

## **DOCUMENTATION DU PROJET BUS DE COM**

**Département SEI**

**Février-Mars 2018**

***Dominant : ISE-OC***

---

***Rapport d'équipe***

***Li Zhengxi***

---

***Jiang Zhuohang***

---

## **Introduction :**

Cette année, nous avons travaillé beaucoup sur les microcontrôleurs, ils nous ont demandé de maîtriser les connaissances pertinentes de msp430. Dans le cadre de ce projet, nous avons réalisé un petit robot qui peut détecter et éviter les obstacles automatiquement ou être contrôlé par un appareil Bluetooth. Nous allons présenter notre petit robot par les spécifications, les matériaux utilisés, les fonctions de programme.

# **Plan :**

## Table des matières

Introduction :	2
Plan :	3
1) Conception général :	5
1-1) Matériels utilisés :	5
1-2) Explication générale du projet :	7
1-3) Présentation de différent Pins :	9
2) Conception spécifiée :	10
2-1) Liste de fichier :	10
2-2) Organigramme :	11
2-2-a) Master :	11
2-2-b) Slave :	13
2-3 ) Présentation de différent module :	14
2-3-a) UART :	14
2-3-b) Partie IR.....	16
2-3-c) SPI :	19
2-3-d) Partie servomoteur :	21
2-3-e) Partie SAMBot(robot) :	26
3) Test unitaire :	28
3-a) Communication entre Bluetooth et MSP430G2553 :	28
3-b) L'infrarouge donne l'information à MSP430G2553(Master) .....	30
3-c) Communication entre MSP430G2553 et MSP430G2231(SPI) .....	31

3-c-1) Master : .....	31
3-c-2) Slave : .....	32
3-d) MSP430G2231 dirige le mouvement de servomoteur.....	33
3-e) MSP430G2553 dirige le mouvement du robot. ....	35
4) Gestion de configuration : .....	39
4-1) Lien de repositories : .....	39
4-2) Repositories utilisées :.....	39
Conclusion :.....	41
Annex : .....	42
Les schémas utilisés :.....	42
Pièces jointes :.....	42

# 1) Conception général :

## 1-1) Matériels utilisés :

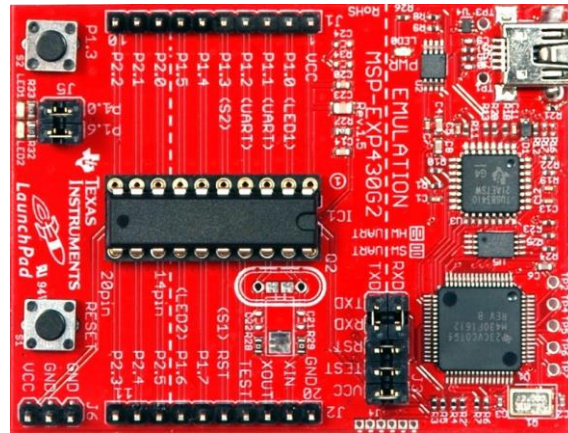


Figure 1 : Launchpad



Figure 2 : MSP430G2231

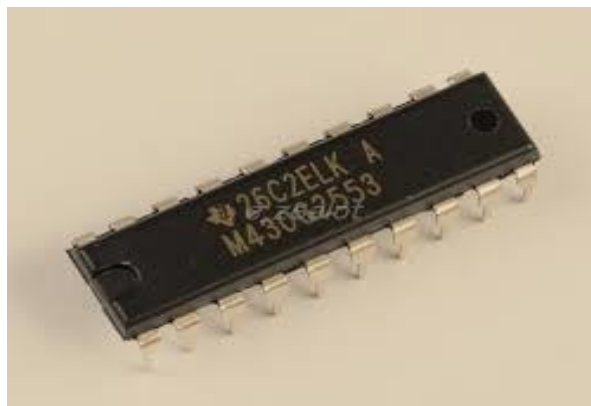


Figure 3 : MSP430G2553



Figure 4 : Servomoteur



Figure 5 : module Bluetooth (installé sur le master)



Figure 6 : L'infrarouge

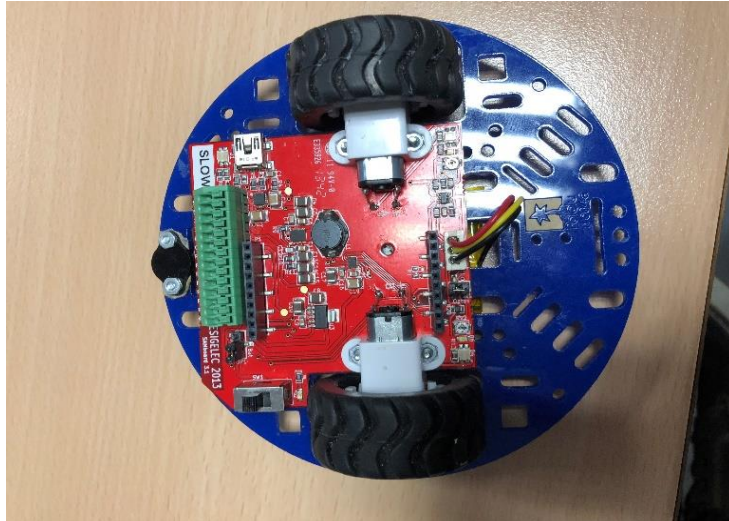


Figure 7 : SAMbot

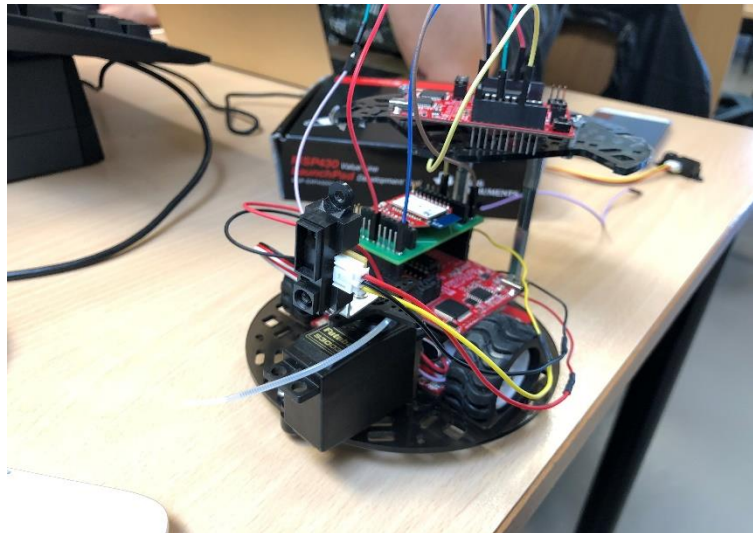


Figure 8 : Construction finale

## 1-2) 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 un 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 détecter 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.

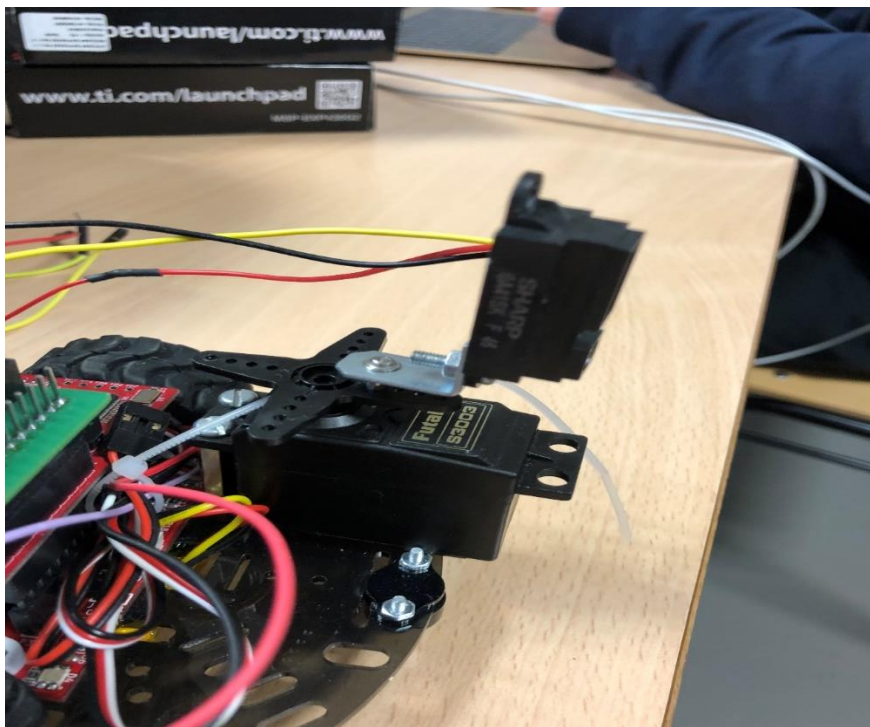


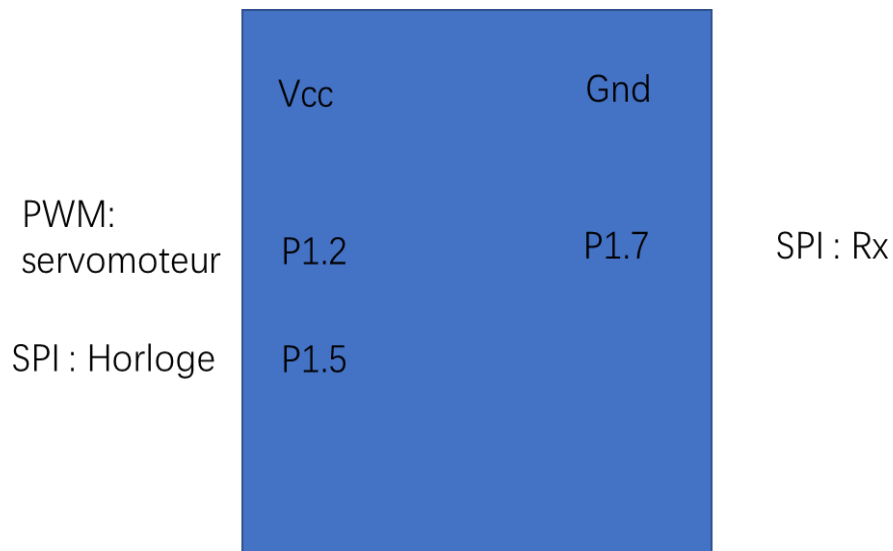
Figure 9 : installation d'infrarouge et servomoteur



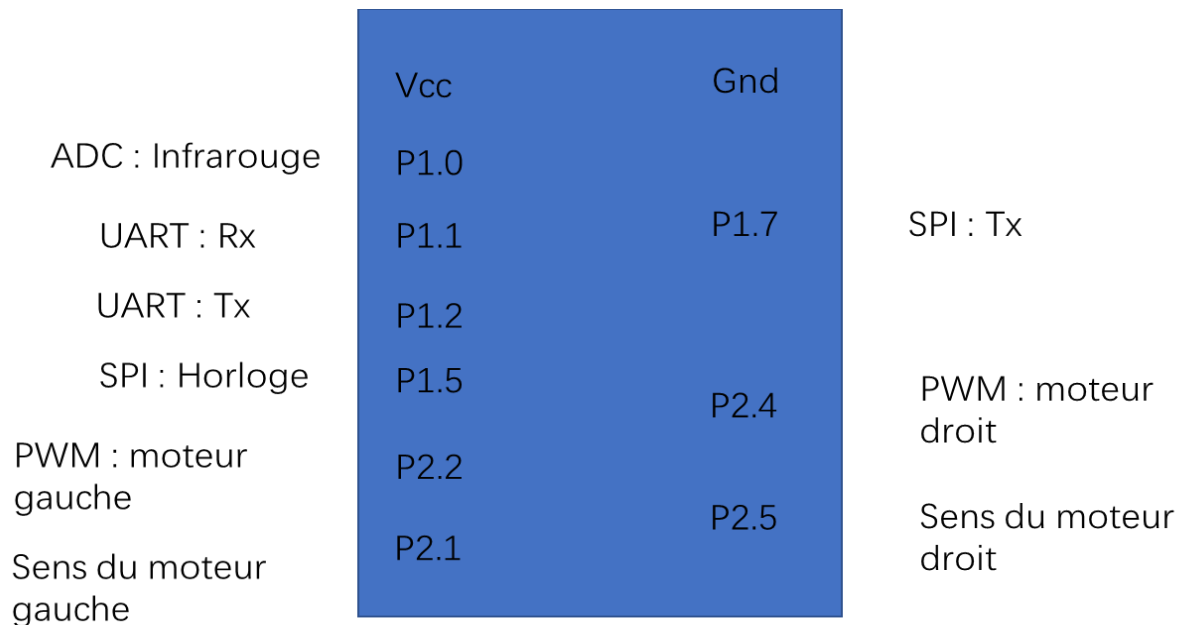
Les parties les plus important sont transporter les informations entre deux cartes de microprocesseurs de types différents par SPI et aussi faire communication entre une carte de microprocesseur et le Bluetooth par UART.

### **1-3) 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 MSP430G2231(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 :

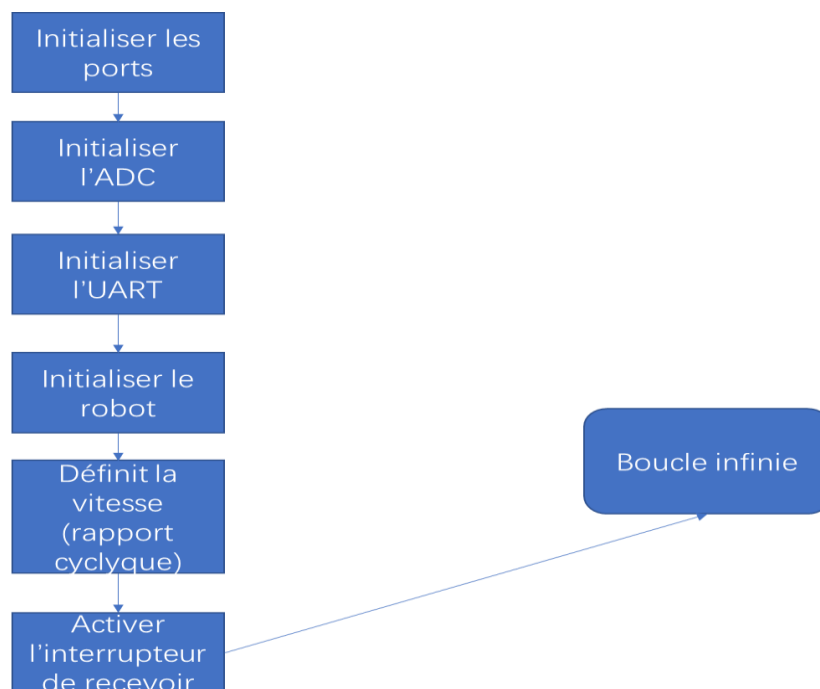


Figure 10 : Organigramme d'initialisation

### Mode autonome :

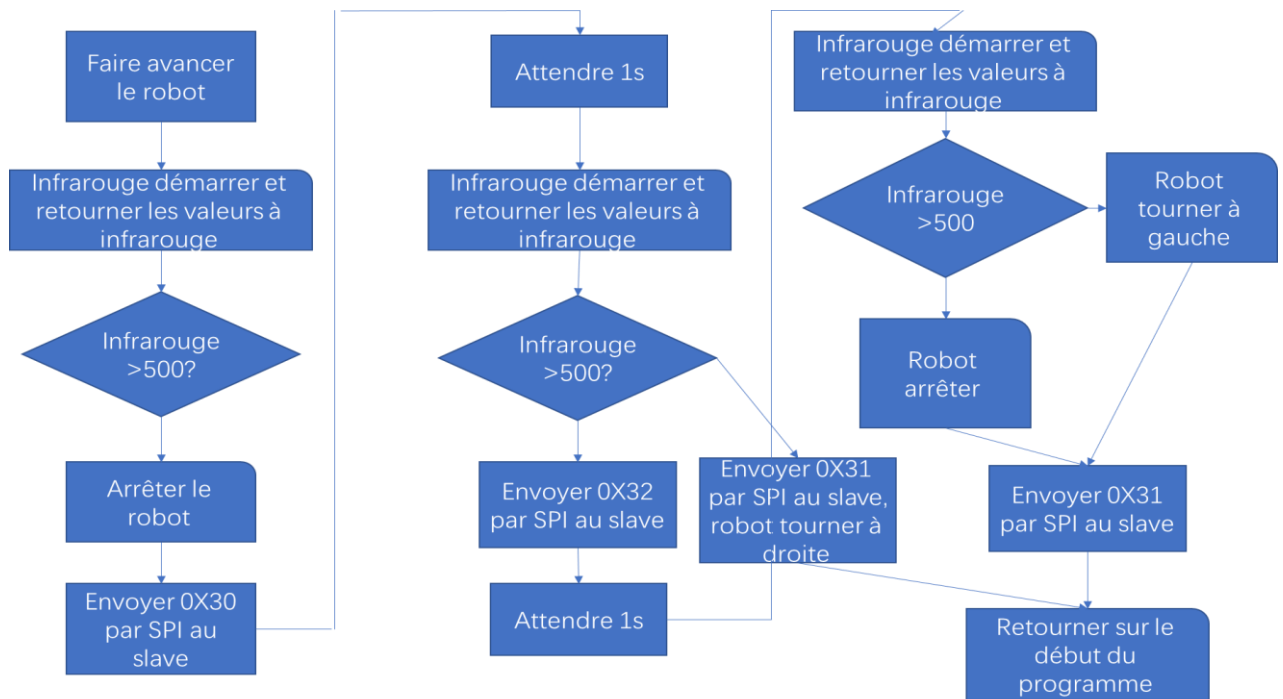


Figure 11 : Organigramme de mode automatique

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 :

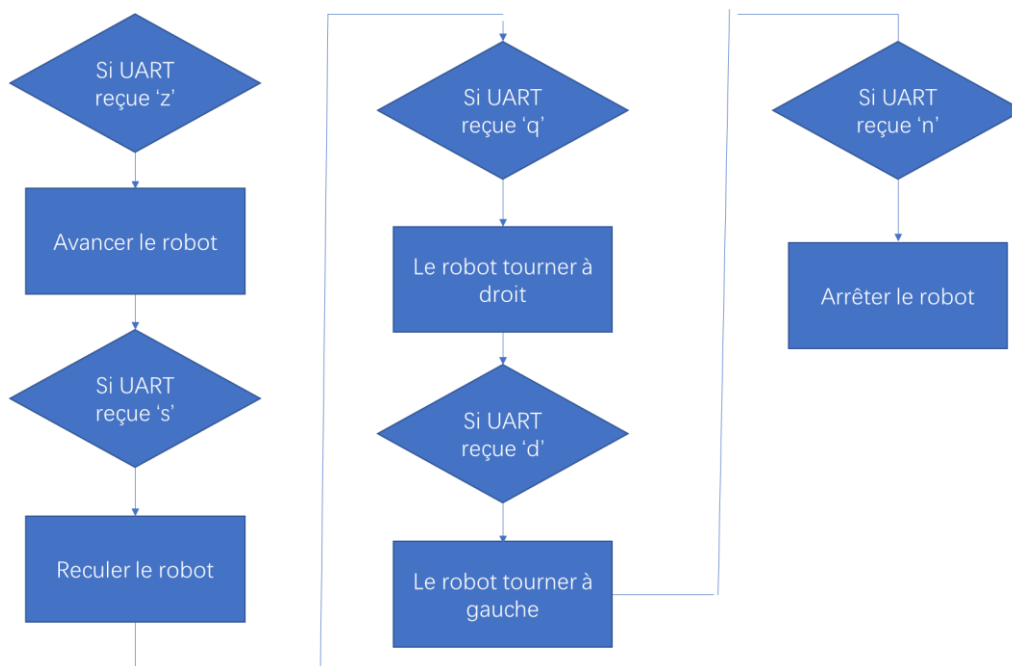


Figure 12 : mode manuel

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

### Menu :

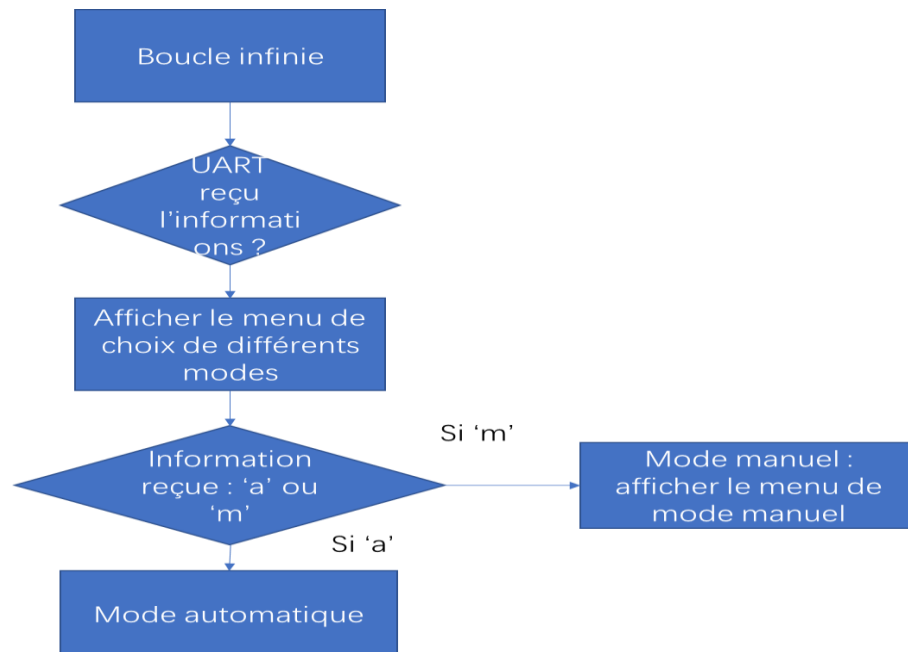


Figure 13 : Organigramme menu

### 2-2-b) Slave :

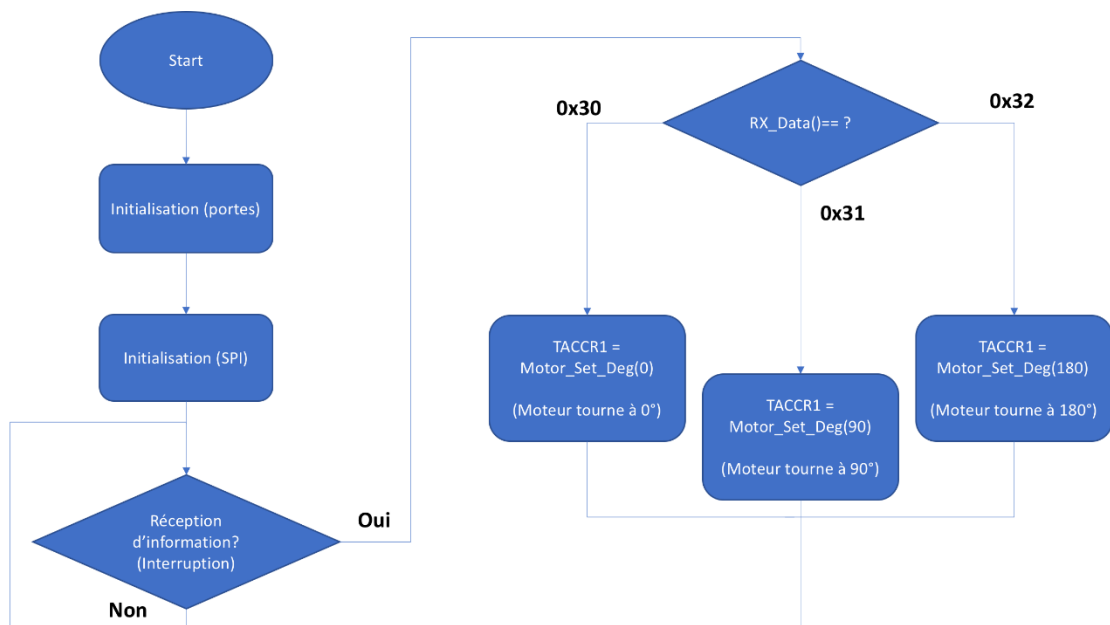


Figure 14 : organigramme de slave

## **2-3 ) Présentation de différent module :**

### **2-3-a) UART :**

#### **1) Présentation générale :**

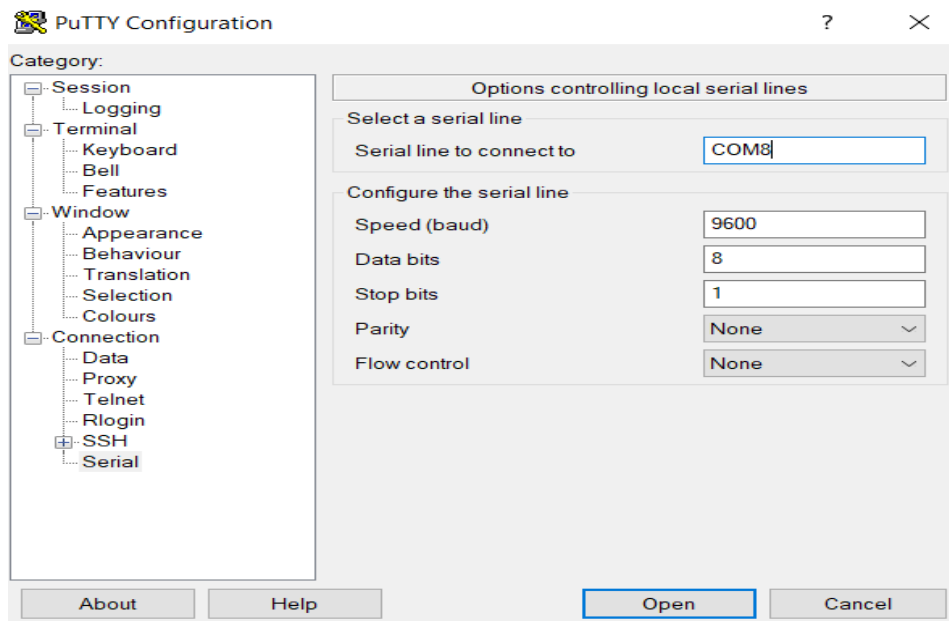
Pour la partie Bluetooth, nous avons décidé d'utiliser le mode UART pour faire la communication entre MSP430G2553 et l'appareil Bluetooth (téléphone portable ou ordinateur avec fonctionnalité Bluetooth). Pour réaliser ce but, nous avons utilisé une carte de connecteur pour connecter le microprocesseur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

#### **2) Spécification détaillée :**

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

- **Configuration**
  - **Débit : 9600**
  - **Données: 7**
  - **Parité : sans**
  - **Stop : 1**

Nous pouvons clairement voir que nous avons configuré le baud rate en 9600. Il n'est pas super vite. Mais c'est largement suffisant pour notre projet. Nous avons mis les bits de donné en 8(Comme il est affiché en ASCII). 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.

### 3) 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 registres 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.

$$\text{CLK} = F/\text{débit} \Rightarrow 1000000/9600 = 104$$

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



## Configuration DCE : UCAxCTLx

- reset : UCA0CTL1 = UCSWRST
- Vitesse : UCA0BRO = 104 ET UCA0BR1 = 0
- Modulation : UCA0MCTL1 = UCBS0
- Fin reset : UCA0CTL1 &= ~UCSWRST

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

### Print\_str :

Nom de la fonction :	<u>Print_str</u>	
Description :	Cette fonction permet d'envoyer et afficher une chaine de caractères sur hyperterminal	
	Type :	Paramètre :
Input :	unsigned	Char*c
Output :	non	non

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

### TXdata :

Nom de la fonction :	TXdata	
Description :	Cette fonction permet d'envoyer et d'afficher un seul caractère sur hyperterminal	
	Type :	Paramètre :
Input :	unsigned	Char c
Output :	non	non

Cette fonction est la fonction basique pour envoyer les caractères. Chaque fois qu'il envoie un caractère, le programme vérifie si le buffer Tx est bien disponible.

## 2-3-b) Partie IR

### 1) Objectif

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

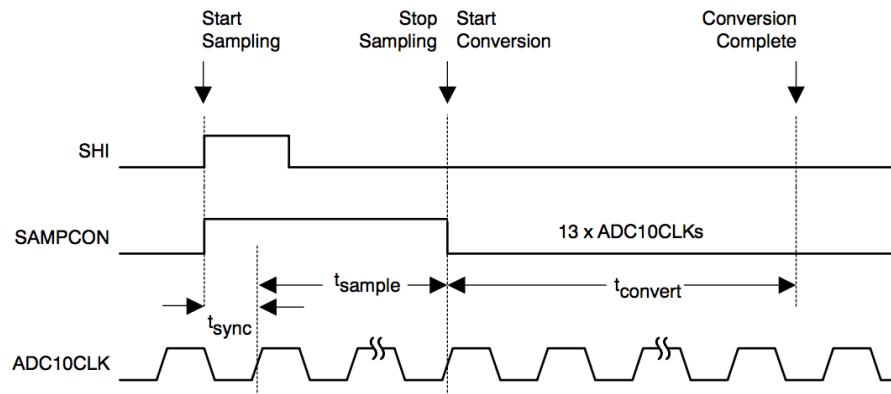
## 2) Spécifications de cette infrarouge

### 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.

Figure 20-3. Sample Timing



### 2-b) Registres concernés :

#### 20.3 ADC10 Registers

The ADC10 registers are listed in Table 20-3.

Table 20-3. ADC10 Registers

Register	Short Form	Register Type	Address	Initial State
ADC10 input enable register 0	ADC10AE0	Read/write	04Ah	Reset with POR
ADC10 input enable register 1	ADC10AE1	Read/write	04Bh	Reset with POR
ADC10 control register 0	ADC10CTL0	Read/write	01B0h	Reset with POR
ADC10 control register 1	ADC10CTL1	Read/write	01B2h	Reset with POR
ADC10 memory	ADC10MEM	Read	01B4h	Unchanged
ADC10 data transfer control register 0	ADC10DTC0	Read/write	048h	Reset with POR
ADC10 data transfer control register 1	ADC10DTC1	Read/write	049h	Reset with POR
ADC10 data transfer start address	ADC10SA	Read/write	01BCh	0200h with POR

Le cœur ADC10 est configuré par deux registres de contrôle, ADC10CTL0 et ADC10CTL1. Le bit ADC10ON s'occupe de l'activation de noyau.

### 3) 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 :

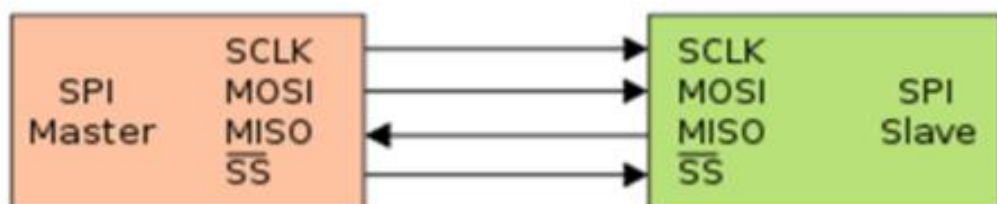
#### 1) Présentation générale :

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. Normalement, le mode SPI demande 3 ou 4 fils de liaisons.

Comme le schéma présenté suivant :

### Requiert 4 liaisons de connexions

- MOSI : *Master Out Slave In*
  - MISO : *Master In Slave Out*
  - SCK : *Serial Clock*
  - SS : *Slave Select*
- 
- Communication **Full-Duplex**

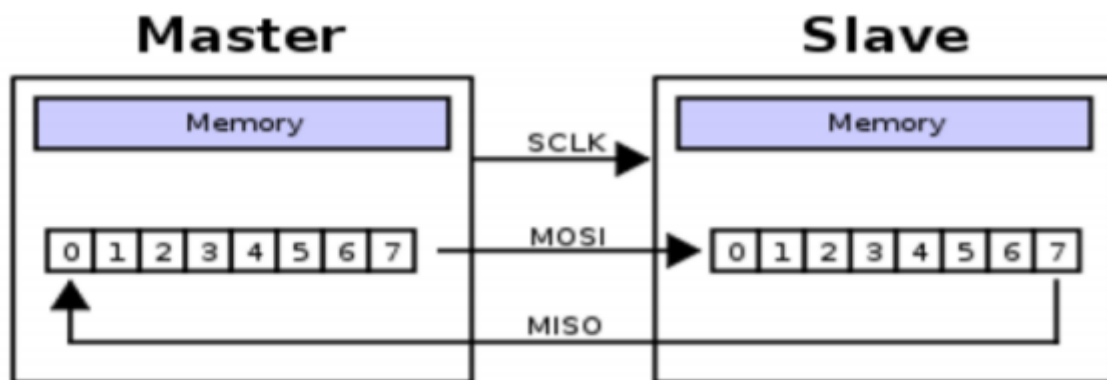


Mais notre projet n'a qu'une seule slave. Il n'a pas besoin de slave select. De plus, nous n'avons pas besoin de retour d'information. Donc nous avons supprimé le fil MISO et il ne reste que le SCK(Clock) et le MOSI(Tx).

*SCK : Serial Clock*

*MOSI : Master Out Slave In*

Le SPI fait 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 registres pour faire la communication SPI.

- **UCA0**
  - P1.1 : UCA0SOMI
  - P1.2 : UCA0SIMO
  - P1.4 : UCA0CLK
- **UCAB0**
  - P1.6 : UCB0SOMI
  - P1.7 : UCB0SIMO
  - P1.5 : UCB0CLK

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


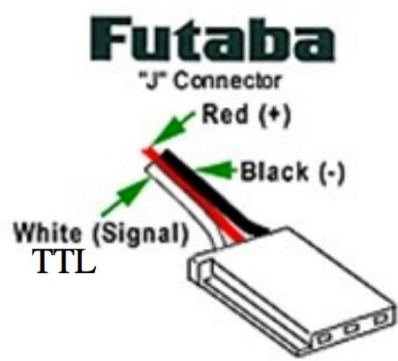
Nom de la fonction :	RX_Data	
Description :	Cette fonction permet de recevoir l'information	
	Type :	Paramètre :
Input :	non	non
Output :	non	non

## **2-3-d) Partie servomoteur :**

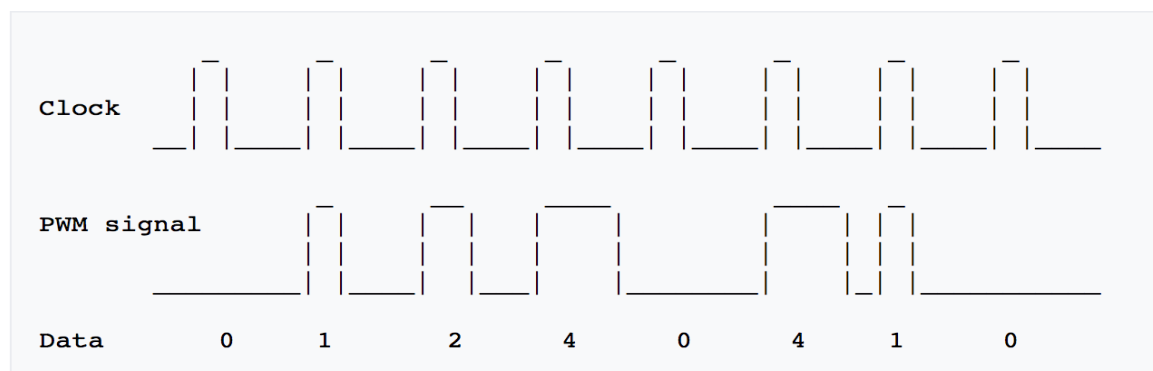
### **1) Objectif :**

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

## 2) Spécifications de ce moteur :

<p>SERVO-MOTEUR FUTABA S3003</p>	<p>Dimensions: 40,4 x 19,8 x 36,0 mm Poids: 37 g Couple: 4,1 Kg.cm Vitesse: 0,19s / 60° Alimentation: 4,8 - 6 V Roulements: Néant Engrenage: Plastique</p> 
<p>Le signal de commande est de niveau TTL (transistor transistor logic).</p>	

### 2-a) Modulation de largeur d'impulsion(PWM) :



### Explication :

Pour un signal PWM, on parle de

- Période(T)



- Largeur d'impulsion(D)

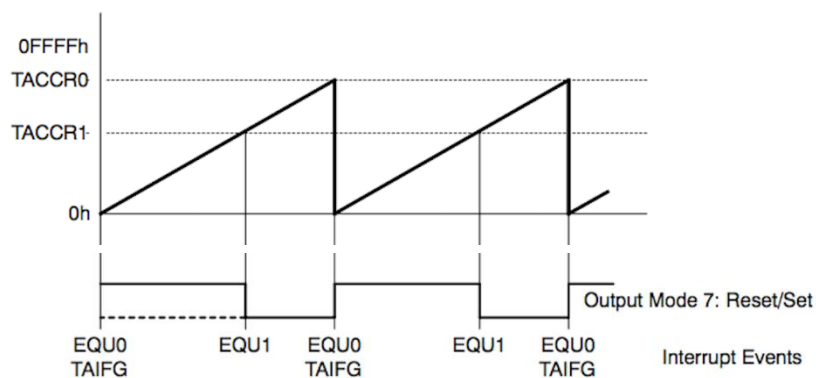
Et

$$\text{Cycle de service (Duty cycle)} = D/T \times 100\%$$

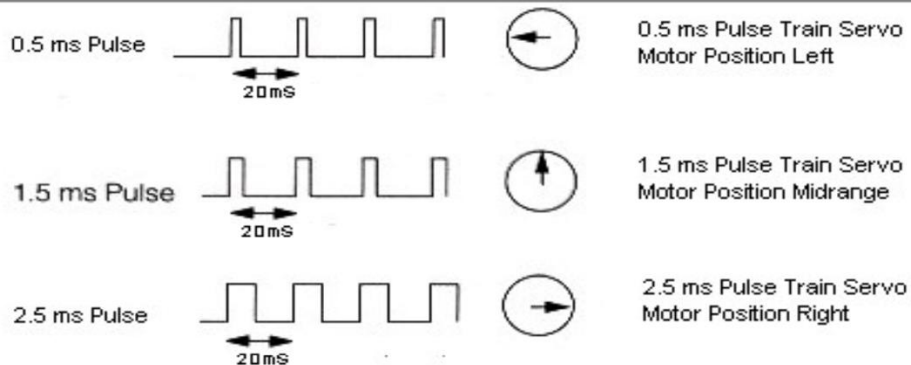
Dans une carte de msp430, on peut réaliser un signal PWM par le TIMERA. On choisit l'Output Mode 7, et

TACCR0 = 20000 ; //Période = 20000 $\mu$ s = 20ms

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



## 2-b) PWM concerné à ce servomoteur



Pulse Width Duty / ms	Angle / degrees
0.5	0
1.0	45
1.5	90 (center)
2.0	135
2.5	180

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 =  $(2500 - 500) / 180 = 11.11 \mu s = 0.0111 ms$

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

### 3) Registres concernés :

#### 12.3 Timer\_A Registers

The Timer\_A registers are listed in Table 12-3.

Table 12-3. Timer\_A Registers

Register	Short Form	Register Type	Address	Initial State
Timer_A control	TACTL	Read/write	0160h	Reset with POR
Timer_A counter	TAR	Read/write	0170h	Reset with POR
Timer_A capture/compare control 0	TACCTL0	Read/write	0162h	Reset with POR
Timer_A capture/compare 0	TACCR0	Read/write	0172h	Reset with POR
Timer_A capture/compare control 1	TACCTL1	Read/write	0164h	Reset with POR
Timer_A capture/compare 1	TACCR1	Read/write	0174h	Reset with POR
Timer_A capture/compare control 2	TACCTL2†	Read/write	0166h	Reset with POR
Timer_A capture/compare 2	TACCR2†	Read/write	0176h	Reset with POR
Timer_A interrupt vector	TAIV	Read only	012Eh	Reset with POR

† Not present on MSP430x20xx Devices

### 4) 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.

#### **2) 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.

#### **3) Registres concernés**

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

#### **4) 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 le 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

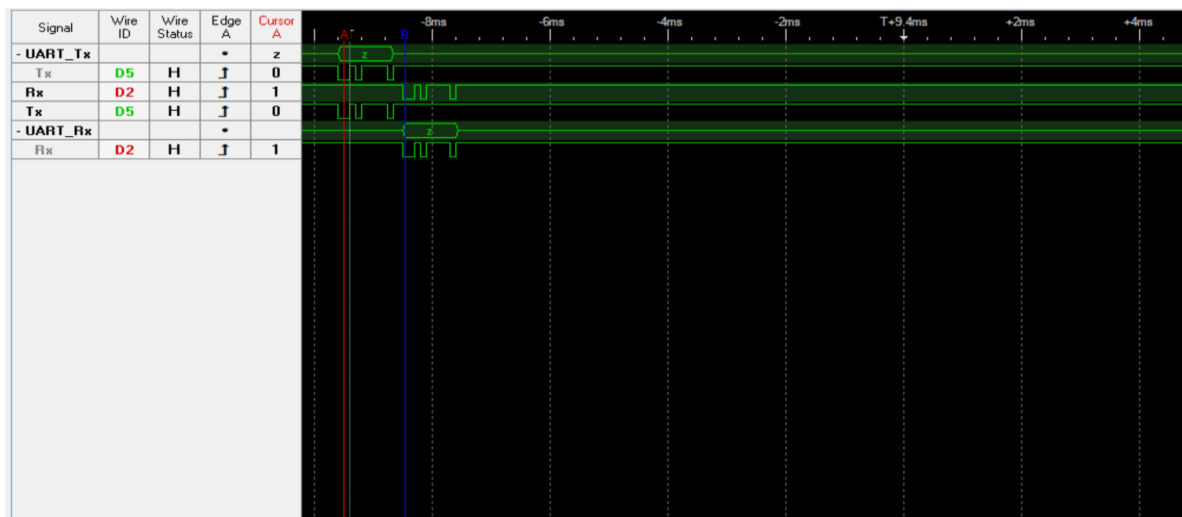
void tempo (unsigned int ms) :

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

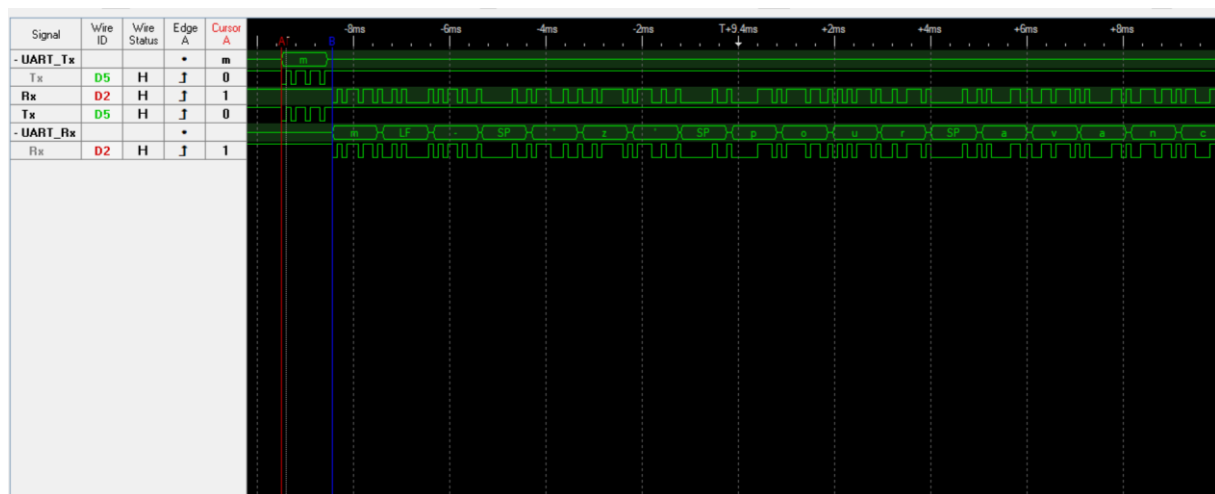
### **3) Test unitaire :**

#### **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 si 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
----------	-----

1010 0101	UCA0CTL1	0x80	USCI A0 Control Register 1 [Memory Mapped]
1010 0101	UCSSEL	10 - UCSSEL_2	USCI0 Clock Source Select 1
1010 0101	UCRXEIE	0	RX Error interrupt enable
1010 0101	UCBRKIE	0	Break interrupt enable
1010 0101	UCDORM	0	Dormant (Sleep) Mode
1010 0101	UCTXADDR	0	Send next Data as Address
1010 0101	UCTXBRK	0	Send next Data as Break
1010 0101	UCSWRST	0	USCI Software Reset
1010 0101	UCA0BR0	0x68	USCI A0 Baud Rate 0 [Memory Mapped]
1010 0101	UCA0BR1	0x00	USCI A0 Baud Rate 1 [Memory Mapped]
1010 0101	UCA0MCTL	0x00	USCI A0 Modulation Control [Memory Mapped]
1010 0101	UCA0STAT	0x00	USCI A0 Status Register [Memory Mapped]
1010 0101	UCA0RXBUF	0x00	USCI A0 Receive Buffer [Memory Mapped]
1010 0101	UCA0TXBUF	0x0A	USCI A0 Transmit Buffer [Memory Mapped]

Tips : 0x68 = 104

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

Nom de la fonction :	ADC_Lire_resultat ()
Méthode de test	Créer un variable global infrarouge(int), vérifier si cette valeur respecte la règle.
Résultat attendu	S'il y a un objet devant, cette valeur est plus élevée, sinon, elle est plus bas
Résultat obtenu	Voir sur le photo
Validé ?	Oui

pression	Type	Value	Address
infrarouge	int	612	0x0200
i	unknown	identifier not found: i	

Tip : un objet devant

infrarouge	int	298	0x0200
------------	-----	-----	--------

Tip : pas d'objet

Nom de la fonction :	ADC_init
Méthode de test	Voir si 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

▲ 1010 0101	ADC10CTL0	0x0070	ADC10 Control 0 [Memory Mapped]
1010 0101	SREF	000 - SREF_0	ADC10 Reference Select Bit: 0
1010 0101	ADC10SHT	00 - ADC10SHT_0	ADC10 Sample Hold Select Bit: 0
1010 0101	ADC10SR	0	ADC10 Sampling Rate 0:200kps / 1:50kps
1010 0101	REFOUT	0	ADC10 Enalbe output of Ref.
1010 0101	REFBURST	0	ADC10 Reference Burst Mode
1010 0101	MSC	0	ADC10 Multiple SampleConversion
1010 0101	REF2_5V	1	ADC10 Ref 0:1.5V / 1:2.5V
1010 0101	REFON	1	ADC10 Reference on
1010 0101	ADC10ON	1	ADC10 On/Enable
1010 0101	ADC10IE	0	ADC10 Interrupt Enalbe
1010 0101	ADC10IFG	0	ADC10 Interrupt Flag
1010 0101	ENC	0	ADC10 Enable Conversion
1010 0101	ADC10SC	0	ADC10 Start Conversion
▲ 1010 0101	ADC10CTL1	0x0010	ADC10 Control 1 [Memory Mapped]
1010 0101	INCH	0000 - INCH_0	ADC10 Input Channel Select Bit: 0
1010 0101	SHS	00 - SHS_0	ADC10 Sample/Hold Source Bit: 0
1010 0101	ADC10DF	0	ADC10 Data Format 0:binary 1:2's complem
1010 0101	ISSH	0	ADC10 Invert Sample Hold Signal
1010 0101	ADC10DIV	000 - ADC10DIV_0	ADC10 Clock Divider Select Bit: 0
1010 0101	ADC10SSEL	10 - ADC10SSEL_2	ADC10 Clock Source Select Bit: 0
1010 0101	CONSEQ	00 - CONSEQ_0	ADC10 Conversion Sequence Select 0
1010 0101	ADC10BUSY	0	ADC10 BUSY

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

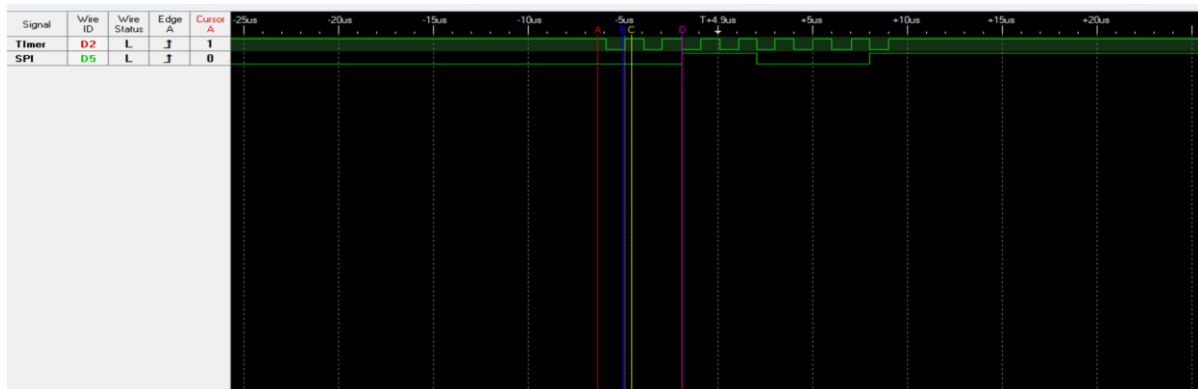
De même comme la spécification détaillée, nous avons séparé le test unitaire en deux parties :

#### **3-c-1) Master :**

Nom de la fonction :	SPI_Init
Méthode de test	Voir si 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

Name	Value	Description
USCI_A0_UART_Mode		
USCI_A0_SPI_Mode		
USCI_B0_SPI_Mode		
UCB0CTL0_SPI	0x69	USCI B0 Control Register 0 [Memory Mapped]
UCCKPH	0	Sync. Mode: Clock Phase
UCCKPL	1	Sync. Mode: Clock Polarity
UCMSB	1	Sync. Mode: MSB first 0:LSB / 1:MSB
UC7BIT	0	Sync. Mode: Data Bits 0:8-bits / 1:7-bits
UCMST	1	Sync. Mode: Master Select
UCMODE	00 - UCMODE_0	Sync. Mode: USCI Mode 1
UCSYNC	1	Sync-Mode 0:UART-Mode / 1:SPI-Mode
UCB0CTL1_SPI	0x80	USCI B0 Control Register 1 [Memory Mapped]
UCSSEL	10 - UCSSEL_2	USCI1 Clock Source Select 1
UCSWRST	0	USCI Software Reset
UCB0BR0_SPI	0x02	USCI B0 Baud Rate 0 [Memory Mapped]
UCB0BR1_SPI	0x00	USCI B0 Baud Rate 1 [Memory Mapped]
UCB0STAT_SPI	0x00	USCI B0 Status Register [Memory Mapped]
UCB0RXBUF_SPI	0x00	USCI B0 Receive Buffer [Memory Mapped]
UCB0TXBUF_SPI	0x00	USCI B0 Transmit Buffer [Memory Mapped]
USCI_B0_I2C_Mode		
Watchdog_Timer		

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



Name	Value	Description
USCI_A0_UART_Mode		
USCI_A0_SPI_Mode		
USCI_B0_SPI_Mode		
UCB0CTL0_SPI	0x69	USCI B0 Control Register 0 [Memory Mapped]
UCB0CTL1_SPI	0x80	USCI B0 Control Register 1 [Memory Mapped]
UCB0BR0_SPI	0x02	USCI B0 Baud Rate 0 [Memory Mapped]
UCB0BR1_SPI	0x00	USCI B0 Baud Rate 1 [Memory Mapped]
UCB0STAT_SPI	0x00	USCI B0 Status Register [Memory Mapped]
UCB0RXBUF_SPI	0x00	USCI B0 Receive Buffer [Memory Mapped]
UCB0TXBUF_SPI	0x32	USCI B0 Transmit Buffer [Memory Mapped]
USCI_B0_I2C_Mode		
Watchdog_Timer		
Calibration_Data		
TLV Calibration Data		

### 3-c-2) Slave :

Nom de la fonction :	Init_SPI
Méthode de test	Voir si 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

USI		
USICTL0	0xE2	USI Control Register 0 [Memory Mapped]
USIPE7	1	USI Port Enable Px.7
USIPE6	1	USI Port Enable Px.6
USIPE5	1	USI Port Enable Px.5
USILSB	0	USI LSB first 1:LSB / 0:MSB
USIMST	0	USI Master Select 0:Slave / 1:Master
USIGE	0	USI General Output Enable Latch
USIOE	1	USI Output Enable
USISWRST	0	USI Software Reset
USICTL1	0x90	USI Control Register 1 [Memory Mapped]
USICKPH	1	USI Sync. Mode: Clock Phase
USI2C	0	USI I2C Mode
USISTTIE	0	USI START Condition interrupt enable
USIIE	1	USI Counter Interrupt enable
USIAL	0	USI Arbitration Lost
USISTP	0	USI STOP Condition received
USISTTIFG	0	USI START Condition interrupt Flag
USIIFG	0	USI Counter Interrupt Flag

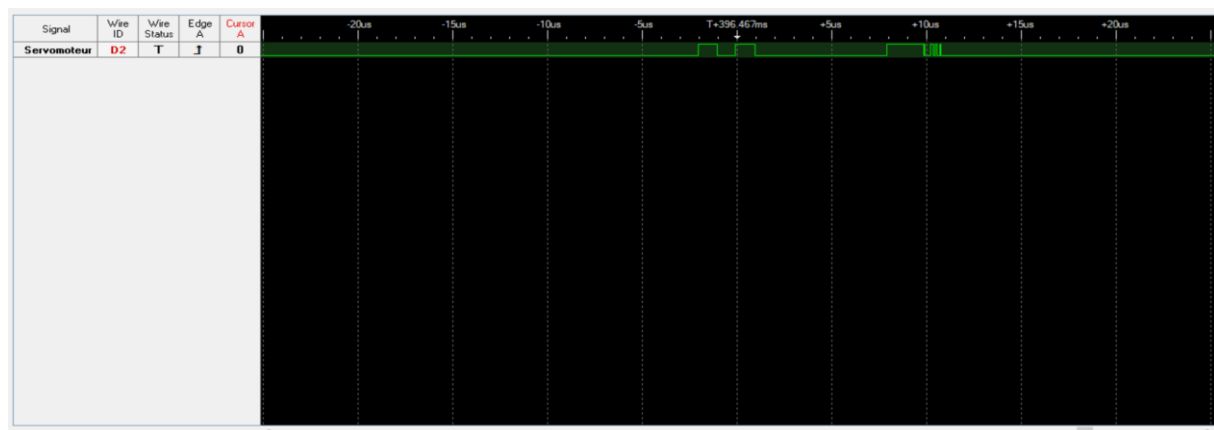
Nom de la fonction :	RX_Data
Méthode de test	Recevoir 0x32 par le slave
Résultat attendu	0x32 est reçu par le slave
Résultat obtenu	Voir sur le photo
Validé ?	Oui

USICTL1	0x91	USI Control Register 1 [Memory Mapped]
USICKCTL	0x00	USI Clock Control Register [Memory Mapped]
USICNT	0x00	USI Bit Counter Register [Memory Mapped]
USISRL	0x32	USI Low Byte Shift Register [Memory Mapped]
USISRH	0x00	USI High Byte Shift Register [Memory Mapped]
USICTL	0x91E2	USI Control Register [Memory Mapped]
USICCTL	0x0000	USI Clock and Counter Control Register [Memory Mapped]
USISR	0x0032	USI Shift Register [Memory Mapped]
Watchdog_Timer		
Calibration_Data		


























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

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 pour 90° 3) 2500 pour 180°
Résultat obtenu	Voir sur le photo
Validé ?	Oui



Nom de la fonction :	Motor_PWM_Init
Méthode de test	Voir si 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

 Timer_A2		
 TAIV	0x0000	Timer A Interrupt Vector Word [Memory Mapped]
 TACTL	0x0211	Timer A Control [Memory Mapped]
 TASSEL	10 - TASSEL_2	Timer A clock source select 1
 ID	00 - ID_0	Timer A clock input divider 1
 MC	01 - MC_1	Timer A mode control 1
 TACLR	0	Timer A counter clear
 TAIE	0	Timer A counter interrupt enable
 TAIFG	1	Timer A counter interrupt flag
 TACCTL0	0x0409	Timer A Capture/Compare Control 0 [Memory Mapped]
 TACCTL1	0x00E1	Timer A Capture/Compare Control 1 [Memory Mapped]
 CM	00 - CM_0	Capture mode 1
 CCIS	00 - CCIS_0	Capture input select 1
 SCS	0	Capture synchronize
 SCCI	0	Latched capture signal (read)
 CAP	0	Capture mode: 1 / Compare mode : 0
 OUTMOD	111 - OUTMOD_7	Output mode 2
 CCIE	0	Capture/compare interrupt enable
 CCI	0	Capture input signal (read)
 OUT	0	PWM Output signal if output mode 0
 COV	0	Capture/compare overflow flag
 CCIFG	1	Capture/compare interrupt flag
 TAR	0x2FD3	Timer A Counter Register [Memory Mapped]
 TACCR0	20000 (Decimal)	Timer A Capture/Compare 0 [Memory Mapped]
 TACCR1	1500 (Decimal)	Timer A Capture/Compare 1 [Memory Mapped]

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

Nom de la fonction :	Init_Robot
Méthode de test	Voir si 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

Name	Value	Description
▲ Timer1_A3		
1010 0101 TA1IV	0x0000	Timer1_A3 Interrupt Vector Word [Memory Mapped]
▲ 1010 0101 TA1CTL	0x0211	Timer1_A3 Control [Memory Mapped]
1010 0101 TASSEL	10 - TASSEL_2	Timer A clock source select 1
1010 0101 ID	00 - ID_0	Timer A clock input divider 1
1010 0101 MC	01 - MC_1	Timer A mode control 1
1010 0101 TACLR	0	Timer A counter clear
1010 0101 TAIE	0	Timer A counter interrupt enable
1010 0101 TAIFG	1	Timer A counter interrupt flag
▶ 1010 0101 TA1CCTL0	0x0001	Timer1_A3 Capture/Compare Control 0 [Memory Mapped]
▲ 1010 0101 TA1CCTL1	0x00E1	Timer1_A3 Capture/Compare Control 1 [Memory Mapped]
1010 0101 CM	00 - CM_0	Capture mode 1
1010 0101 CCIS	00 - CCIS_0	Capture input select 1
1010 0101 SCS	0	Capture synchronize
1010 0101 SCCI	0	Latched capture signal (read)
1010 0101 CAP	0	Capture mode: 1 / Compare mode : 0
1010 0101 OUTMOD	111 - OUTMOD_7	Output mode 2
1010 0101 CCIE	0	Capture/compare interrupt enable
1010 0101 CCI	0	Capture input signal (read)
1010 0101 OUT	0	PWM Output signal if output mode 0
1010 0101 COV	0	Capture/compare overflow flag
1010 0101 CCIFG	1	Capture/compare interrupt flag
▲ 1010 0101 TA1CCTL1	0x00E1	Timer1_A3 Capture/Compare Control 1 [Memory Mapped]
1010 0101 CM	00 - CM_0	Capture mode 1
1010 0101 CCIS	00 - CCIS_0	Capture input select 1
1010 0101 SCS	0	Capture synchronize
1010 0101 SCCI	0	Latched capture signal (read)
1010 0101 CAP	0	Capture mode: 1 / Compare mode : 0
1010 0101 OUTMOD	111 - OUTMOD_7	Output mode 2
1010 0101 CCIE	0	Capture/compare interrupt enable
1010 0101 CCI	0	Capture input signal (read)
1010 0101 OUT	0	PWM Output signal if output mode 0
1010 0101 COV	0	Capture/compare overflow flag
1010 0101 CCIFG	1	Capture/compare interrupt flag
▲ 1010 0101 TA1CCTL2	0x00E9	Timer1_A3 Capture/Compare Control 2 [Memory Mapped]
1010 0101 CM	00 - CM_0	Capture mode 1
1010 0101 CCIS	00 - CCIS_0	Capture input select 1
1010 0101 SCS	0	Capture synchronize
1010 0101 SCCI	0	Latched capture signal (read)
1010 0101 CAP	0	Capture mode: 1 / Compare mode : 0
1010 0101 OUTMOD	111 - OUTMOD_7	Output mode 2
1010 0101 CCIE	0	Capture/compare interrupt enable
1010 0101 CCI	1	Capture input signal (read)
1010 0101 OUT	0	PWM Output signal if output mode 0
1010 0101 COV	0	Capture/compare overflow flag
1010 0101 CCIFG	1	Capture/compare interrupt flag
1010 0101 TA1R	0x003A	Timer1_A3 Counter Register [Memory Mapped]
1010 0101 TA1CCR0	200 (Decimal)	Timer1_A3 Capture/Compare 0 [Memory Mapped]
1010 0101 TA1CCR1	190 (Decimal)	Timer1_A3 Capture/Compare 1 [Memory Mapped]
1010 0101 TA1CCR2	190 (Decimal)	Timer1_A3 Capture/Compare 2 [Memory Mapped]

Nom de la fonction :	Choix_direction
Méthode de test	Voir si 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

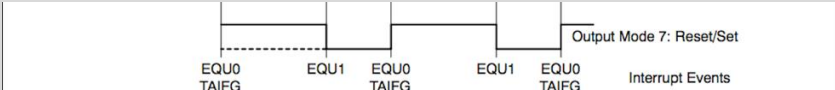
Nom de la fonction :	Vitesse_moteurs
Méthode de test	Voir si les registres est bien identique que ceux qu'on veut. (TA1CCR1 et TA1CCR2)
Résultat attendu	Comparer les valeurs de registre (TA1CCR1 et TA1CCR2) avec les entrées de fonction (vit_gauche et vit_droite). Voir sur la spécification : vit_gauche = 190, vit_droite = 190.
Résultat obtenu	Voir sur le photo
Validé ?	Oui

1010 0101	TA1CCR0	200 (Decimal)	Timer1_A3 Capture/Compare 0 [Memory Map]
1010 0101	TA1CCR1	190 (Decimal)	Timer1_A3 Capture/Compare 1 [Memory Map]
1010 0101	TA1CCR2	190 (Decimal)	Timer1_A3 Capture/Compare 2 [Memory Map]

Nom de la fonction :	Arret_robot
Méthode de test	Voir si 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
----------	-----

OUTMOD	000 - OUTMOD_0	Output mode 2
--------	----------------	---------------

Nom de la fonction :	Demarrer_robot
Méthode de test	Voir si 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. 
Résultat obtenu	Voir sur le photo
Validé ?	Oui

Name	Value	Description
Timer1_A3		
TA1IV	0x0000	Timer1_A3 Interrupt Vector Word [Memory Mapped]
TA1CTL	0x0211	Timer1_A3 Control [Memory Mapped]
TASSEL	10 - TASSEL_2	Timer A clock source select 1
ID	00 - ID_0	Timer A clock input divider 1
MC	01 - MC_1	Timer A mode control 1
TACLR	0	Timer A counter clear
TAIE	0	Timer A counter interrupt enable
TAIFG	1	Timer A counter interrupt flag
TA1CCTL0	0x0001	Timer1_A3 Capture/Compare Control 0 [Memory Mapped]
TA1CCTL1	0x00E1	Timer1_A3 Capture/Compare Control 1 [Memory Mapped]
CM	00 - CM_0	Capture mode 1
CCIS	00 - CCIS_0	Capture input select 1
SCS	0	Capture synchronize
SCCI	0	Latched capture signal (read)
CAP	0	Capture mode: 1 / Compare mode: 0
OUTMOD	111 - OUTMOD_7	Output mode 2
CCIE	0	Capture/compare interrupt enable
CCI	0	Capture input signal (read)
OUT	0	PWM Output signal if output mode 0
COV	0	Capture/compare overflow flag
CCIFG	1	Capture/compare interrupt flag

TA1CCTL1	0x00E1	Timer1_A3 Capture/Compare Control 1 [Memory Mapped]
OUTMOD	111 - OUTMOD_7	Output mode 2
CCIE	0	Capture/compare interrupt enable

Tip : moteur gauche

TA1CCTL0	0x0001	Timer1_A3 Capture/Compare Control 0 [Memory Mapped]
OUTMOD	111 - OUTMOD_7	Output mode 2
CCIE	0	Capture/compare interrupt enable

Tip : moteur droite

## 4) Gestion de configuration :

### 4-1) Lien de repositories :

Master (G2553):

[https://github.com/LiZhengxi/MSP\\_Projet\\_G2553.git](https://github.com/LiZhengxi/MSP_Projet_G2553.git)

Slave (G2231) :

[https://github.com/LiZhengxi/MSP\\_Projet\\_G2231.git](https://github.com/LiZhengxi/MSP_Projet_G2231.git)

### 4-2) Repositories utilisées :

Pour notre projet, nous avons utilisé le GitHub pour gérer les différentes versions du programme. Parce que le système d'ordinateur personnel de chaque membre n'est pas même (MacOs et Windows). Donc nous décidons d'utiliser un repositories compatible pour les deux systèmes. Mais en deux logiciels différents : TortoiseSVN(Windows) et Gitkraken(MacOs).

### 4-3) Différents dossiers sur repositories :

Nous avons séparé les codes en deux repositories différents. Les codes sont présentés par archive file. Donc nous pouvons importer directement dans le logiciel CCS

#### **Master (G2553) :**








Dans ce repositories, nous avons mis 3 archive files plus un dossier de rapport. Les 3 archive files sont :

- 1) Master(infrarouge\_spi).zip ;
- 2) Master(bluetooth(uart)\_SAMbot).zip
- 3) Master (version finale).zip

Le premier dossier est le code pour le mode autonome. Il possède les codes infrarouge et SPI(master).

Le deuxième dossier est le code pour le mode manuel. Il possède les codes Bluetooth et SAMbot.

Et le dernier dossier est le projet final. Il a mélangé les deux premiers et faire les petites modifications. De plus, nous avons rajouté les commentaires. Pour qu'il soit plus lisible.

 LiZhengxi	No commit message	Latest commit dae3fa4 18 minutes ago
 Rapport projet bus de com	No commit message	17 minutes ago
 Master(bluetooth(uart_SAMbot).zip	No commit message	17 minutes ago
 Master(infrarouge_spi).zip	No commit message	17 minutes ago
 Master(version finale).zip	Dernière version avec les commentaires du codes. Il manque juste de m...	4 days ago
 README.md	Initial commit	17 days ago
 uart g2553(Bus de com_TP1).c	No commit message	17 minutes ago

## Slave (G2231) :






Dans ce repositories, nous avons mis 2 archive files.

Les 2 archive files sont :

- 1) SPI\_Slave(version1.2).zip
- 2) Slave (version finale).zip

Le premier dossier est notre TP2 de cours SPI et servomoteur (une seule commande).

Le deuxième dossier est le projet final, il a été modifié à partir de premier dossier. Il a ajouté quelques différentes commandes de commander le servomoteur tourner à différent angle.

 LiZhengxi	Version finale : remettre tous les fonctions dans les différents bibl... 	Latest commit 8867238 7 days ago
 README.md	Create README.md	17 days ago
 SPI_Slave(version1.2).zip	Permet de recevoir le caractère 0X10 et commander le servomoteur tour...	10 days ago
 Slave(version finale).zip	Version finale : remettre tous les fonctions dans les différents bibl...	7 days ago

## **Conclusion :**

Globalement, notre projet a 5 parties : SAMbot, SPI entre deux microprocesseurs, Bluetooth(UART), servomoteur et capteur IR. Parmi eux, la partie la plus difficile est la communication comme les modules UART et SPI. De plus, le contrôle du signal PWM est également une partie importante. Heureusement, nous avons fini ce projet étape par étape et appris beaucoup de connaissances de microcontrôleur avec ce projet.

## **Annex :**

### **Les schémas utilisés :**

Figure 1 : Lauchpad .....	5
Figure 2 : MSP430G2231 .....	5
Figure 3 : MSP430G2553 .....	5
Figure 4 : Servomoteur .....	6
Figure 5 : module Bluetooth (installé sur le master) .....	6
Figure 6 : L'infrarouge.....	6
Figure 7 : SAMbot .....	7
Figure 8 : Construction finale.....	7
Figure 9 : installation d'infrarouge et servomoteur .....	8
Figure 10 : Organigramme d'initialisation.....	11
Figure 11 : Organigramme de mode automatique.....	12
Figure 12 : mode manuel.....	12
Figure 13 : Organigramme menu .....	13
Figure 14 : organigramme de slave.....	13

### **Pièces jointes :**

- 1) Code Review Report\_master.mht
- 2) Code Review Report\_slave.mht
- 3) Master test\_manager.mht
- 4) Slave test\_manager.mht