix. Nous devons utiliser celui de P1.6, P1.7, P1.5.

**Fonction :**SPI\_Init :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | SPI\_Init | |
| Description : | Cette fonction permet d’initialiser le master de SPI | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

**Slave :**

**Fonction :**

Init\_SPI :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Init\_SPI | |
| Description : | Cette fonction permet d’initialiser le slave de SPI | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

RX\_Data

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | RX\_Data | |
| Description : | Cette fonction permet de recevoir l’information | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

### 2-3-d) Partie servomoteur :

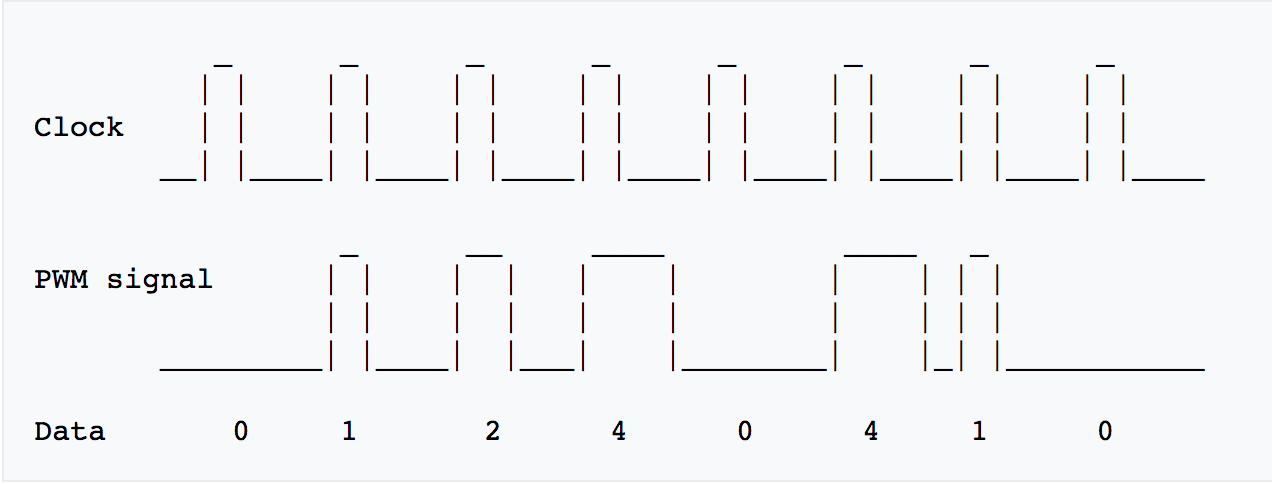
1. **Objectif :**

Déterminer la fréquence optimale afin de générer un signal de type modulation en largeur d’impulsion assurant un déplacement régulier.

1. **Spécifications de ce moteur :**

|  |  |
| --- | --- |
| SERVO-MOTEUR FUTABA S3003 | ../../屏幕快照%202018-03-16%2015.03.40.png |
| Le signal de commande est de niveau TTL (transistor transistor logic). | ../../屏幕快照%202018-03-16%2015.04.00.png |

**2-a) Modulation de largeur d’impulsion(PWM) :**



**Explication :**

Pour un signal PWM, on parle de

* Période(T)
* Largeur d’impulsion(D)

Et

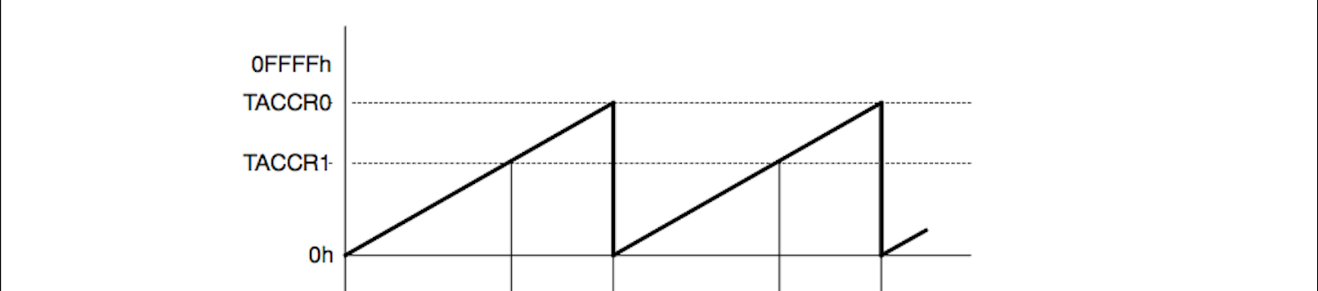
Cycle de service (Duty cycle) = D/T x 100%

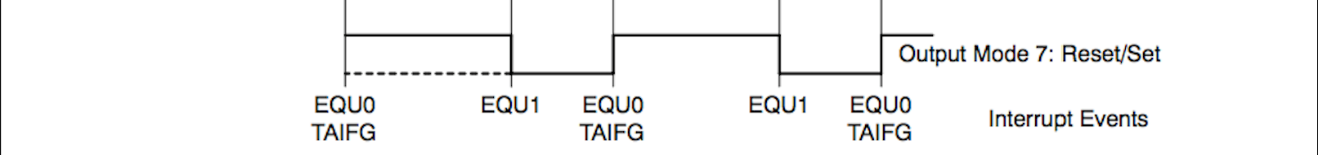
Dans une carte de msp430, on peut réaliser un signal PWM par le TIMERA.

On choisit l’Output Mode 7, et

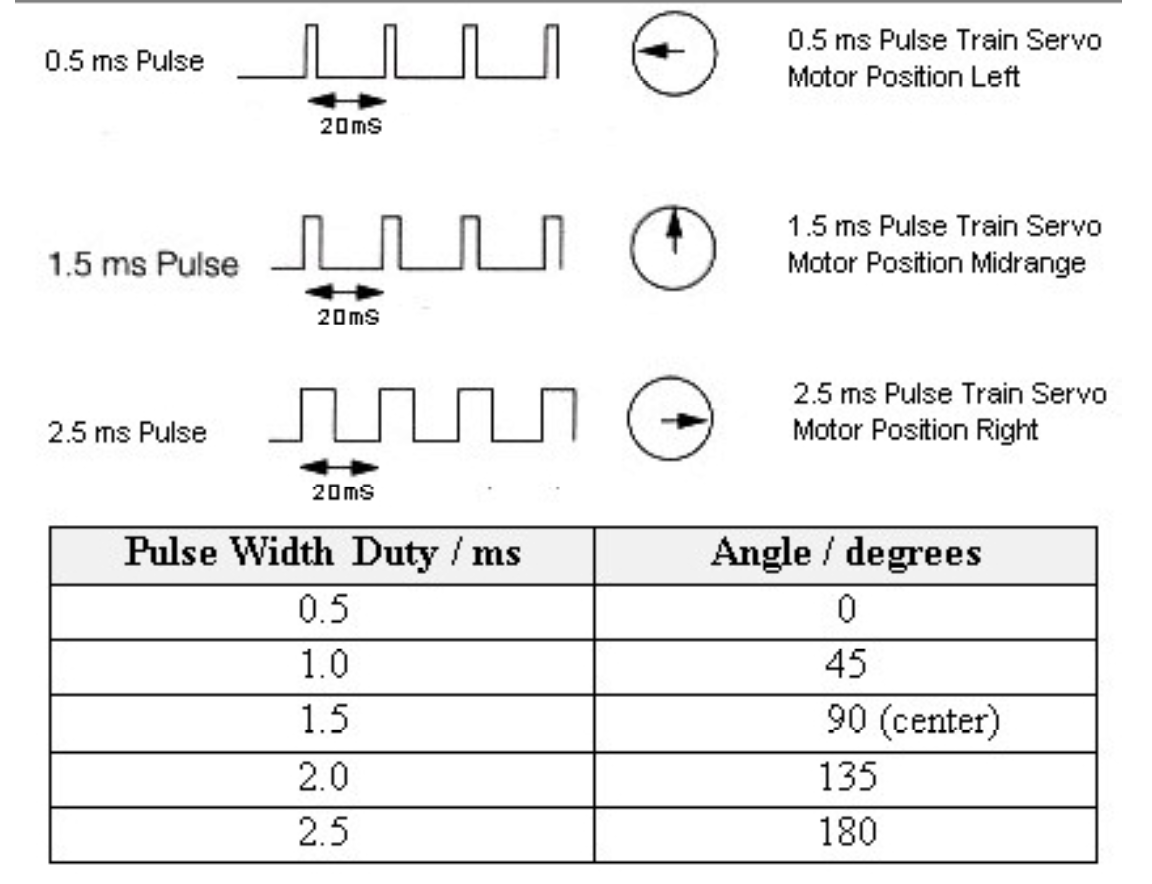
TACCR0 = 20000 ; //Période = 20000μs = 20ms

TACCR1 = MOTOR\_INIT\_DEG = 500 ; //Largeur d’impulsion = 0.5ms





**2-b) PWM concerné à ce servomoteur**



**Explication :**

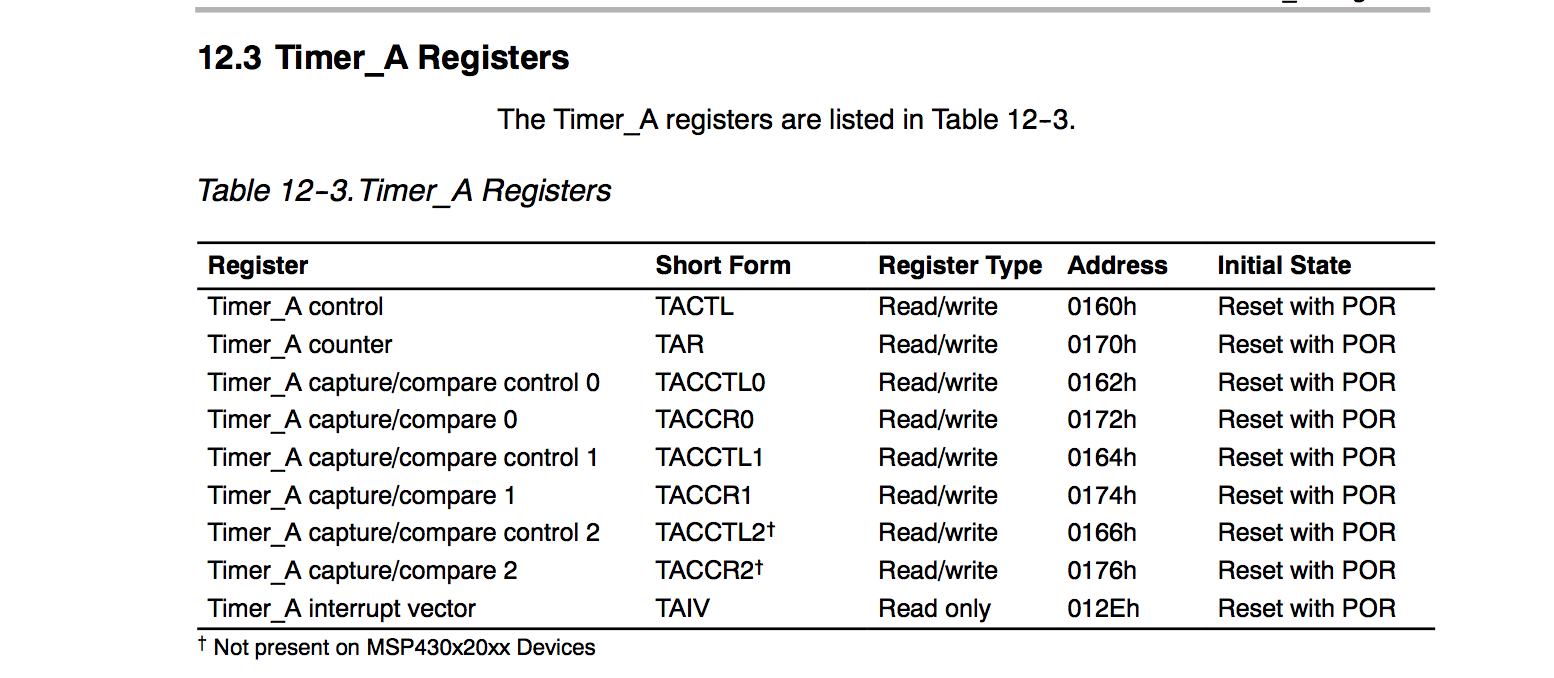
Pour commander la position angulaire, il faut envoyer sur le fil de commande des impulsions de largeur variable (entre 0.5 et 2.5 ms) toutes les 20 ms (50 HZ).

En l’absence d’impulsions, le Servo reste en position mais pour bénéficier du couple maxi il faut envoyer les impulsions en continu.

Step Angle = (2500 – 500) /180 = 11.11 μs = 0.0111 ms

Donc, angle/largeur d’impulsion = 0.0111ms / 1 deg

1. **Registres concernés :**

****

1. **Fonctions :**

Void Motor\_PWM\_Init(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Motor\_PWM\_Init | |
| Description : | Cette fonction permet d’initialiser le TIMERA, donc fixer la période et la largeur d’impulsion de PWM. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

Void Motor\_Stop(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Motor\_Stop | |
| Description : | Cette fonction permet de définir le P1.2 en entrée | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

Void Motor\_Start(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Motor\_Start | |
| Description : | Cette fonction permet de définir le P1.2 en sortie | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

int Motor\_Set\_Deg(int deg) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Motor\_Set\_Deg | |
| Description : | Cette fonction permet de sortir un signal PWM de différentes largeurs d’impulsion. | |
|  | Type : | Paramètre : |
| Input : | int | deg |
| Output : | int | taccr |

void main (void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | main | |
| Description : | Cette fonction comporte tous les initialisations de SPI et de servomoteur. Elle comporte aussi un interrupteur pour que la module SPI puisse commander le servomoteur. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

### 2-3-e) Partie SAMBot(robot) :

**1) Objectif :**

Déplacer comme une voiture.

1. **Spécification de ce ROBOT :**

**2-a) Module ROBOT**

Le robot comporte 2 moteurs, les signaux de PWM sont fournis par le TIMERA de carte MSP430g2553.

|  |  |
| --- | --- |
| Fonctionnalités | Descriptions |
| Avancer | Les deux moteurs ont le même sens(avancé) |
| Reculer | Les deux moteurs ont le même sens(reculé) |
| Tourner à gauche | Le moteur à gauche a le sens reculé, l’autre est le contraire. |
| Tourner à droite | Le moteur à droite a le sens reculé, l’autre est le contraire. |

1. **Registres concernés**

**De même comme le PWM en slave(servomoteur) .**

1. **Fonctions**

void Init\_Robot(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Init\_Robot | |
| Description : | Cette fonction permet de définir les sorties de signal PWM et de configurer le TIMERA. | |
|  | Type : | Paramètre : |
| Input : | int | deg |
| Output : | int | taccr |

void Choix\_direction(unsigned char sens) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Choix\_direction | |
| Description : | Cette fonction permet de choisir la direction de moteur.  (2 pour avancer, 1 pour reculer, 2 pour tourner à droite, 3 pour tourner à gauche) | |
|  | Type : | Paramètre : |
| Input : | unsigned char | sens |
| Output : | non | non |

void Vitesse\_moteurs(unsigned int vit\_gauche, unsigned int vit\_droite) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Vitesse\_moteurs | |
| Description : | Cette fonction permet de changer la rapport cyclique de moteur à gauche et celui à droite. | |
|  | Type : | Paramètre : |
| Input : | unsigned int | vit\_gauche, vit\_droite |
| Output : | non | non |

void Arret\_robot(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Arret\_robot | |
| Description : | Cette fonction permet de changer le OUTMODE de PWM pour arrêter les moteurs. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

void Demarrer\_robot(void) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | Demarrer\_robot | |
| Description : | Cette fonction permet de choisir le bon OUTMODE de PWM pour démarrer les moteurs. | |
|  | Type : | Paramètre : |
| Input : | non | non |
| Output : | non | non |

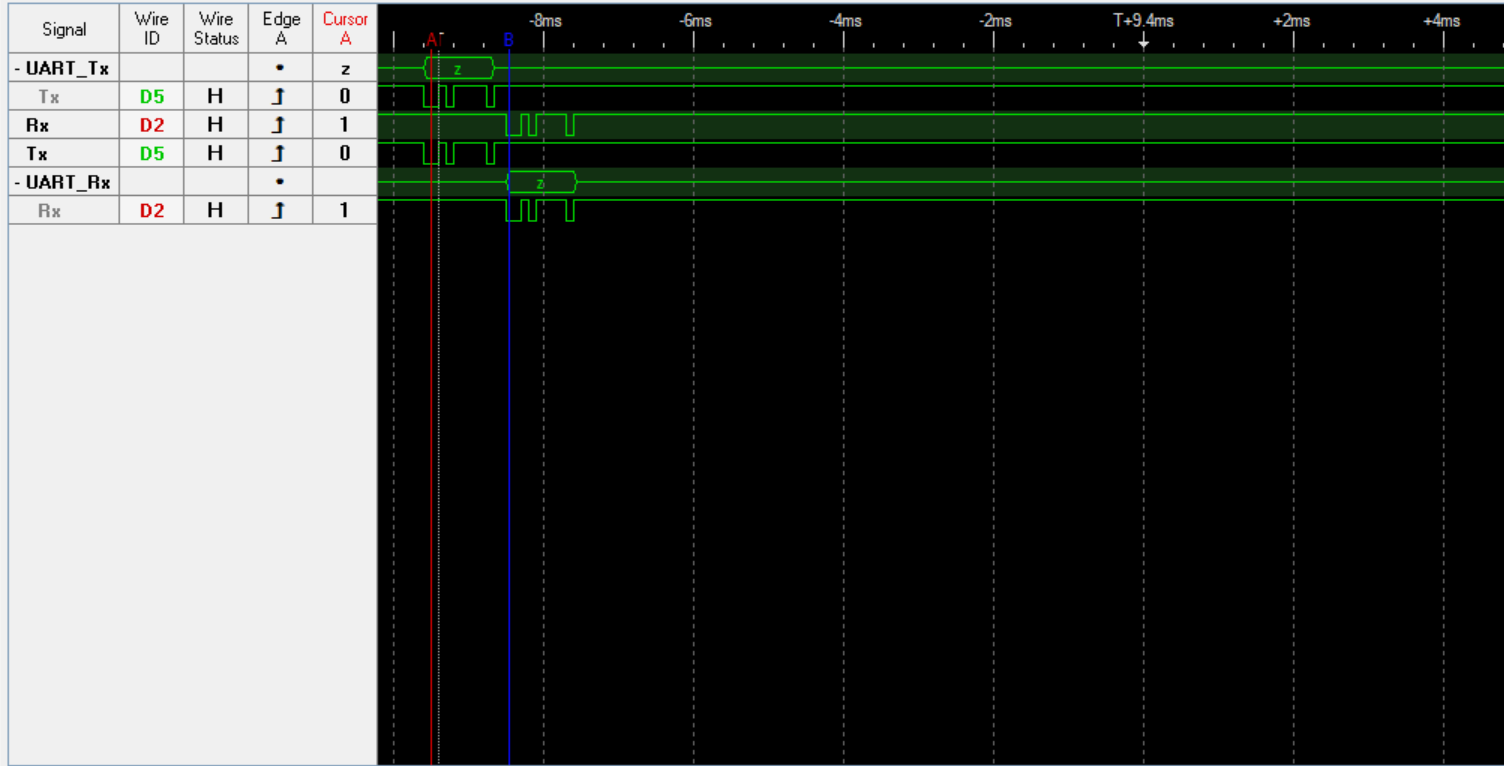
void tempo (unsigned int ms) :

|  |  |  |
| --- | --- | --- |
| Nom de la fonction : | tempo | |
| Description : | Cette fonction permet de faire le programme attendre un certain temps (ms). | |
|  | Type : | Paramètre : |
| Input : | unsigned int | ms |
| Output : | non | non |

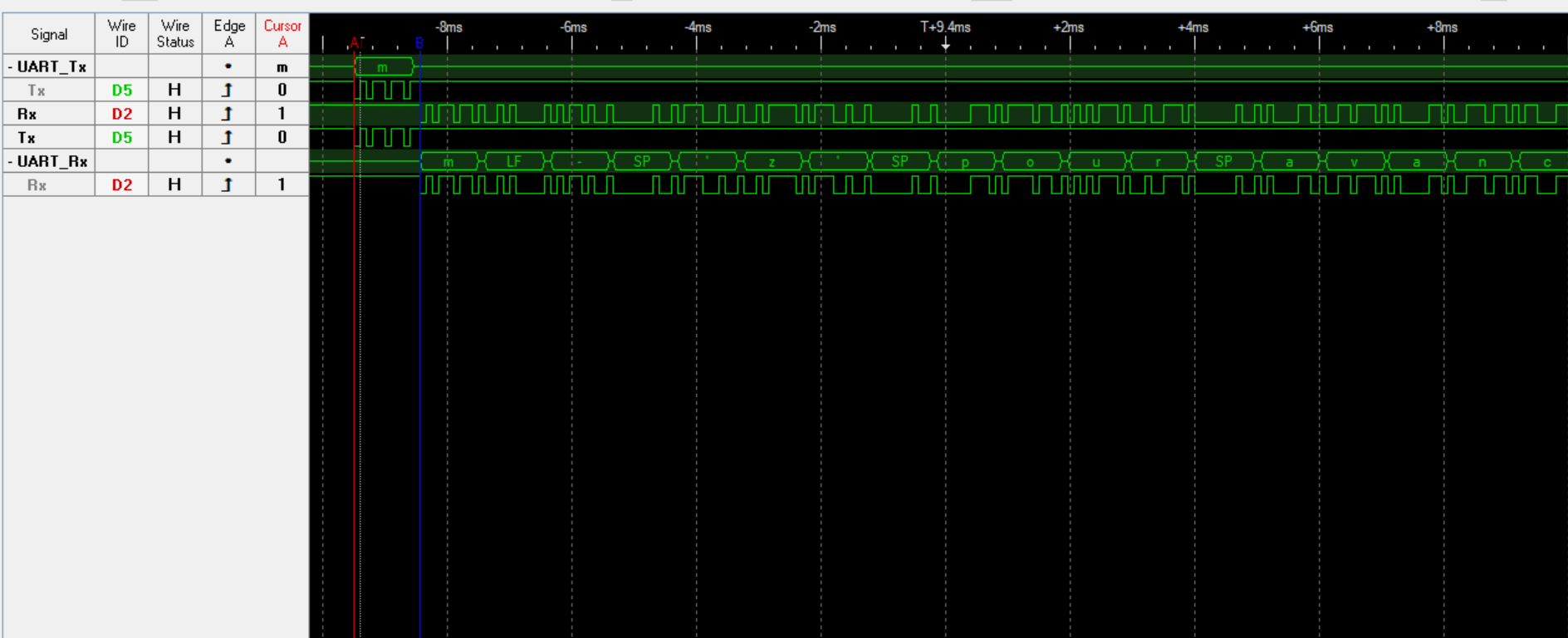
# 3) Test unitaire :

## 3-a) Communication entre Bluetooth et MSP430G2553 :

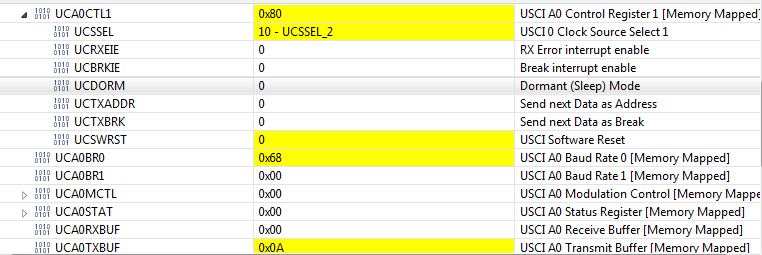
|  |  |
| --- | --- |
| Nom de la fonction : | Txdata |
| Méthode de test | Envoyer lettre ‘z’ au UART |
| Résultat attendu | Lettre ‘z’ est affiché sur PuTTY |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



|  |  |
| --- | --- |
| Nom de la fonction : | Print\_str |
| Méthode de test | Envoyer une chaine caractère au UART |
| Résultat attendu | Une chaine est affichée sur PuTTY |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



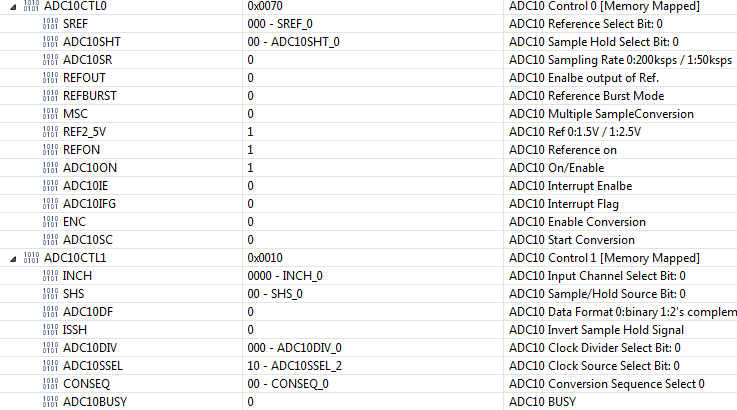
|  |  |
| --- | --- |
| Nom de la fonction : | InitUART |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut (UCA0CTL1 et UCA0BR0 et UCA0BR1) |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (UCA0CTL1 voir sur la spécification), (UCA0BR0=104 et UCA0BR1=0) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



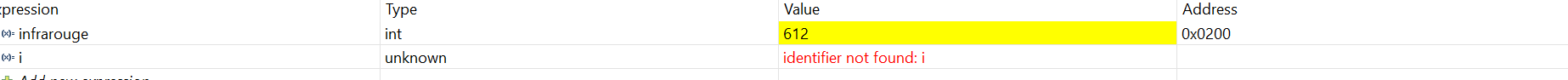
Tips : 0x68 = 104

## 3-b) L’infrarouge donne l’information à MSP430G2553(Master)

|  |  |
| --- | --- |
| Nom de la fonction : | ADC\_init |
| Méthode de test | Voir les registres est bien identique |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (ADC10CTL0 et ADC10CTL1 voir sur la spécification) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



|  |  |
| --- | --- |
| Nom de la fonction : | ADC\_Lire\_resultat () |
| Méthode de test | Créer un variable globale infrarouge(int) , vérifier que cette valeur respecte la règle . |
| Résultat attendu | S’il y a un objet devant, cette valeur plus élevé, sinon, la valeur est plus bas |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



Tip : un objet devant



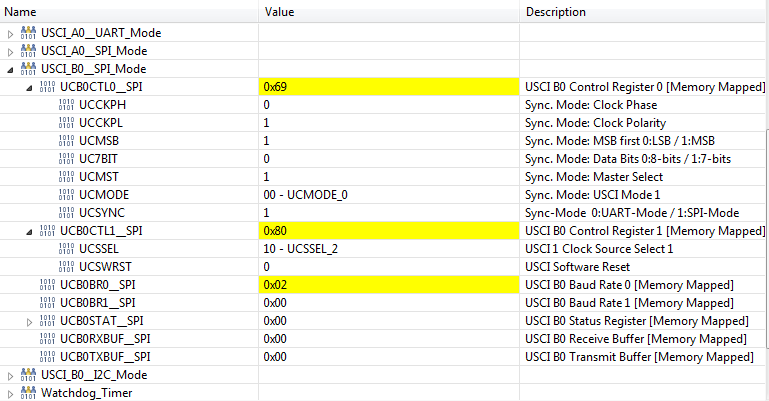
Tip : pas d’objet

## 3-c) Communication entre MSP430G2553 et MSP430G2231(SPI)

De même comme la spécification détaillée, nous avons séparé le test unitaire en deux parties :

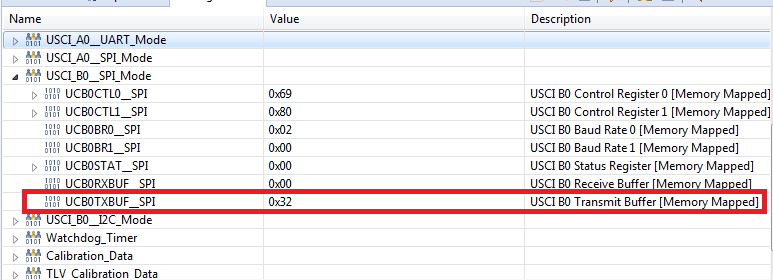
### 3-c-1) Master :

|  |  |
| --- | --- |
| Nom de la fonction : | SPI\_Init |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut (UCB0CTL1, UCB0CTL0 et UCA0BR0 et UCA0BR1) |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (UCA0CTL1, UCB0CTL0 voir sur la spécification), (UCA0BR0=0x02 et UCA0BR1=0) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



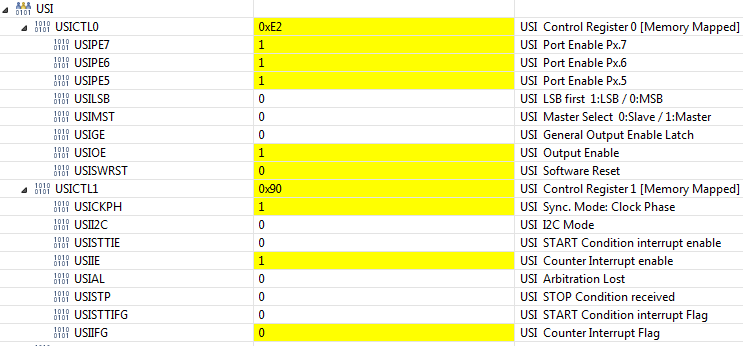
|  |  |
| --- | --- |
| Nom de la fonction : | SPI\_Tx |
| Méthode de test | Envoyer 0x32 à slave |
| Résultat attendu | 0x32 est reçu par le slave |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |





### 3-c-2) Slave :

|  |  |
| --- | --- |
| Nom de la fonction : | Init\_SPI |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut (USCICTL0, USCICTL1 et UCA0BR0 et UCA0BR1) |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (USCICTL0, USCICTL1 voir sur la spécification) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |

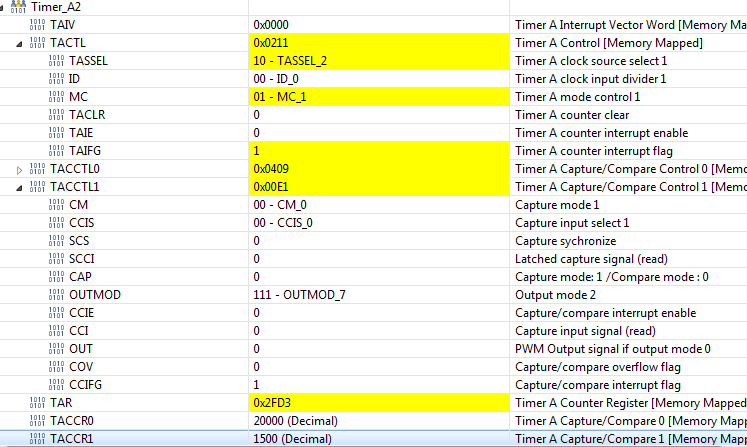


|  |  |
| --- | --- |
| Nom de la fonction : | RX\_Data |
| Méthode de test | Recevoir 0x32 par le slave |
| Résultat attendu | 0x32 est reçu par le slave |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |

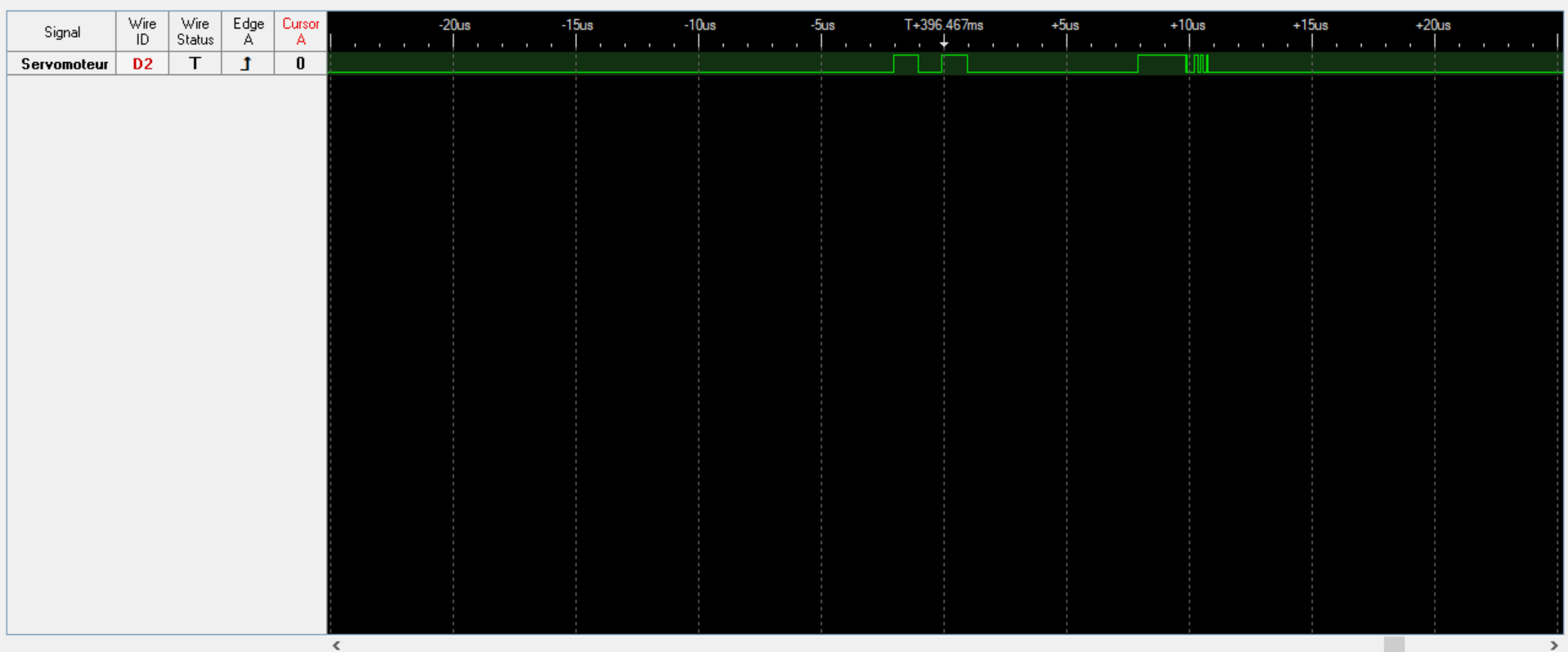


## 3-d) MSP430G2231 dirige le mouvement de servomoteur.

|  |  |
| --- | --- |
| Nom de la fonction : | Motor\_PWM\_Init |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut (TACTL, TACTL1 et TACCR0 et TACCR1) |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (TACTL, TACTL1 voir sur la spécification. (TACCR0=20000(période)),( TACCR1=1500(rapport cyclique)) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |

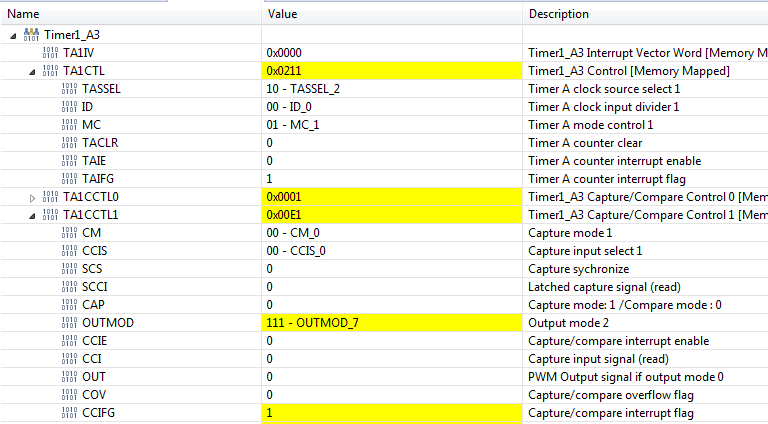


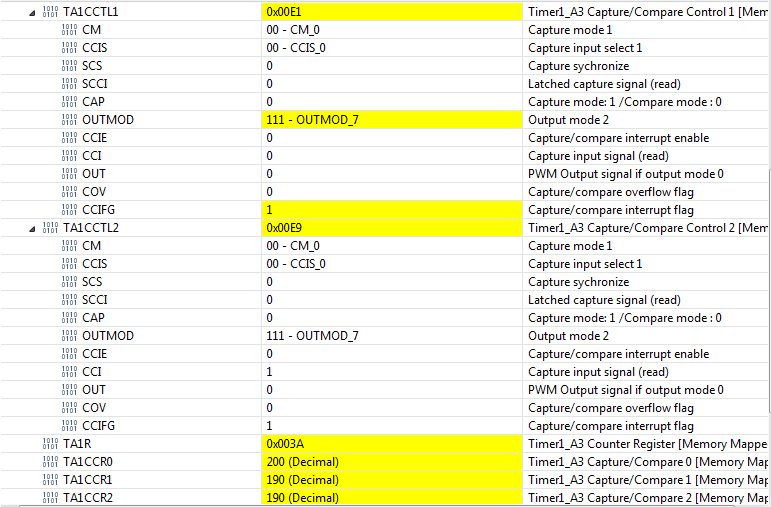
|  |  |
| --- | --- |
| Nom de la fonction : | Motor\_Set\_Deg |
| Méthode de test | Vérifier si le nombre retourner est bien le rapport cyclique qu’on veut |
| Résultat attendu | 1. 500 pour 0° 2. 1500 pour90° 3. 2500 pour 180° |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



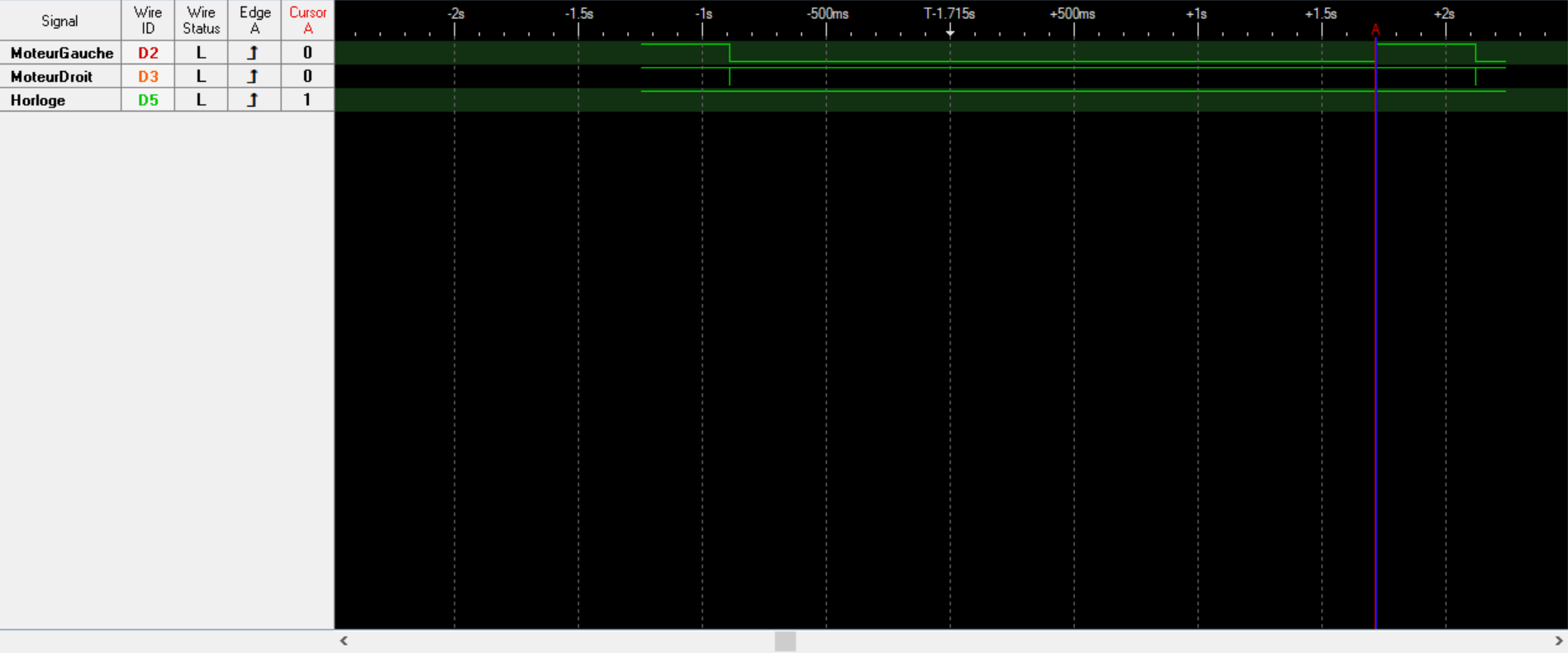
## 3-e) MSP430G2553 dirige le mouvement du robot.

|  |  |
| --- | --- |
| Nom de la fonction : | Init\_Robot |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut (TA1CTL, TA1CCR0, TA1CCR1 et TA1CCR2) |
| Résultat attendu | Les registres sont bien configurés d’après la spécification (TA1CTL voir sur la spécification).  (TA1CCR0=200 (période)),  (TA1CTL = TASSEL\_2 | MC\_1 (configuration)). |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |





|  |  |
| --- | --- |
| Nom de la fonction : | Choix\_direction |
| Méthode de test | Voir les P2.1 et P2.5 sont bien les valeurs correspondantes au sens rotation. |
| Résultat attendu | Avancer (P2.1 = 1, P2.5 = 1),  Reculer (P2.1 = 0, P2.5 = 0),  Tourner à gauche (P2.1 = 0, P2.5 = 1),  Tourner à droite (P2.1 = 1, P2.5 = 0) |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



Tip : avancer puis tourner à gauche

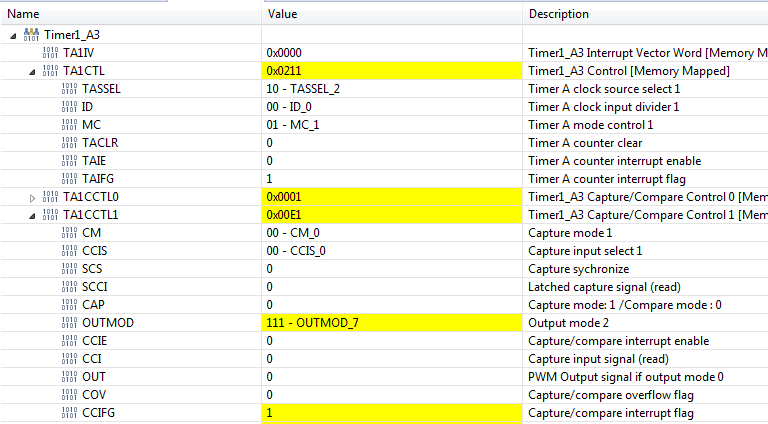
|  |  |
| --- | --- |
| Nom de la fonction : | Vitesse\_moteurs |
| Méthode de test | Voir les registres est bien identique que ceux qu’on veut.  (TA1CCR1 et TA1CCR2) |
| Résultat attendu | Comparer les valeurs de registre (TA1CCR1 et TA1CCR2) avec les entrées de fonction (vit\_gauche et vit\_droite).  Voir sur la spécification :  vit\_gauche = 190, vit\_droite = 190. |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



|  |  |
| --- | --- |
| Nom de la fonction : | Arret\_robot |
| Méthode de test | Voir le signal sorti est bien correspondant au mode choisit.  (TA1CCR1 et TA1CCR2) |
| Résultat attendu | Comparer le signal sorti avec le signal PWM de OUTMOD\_0. |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



|  |  |
| --- | --- |
| Nom de la fonction : | Demarrer\_robot |
| Méthode de test | Voir le signal sorti est bien correspondant au mode choisit.  (TA1CCR1 et TA1CCR2) |
| Résultat attendu | Comparer le signal sorti avec le signal PWM de OUTMOD\_7.  ../../屏幕快照%202018-03-20%2008.04.22.png |
| Résultat obtenu | Voir sur le photo |
| Validé ? | Oui |



Tip : moteur gauche





Tip : moteur droite

# Gestion de configuration :

# Conclusion :

# Annex :

## Les schémas utilisées :

[Figure 1 : Lauchpad 5](#_Toc509498515)

[Figure 2 : MSP430G2231 5](#_Toc509498516)

[Figure 3 : MSP430G2553 6](#_Toc509498517)

[Figure 4 : Servomoteur 6](#_Toc509498518)

[Figure 5 : module Bluetooth (installé sur le master) 6](#_Toc509498519)

[Figure 6 : L'infrarouge 7](#_Toc509498520)

[Figure 7 : SAMbot 7](#_Toc509498521)

[Figure 8 : Construction finale 8](#_Toc509498522)

[Figure 9 : installation d'infrarouge et servomoteur 9](#_Toc509498523)

[Figure 10 : Organigramme d’initialisation 12](#_Toc509498524)

[Figure 11 : Organigramme de mode automatique 12](#_Toc509498525)

[Figure 12 : mode manuel 13](#_Toc509498526)

[Figure 13 : Organigramme menu 13](#_Toc509498527)

[Figure 14 : organigramme de slave 14](#_Toc509498528)