

Le Scenix d'Asservissement

Blaise Gassend

10 juillet 2004

Table des matières

1	Aperçu	2
2	Brochage	2
3	Répartition de la mémoire	2
3.1	Memoire programme (EEPROM)	2
3.2	Memoire vive (RAM)	3
3.2.1	Organisation des banques de memoire	3
3.2.2	La zone de memoire commune	3
4	Les blocs fonctionnels	3
4.1	L'initialisation	3
4.2	La boucle principale	4
4.3	La routine d'interruption	4
4.4	L'asservissement	5
4.5	La génération de trapèze	6
4.6	Le calcul de position	7
4.7	Le protocole de communication	9
4.7.1	Le SPI	9
4.7.2	Les commandes	10
4.8	Le code du SPI	11
5	Les fichiers source	11
6	Les modes de fonctionnement	12
7	Améliorations possibles	13
8	Glossaire	13

1 Aperçu

Le SX d'asservissement a été mis au point pour remplacer deux LM629, chers et encombrants par un seul petit composant bon marché, qui peut de surcroît assurer des tâches supplémentaires telles que le calcul de position. Sa fonction est d'asservir en position deux moteurs munis de roues codeuses, suivant les commandes qu'il reçoit par une interface série SPI.

Les commandes reçues donnent une distance à parcourir, une vitesse et une accélération limite. Ces informations sont utilisées par un générateur de trapèze, pour fournir un point de référence à un filtre PID qui assure la commande des moteurs.

2 Brochage

Pin	Nom	Nom Scenix	Direction	Description
1	PWM2	RA2	Sortie	Signal PWM 2
2	DIR2	RA3	Sortie	Direction 2
3	\emptyset	RTCC	Entrée	Non utilisé
4	<i>MCLR</i>	<i>MCLR</i>	Entrée	0V pour remettre à zero
5	V_{dd}	V_{dd}	Alim.	Masse
6	PWM1	RB0	Entrée	Roue codeuse 1
7	DIR1	RB1	Entrée	Roue codeuse 1
8	PWM2	RB2	Entrée	Roue codeuse 2
9	DIR2	RB3	Entrée	Roue codeuse 2
10	CK	RB4	Sortie	Horloge SPI
11	RX	RB5	Entrée	Entrée du signal SPI
12	TX	RB6	Sortie	Sortie du signal SPI
13	<i>SS</i>	RB7	Entrée	Déclenchement SPI
14	V_{dd}	V_{dd}	Alim.	Alimentation +5V
15	OSC2	OSC2	Sortie	Programmation (et quartz)
16	OSC1	OSC1	Entrée	Horloge et programmation
17	QA1	RA0	Sortie	Signal PWM 1
18	QA0	RA1	Sortie	Direction 1

3 Répartition de la mémoire

3.1 Memoire programme (EEPROM)

Adresse	Contenu
000h	Initialisation, interruptions, boucle principale, calcul de position
200h	PID
400h	Génération de trapèze
600h	Execution des commandes SPI

3.2 Memoire vive (RAM)

3.2.1 Organisation des banques de memoire

Banque	Contenu
000h	Registres du SX et memoire temporaire.
010h	Gestion d'interruption.
030h	Entrées sorties SPI.
050h	PID pour la voie 1.
070h	PID pour la voie 2.
090h	Génération de trapèze voie 1.
0b0h	Génération de trapèze voie 2.
0d0h	Calcul de position.
0f0h	Calcul de position (suite).

Notes :

- Les détails de chaque banque seront donnés dans la section correspondante.
- Les variables de plus de 8 bits sont codées avec l'octet de poids faible à l'adresse la plus faible.

3.2.2 La zone de memoire commune

La memoire de 00h à 0Fh est accessible en permanence. Voici son organisation :

Adresse	Nom	Contenu
00h	INDF	Sert pour faire de l'adressage indirect (pointeurs).
01h	RTCC	Horloge du SX.
02h	PC	8 bits de poids faible du compteur d'instruction.
03h	STATUS	Contient les différents flags du SX.
04h	FSR	L'adresse qui apparait en 00h.
05h	RA	Port A. Utilisé pour le PWM.
06h	RB	Port B. Utilisé pour le SPI et les roues codeuses.
07h	inttmp	Mémoire temporaire de la routine d'interruption. Sur un SX28 ce serait le port C. Le code d'asservissement ne marche donc que sur un SX18.
08h à 0Fh	temp	Zone temporaire à usage général.

4 Les blocs fonctionnels

4.1 L'initialisation

Le code d'initialisation se contente d'effacer la memoire, de préparer les interruptions, de choisir le mode de fonctionnement du scenix et de ses pattes, et d'initialiser quelques variables nécessaires au bon fonctionnement du système.

4.2 La boucle principale

La boucle principale dépend beaucoup du mode de fonctionnement du SX. En effet il y a des modes où différentes fonctions sont désactivées pour être commandées par un PC, par exemple.

En fonctionnement normal le générateur de trapèze est appelé, puis le PID. On attend ensuite le moment de passer au cycle suivant d'asservissement en gérant éventuellement les commandes SPI qui arriveraient.

Dans le mode où le PID se fait sur PC, la boucle principale gère les communications entre le PC et le SX.

4.3 La routine d'interruption

La routine d'interruption est sans doute la partie la plus confuse, et celle qui a fait l'objet du plus de soins lors de la conception. Elle est appelée sur passage à zéro de RTCC, et aussi quand la patte \overline{SS} change d'état¹. Elle a les tâches suivantes :

- Générer les sorties PWM.
- Mettre à jour l'horloge 16 bits qui est utilisée pour savoir quand faire un cycle d'asservissement.
- Faire les échanges SPI.
- Lire les roues codeuses.
- Appeler la routine de calcul de position.

Il faut remarquer que j'ai été sévèrement limité en mémoire. La routine d'interruption utilise donc l'adresse correspondant au port C comme zone temporaire. Ceci est conforme à la documentation du SX qui stipule que sur le SX18 cette zone correspond à une mémoire normale. Il faut aussi sauvegarder par moments la valeur du registre M. Pour cela j'utilise FSR qui ne contient pas de valeur intéressante. Il ne faut donc pas s'étonner d'échanges suspects entre M et FSR dans la routine d'interruption.

Les variables utilisées sont les suivantes :

¹Je ne sais plus si ce deuxième cas est fait exprès, ni s'il est nécessaire. Il y a eu de nombreux problèmes de synchronisation et c'est peut-être juste un oubli.

Adresse	Nom	Contenu
10h à 11h	count1	Nombre de tops de la voie 1 depuis le dernier rafraichissement.
12h à 13h	count2	Nombre de tops de la voie 2 depuis le dernier rafraichissement.
14h	wkenech	Anciennement contenant une copie de WKEN du port B (Interruptions activées ou non). Maintenant ne sert que pour son bit 7 en rapport avec la gestion de \overline{SS} .
15h	oldrb	Entre interruptions, stocke l'état du port B pour pouvoir voir les changements.
16h à 17h	speed1	Valeur absolue de la tension à sortir sur la voie 1. De 0 à 1023.
18h à 19h	speed1	Valeur absolue de la tension à sortir sur la voie 2. De 0 à 1023.
1ah à 1bh	pwmerr1	Toto
1ch à 1dh	pwmerr2	Retient l'erreur commise jusqu'à présent dans la sortie PWM 2. On choisira l'état de la sortie PWM correspondante directement à partir du signe de cette valeur.
1eh à 1fh	clock	Horloge 16 bits. Dans la même unité de temps que le RTCC.

La routine d'interruption est dans servo4.asm.

4.4 L'asservissement

L'asservissement est un asservissement en position. C'est à dire qu'on tente à chaque instant de tourner le moteur dans une direction précise (on considère naturellement comme différentes des directions correspondant à un ou plusieurs tours de roue de décalage) qui peut dépendre du temps. Ici on met en oeuvre un filtre PID pour chacune des deux voies commandées. C'est un filtre linéaire dont la sortie se déduit de l'entrée par :

$$y(t) = a \int x(t) dt + b x(t) + c \frac{dx}{dt}(t)$$

Chaque terme de cette somme correspond à un besoin précis :

- Le terme Proportionnel agit comme une force de rappel. Si on n'est pas là où il faut, on s'en rapproche.
- Le terme Dérivée sert de force de frottement. En effet, si on n'a que le terme proportionnel le système oscille autour de la position d'équilibre ce qui n'est pas ce qu'on souhaite. Idéalement on essaye de se mettre dans le cas critique qui correspond à une approche non oscillante aussi rapide que possible.
- Les deux premiers termes donnent un bon asservissement quand on essaye d'atteindre un point indépendant du temps. Si le point cible bouge à vitesse uniforme, on aura à l'équilibre un retard constant par rapport au point cible. Le terme intégral va permettre de compenser ce décalage, pour qu'on soit exactement sur le point cible à l'équilibre.

Il y a quelques ajouts par rapport à cette théorie simple :

- La tension qu'on peut envoyer aux moteurs est limitée, il y a donc une saturation de $y(t)$ qui se fait lorsqu'il est trop grand.
- Si une des voies se bloque pour une raison mécanique extérieure, le terme intégral va croître vers des très grandes valeurs. Lorsqu'on débloque, la combinaison de cette grande valeur avec la saturation qui a lieu dans le point précédent fait qu'on a des oscillations violentes qui ne s'amortissent presque pas car on est toujours hors zone linéaire. Pour éviter ceci je limite les variations que peut avoir l'intégrale. Une limite raisonnable est de permettre au terme intégral de mener tout juste à la saturation de $y(t)$ (On le limite alors à y_{max}/a). Des limites plus petites peuvent cependant être prises si elles donnent de meilleurs résultats.
- Comme il y a des frottements solides, on a une non-linéarité qui s'introduit dans la réponse du système à $y(t)$. Pour des petites valeurs de $y(t)$ il se peut qu'il n'y ait aucune réponse. Ceci pose notamment problème en approche finale du point cible car on se retrouve en train d'avancer par accoups quand le terme intégral devient assez fort pour faire avancer (Un peu comme pour une corde de violon excitée par un archet). Pour compenser je rajoute à $y(t)$ un terme constant du signe de la valeur théorique de $y(t)$. Cette approche n'est pas théoriquement bonne quand on est en mouvement car en phase de freinage on est en train d'ajouter un terme qui renforce les frottements solides, elle est cependant validée par les essais pratiques.
- Pour permettre une stabilisation au point d'arrivée, je fais subir au terme intégral une petite décroissance. Ainsi si on est au bon endroit avec une intégrale non nulle, on aura annulation progressive de l'intégrale ce qui permettra l'arrêt complet du système.

Les constantes a , b et c sont codées directement dans le source à l'aide d'une macro. Ceci a le défaut d'obliger à recompiler quand on veut les modifier.

Voici une description de la mémoire utilisée par le PID. Deux banques sont utilisées, 50h pour la voie 1 et 70h pour la voie 2. Les trois derniers octets de la zone temporaire (0Dh à 0Fh) sont utilisées pour les bits de poids faible de $y(t)$.

²

Adresse	Nom	Contenu
0dh à 0fh	fspeed	Partie fractionnaire de $y(t)$.
50h à 51h	ispeed	Partie entière de $y(t)$.
52h à 54h	perr	Erreur en position.
55h à 58h	ierr	Erreur intégrale.
59h à 5ah	derr	Erreur dérivée.
5bh à 5ch	dist	Distance parcourue depuis le dernier calcul.
5dh à 5fh	abspos	Déplacement total depuis l'allumage du SX.

Le code du PID se trouve dans pid.asm

4.5 La génération de trapèze

Le générateur de trapèze est utilisé pour fixer le point cible du PID. A partir d'un ordre de déplacement comprenant une distance à parcourir, une vitesse à laquelle la parcourir et une accélération maximale à ne pas dépasser, il génère une trajectoire qui part de la vitesse actuelle, atteint la vitesse cible aussi vite que possible, puis freine à l'accélération maximale pour s'arrêter au point prévu.

²Les adresses sont données pour la voie 1

Pour faire son travail, le générateur de trapèze à un certain nombre de comportements selon le point de l'espace des phases (position, vitesse) où il se trouve. La partie la plus délicate est le freinage, puisqu'il nécessite normalement le calcul d'une racine carrée. Je m'en tire avec une version approximative qui freine un tout petit peu trop tôt.

Il faut bien noter que le générateur de trapèze ne connaît pas du tout la position des roues. Il sait où il pense être et génère sa trajectoire en fonction de celà. Une amélioration possible, mais difficile pourrait être de prendre en compte la position réelle dans le calcul de la trajectoire. Ceci éviterait des situations dangereuses dans le cas où les roues se bloquent temporairement.

Voici une description de la memoire utilisée par le générateur de trapèze. Deux banques sont utilisées, 90h pour la voie 1 et 0b0h pour la voie 2. La plus part des grandeurs sont codées avec un octet après la virgule ce qui permet de régler très finement la vitesse (je noterai x256 dans ce cas).³

Adresse	Nom	Contenu
90h à 93h	pos	position actuelle x256 (par rapport au point cible).
94h à 96h	speed	vitesse actuelle x256.
97h à 99h	rspeed	vitesse cible x256 (en valeur absolue).
9ah à 9bh	raccel	accélération maximale x256 (en valeur absolue).
9ch	flags	Indicateurs divers. En pratique seul le bit 0 est utilisé pour indiquer qu'en cas de dépassement lors du freinage on ne doit pas chercher à générer une trajectoire retour. L'intérêt est de gérer des légères erreurs de calcul. Le bit 1 devait indiquer une trajectoire sans arrêt mais c'est buggé.
9dh	perrlow	8 bits de poids faible de perr du PID. En effet, à chaque cycle de l'asservissement speed est ajouté à perr. Pour ne pas introduire d'erreurs de calcul il faut retenir les bits de poids faible de cette somme que perr n'a pas.
9eh à 9fh	temp2	Zone temporaire supplémentaire.

Le code du générateur de trapèze se trouve dans trapeze.asm

4.6 Le calcul de position

Un grand avantage de l'asservissement fait par un SX par rapport à l'utilisation de deux LM629, par exemple, c'est qu'il nous reste du temps de calcul pour faire autre chose, par exemple le calcul de position.

Le calcul de position est fait en connaissant à chaque instant un vecteur \vec{u} vers l'avant \vec{u}

$$\vec{u} = \begin{pmatrix} u_x \\ u_k \end{pmatrix}$$

A chaque top de roue codeuse on avance un peu et on tourne un peu (car un top ne correspond qu'à une roue). On va donc ajouter un vecteur vers l'avant à la position à chaque top. Par contre on ne va pas faire tourner \vec{u} à chaque fois. En effet, une rotation de \vec{u} est une opération lourde en calculs et dans laquelle il y a une perte de précision.

³Les adresses sont données pour la voie 1

Comme la rotation correspondant à un top de roue codeuse est très petite, on va pouvoir approcher la matrice de rotation qu'on utilise par :

$$\begin{pmatrix} \cos \theta & -\sin \theta \\ \sin \theta & \cos \theta \end{pmatrix} \approx \begin{pmatrix} 1 & -\theta \\ \theta & 1 \end{pmatrix}$$

Pour simplifier encore les calculs on peut s'arranger pour que θ soit une puissance de 2. Pour avoir une très (trop ?) grande précision sur obelix j'ai pris $\theta = 1/65536$. Pour Asterix un top de roue codeuse correspond à une rotation plus grande que $1/65536$ donc j'ai pris $\theta = 1/256$. Faire tourner \vec{u} revient donc simplement à faire des sommes ou des soustractions.

Avec l'approximation qu'on a faite, il peut y avoir à la longue une augmentation de la norme de \vec{u} qui va réduire la précision du calcul de position. On va donc renormaliser \vec{u} chaque fois qu'il est colinéaire à un des axes du repère.

J'ai dit tout à l'heure qu'on n'allait pas faire tourner \vec{u} à chaque top de roue codeuse. J'avais parlé de temps de calcul et de perte de précision. En fait, on vient de voir que le temps de calcul n'est pas un problème. Les erreurs de précision le sont toujours. Imaginez la perte de précision si on va en ligne droite et qu'on fait tourner \vec{u} à chaque top de roue codeuse. Tous les quelques microns on aura un top à gauche suivi d'un top à droite. Pas de changement de direction, mais deux approximations sur \vec{u} . A la longue on risque d'avoir des problèmes, même en ligne droite. Il ne va donc falloir faire tourner \vec{u} qu'après plusieurs tops dans une même direction. A ce problème s'ajoute le fait qu'avec notre méthode d'approximation, il va falloir passer d'angles correspondant à un top de roue codeuse à des angles qui sont une puissance de 2. Voici comment c'est fait :

Soit α l'angle correspondant à un top de roue codeuse. On a : $\alpha/\theta \approx n/d$. Soit ϵ l'erreur angulaire qu'on a accumulé à un instant donné. Quand on reçoit un top de roue codeuse on a :

$$\Delta(\epsilon d/\theta) = \pm \alpha d/\theta \approx n$$

Si on fait tourner \vec{u} d'un angle θ on a :

$$\Delta(\epsilon d/\theta) = \pm d$$

Ainsi si on retient l'entier $\epsilon d/\theta$ on sait à tout instant quelle erreur on commet, et on peut mettre facilement à jour cette erreur quand on reçoit un top de roue codeuse, ou quand on décide de faire tourner \vec{u} .

Il ne reste plus qu'à décider quand faire tourner \vec{u} , le plus simple c'est de fixer un seuil sur $\epsilon d/\theta$.

Voici l'occupation de la memoire pour faire ce travail. Les banques 0D0h et 0F0h sont utilisées.

Adresse	Nom	Contenu
0d0h à 0d3h	dirx	Coordonnée x de \vec{u}
0d4h à 0d7h	diry	Coordonnée y de \vec{u}
0d8h à 0dah	posxl	Les 3 octets de poids faible de la coordonnée x de la position.
0dbh à 0ddh	posyl	Les 3 octets de poids faible de la coordonnée y de la position.
0deh à 0dfh	postmp	Sert de zone temporaire quand on tourne \vec{u} .
0f0h à 0f2h	angfrac	La valeur de $\epsilon d/\theta$.
0f3h à 0f6h	posxh	Les 4 octets de poids fort de la coordonnée x de la position.
0f7h à 0fah	posyh	Les 4 octets de poids fort de la coordonnée y de la position.
0fbh à 0fdh	ang	La différence entre les nombres de tops à gauche et à droite.

Le code est dans poscalc.asm et est intégralement différent pour Astérix et pour Obelix. La variable SMALLTICKS choisit le mode.

4.7 Le protocole de communication

4.7.1 Le SPI

Le SPI est un protocole de transfert série synchrone mis au point par Motorola. Il permet d'échanger des octets sur trois lignes : une horloge, une pour les données entrantes, une pour les données sortantes. Dans la version que j'utilise (qui correspond à SPCR=0C4h sur un 68HC11 qui communiquerait avec le SX) le SX est maître. C'est à dire que c'est lui qui génère le signal d'horloge.

Il y a toutefois une particularité. En effet, le SX ne décide pas quand transmettre, c'est son entrée \overline{SS} qui en passant au niveau bas déclenche la transmission. Ceci a été fait pour que le 68HC11 qui est lent fixe le rythme de la communication, tout en évitant au SX de devoir suivre avec précision une horloge fixée par le 68HC11 (qui étant équipé d'un circuit SPI n'a pas de problème pour suivre le transfert d'un octet).

A condition de mettre à 1 la variable de compilation MULTISPI, on a le comportement suivant : Quand \overline{SS} est haut, le SX ne fixe pas ses sorties CK et TX, ce qui permet en multipliant les lignes \overline{SS} de communiquer avec plusieurs composants sur les trois mêmes lignes SPI. Il faut prévoir une pulldown sur CK dans ce cas.

La transmission des octets est toujours bidirectionnelle (chaque parti reçoit et envoie un octet). Quand l'horloge est haute, chaque composant doit préparer un bit sur sa sortie. Quand l'horloge est basse chaque composant doit lire la valeur de son bit d'entrée. On transmet ainsi 8 bits consécutifs, du plus fort au plus faible. Puis \overline{SS} doit repasser à l'état haut pour permettre de synchroniser les coupures entre octets et si le SX communique avec un 68HC11, pour permettre à la même ligne \overline{SS} de servir pour le SX et le 68HC11. Je récapitule la transmission d'un octet entre le SX et un 68HC11 :

1. Le 68HC11 met \overline{SS} à l'état bas.
2. Le SX met CK à l'état haut.

3. Les deux composants fixent leurs sorties respectives.
4. Le SX met CK à l'état bas.
5. Les deux composants lisent leurs entrées respectives.
6. On répète les étapes d'écriture et de lecture pour chaque bit de l'octet.
7. Le 68HC11 met \overline{SS} à l'état haut.
8. Le SX met en haute impédance ses sorties pour libérer la voie de communication.

4.7.2 Les commandes

Initialement le SX attend un octet commande qui définit la nature des informations qui vont suivre (toujours avec l'octet de poids fort en premier). Les commandes sont les suivantes :

Commande	Description
00h	Commande nulle. Le prochain octet émis par le SX sera 35h. Le prochain octet attendu sera une commande. Avant d'envoyer une commande au SX il est conseillé d'envoyer des 00h jusqu'à recevoir un 35h pour assurer qu'on est bien synchronisés.
01h	Lire la memoire. Le SX attendra en entrée une suite d'adresses terminées par un zero. A l'échange qui suit chaque adresse, il renverra la valeur de cette adresse, tout en recevant l'adresse suivante.
10h	Suivi de vitesse2, vitesse1, distance2, distance1 (tous en 24 bits). Donne l'ordre de rajouter le déplacement ainsi défini à l'ordre en cours d'exécution.
11h	Comme ci-dessus, sauf qu'on annule le déplacement en cours.
12h	Comme ci-dessus, sauf qu'on remet à zero le PID.
20h	Fixe les accélérations limites du générateur de trapèze. Suivi de accel2 et accel1 sur 16 bits chacun.
30h	Ajoute les coordonnées y puis x fournies sur 32 bits aux 32 bits de poids fort de la position. (Non testé, ne marche peut-être pas)
31h	Fixe la position actuelle avec les coordonnées y puis x fournies sur 32 bits. (Non testé, ne marche peut-être pas)
32h	Fixe ang (24 bits), diry et dirx (32 bits) à partir des valeurs fournies.
33h	Permet de lire la position. Les unités dépendent de paramètres de compilation, afin d'avoir des unités convenables pour Asterix et Obelix en réduisant la bande passante. Donne diry, dirx, posy, posx (tous sur 16 bits), ang (sur 24 bits). (Je conseille de décomposer cette commande en plusieurs commandes et de tout envoyer à chaque fois pour éviter la confusion sur les ordres de grandeur)
fdh	Arrête brutalement le robot. Pas de paramètres.
feh	Arrête lentement le robot (il ralentit grace au générateur de trapèze). Pas de paramètres.
ffh	Fait un reset logiciel du SX d'asservissement.

4.8 Le code du SPI

La partie reception d'un octet se fait dans la routine d'interruption dans le fichier servo4.asm. La partie traitement des données reçues, appelée à partir de la boucle principale, se fait dans sercmd.asm, hors interruption. Les variables utilisées sont dans la banque 30h :

Adresse	Nom	Contenu
30h à 3Bh	iodata	Stoque les données à transmettre et et les données reçues.
3Ch	spidelay	Determine le temps restant avant le prochain changement d'état de CK. vaut 0FFh si le SPI est inactif.
3Dh	cmd	Retient la commande en cours d'execution.
3Eh	iobyte	L'octet en cours de reception et émission. Il est décalé 8 fois lors de l'échange d'un octet.
3Fh	iostat	Determine l'état de la transmission. Les 4 bits de poids faible sont utilisés lors de la transmission d'un octet. Ainsi, le bit de poids faible est l'état de CK et les trois bits suivants comptent le nombre de bits restant à transmettre. Les 4 bits de poids fort indiquent combien d'octets il reste à échanger dans iodata. Les adresses les plus fortes sont échangées en premier. Il y a des valeurs spéciales choisies astucieusement pour les bits de poids fort : 0eh indique qu'on attend une commande, 0dh indique qu'on vient de recevoir une commande, 0fh indique qu'on a échangé tout ce qu'il y avait dans iodata. Il faut regarder le code pour bien comprendre ce qui se passe ici.

Pour éviter de réécrire plusieurs fois le code qui travaille avec iodata dans les différentes commandes, il y a des routines (ClearRange, AddRange, SubRange, SetRange, GetRange) faites sur mesure.

Quand elles opèrent sur iodata, il faut préciser dans FSR l'adresse de l'octet de iodata à partir duquel on travaille, et dans M:W un pointeur vers un mot en memoire programme qui définit dans ses 4 bits de poids fort le nombre d'octets à traiter et dans les 8 bits restants l'adresse de la destination. Le mieux c'est de regarder comment c'est utilisé, pour donner du code très compact. Il faut noter que FSR et M sont fixés à l'avance donc on n'a pas besoin d'y toucher la plus part du temps. FSR est aussi automatiquement incrémenté par ces opérations.

5 Les fichiers source

La version actuelle est décomposée en plusieurs fichiers .asm qui s'incluent les uns les autres :

sasterix.asm et **sobelix.asm** définissent les différentes constantes spécifiques

à chaque robot et incluent servo4.asm

servo4.asm contient la routine d'interruption, le code d'initialisation, la boucle principale et les déclarations de variables. Il inclut les fichiers suivants.

pid.asm contient le filtre PID, ainsi qu'une routine qui prépare les variables de la zone memoire PID et met en application le resultat des calculs PID.

trapeze.asm contient le générateur de trapèze.

sercmd.asm contient les routines de décodages et d'exécution des commandes SPI.

poscalc.asm contient les routines de calcul de position.

macro.inc contient quelques macros.

6 Les modes de fonctionnement

Un certain nombre de variables de compilation permettent de configurer le programme du SX pour différents environnements :

Nom	Description
SLOWQUAD	Indique qu'on aura \llcorner peu \lrcorner de tops de roue codeuse, ce qui permet de ralentir le polling.
INVPWM	Permet d'inverser les sorties PWM1 et PWM2 si le hacheur est inverseur.
INVCH1	Permet d'inverser le signal reçu des roues codeuses de la voie 1, ainsi que la direction de rotation en sortie, suite à une erreur sur Asterix.
INVCH2	Permet d'inverser la direction de rotation en sortie, n'est pas utilisé pour le moment. Il pourrait être intéressant de revoir ces variables en distinguant l'inversion de l'entrée et l'inversion de la sortie pour pouvoir faire face à tous les cas de figure.
MULTANG	Normalement plus utilisé. Correspondait à une stratégie de calcul de position où les rotation du vecteur \llcorner vers l'avant \lrcorner pouvaient avoir un angle plus petit que le top de roue codeuse. Il faut décommenter du source pour s'en servir.
MULTISPI	Précise que les lignes SPI seront <u>partagées</u> et qu'il faut donc libérer les sorties quand \overline{SS} est haut. Si on est dans ce cas il faut prévoir une pulldown sur CK.
DOEXTPID	Indique que le PID va être fait par un composant externe. Utile pour expérimenter les constantes. Voir le source pour les détails du protocole (s'il marche toujours car c'est ancien).
SMALLTICKS	Indique quels bits des coordonnées choisir pour la commande 33h quand on lit la position.

Il y a aussi des paramètres qu'il faut fixer en fonction des caractéristiques dynamiques et géométriques du robot. Les variables de plus de 16 bits ont du être découpées car sasm ne les gère pas directement.

Nom	Description
IERRLIM	ierr est limité pour ne pas dépasser $65536 \times IERRLIM$
PCONST	Constante du PID pour la partie proportionnelle. Il y a en fait 3 variables PCONSTH pour le bit de poids fort, PCONST pour les bits 16 à 31 et PCONSTL pour les bits 0 à 15.
ICONST	Constante du PID pour la partie intégrale. Il y a en fait 3 variables ICONSTH pour le bit de poids fort, ICONST pour les bits 16 à 31 et ICONSTL pour les bits 0 à 15.
DCONST	Constante du PID pour la partie dérivée. Il y a en fait 3 variables DCONSTH pour le bit de poids fort, DCONST pour les bits 16 à 31 et DCONSTL pour les bits 0 à 15.
FRICTION	Terme qu'on rajoute au resultat du PID pour compenser les frottements solides.
ANGFRACLIM	Limite à partir de laquelle on fait tourner le vecteur jivers l'avantj. Son comportement dépend de la valeur de SMALLTICKS.
ANGFRACNUM	Voir le calcul de position, c'est le numérateur de α/θ Il est en fait en deux parties ANGFRACNUMH et ANGFRACNUML
ANGFRACDENOM	Voir le calcul de position, c'est le numérateur de α/θ Il est en fait en deux parties ANGFRACDENOMH et ANGFRACDENOML

7 Améliorations possibles

- Changer de protocole de communication. Notamment, il pourrait être intéressant de passer à quelquechose qui serait compatible I_2C .
- L'idée de base qui consiste à faire deux LM629 n'est pas du tout adaptée à l'asservissement d'un robot. Il pourrait être intéressant de passer à un DSP, et de faire du calcul ou on réfléchit plutôt en moment cinétique et en quantité de mouvement global du robot. Une version simplifiée de ce vaste chamboulement pourrait être d'asservir la somme et la différence des voies pour faire ressortir le comportement dynamique linéaire et rotatoire du robot.
- Rendre la communication plus robuste en ajoutant des timeout, pour que le SX génère des faux CK s'il voit que \overline{SS} ne remonte pas en fin de transfert.

8 Glossaire

PID (Proportional Integral Derivative) Filtre linéaire de la forme :

$$y(t) = a \int x(t) dt + b x(t) + c \frac{dx}{dt}(t)$$

PWM (Pulse Width Modulation) Il est plus facile de produire et d'amplifier des signaux digitaux que des signaux analogiques. Pour appliquer une tension donnée aux bornes d'un moteur, on pourra se contenter d'envoyer un signal dont la moyenne temporelle est la tension qui nous intéresse. On pourra donc se contenter de générer des créneaux dont la longueur est calculée pour que la valeur moyenne de la tension aux bornes du moteur soit celle recherchée.

Roue codeuse Disque dont la périphérie est percée de trous équirépartis. En regardant en deux points de la périphérie écartés d'un demi trou, on arrive à suivre le mouvement de la roue, en regardant le défilement des trous. Ouvrir une souris à boule pour voir.

SPI (Serial Peripheral Interface) Protocole de communication série synchrone (il y a un signal horloge) mis au point par Motorola.