ENS Cachan
Département EEA

Guillaume HERAULT
Loïc SIMON
Vincent THOMAS
Julien VILLEMEJANE

T.E.R.

GESTION DE L'ÉNERGIE EMBARQUÉE SUR UN ROBOT MOBILE

Encadrants

G. Pradel

P. Varoqui

Année 2005

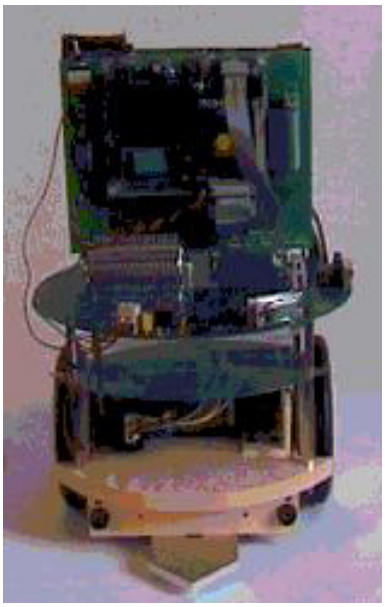
T.E.R.
Gestion de l'énergie embarquée sur un robot mobile

Guillaume HERAULT – Loïc SIMON
Vincent THOMAS – Julien VILLEMEJANE

12 mai 2005

Introduction

Dans le cadre de ce TER, nous allons travailler sur un robot mobile et autonome, destiné à être utilisé auprès d'enfants autistes, afin d'étudier leurs comportements et leurs réactions face aux différentes actions du robot.



Actuellement, ce robot est équipé de deux batteries au plomb : l'une pour la partie "moteur" et l'autre pour la partie "électronique". Ces deux batteries ont respectivement 45 min et 20 min d'autonomie.

Le problème se situe donc sur la partie "électronique" qui a une autonomie trop faible pour l'utilisation souhaitée. En effet, ce robot étant destiné à "jouer" avec des enfants autistes, une durée de jeu de moins d'une heure n'est pas convenable.

Cependant, l'autonomie d'une batterie ne peut être modifiée. La seule possibilité pour améliorer cette caractéristique sur un robot est de modifier la capacité de la batterie. Il est donc envisagé de remplacer la batterie de la partie "électronique" au plomb par une batterie au lithium-polymère, d'une plus grande capacité.

Avec ce changement, on espère que le robot aura alors une autonomie supérieure à 45 min (voire 1h).

Cette unique solution ne ferait que déplacer le problème. Cependant, à terme, il est envisagé un mode de fonctionnement à plusieurs robots. Lorsqu'un robot n'a plus d'énergie, il enverra un signal à ceux aux alentours (non-utilisés et rechargés) pour que l'un d'entre eux vienne prendre le relais, afin que ce premier puisse se recharger. Et ainsi de suite.

Pour cela, il est donc nécessaire de connaître en permanence la quantité d'énergie restante dans chacun des robots, et lorsque cette énergie devient insuffisante, que le robot puisse trouver seul une balise pour se recharger dans la pièce où il se trouve.

Notre travail, à travers ce TER, sera de réaliser un circuit capable de calculer l'énergie restante ou dépensée par le robot, de lancer une procédure de recherche d'une balise de rechargement, une fois l'énergie emmagasinée trop faible pour faire évoluer le robot et de se diriger vers cette balise.

Ce rapport est divisé en 3 grandes parties :
- une partie "théorique", s'appuyant sur le travail de recherche effectué ;
- une partie "technique", s'appuyant sur la réalisation du module de "rechargement" durant les deux semaines dédiées à cela ;
- une partie regroupant les essais réalisés sur nos maquettes et les performances de notre système.

Ce rapport est suivi des références bibliographiques auxquelles nous avons fait appel et d'annexes regroupant des informations techniques, l'intégralité des codes écrits durant ce TER, les schémas complets des systèmes réalisés.

Table des matières

I Généralités **5**

1 Cahier des Charges **7**

2 Recherche de solutions **9**

2.1 Rappel sur les batteries 10

2.1.1 Batteries au plomb 10

2.1.2 Batteries au lithium-polymère 12

2.1.3 Comparatif entre les deux types de batteries 14

2.2 Contrôle de l'énergie 15

2.2.1 Contrôle du niveau de tension 15

2.2.2 Contrôle de l'énergie dépensée 19

2.3 Détection de la balise de rechargement 23

2.3.1 Systèmes envisageables 23

2.3.2 Les contraintes de notre projet et le choix du système 24

2.4 Choix possible 32

3 Choix de conception **33**

3.1 Système de contrôle de l'énergie 33

3.2 Système de détection de la balise de rechargement 35

4 Structures réalisées **37**

4.1 Système de contrôle de l'énergie 37

4.2 Système de détection de la balise de rechargement 39

II Etude des différentes fonctions **40**

5 Système de contrôle de l'énergie **42**

5.1 Acquisition et conversion de U et de I 44

5.2 Traitement des données - Signaux de sortie 47
5.3 Schéma électrique complet et typon 50

6 Système de détection de la balise de rechargement 52
6.1 Système d'émission de la balise de rechargement 52
6.2 Système de réception de la balise de rechargement 53
6.3 Programme de détection de la balise 61

III Essais et Performances 70

7 Système de contrôle de l'énergie 72
7.1 Premier test 73
7.2 Réalisation finale 74
7.3 Performances 76

8 Système de détection de la balise de rechargement 78
8.1 Les essais avec la maquette 78
8.2 Les essais avec le robot 81
8.3 Performances 83

Conclusion 84

Bibliographie 88

Annexes 92

Annexe A - Prise en main des PSoC 92
Structure globale d'un PSoC 93
Fonctions principales des PSoC 95
Prise en main et développement 96

Annexe B - Listing du programme PSoC 98

Annexe C - Fonctions utilisées sur les PSoC et leur implantation 102

Annexe D - Listing complet des programmes du robot 105

Annexe E - Composants utilisés : Doc. Technique 141

Première partie

Généralités

Dans cette première partie, nous vous présenterons le projet dans sa globalité et de façon relativement “théorique”. La partie réalisation, donc plus “technique”, est renvoyée à la partie II.

Cette première partie est divisée en 4 chapitres.

Dans le premier, vous trouverez le cahier des charges que doit remplir le projet.

Le second chapitre retrace la partie recherche de solutions satisfaisant le cahier des charges.

Puis nous avons dû faire des choix avant de commencer la conception. Ces choix sont expliqués dans un troisième chapitre.

Enfin, vous trouverez dans le dernier chapitre de cette partie la structure qui a été réalisée au cours des deux semaines dédiées à la réalisation de ce projet.

Voici donc pour commencer, le cahier des charges que doit satisfaire notre module de “rechargement”.

Chapitre 1

Cahier des Charges

Comme on l'a vu dans l'introduction, le robot sur lequel nous allons travaillé est destiné à être utilisé auprès d'enfants autistes.

L'autonomie des batteries actuellement utilisées pour la partie "électronique" n'est pas suffisante pour l'utilisation souhaitée. Pour ne pas gêner l'enfant dans son jeu, une autonomie supérieure à 45 min serait plus convenable.

Dans un premier temps, la batterie au plomb d'une capacité de 0,8 Ah sous 12 V de la partie "électronique" est remplacée par une batterie au Lithium-Polymère d'une capacité de 3,2 Ah sous 11,1 V. L'autonomie est alors améliorée.

Cependant, pour l'utilisation multi-robots souhaitée à terme, il est nécessaire de connaître en permanence la quantité d'énergie restante dans les batteries des robots et d'envoyer une procédure de recherche lorsque cette énergie descend en dessous d'un certain seuil.

Nous devons donc réaliser un circuit capable de calculer l'énergie stockée et restante dans les batteries, dans un premier temps uniquement celle au Lithium-Polymère, de trouver une balise de rechargement dans une pièce sans obstacle et de se diriger vers elle.

Nous ne nous occuperons pas du positionnement du robot une fois devant la balise, ni de la procédure de rechargement. Ces étapes seront effectuées manuellement, une fois le robot au pied de la balise.

Notre circuit doit satisfaire les conditions suivantes :
– ne pas consommer trop d'énergie ;
– être adaptable sur les différents robots utilisés ;
– être adaptable sur les deux types de batteries que nous utiliserons ;
– être utilisable dans un milieu dégagé ;
– être alimenté par une batterie 12 V.

Notre circuit doit pouvoir :
– calculer de manière fiable l'énergie emmagasinée dans les batteries ;
– lancer une procédure de recherche de balise de rechargement, à l'aide de fonctions déjà implantées sur le robot ou à développer, lorsque l'énergie stockée est trop faible pour continuer son "jeu" ;
– se diriger vers la balise repérée à une distance maximum de 3m ;
– être assez rapide pour arriver jusqu'à la balise (1mn environ) ;

Chapitre 2

Recherche de solutions

Afin de remplir le cahier des charges vu précédemment, nous avons procédé à quelques recherches sur les différentes techniques possibles pour réaliser ce module de “rechargement”.

Afin de rendre le travail plus simple, nous avons divisés le projet en deux grandes parties :

- une partie axée sur le contrôle de l'énergie dans les batteries ;
- une autre partie basée sur la recherche des balises de rechargement.

Notre travail de recherche se divise en 3 rubriques. La première fait un bref rappel sur les deux types de batterie que nous allons utiliser : leurs caractéristiques principales, leurs modes de rechargement. La seconde s'intéressera aux méthodes de contrôle de la tension et de l'énergie sur les batteries. Et enfin la dernière présentera les différents moyens de repérage dans l'espace et leurs avantages et inconvénients.

2.1 Rappel sur les batteries

Dans les batteries, le conducteur ionique ou électrolyte, qui sépare les électrodes, est un élément clé. D'une part, son état, liquide ou solide, affecte la sûreté du système et d'autre part, sa conductivité détermine la gamme de température de fonctionnement.

2.1.1 Batteries au plomb

Inventé en 1859 par Gaston Planté, l'accumulateur au plomb est sans doute le plus répandu, surtout dans les applications où sont demandées :

– une forte intensité
– une grande capacité de stockage.

a - Composition et caractéristiques principales

Les accumulateurs au plomb sont souvent vendus sous forme de batterie de plusieurs éléments de 2V chacun. Un élément chargé présente une tension de 2,1V, et on peut le décharger jusqu'à une tension de 1,9V sans risque, le minimum absolu étant à 1,65V. La version la plus courante est la batterie à 6 éléments, donc dite batterie 12V. La tension de ce type de batterie peut donc varier de 11V à 12,6V. La tension d'une batterie au plomb 12V ne doit jamais descendre en dessous du seuil minimum de 10V (extrême limite). Les batteries utilisées par le robot actuellement ont une capacité de 0,8 Ah, sous 12 V. Elles peuvent normalement fournir 0,8 A pendant une heure. Pour la partie “entraînement”, ces batteries peuvent tenir un peu plus de 45 min. Pour la partie “électronique”, ces batteries ne tiennent qu'environ 20 min.

b - Charge et décharge

La charge d'une batterie au plomb se fait sous tension constante de l'ordre de 2,3-2,4 V par élément, soit 14 V pour une batterie de 12 V. La limitation du courant durant la charge est nécessaire, car la résistance interne d'une telle

batterie est faible. La valeur normale de limitation de ce courant doit varier entre 1/4 et 1/5 de la capacité nominale pour une durée de charge de 20h.

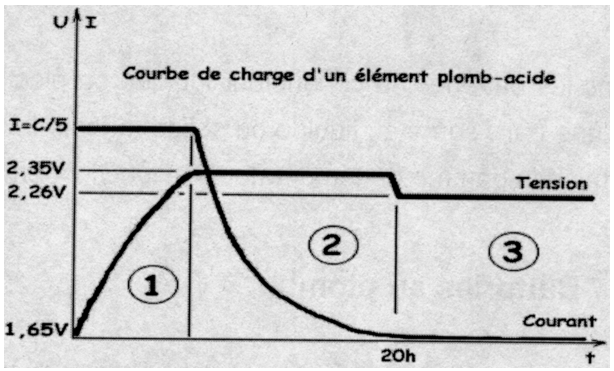


Fig. 2.1 – Courbe de charge d'une batterie au plomb

Une fois chargé, chaque élément présente une tension de 2,1 V à ses bornes, soit 12,6 V pour une batterie de 12 V. Pour connaître l'état de charge d'une batterie, on peut aussi mesurer la densité de l'électrolyte à l'aide d'un pèse-acide. En dessous de 23 degrés Baumé (d=1,19) la batterie est déchargée. Au dessus de 30 degrés Baumé (d=1,26) on considère qu'elle est chargée. Mais il n'existe pas de capteur pour ce genre d'application qui soit facile à mettre en oeuvre sans risque.

Durant l'utilisation de la batterie, donc de sa décharge, on peut noter que la tension à ses bornes diminue (voir courbe suivante).

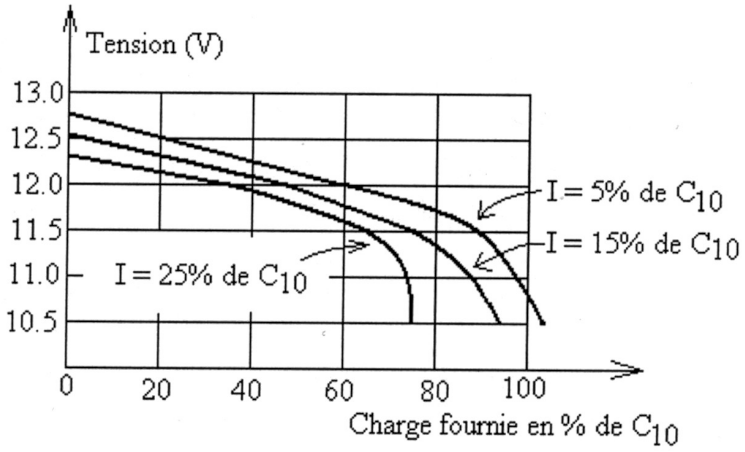


Fig. 2.2 – Courbe de décharge d'une batterie au plomb

Voyons maintenant les caractéristiques d'une batterie au Lithium-Polymère.

2.1.2 Batteries au lithium-polymère

Ce type de batterie, encore peu répandu il y a quelques années, est de plus en plus présent dans les systèmes portables actuels (téléphone, ordinateur portable...). Ces accumulateurs présentent notamment une plus grande autonomie et une plus longue durée de vie.

a - Composition et caractéristiques principales

Les batteries Lithium Metal Polymer se composent, comme les batteries au plomb, de plusieurs cellules faisant chacune 3,7 V. Une fois chargée, une cellule peut atteindre 4,2 V (à vide) et ne doit pas descendre lors de sa décharge sous le seuil de 2,8 V. Pour une batterie de 3 éléments, on peut ainsi obtenir une tension, une fois chargée de l'ordre de 12,5 V.
La batterie qui sera utilisée sur le robot pour la partie électronique a une capacité de 3,2 Ah sous 11,1 V. On peut donc espérer une durée de fonctionnement de l'ordre d'une heure (voir un peu plus).

b - Charge et décharge

Durant la charge de la batterie, le courant et la tension doivent être contrôlés en permanence, car les produits mis en jeu sont facilement inflammables. La charge d'une batterie au Lithium se fait à tension constante. Pour une cellule au polymère, une tension de 4,1 V convient. Il faut pouvoir contrôler le courant dans la batterie, afin de pouvoir arrêter la charge lorsque le courant de charge atteint les 0,03 C (C étant la capacité de la batterie).

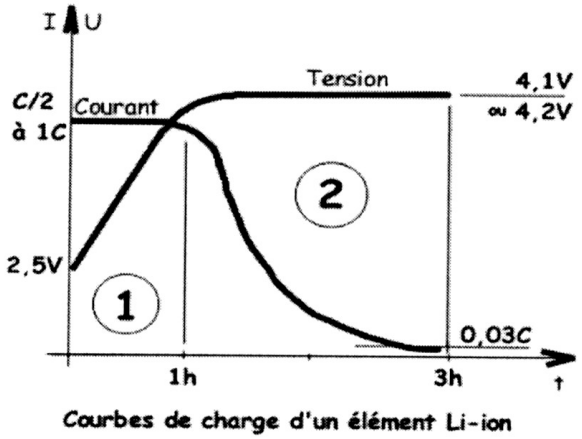


Fig. 2.3 – Courbe de charge d'une batterie au lithium-polymère

Une cellule chargée présente à ses bornes une tension d'environ 4,1 V, soit 12,3 V pour une batterie de 11,1 V. Durant l'utilisation de la batterie, donc de sa décharge, on peut noter que la tension à ses bornes diminue (voir courbe suivante).

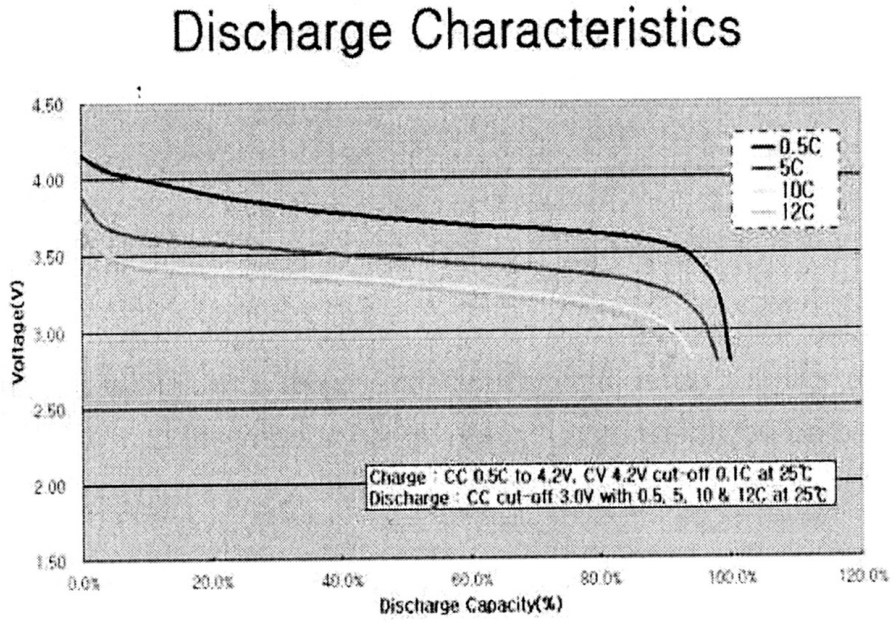


Fig. 2.4 – Courbe de décharge d'une batterie au lithium-polymère

Pour garantir un niveau correct de charge et une durée de vie de la batterie suffisamment longue, il ne faut pas que la tension passe au dessous du seuil de 2,8 V.

Comparons à présent les deux types de batteries présentés ci-dessus.

2.1.3 Comparatif entre les deux types de batteries

Pour une cellule (pour une batterie 12V)

	Plomb	Lithium Polymere
Tension nominale	2 V (12 V)	3,7 V (11,1 V)
Tension maximale	2,1 V (12,6 V)	4,2 V (12,6 V)
Tension critique	1,65 V (9,9 V)	2,8 V (8,4 V)
Contrôle possible de la charge	- tension - énergie consommée - pèse-acide	- tension - énergie consommée

Fig. 2.5 – Comparatif Pb / Li-Po

La capacité de la batterie au polymère étant de l'ordre de 3,2 Ah, on peut espérer atteindre un temps d'utilisation d'un peu plus d'une heure pour la partie électronique, par rapport à celle déjà utilisée actuellement au plomb qui a une charge de 0,8 Ah et avec laquelle on obtient un temps d'utilisation d'environ 20 min.

Pour mesurer l'état de charge des batteries, que ce soit une batterie au plomb ou au lithium polymère, on pourra utiliser deux types de circuit :

- soit un circuit relevant la tension de la batterie au cours de son utilisation, et qui, une fois un seuil critique à définir pour chaque type de batterie, envoie une alerte au microcontrôleur afin qu'il passe dans un mode de veille et qu'il aille faire recharger le robot ;
- soit un circuit relevant l'énergie consommée à chaque instant par le système et le comparant à un seuil pour lequel le robot devra aller se recharger.

Ce sont ces deux types de circuit qui seront étudiés dans la partie suivante.

2.2 Contrôle de l'énergie

Afin de réaliser un circuit qui permet de contrôler l'état de charge d'une batterie, on pourra s'appuyer sur deux techniques différentes. Dans la partie précédente, nous avons vu que, lors de l'utilisation de la batterie la tension à ses bornes était en constante diminution. On pourra donc utiliser cette caractéristique pour pouvoir savoir à quel niveau de charge en est la batterie, et définir alors un seuil de tension (ou plutôt deux seuils) pour lequel il sera nécessaire d'aller recharger la batterie. Une autre méthode pourra consister à évaluer l'énergie consommée par le système et définir là encore un seuil d'énergie pour lequel le microcontrôleur du système décidera d'aller recharger le robot.

Dans les deux cas, le circuit devra fournir au robot une information sur l'état de charge de ses batteries, soit en lui précisant exactement quelle quantité d'énergie il reste ou a été dépensée, soit en lui envoyant deux signaux, que l'on appellera seuils : un premier seuil permettant au robot de savoir qu'il doit cesser ce qu'il est en train de faire et commencer la procédure de recherche de la base de rechargement ; un deuxième seuil permettant de lui dire que ces batteries sont au plus bas et qu'il est vital pour lui de se recharger.

2.2.1 Contrôle du niveau de tension

Le but d'un tel circuit est de relever la tension aux bornes de la batterie, et de la comparer à un seuil pour lequel le robot devra aller se recharger. Ce circuit ne doit pas consommer trop de courant et doit pouvoir être autonome ; c'est à dire qu'il doit fournir l'information "batterie faible" au microcontrôleur sans que celui-ci ait de calculs à faire.

a - Circuits intégrés spécialisés

On trouve dans le commerce des circuits utilisant une série de comparateurs et permettant d'indiquer la tension aux bornes de son entrée sur un vu-mètre à leds. La consommation des leds étant importantes pour notre application, on pourra

simplement utiliser l'information disponible en sortie d'un tel circuit et la rendre compatible avec l'utilisation souhaitée. On peut utiliser par exemple les circuits intégrés LM3914 et LM3916 de National Semiconductor, dont voici un exemple d'application qui ressemble à celui que nous recherchons : un testeur de batterie à leds qui peut fonctionner avec des batteries de 12 V et des batteries de 24 V. Dans notre cas, certains composants pourront être supprimés. Le timer permet "d'échantillonner" l'acquisition de la tension aux bornes de la batterie. Avec un timer ICM7555, la consommation de ce circuit est d'environ 0,5 mAh.

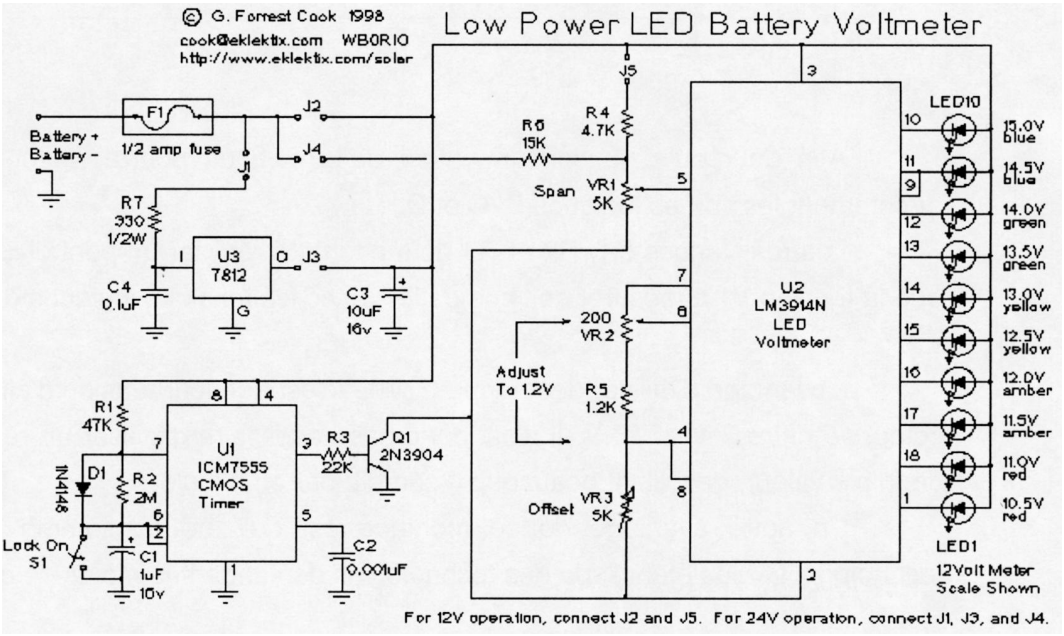


Fig. 2.6 – Montage testeur de tension à base du LM3914

Le timer pourra être supprimé, et alors le LM3914 vérifiera en permanence la tension aux bornes de la batterie. On pourra alors s'intéresser à la différence de consommation entre les deux montages.

b - Autres montages

On trouve aussi d'autres testeurs de batteries réalisés avec des circuits logiques ou d'autres montages à base d'AOP.

Voici le schéma de l'un de ces testeurs, réalisé avec des circuits logiques :

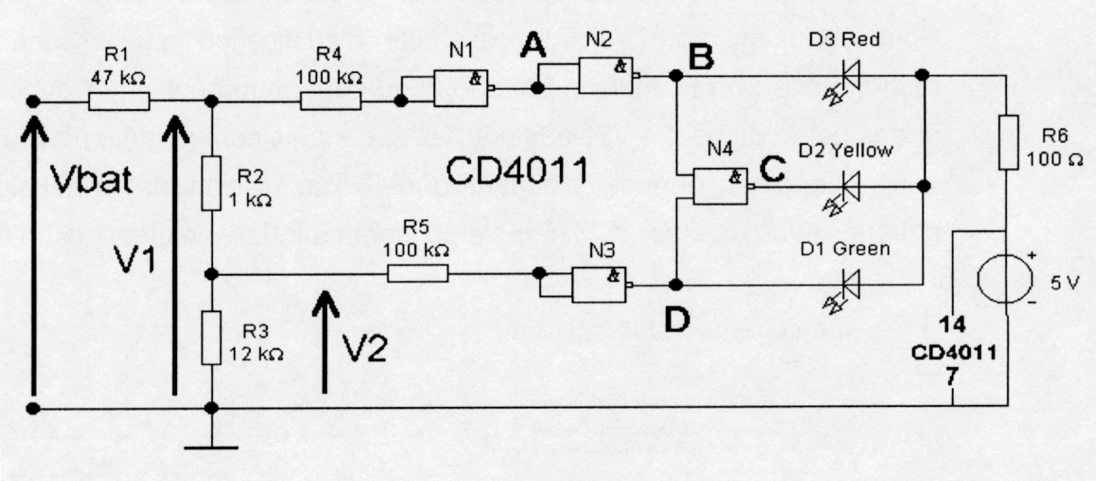


Fig. 2.7 – Montage testeur de tension basé sur un circuit logique CMOS

Afin de réduire la consommation du circuit, on pourra retirer les leds et utiliser directement les sorties logiques B, C et D. Les résistances R1, R2 et R3 permettent de réaliser un pont diviseur de tension afin que la tension V1 à mesurer soit compatible avec les tensions logiques des circuits CMOS.

L'avantage d'utiliser des portes CMOS, c'est que leur tension d'alimentation peut être comprise entre 3 V et 18 V. Il n'est donc pas nécessaire d'utiliser un régulateur de tension de faible valeur, gaspillant beaucoup d'énergie par effet joule.

Un autre avantage de ce montage est qu'il ne consomme quasiment rien, la technologie CMOS étant l'une des technologies demandant le moins de courant.

Un deuxième montage du même type mais utilisant des amplificateurs opérationnels (comme comparateurs) peut réaliser la même fonction, et envoyer alors deux signaux logiques au microcontrôleur : un pour le premier seuil, et un autre pour le second seuil, définis dans l'introduction de cette partie.

On retrouve en entrée de ce montage, un pont diviseur de tension (R1, R2)

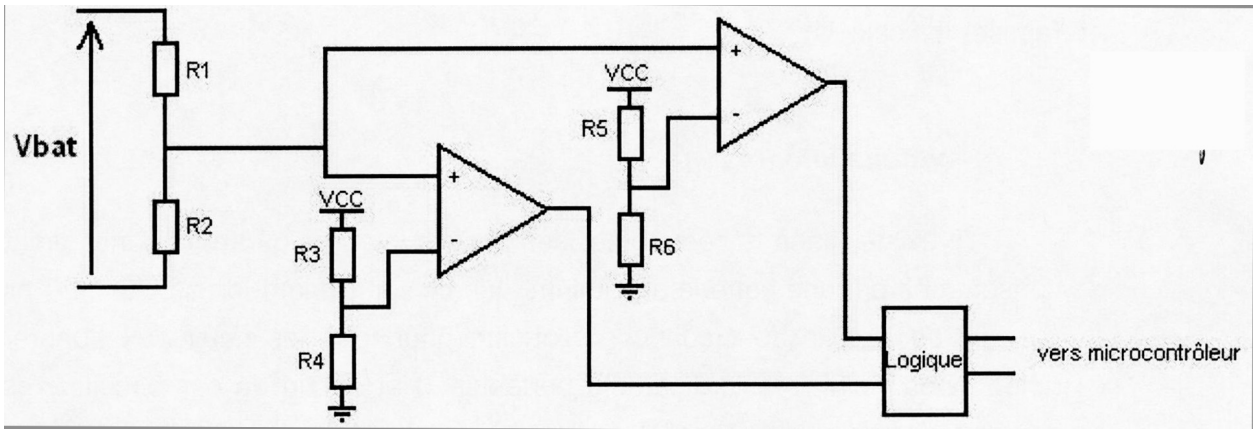


Fig. 2.8 – Montage testeur de tension basé sur des comparateurs

afin d'adapter la tension à mesurer à celle que pourront comparer les deux AOP. Ces deux AOP compareront la tension V_{bat} (à un coefficient près) à deux tensions de référence qu'il faudra fixer, ces deux tensions de référence s'appuyant sur une tension V_{CC} qui devra être fixe. Cette tension V_{CC} pourra être obtenue à l'aide d'une diode Zener ou d'un régulateur de tension du type 78LXX.

La partie logique permet de rendre compatible les signaux obtenus en sortie des AOP avec les entrées logiques du microcontrôleur. On pourra notamment utiliser des AOP à collecteur ouvert et ainsi réutiliser la tension V_{CC} en rajoutant des résistances de tirage en sortie de ces AOP.

En prenant des AOP qui consomment peu et en réalisant des ponts diviseur de tension ayant des valeurs suffisamment élevées, ce montage n'est pas très demandeur en énergie, ce qui est l'un des points à vérifier pour pouvoir rentrer dans notre cahier des charges.

2.2.2 Contrôle de l'énergie dépensée

Une deuxième solution consiste à calculer l'énergie dépensée ou consommée par le système à partir des mesures de la tension et du courant en sortie de la batterie, et de la comparer à l'énergie que peut fournir la batterie. Le dispositif qui sera utilisé devra bien sûr consommer peu d'énergie afin de ne pas fausser les calculs.

a - Circuits intégrés

Il existe, dans le commerce, des circuits intégrés gérant la consommation d'énergie consommée par une batterie au Lithium (que ce soit Lithium-Ion ou Lithium-Polymer). Ces circuits sont conçus à l'origine pour tous les nouveaux appareils portables (téléphones cellulaires, ordinateurs portables...) et aucun de ces circuits n'est prévu pour des batteries de plus de 2 cellules, c'est à dire pour des tensions allant de 3 à 10 V. Or, les batteries que nous utilisons ont des tensions de l'ordre de 12 V. Ce type de circuit, comme le DS2751 de Dallas Semiconductor ou encore le MCP79841 de Microchip, ne pourra donc pas être utilisé dans notre application.

Il existe aussi une interface normalisée de gestion de l'énergie embarquée, appelée ACPI (Advanced Configuration and Power Interface), utilisée dans les ordinateurs portables afin que le système opérationnel connaisse l'état de charge des batteries, pour pouvoir ou non limiter l'utilisation de ses ressources. Les batteries embarquées dans les portables possèdent alors un circuit dit "intelligent" qui permet d'informer la partie logicielle, via un bus spécifique, de l'énergie consommée et restante. De là, la partie logicielle peut calculer le temps restant avant la mise en "état de veille" des processus en cours. Cependant, l'accès à ces circuits spécialisés est restreint et réservé aux fabriquants de batteries. Il n'est donc pas possible d'obtenir les schémas de principe sans accord.

L'utilisation de tels circuits ne sera donc pas possible dans notre système.

b - Mesure de U et de I

Pour relever le courant on utilise une résistance de faible valeur (environ 0,1Ω) afin de limiter les pertes dans cette résistance. Ensuite on fait le produit de cette tension image du courant et de la tension aux bornes de la batterie à l'aide d'un multiplieur.

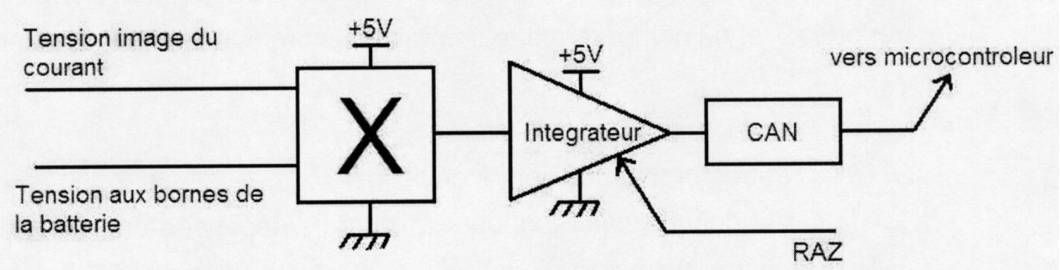


Fig. 2.9 – Montage pour contrôler l'énergie

Un premier problème se pose quant aux valeurs de tensions d'alimentation du multiplieur :

- les tensions d'entrée et de sortie ne doivent pas dépasser les tensions d'alimentation ;
- il faut vérifier que le multiplieur puisse fonctionner sous 0/+5V et qu'il ne consomme pas trop de courant pour fonctionner.

Le premier point se règle en utilisant un simple pont diviseur de tension.

Pour le second, la famille des MAX232 de chez Maxim (voir **Annexe E**) permet à partir d'une tension simple de réaliser une tension symétrique de valeur +/- 2xVCC (-/+ 1,5V). Un tel circuit (ou le couplage de plusieurs comme celui-ci, afin d'obtenir la puissance souhaitée) pourra être utilisé si le multiplieur ne peut fonctionner en tension simple. Puis on intègre le résultat sur une période T = 1 min pour avoir l'énergie consommée pendant une minute, par exemple. On convertit ce résultat à l'aide d'un CAN, dont l'ordre de conversion sera envoyé par le microcontrôleur tous les T, ce dernier se chargeant aussi de récupérer la

donnée convertie. Il faut préciser que cette donnée est l'image de l'énergie à un coefficient multiplicatif près dû aux méthodes d'acquisition.

Le travail du microcontrôleur sera de sommer les énergies calculées toutes les périodes T et de comparer cette somme à l'énergie totale emmagasinée lors de la charge.

D'autres problèmes sont à envisager :

- la donnée correspondant à l'énergie dépensée étant stockée dans la RAM de la carte microcontrôleur, celle-ci ne doit pas être arrêtée car cela provoquerait une perte des données dans cette RAM. Une solution est l'utilisation d'une RAM sur batterie, comme il en existe une sur la carte microcontrôleur utilisée ;
- la connaissance exacte de l'énergie disponible/emmagasinée lors de la charge est indispensable au bon fonctionnement. Si cette donnée n'est pas bonne, et en cas par exemple de rechargement incomplet de la batterie, le robot n'ira pas se recharger même si ces batteries sont quasiment à plat. Pour éviter cela, on pourra alors utiliser le même type de circuit de contrôle dans la base de rechargement afin de connaître la charge emmagasinée dans la batterie lors de son rechargement, ce qui implique une liaison entre la station de rechargement et le microcontrôleur embarqué.

Une deuxième méthode s'appuyant là encore sur la mesure de I et de U serait d'envoyer directement au microcontrôleur les données sur la tension et le courant. Le courant ayant une valeur moyenne quasi constante et la tension ne diminuant pas brusquement, cette acquisition ne pourrait se faire que toutes les minutes par exemple. Les informations sur U et I seraient alors acquises via deux CAN (ou un CAN dont l'entrée pourrait être multiplexée).

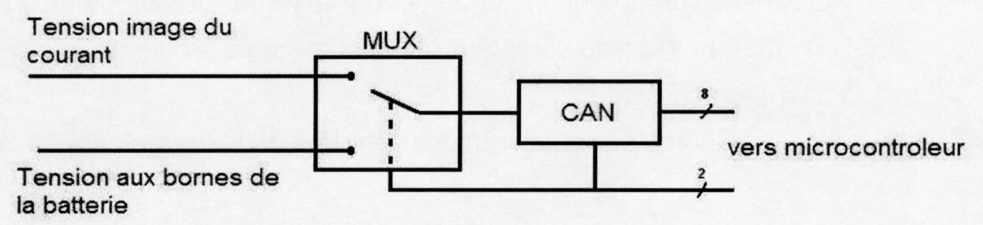


Fig. 2.10 – Montage pour contrôler l'énergie

Le microcontrôleur devrait alors traiter ces deux données en les multipliant et en ajoutant ce résultat aux précédents et alors le comparer à l'énergie totale emmagasinée. Cette méthode requiert donc de connaître l'énergie embarquée par le robot, c'est à dire, soit de la mesurer à chaque rechargement (par une méthode similaire), soit de la définir comme une constante et de prendre une marge de sécurité au niveau des différents seuils.

Pour ne pas surcharger le microcontrôleur principal, on pourra allouer à ce circuit un CPLD ou FPGA autonome, qui enverra uniquement à la carte microcontrôleur l'ordre d'aller se recharger.

Après vous avoir présenté des méthodes de contrôle de la tension et de l'énergie d'une batterie, voyons maintenant la deuxième partie de notre TER qui consiste, une fois l'ordre de rechargement envoyé par ce premier module de contrôle d'énergie, à engager une procédure de recherche d'une balise de rechargement.

Voici donc certaines solutions trouvées.

2.3 Détection de la balise de rechargement

Nous devons par ailleurs trouver un système permettant au robot de détecter la station de recharge. Ceci comprend, en premier lieu la détection de la direction de cette station et éventuellement la mesure de la distance qui la sépare du robot. Ce dernier objectif n'est pas indispensable, car il suffit que le robot soit capable de détecter l'arrivée à la station ce qui peut être réalisé par un simple système mécanique (capteur tactile).

2.3.1 Systèmes envisageables

Dans le domaine de la détection de cible ou d'obstacles, il existe essentiellement deux types de systèmes (en tout cas si on veut s'en tirer à coût raisonnable) :

- ceux basés sur des capteurs à ultrasons ;
- ceux reposants sur des capteurs à infrarouges.

Donnons à présent un rapide descriptif de ces deux types de système.

a - Les ultrasons

Avantages :

- grande portée pour la mesure de distance (mais on néglige ce point pour l'instant) ;
- peu cher (environ 30 €) mais plus que ceux à infrarouges.

Inconvénients :

- directivité faible (peut aussi s'avérer utile) ;
- grande consommation ;
- sensibles aux interférences.

b - Les infrarouges

Avantages :
– très directifs en emission (20 °) ;
– peu chers (environ 15 €) ;
– faible consommation ;
– possibilité de modulation pour insensibiliser aux interférences ;
– très peu encombrant.

Inconvénients :
– peu directifs en reception (90 °) ;
– plusieurs émetteurs pour couvrir un grand domaine ;
– nécessite un environnement sans obstacle ;
– espace de dimension raisonnable.

2.3.2 Les contraintes de notre projet et le choix du système

a - Cahier des charge

Une première contrainte importante que nous devons prendre en compte dans notre cas est d'éviter d'augmenter la consommation d'énergie du robot. Ceci implique alors d'utiliser un capteur passif (i.e. uniquement récepteur) au niveau du robot. On placera alors l'émetteur sur une balise placée sur la station de recharge. Cependant cette première contrainte ne pénalise pas plus l'un que l'autre des systèmes envisagés. Par contre, le manque de directivité du capteur à ultrasons le rend moins intéressant que celui à infrarouge. En effet la directivité est essentielle pour notre application : il faut que le robot connaisse avec précision la direction de la balise pour y parvenir sans trop tâtonner. Par ailleurs, les capteurs à infrarouge sont généralement moins chers que ceux à ultrasons et des connexions à un microcontrôleur sont généralement prévue de base.

Tout nous pousse donc à utiliser un capteur à infrarouge il nous reste donc à exposer le principe de détection de la balise.

b - Principe du système de détection

Pour simplifier la présentation de ce système nous allons en donner un schéma de principe.

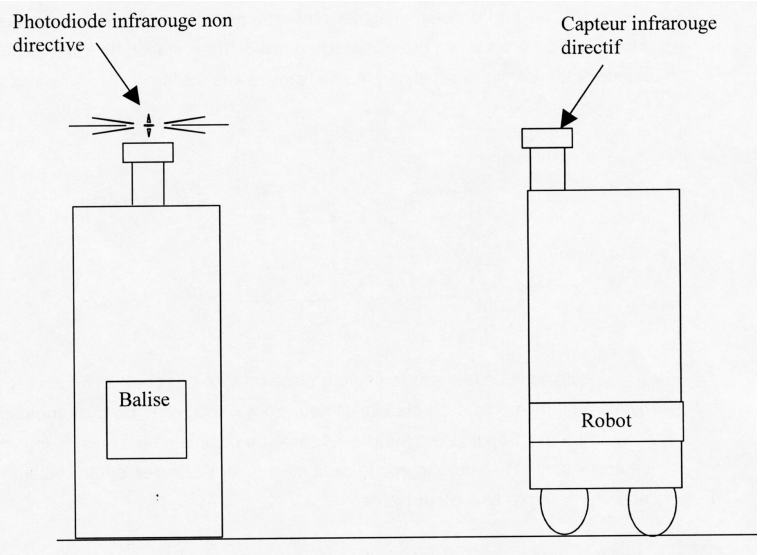


Fig. 2.11 – Principe de détection

Pour qu'un tel système puisse fonctionner il faut d'une part que la diode ne soit pas directive pour être sûr qu'elle émette toujours vers le robot. Ainsi le robot n'a qu'une rotation à effectué pour chercher le faisceau d'émission de la balise. Il faut par contre que le capteur soit directif. Si ce n'est pas le cas dès le départ on peut toujours rajouter une obstruction qui ne laissera passer qu'un cône très fin de rayons lumineux comme sur le schéma 2.12.

La taille de l'obstruction pouvant être choisie d'autant plus petite que la longueur d'onde l'est on comprend pourquoi les capteurs à ultrasons sont bien moins directifs que ceux à infrarouge.

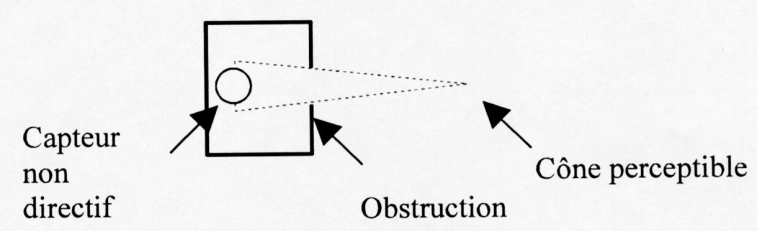


Fig. 2.12 – Capteur

D'autre part on peut imaginer un système a deux capteurs et deux photodiodes permettant de détecter en premier lieu la direction de la station de travail et enfin l'arrivée à cette station. Voici la situation près de l'arrivée à la station.

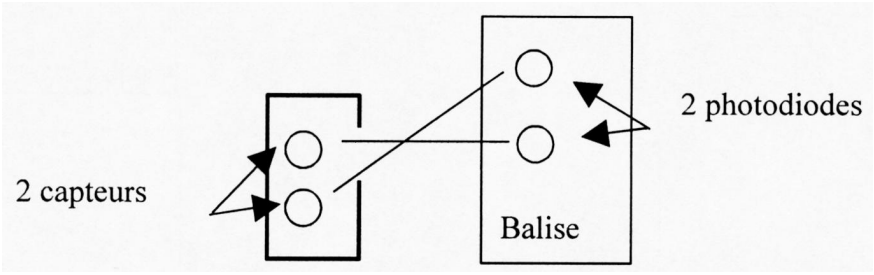
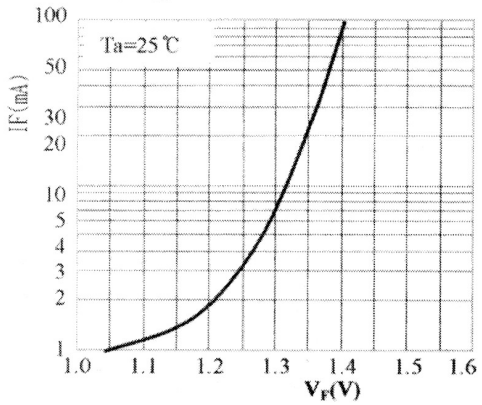
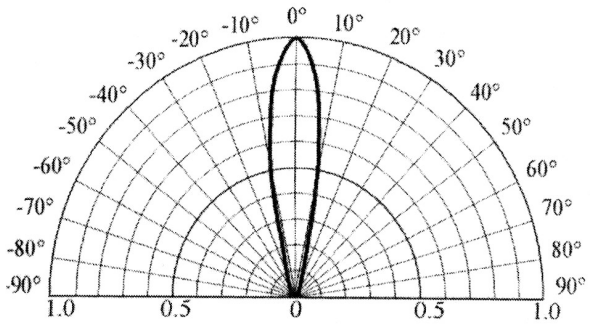


Fig. 2.13 – Capteur plus précis

Loin de la balise, seul le capteur placé en face de l'obstruction peut recevoir des rayons alors que près de la station les deux sont "éclairés". Ce système risque de s'avérer un peu trop imprécis mais peut servir de complément a un système mécanique (capteurs tactiles) : ainsi l'approche de la station sera détectée par le système optique et l'arrivée réelle par le capteur tactile.

Afin de réaliser ces montages on pourra utiliser les deux composants suivants dont les caractéristiques sont données par la suite. Pour l'émission on peut utiliser une diode émettrice OPE5594A dont les caractéristiques sont les suivantes :





directivité de l'emetteur

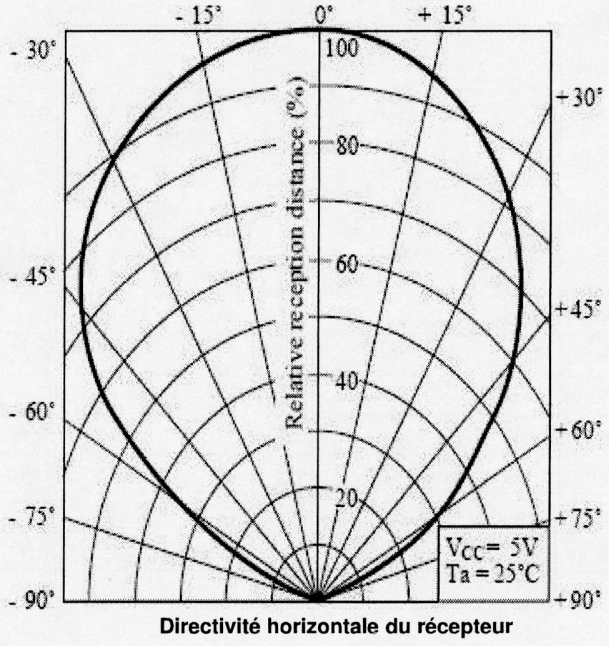
Item	Symbol	Conditions	Min.	Typ.	Max.	Unit
Forward voltage	V_F	I_F=100mA		1.4	1.7	V
Reverse current	I_R	V_R= 5V			10	μA
Capacitance	Ct	f = 1MHz		20		pF
Radiant intensity	Ie	I_F=100mA		80		mW/sr
Peak emission wavelength	λ_p	I_F= 50mA		940		nm
Spectral bandwidth 50%	$\Delta\lambda$	I_F= 50mA		45		nm
Half angle	$\Delta\Theta$	I_F=100mA		±10		deg.

Fig. 2.14 – Caractéristiques de la diode émettrice OPE5594A

On pourra prendre comme point de polarisation : i=100mA et u=1,4V.

Pour la réception on pourra utiliser le module GP1U26X dont les caractéristiques sont les suivantes :

Parameter	Symbol	Conditions	MIN.	TYP.	MAX.	Unit
Dissipation current	I_{CC}	No input light	-	-	5.0	mA
High level output voltage	V_{OH}	*3	V_{CC} - 0.5	-	-	V
Low level output voltage	V_{OL}		-	-	0.45	V
High level pulse width	T_1		400	-	800	µs
Low level pulse width	T_2		400	-	800	
B.P.F. center frequency	f_O	-	-	*4	-	kHz



Directivité horizontale du récepteur

Fig. 2.15 – Caractéristiques du module récepteur GP1U26X

Le problème avec la détection infrarouge est la recherche du rayon infrarouge en effet le robot ne saura pas par où commencer sa recherche, car une diode émettrice a une directivité comprise entre 20° et 40°.

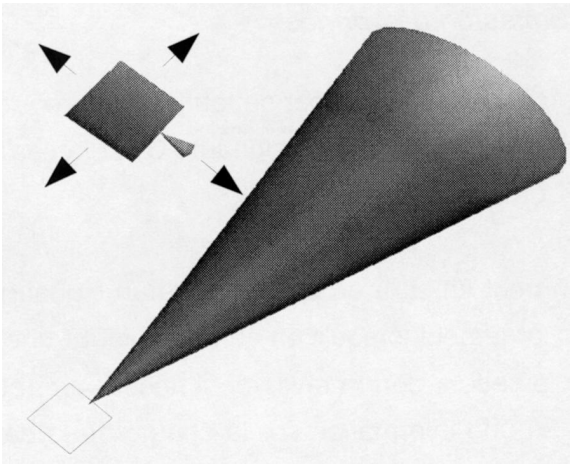


Fig. 2.16 – Fonctionnement avec une seule diode émettrice

Pour pouvoir balayer toute la pièce il faut donc installer plusieurs diodes émettrices. Pour être plus précis sur la direction de la balise il faut installer au moins 2 modules de réception sur le robot. Ensuite pour la detection de la direction de la balise on peut faire pivoter le robot et quand les deux modules sont éclairés le robot s'arrête de pivoter et avance tout droit. Si l'un des modules de réception n'est plus éclairé alors le robot doit se rediriger pour que les deux modules soient éclairés jusqu'à arriver à la balise.

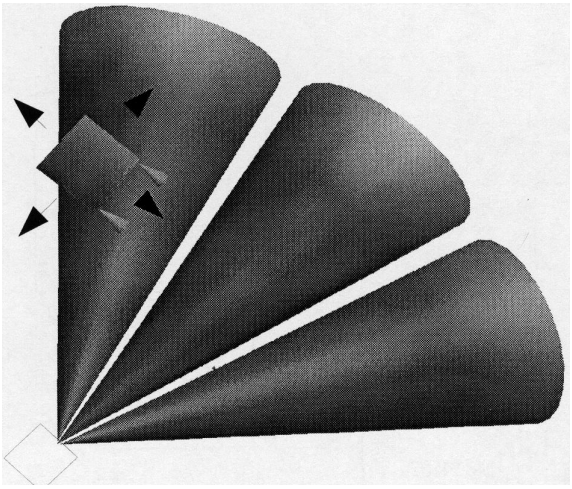


Fig. 2.17 – Fonctionnement avec trois diodes émettrices

Comme ça le problème a l'air réglé mais il reste un problème qui est la réflexion

du signal infrarouge car le signal peut se réfléchir sur les murs ou autres objet et ainsi éclairer le module de réception sans que cela soit utile. Il faudrait essayer de détecter un maximum de puissance optique dans les modules de réception et diminuer la directivité de la réception du récepteur.

c - Principe du système de détection

Il existe dans le commerce des circuits permettant d'émettre des données via infrarouge, et d'autres qui permettent de décoder ces signaux. C'est le cas notamment des circuits MC 145026 et MC 145027 de chez Motorola.

Avec ce type de circuit, on peut émettre en permanence un signal qui permettra au robot de retrouver la balise de rechargement lorsqu'il en aura besoin. Et une fois connecté à la balise pour son rechargement, l'émetteur pourra envoyer si nécessaire (et dans le cas de l'utilisation d'un des circuits du 2-2-2/) une information sur la charge de la batterie à la carte microcontrôleur embarquée sur le robot.

Voici une application à base de ces circuits :

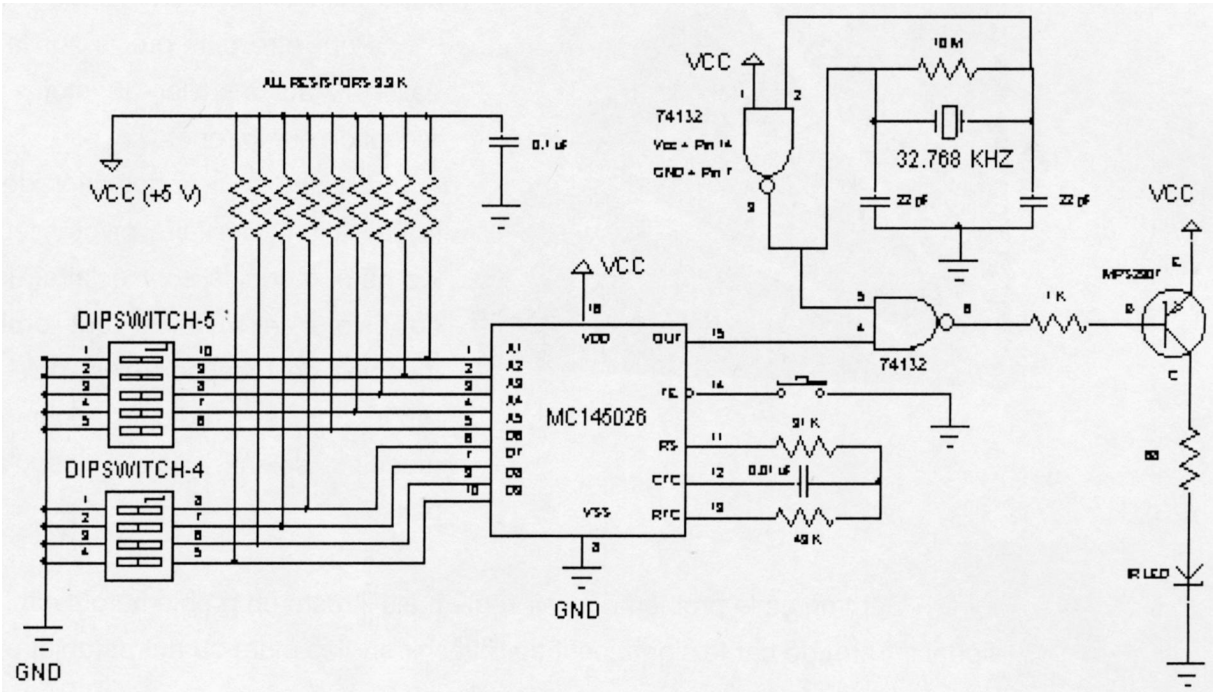


Fig. 2.18 – Schéma de l'émetteur

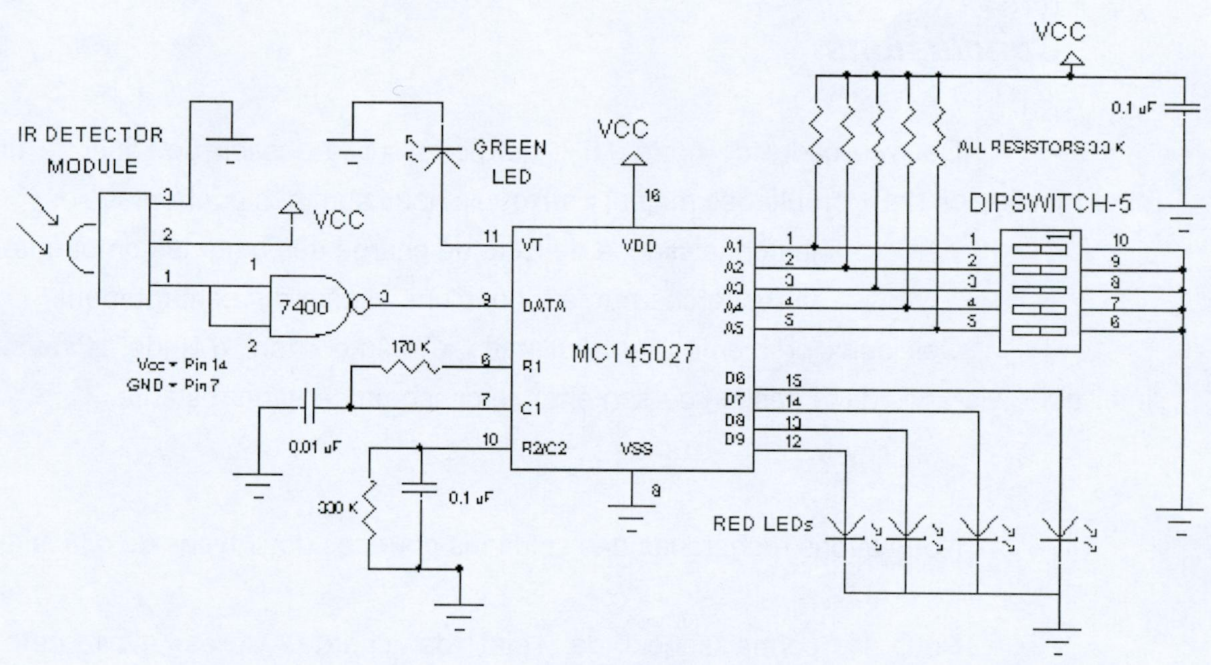


Fig. 2.19 – Schéma du récepteur

On peut donc avec ce type de montage réaliser deux fonctions à la fois :

– repérage de la balise par le robot ;

– “discussion” entre la balise de rechargement et le robot pour connaître l’énergie stockée dans la batterie.

2.4 Choix possible

Dans le cadre de ce TER sur la gestion de l'énergie embarquée sur un robot mobile, les deux problèmes majeurs auxquels nous sommes confrontés sont :

– la connaissance de l'état de charge des batteries embarquées ;
– la recherche par le robot d'une station de rechargement.

Nous avons recherchés des solutions pour ces deux types de difficultés.

Pour la connaissance de l'état de charge, après avoir noté les différentes caractéristiques de décharge des batteries au plomb ou au polymère, nous avons retenu deux types de circuit pouvant servir à mesurer l'état d'une batterie :

– l'un s'appuyant sur le contrôle de la tension aux bornes de cette batterie, puis une comparaison à des seuils à fixer ;
– l'autre s'appuyant sur le calcul de l'énergie consommée, par la mesure de la tension et du courant fournis par la batterie au cours de son utilisation.

Nous réaliserons probablement ces deux types de montage afin d'évaluer leur consommation et de voir lequel est le plus fiable.

Pour la partie détection de la balise de rechargement en environnement libre, deux systèmes différents nous sont venus à l'esprit :

– l'un réalisé à base d'émetteur et de récepteur à ultrasons ;
– l'autre basé sur l'infrarouge.

Après une brève étude sur chacun des systèmes, nous pensons plutôt retenir celui à infrarouge, les ultrasons ayant l'inconvénient d'un manque de directivité notable.

Au jour où nous avons réalisé ces recherches, nos choix se portent donc plutôt vers le simple contrôle de tension pour la partie "énergie", pour sa simplicité de mise en oeuvre, et sur un système de guidage par infrarouge pour la recherche de la base de recharge ; ces choix pouvant évoluer au cours du TER.

Chapitre 3

Choix de conception

Avant de commencer la conception des deux montages dont il a été question dans le chapitre précédent, nous avons dû faire des choix techniques et technologiques.

Voyons donc à travers ce chapitre ces différents choix, avec dans un premier temps, ceux liés au contrôle de l'énergie et dans un second temps, ceux qui ont été faits pour la détection de la balise de rechargement.

3.1 Système de contrôle de l'énergie

Pour cette partie, nous avions le choix entre deux types de circuit :
- un testeur de tension ;
- un calculateur d'énergie, emmagasinée ou dépensée.

Bien que les circuits testeurs de tension soient plus simples à mettre en oeuvre, nous avons opté pour un circuit capable de calculer l'énergie transmise entre la batterie et le robot, car ce dispositif est plus fiable.

Dans un premier temps, nous avons choisi de développer notre application autour de composants discrets et classiques : AOP, CAN, composants passifs... et d'envoyer les données ainsi acquises à la carte microcontrôleur embarquée sur le robot.

Le traitement de ces informations serait alors laissé au microcontrôleur principal, dédié en priorité à la gestion des modules PWM associés aux deux moteurs de traction.

Mais les calculs à effectuer par ce microcontrôleur nécessitaient d'utiliser trop souvent et de manières importantes les ressources CPU, au détriment de son utilisation principale : la gestion des moteurs.

La présence d'un microcontrôleur ou d'un organe capable de faire des calculs dans notre montage a alors été envisagé.

Cependant, développer notre circuit autour d'un 80C552 n'était pas concevable : la présence d'une deuxième carte basée sur ce même microcontrôleur aurait pris beaucoup de place sur le robot et aurait consommé trop de courant pour l'utilisation souhaitée, ce qui n'était pas en accord avec notre cahier des charges.

C'est donc en recherchant un microcontrôleur plus simple à mettre en oeuvre que nous sommes tombés sur les nouveaux microcontrôleurs de chez *Cypress MicroSystems* (19) : les PSoC.

Ces nouveaux circuits intégrant à la fois de l'analogique et du numérique sur une même "puce" de silicium (voir **Annexe A** et **Annexe C**), vont nous permettre de réduire considérablement le nombre de circuits intégrés, et donc à la fois la taille et la consommation de notre montage.

De plus, ce sont de nouveaux produits tournés vers l'avenir.

Nous avons donc choisi de développer ce système de calcul de l'énergie embarquée autour de l'un de ces nouveaux microcontrôleurs, les PSoC, ce qui va nous permettre de réduire considérablement la taille et la consommation du montage, mais aussi nous permettre de rendre notre système totalement autonome.

3.2 Système de détection de la balise de rechargement

En accord avec la bibliographie, pour la détection de la balise nous avons choisi un système infra-rouge, car il est plus facile à mettre en place et moins coûteux.

Pour cela, dans un premier temps nous avions choisi une méthode analogique (récepteur, suiveur, filtre, amplificateur, détecteur de crête) ce qui devait créer une interruption transmise au microcontrôleur, mais cette technique s'avéra trop aléatoire. En effet un peigne de Dirac apparaissait aléatoirement au moment de la détection. Nous nous sommes donc tournés vers une détection numérique (récepteur, suiveur, filtre, amplificateur, détecteur d'enveloppe, CAN, programme de détection de la balise) qui s'avéra simple à mettre en place et qui, sans les problèmes dus à la commande en rotation du robot, donna des résultats assez satisfaisants.

Pour l'emission, contrairement à la bibliographie, nous avons choisi une diode infra-rouge SFH485. Nous avons choisi cette diode car l'une de ces applications principales est la télécommande par infra-rouge pour téléviseur et chaîne Hi-Fi, donc la distance de détection peut-être de 3m voire plus ; l'émission de cette diode est de 40° ce qui est bien pour notre application car le robot doit pouvoir détecter la balise n'importe où dans l'espace.

Pour la reception, contrairement à la bibliographie, nous avons choisi un phototransistor BPX25 car celui-ci reçoit les signaux dans un angle de 10° ce qui permet d'être très précis dans la détection de la balise. Pour des raisons pratiques (réflexion de la lumière sur les objets), nous avons décider d'utiliser ce transistor en amplificateur et non en saturation afin de détecter le maximum d'amplitude de la lumière émise.

Nous avons choisi d'intégrer un contrôle du gain à l'aide de switchs analogiques qui permettent de choisir un gain entre 1, 10, 100 et 1000. Nous avons choisi d'intégrer ce système car l'amplitude des signaux varie en fonction de la distance

entre la balise et le robot, ainsi l'AOP qui sert d'amplificateur saturait lorsque le robot était trop près de la balise. Nous avons préféré cette solution à un contrôle automatique de gain car la détection de maximum aurait été impossible.

Dans notre application nous avions besoin de deux amplificateurs opérationnels, pour des raisons de compacité nous avons donc choisi d'utiliser un TL082.

Afin d'alimenter les amplificateurs opérationnels de façon symétrique et ainsi éviter les saturations indésirables nous avons placé derrière le 7805, qui alimente la plaquette en 0-5V, un circuit MAX232 qui fournit à la carte du $\pm$10V.

Le principal inconvénient de cette étude est de minimiser l'énergie consommée pendant le fonctionnement "normal" du robot, pour cela nous avons inséré, entre le 7805 et le MAX232, un interrupteur commandé par le microcontrôleur, ainsi la plaquette destinée à la détection de la balise ne fonctionne que pour sa fonction.

Chapitre 4

Structures réalisées

4.1 Système de contrôle de l'énergie

Comme annoncé dans le chapitre précédent, nous avons retenu d'utiliser un microcontrôleur intégrant à la fois des fonctions analogiques et numériques.

Le principe que nous avons choisi de mettre en oeuvre est le calcul de l'énergie :

$$E = T.\int_0^T U(t).I(t).dt \tag{4.1}$$

Et comme nous voulons traiter ces données numériquement, nous commençons d'abord par venir échantillonner U(t) et I(t).

Les signaux U(t) et I(t) qui nous intéressent évoluent très lentement, c'est à dire que leurs variations sont de l'ordre de la seconde. Pour pouvoir acquérir ces deux signaux de façon fiable, une fréquence de conversion supérieure à 10 Hz suffirait.

Nous faisons ces acquisitions à une fréquence de 300 Hz, fréquence maximale que nous avons pu atteindre afin de laisser le temps au processeur d'effectuer tous les calculs qui lui sont demandés entre deux acquisitions.

Nous ne calculerons donc pas la valeur réelle de l'énergie E, mais une approximation de cette intégrale par la méthodes des rectangles (voir figure 4.1).

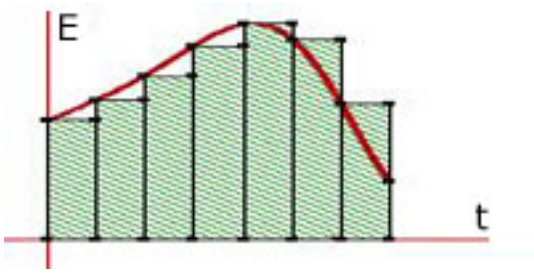


Fig. 4.1 – Approximation d'une intégrale par la méthode des rectangles

Pour cela, nous calculerons chaque échantillon E_n de cette façon :

$$E_n = \frac{U_n.I_n}{F} \tag{4.2}$$

que nous sommerons ensuite pour obtenir l'énergie totale :

$$E = \sum_{n=1}^{n=N} \frac{U_n.I_n}{F}$$

La partie calcul de E_n et la sommation de tous les échantillons recueillis sont réalisées logiciellement et seront programmées sur le PSoC.

A chaque fois que cette énergie est calculée, elle est ensuite comparée à une valeur de référence. Une fois cette valeur atteinte, une interruption est envoyée à la carte principale, afin qu'elle lance la procédure de recherche de la balise.

Cette valeur de référence est choisie de façon à laisser au robot environ 5 minutes afin qu'il puisse rechercher une balise de rechargement et s'y rendre.

Voici le schéma fonctionnel de cette partie :

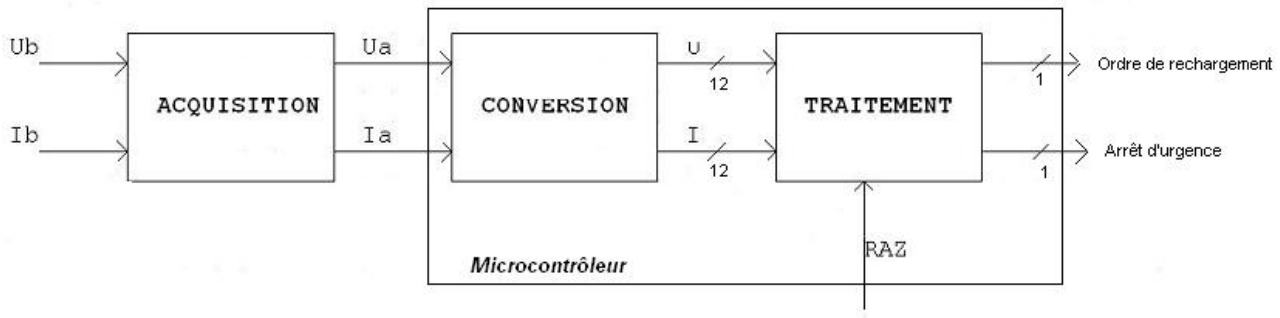


Fig. 4.2 – Schéma bloc du système de calcul de l'énergie embarquée

4.2 Système de détection de la balise de rechargement

Une fois l'ordre de lancement de la procédure de recherche effectué, on doit commencer la recherche de la balise de rechargement.

Pour cela, comme énoncé dans le chapitre 3 on choisit une détection numérique. Ce système comporte une récepteur, un suiveur, un filtre, un amplificateur dont on peut commander le gain, et un détecteur d'enveloppe comme on peut le voir sur ce schéma :

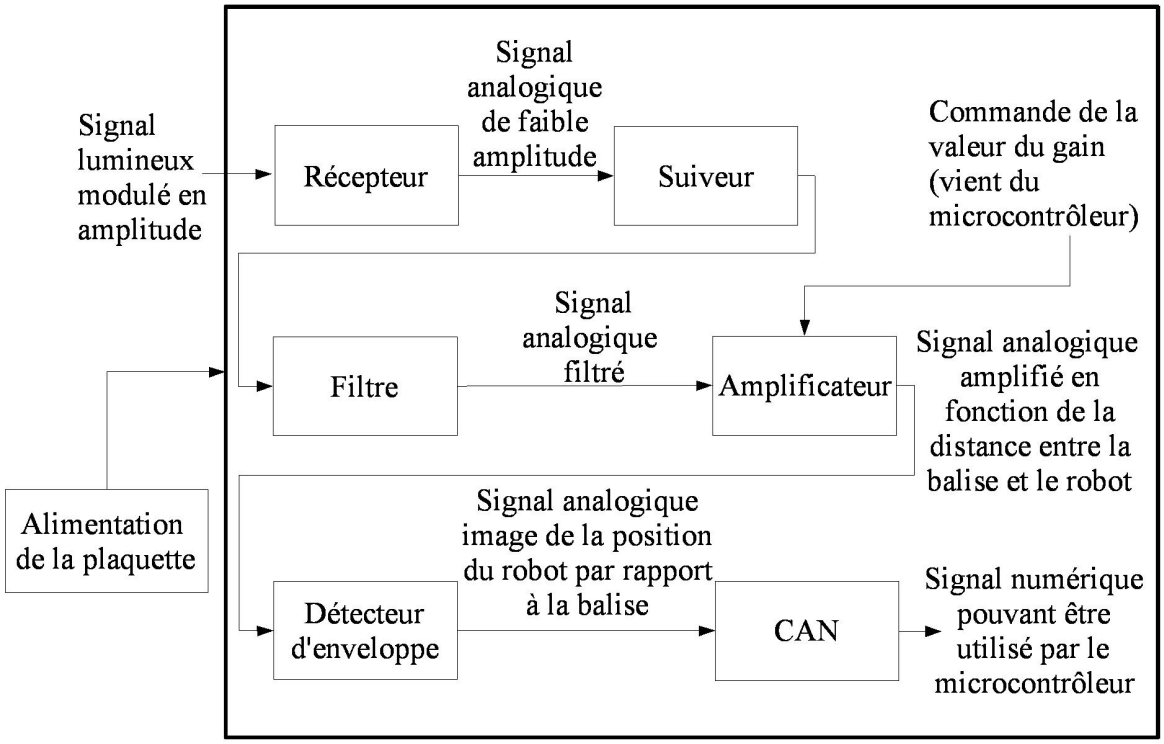


Fig. 4.3 – Schéma bloc du détecteur de balise

Deuxième partie

Etude des différentes fonctions

Cette partie est divisée en 2 chapitres.

Le premier traite du système de contrôle de l'énergie qui a été développé par Vincent THOMAS et Julien VILLEMEJANE autour d'un PSoC (P S on Chip) de Cypress.

Étant les premiers dans l'établissement à développer sur ce type de microcontrôleur, une prise en main du logiciel de développement, à travers des exemples simples, était nécessaire. Ce travail a été réalisé avec l'aide de Lionel CIMA (laboratoire SATIE). Vous trouverez un descriptif des PSoC et de leur prise en main en **Annexe A**.

Le second chapitre concerne le système de recherche de la balise de rechargement réalisé par Guillaume HERAULT et Loïc SIMON.

Chapitre 5

Système de contrôle de l'énergie

Dans cette partie, vous trouverez le détail et la réalisation des fonctions de la partie I.4.

Nous traiterons dans un premier temps la partie Acquisition et Conversion des signaux d'entrée U et I qui est l'interface entre la partie purement analogique et la partie de traitement de ces données réalisée par le microcontrôleur.

Une deuxième partie présentera les fonctions de traitement et de création des signaux logiques de sortie. Cette section est portée essentiellement sur le code écrit pour le microcontrôleur. Vous trouverez l'intégralité de ce code commenté en **Annexe B** et les descriptifs des fonctions utilisées dans le PSoC en **Annexe C**.

Comme indiqué dans la partie I.3.1, notre premier choix ne s'est pas porté sur le PSoC, car nous n'avions pas connaissance de la présence d'une plateforme de développement dans les locaux, ni même des capacités et performances de ces nouveaux microcontrôleurs.

Nous avons donc, dans un premier temps, développé une solution "analogique" d'un contrôleur d'énergie, s'appuyant sur une méthode d'intégration.

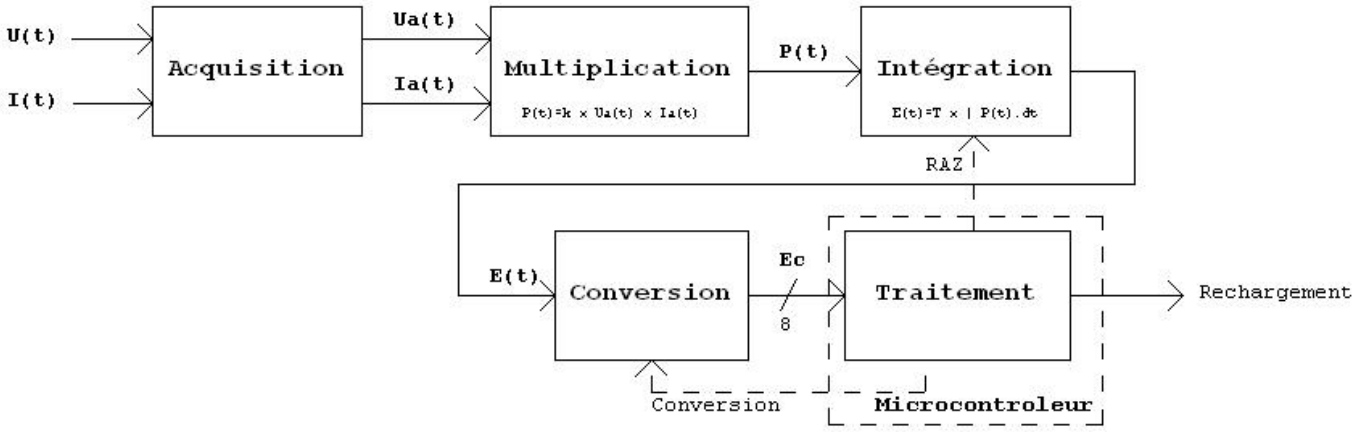


Fig. 5.1 – Schéma fonctionnel du premier montage réalisé

Ce montage nécessitait alors un multiplieur analogique, un montage intégrateur, un convertisseur analogique-numérique, quelques amplificateurs opérationnels pour la partie *Acquisition* et la présence d'un bus de données (9 bits minimum en parallèle ou 2 bits avec l'utilisation d'un bus I2C) pour l'envoi des ordres de conversion et la réception des échantillons convertis.

De plus, la partie traitement devait être effectuée par le microcontrôleur principal du robot ; ce qui rendait notre module non-autonome et qui nécessitait l'utilisation de trop de ressources CPU sur le module de commande du robot.

Nous avons alors développés notre application autour des PSoC. Avec ces derniers, nous avons pu réduire le nombre de circuits intégrés considérablement (et donc la consommation) mais aussi rendre notre module de contrôle d'énergie complètement autonome.

Voici le résultat.

5.1 Acquisition et conversion de U et de I

Comme on l'a vu dans la partie précédente, on souhaite réaliser une structure capable de calculer l'énergie emmagasinée dans une batterie servant à alimenter un robot mobile. Pour calculer cette énergie E, il faut dans un premier temps faire l'acquisition de U et I aux bornes de cette batterie.

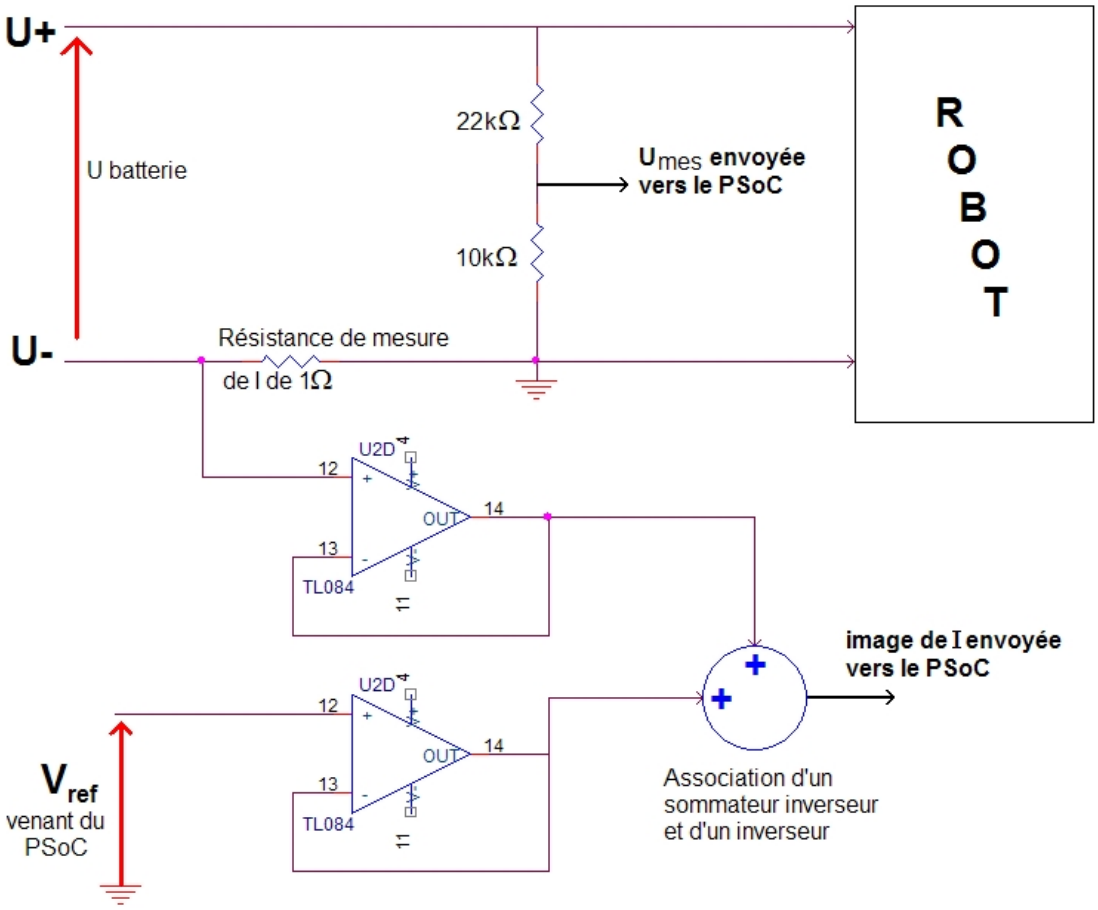


Fig. 5.2 – Schéma d'acquisition de U et I

On effectue l'acquisition d'une image de la tension aux bornes d'un pont diviseur de tension de rapport $\frac{1}{3}$ afin de la rendre compatible avec les entrées 0-5V du PSoC. Ainsi pour une tension de 15 V en entrée, ce qui laisse une marge de sécurité par rapport à la tension nominale de la batterie, on a une tension de 5V en entrée du PSoC.

Pour l'acquisition de l'image du courant, nous utilisons une résistance de "visualisation" de 1 Ω. La tension aux bornes de cette résistance est donc l'image du courant et elle est envoyée sur un montage suiveur afin de ne pas fausser la mesure en absorbant un courant dû à la résistance totale du circuit.

Le courant ne dépassant pas 0,5A en utilisation et 1,6A en charge, nous avons une tension en entrée du PSoC comprise entre -0,5V et 1,6V. Or le PSoC ne supporte pas de tensions négatives sur ses entrées, il a donc fallu trouver un artifice. Les CAN utilisés en entrée sur le PSoC convertissant des valeurs d'entrée entre 0 et 5V sur une plage de -2047 à +2048 en sortie, on peut décaler l'image du courant de 2,5V. Ainsi on obtiendra 0 en sortie du CAN pour un courant nul et le résultat de la conversion sera un nombre négatif lorsque le courant sera négatif et positif dans le cas contraire.

Pour créer cette tension de 2,5V, on utilise une des sorties du PSoC. Cette tension passe elle aussi par un montage suiveur pour éviter toute chute de tension et garantir une tension stable. Elle est ensuite sommée avec l'image du courant à l'extérieur. Puis on réinjecte la tension obtenue sur une entrée du PSoC.

La présence d'AOP et surtout de tensions négatives (le courant et la tension inversée du sommateur) nous ont obligé à rajouter dans notre montage un composant permettant à partir d'une tension continue et positive, de créer deux tensions symétriques égales à +/- 2 x V_{CC} (V_{CC} étant la tension d'alimentation de ce composant). Le composant que nous avons choisi est un MAX 232 (voir figure 5.3 et aussi **Annexe E**).

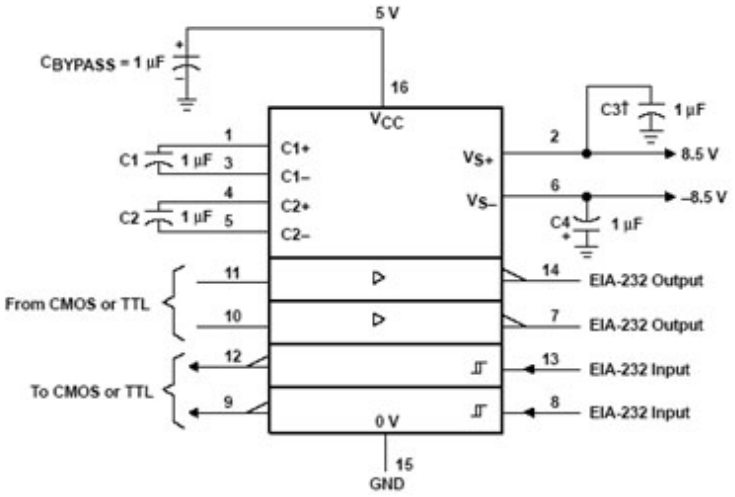


Fig. 5.3 – MAX232

Nous avons aussi dû implanter un convertisseur DC-DC, un 7805, afin de pouvoir alimenter le microcontrôleur en 5V, la seule tension disponible sinon étant celle de la batterie, soit environ 12V.

Les deux tensions, images du courant et de la tension aux bornes de la batterie, sont ensuite converties sur 12 bits par les deux CAN présents et prévus dans le PSoC. Les données sont ensuite traitées par le programme.

5.2 Traitement des données - Signaux de sortie

A l'intérieur du programme, les données récupérées U et I sont contenues dans deux variables de type long portant respectivement les mêmes noms. Le programme commence par une phase d'initialisation du PSoC. Durant cette phase le programme démarre chacun des modules utilisés, comme les CAN, l'EPROM, les différents ports d'entrée/sortie...

Il vient aussi lire dans l'EPROM la valeur de l'énergie mémorisée et la recharge dans E.

Enfin, une première acquisition des données est effectuée.

Suivant le signe du courant, le programme va se placer dans la "boucle de décharge" ou dans la "boucle de charge". A l'intérieur de ces boucles on va, dans les deux cas, calculer de façon discrète l'énergie que voit passer le dispositif. Après une acquisition de U et de I, on calcule dans un premier temps l'énergie "instantanée" que l'on stocke dans une variable de type long :

$$E_n = U * I/(k * f) \tag{5.1}$$

avec f la fréquence d'échantillonnage, qui est la fréquence maximale que nous avons pu atteindre permettant le bon fonctionnement du programme, et k un coefficient constant et fixé permettant de rabaisser la valeur de l'énergie (pour cette application nous l'avons pris égal à 4). Il a été calculé de telle sorte qu'à courant maximal en utilisation (0,5 A) et à tension nominale (12 V), le calculateur ait une capacité de stockage des données largement supérieure à 1h.

En récupérant 300 échantillons à la seconde, pour des valeurs de $I_{max} = 500$ et $U_n = 3500$ (correspondant à I maximal et U nominal), avec une variable de type long signée (soit sur 32 bits) on calcule ainsi une durée maximale où le calculateur n'aura pas de problème de dépassement :

$$T = \frac{E_{max}.k.F^2}{U_n.I_{max}} \qquad (ensecondes) \tag{5.2}$$

où $E_{max} = 2^{16}$, soit T = 224 minutes (un peu plus de 3h30).

On vient ensuite sommer cette dernière valeur calculée à la valeur de l'énergie totale calculée depuis le debut : $E = E + E_n$

Grâce au coefficient k, on repousse la limite de débordement de la valeur de l'énergie capable d'être stockée dans la variable E. Ce coefficient étant présent dans le calcul de l'énergie à la fois dans la phase de charge et de décharge, il ne fausse en rien les calculs.

A chaque itération de ces deux boucles, le programme scanne le bit 2 du port 1 pour savoir si un ordre de réinitialisation de la valeur de l'énergie a été demandé et toutes les 30 secondes la valeur de l'énergie E est mémorisée dans l'EPROM du PSoC, afin qu'elle soit à nouveau disponible après un arrêt du robot, la RAM étant totalement effacée lors d'une coupure d'alimentation du PSoC.

Étudions un peu plus en détail les deux sous-programmes "boucle de décharge" et "boucle de charge" : d'abord la "boucle de décharge" où deux tests sont effectués.

Le premier test porte sur la tension U, si celle-ci passe en dessous du seuil limite fixé U_{ref}, le programme attend 300 mesures supplémentaires avant de mettre à 1 le bit 1 du port 1 et ainsi donner l'ordre de rechargement au robot. Cette précaution est nécessaire car d'une part en cas de pics de courants transitoires on aurait des chutes de tensions qui les accompagneraient, et d'autre part, d'après les courbes 5.4, il se peut qu'il y ait des erreurs d'échantillonnage.

Ces courbes ont été obtenues en réalisant un test sur la batterie au lithium-polymère. Nous l'avons fait débiter dans une charge équivalente à la notre (soit environ 0,5 A) et nous avons mesuré la tension et le courant toutes les minutes aux bornes de cette batterie.

De plus, les batteries ayant un seuil de tension minimale à ne pas franchir pour ne pas les détériorer, si la valeur de U devient inférieure à une valeur de tension critique U_{bas}, on envoie l'ordre d'arrêt d'urgence au microcontrôleur du robot en

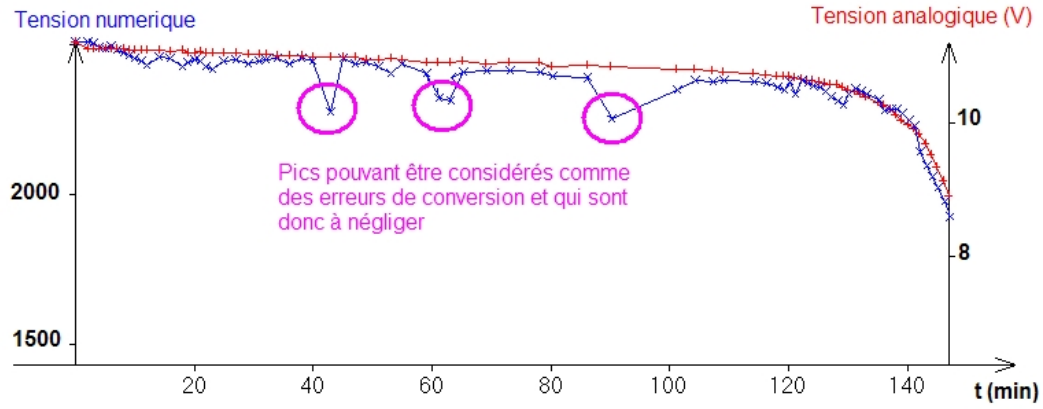


Fig. 5.4 – Relevé de la tension aux bornes de la batterie au cours du temps : valeurs analogiques et numérisées

mettant à 1 le bit 2 du port 1. Cette tension U_{bas} a été définie en fonction des courbes obtenues lors des tests de la batterie (voir figure 5.5).

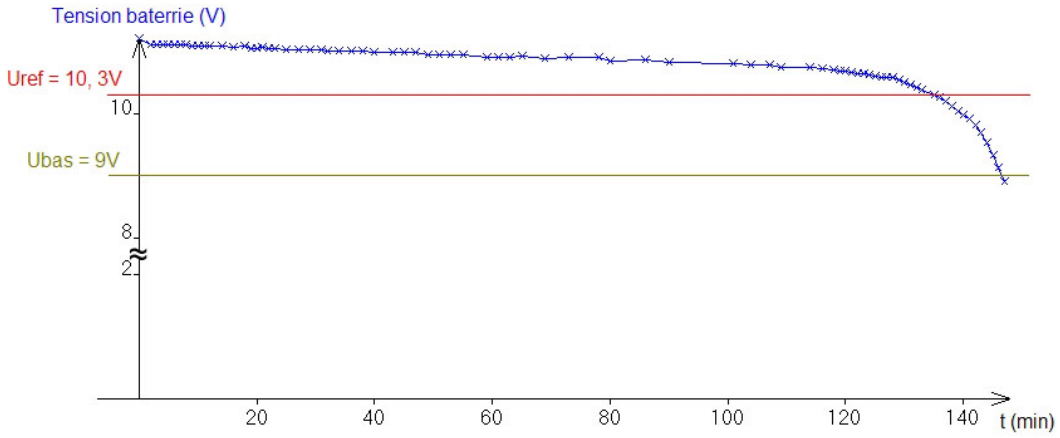


Fig. 5.5 – Choix des tensions critiques

Le deuxième test est effectué sur la valeur de l'énergie E que l'on vient comparer à une valeur de référence E_{ref}. Si E passe en dessous de E_{ref}, on envoie là aussi l'ordre de rechargement en mettant à 1 le bit 1 du port 1.

Ensuite dans la "boucle de charge", dans laquelle on rentre dès que le courant est positif, on teste à nouveau l'énergie. On passe le bit 0 du port 1 à 1 aussitôt que l'énergie E est redevenue supérieure au seuil critique E_{ref}.

5.3 Schéma électrique complet et typon

Voici le schéma électrique complet de cette partie :

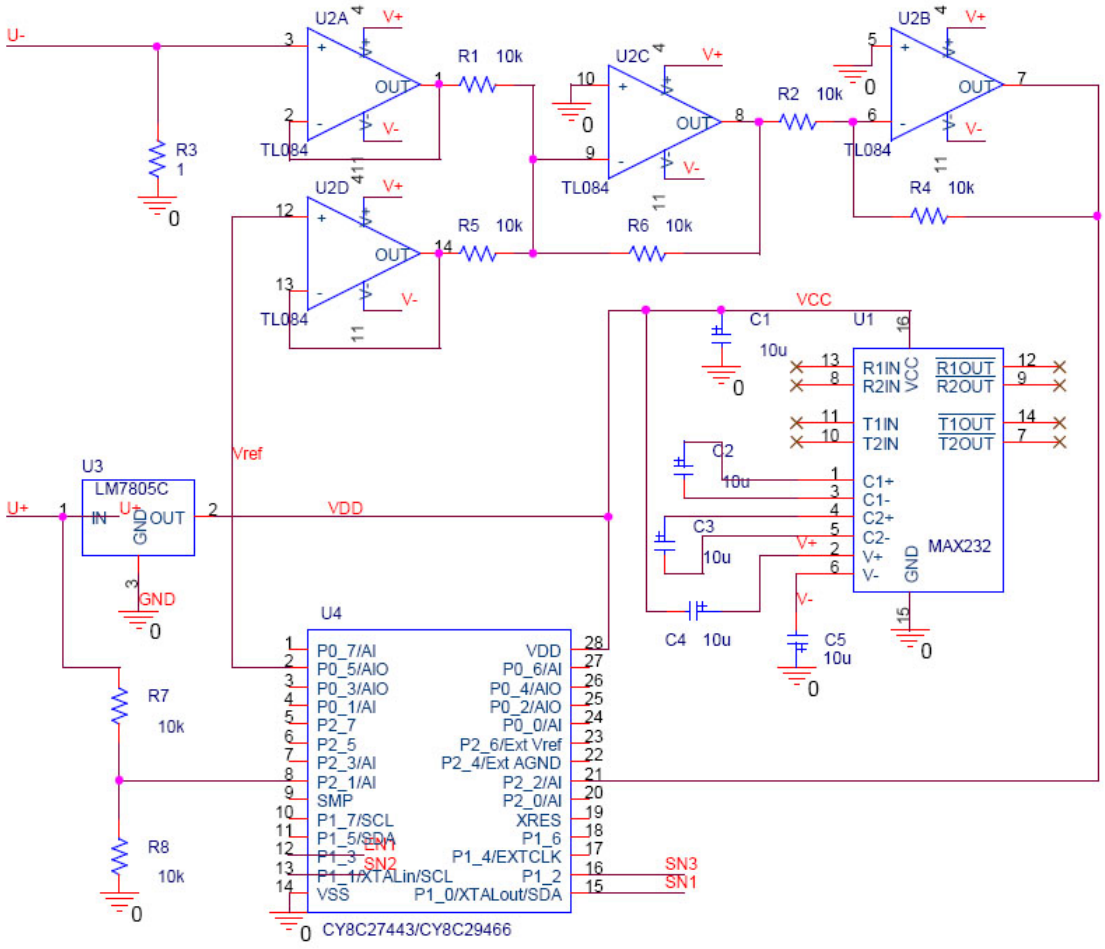


Fig. 5.6 – Schéma électrique de la partie calcul de l'énergie

Ainsi que le typon du circuit imprimé réalisé :

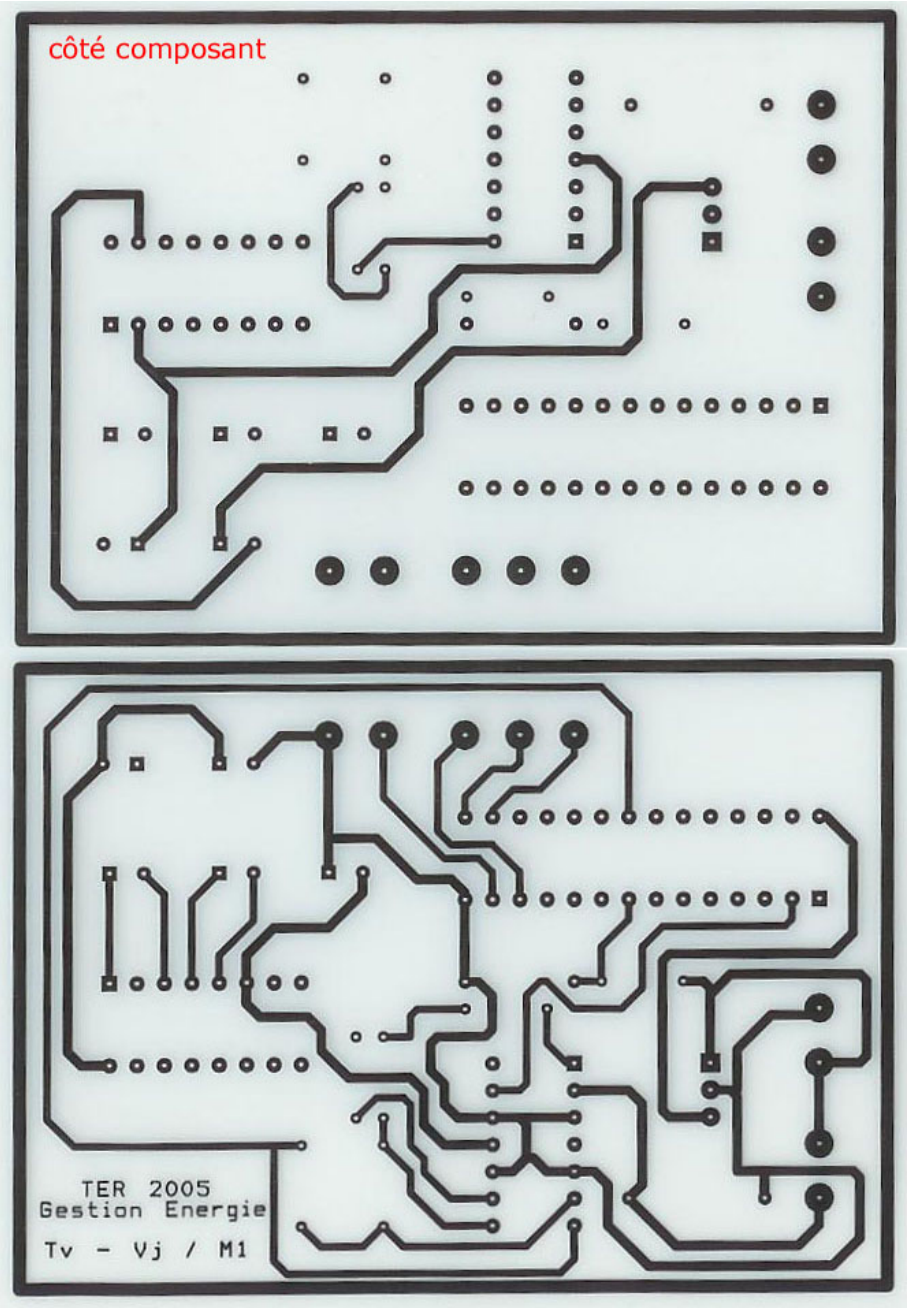


Fig. 5.7 – Typon de la partie calcul de l'énergie

Chapitre 6

Système de détection de la balise de rechargement

Dans cette partie nous allons voir les différentes parties du schéma bloc de la partie 4.2.

6.1 Système d'émission de la balise de rechargement

voici le schéma électrique du système d'émission :

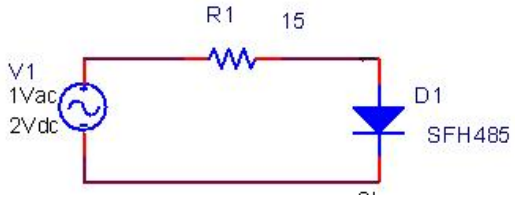


Fig. 6.1 – Schéma du système d'émission

Il y a eu plusieurs paramètres à régler suivant certaines contraintes.

- Tout d'abord la valeur de la résistance qui a été dicté par la valeur limite du courant dans la diode.
- Ensuite, ayant fixé approximativement la valeur de la tension de polarisation à 2 Volts (soit légèrement au dessus de la tension de seuil de la diode)

nous avons effectué les tests portant sur l'amplitude du signal en sortie de l'amplificateur (gain réglé sur 1000).

Nous détaillerons ces tests dans le chapitre 8 mais remarquons qu'ils nous ont permis de fixer l'amplitude du "petit signal" à l'émission (sur 1 Volts) et par conséquent la polarisation définitive de la diode IR. En effet, on comprend rapidement que celle-ci doit être égale à la somme de la tension de seuil de la diode et de l'amplitude du "petit signal". La choisir inférieure mènerait à un régime non linéaire dû au blocage occasionnel de la diode.

6.2 Système de réception de la balise de rechargement

a - Schéma de l'alimentation de la carte de détection de la balise

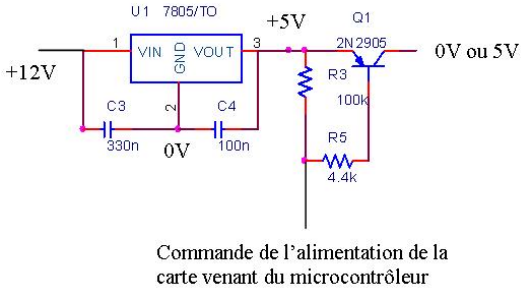


Fig. 6.2 – Commande de l'alimentation de la carte de détection de balise

L'alimentation de la carte est contrôlée par le microcontrôleur, c'est-à-dire que quand le microcontrôleur envoie un niveau logique haut '1', le transistor est bloqué donc la carte n'est pas alimentée. Et si le microcontrôleur envoie un niveau logique bas '0', alors le transistor est passant et ainsi la carte est alimentée.
Ce montage permet donc d'économiser de l'énergie lorsque le robot est en fonctionnement "normal".

b - Schéma de polarisation du phototransistor

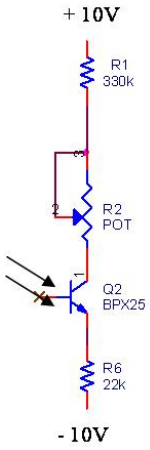


Fig. 6.3 – Polarisation du phototransistor BPX25

Tout d'abord expliquons le principe de fonctionnement d'un phototransistor : un phototransistor est un transistor dont le nombre de porteur minoritaire de la jonction base-émetteur dépend de l'éclairement auquel elle est soumise (cette dépendance n'est effective que dans une gamme donnée de longueurs d'ondes, voir figure 8.6). De ce fait le courant de base est fonction de cet éclairement. D'un point de vue plus simpliste, on peut dire qu'une fois le phototransistor polarisé, l'éclairement joue le rôle d'entrée petit signal (c'est en tout cas l'utilisation qu'on en a faite). Voici maintenant le descriptif du montage que l'on a mis en place pour notre détecteur.

Dans la littérature on trouve deux types de polarisation en émetteur commun qui diffèrent simplement par la présence ou non de polarisation de la base. Les tests des deux types de montages nous donnaient des amplitudes de signaux comparables (voire même à l'avantage du montage sans polarisation de la base). Comme dans notre cahier des charges l'un des points clefs était la minimisation de l'énergie consommée et que tout dispositif de polarisation contribue à cette consommation, nous avons choisi de ne pas polariser la base. Pour ce qui est du choix de la polarisation du collecteur (ie le choix du courant I_{C0} et de la tension

V_{C0}) nous avons pris en compte les point suivants :

- Les caractéristiques du BPX25 nous limitaient en courant (la valeur limite donnée par notre notice est de : 2mA). D'où le choix :

$$(R_E + R_C) > \underbrace{\frac{V_{cc+} - V_{cc-}}{I_{lim}}}_{= 8,5k\Omega}$$

- Le traitement ultérieure du signal "petit signal" en tension de collecteur (image de la lumière incidente) nous a poussé à choisir une tension V_{C0} nulle.

En effet la suite nous contraignait à utiliser des amplificateurs opérationnels que nous allions alimenter de façon symétrique. De ce fait, choisir une polarisation nulle devait nous permettre d'augmenter la dynamique des petits signaux (c'est-à-dire l'amplitude qui nous permettrait de rester dans un régime de fonctionnement non saturé). La solution classique utilisée, consiste non pas à se placer sur une polarisation nulle, mais à rajouter en sortie du collecteur un condensateur dit de liaison (qui coupe la composante continue du signal). Cette solution a été testée mais en série avec l'étage suivant (un suiveur), le condensateur avait tendance à se charger : ainsi le suiveur finissait toujours par saturer. On a donc opté pour la solution d'une polarisation nulle et repousser la capacité de liaison en sortie du suiveur. Expliquons rapidement les risques d'une telle solution et pourquoi dans notre cas nous pouvions ne pas nous en soucier :

Le vrai risque d'une telle approche est que les composants utilisés pour le réglage de la polarisation ont des caractéristiques qui varient avec la température. De ce fait, au cours de l'utilisation du robot le système risque de devenir non opérationnel si l'un des AO passe en régime saturé. Cependant dans notre cas le premier AO est un suiveur et la dynamique des signaux qui lui sont injectés est faible (moins d'un volts en situation extrême : ie robot à 20 cm de la balise). Le deuxième AO quant à lui est monté en amplificateur de gain réglable (par le microcontrôleur) pouvant aller jusqu'à 1000. On comprend donc que, pour cet AO,

le risque de saturation est grand ; c'est pourquoi nous avons placé une capacité de liaison entre le suiveur et l'amplificateur.

c - Intérêt du suiveur et de l'amplificateur

Une fois ce montage validé, ayant remarqué que l'amplitude des signaux décroissait rapidement avec la distance entre la diode émettrice et le récepteur, nous avions en évidence la nécessité d'un amplificateur à gain réglable (en fonction de la distance). Cependant, la solution d'un amplificateur inverseur classique à partir d'un AO modifiait énormément les signaux. Pour pallier à ce défaut nous avons donc introduit un suiveur (impédance d'entrée infinie).

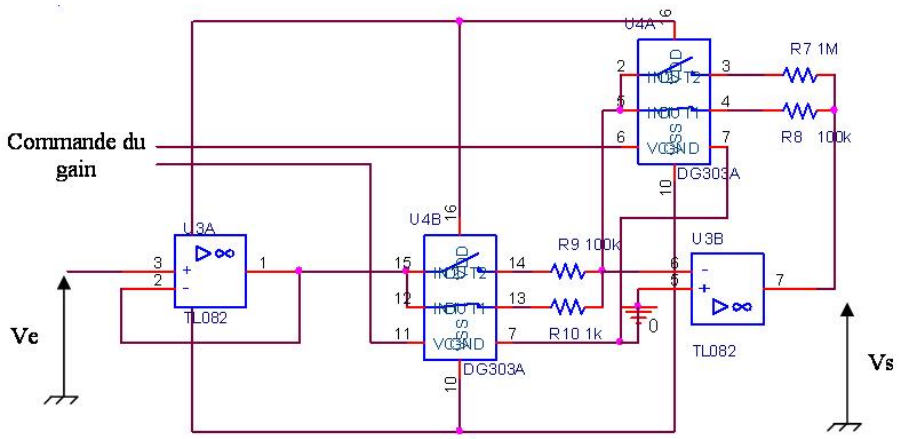


Fig. 6.4 – Schéma de l'amplificateur avec en entrée un suiveur

d - Pourquoi un filtre ?

L'alimentation disponible sur le robot provient des batteries en 0-12V (un 7805 nous donne accès à du 0-5V). Cependant pour augmenter la dynamique des signaux nous avions intérêt à obtenir une alimentation symétrique. Par suite, nous avons décidés de fabriquer à partir de la tension des batteries, une alimentation $\pm$10V. Nous avons, pour ce faire, utilisé en série, un 7805 et un MAX232. Malgré les condensateurs de filtrage (de 10μF), cette tension restait bruitée (bruit de l'ordre de 5mV ce qui correspond à peu près au signal en sortie du phototransistor lorsque la balise est à 1m). Par conséquent, le signal en sortie du suiveur était

lui aussi bruité ce qui aurait pu causer des problème au niveau de la détection d'enveloppe et rendre le système pour le moins imprécis. Deux solutions étaient alors envisageables :

– filtrer directement l'alimentation par un filtre passe-bas passif (R-C).
– filtrer uniquement le signal en sortie du suiveur.

L'inconvénient de cette dernière solution était évident : la fréquence de coupure du filtre devait être supérieure à la fréquence du signal d'émission. Ainsi le filtre allait être moins efficace.

Cependant en filtrant directement l'alimentation, celle-ci tombait autour de 3 volts quand on alimentait le système complet. Cette solution était donc définitivement proscrite.

Nous avons donc choisi de filtrer le signal après le suiveur de tension.

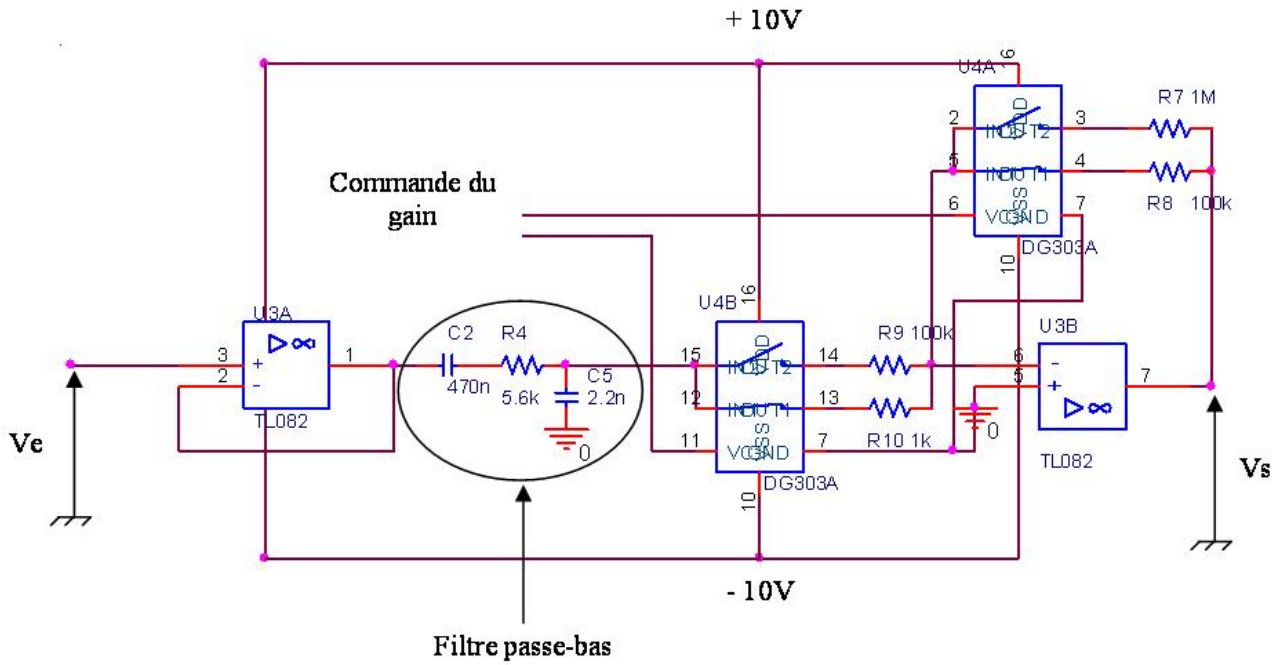


Fig. 6.5 – Schéma avec le filtre passe-bas

e - Le détecteur d'enveloppe

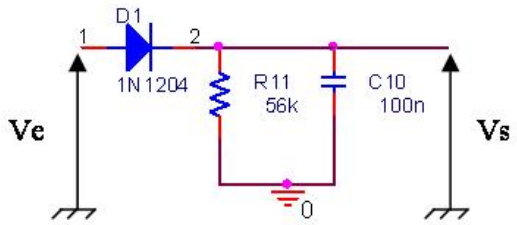


Fig. 6.6 – Schéma du détecteur d'enveloppe

Ayant rapidement favorisé un traitement numérique des acquisitions face à la complexité d'un système de détection entièrement analogique, nous devions adapter le signal à convertir au traitement qui allait en être fait (détection du maximum comme établi dans le rapport bibliographique). Et de ce fait cette adaptation devait transformer le signal alternatif en un signal continu dont l'amplitude serait la même (et donc serait image de l'écart angulaire par rapport à la balise). Ce rôle est bien évidemment celui d'un détecteur d'enveloppe. Pour la réalisation de celui-ci nous avons mis en évidence plusieurs contraintes importantes :

- La tension de seuil de la diode utilisée devait être la plus faible possible ; en effet la tension de sortie est égale à l'enveloppe du signal d'entrée moins cette tension de seuil. Donc pour détecter une enveloppe, il faut que celle-ci soit supérieure à la tension de seuil. Ainsi, plus cette tension de seuil est grande plus le système perd en précision.
- Le choix de la capacité et de la résistance n'était pas arbitraire et était dicté par la contrainte suivante :

$$\frac{1}{2\pi RC} \leq f$$

où f est la fréquence du signal dont on veut détecter l'enveloppe.

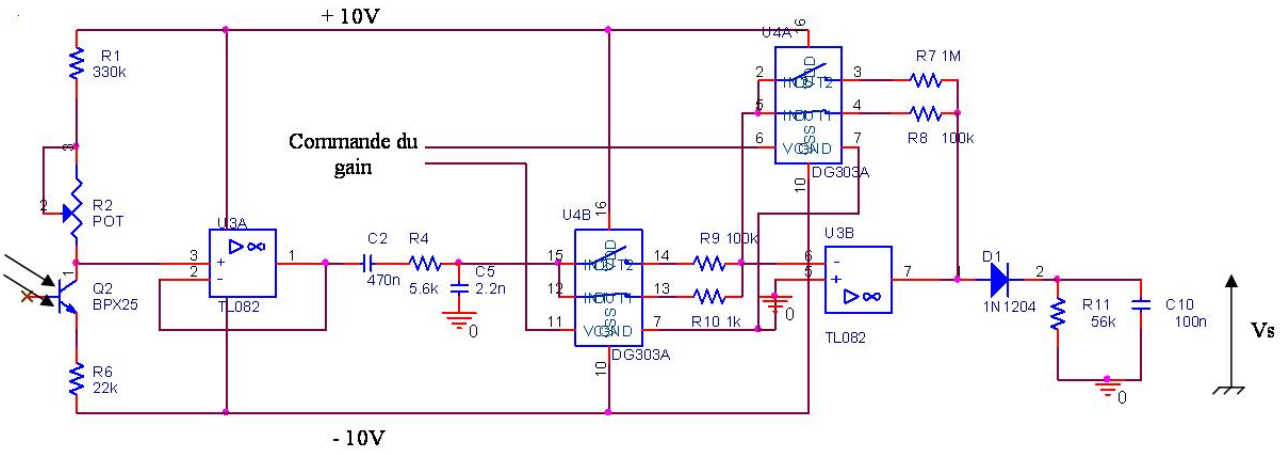


Fig. 6.7 – Schéma complet analogique de la détection de balise

f - Le CAN et le microcontrôleur

Pour ce qui est de la partie numérique, on a utilisé le microcontrôleur du robot (un 80C552) ainsi qu'un des CAN qu'il possède de base. On ne présentera donc que la structure du programme permettant d'effectuer le traitement numérique mais cela sera fait dans la partie III. Au final en sortie du détecteur d'enveloppe nous avons un signal analogique qui varie en fonction de la position du robot par rapport à la balise, ce signal est ensuite convertie en signal numérique grâce au CAN et le microcontrôleur pourra l'utiliser dans le programme de détection de la balise.

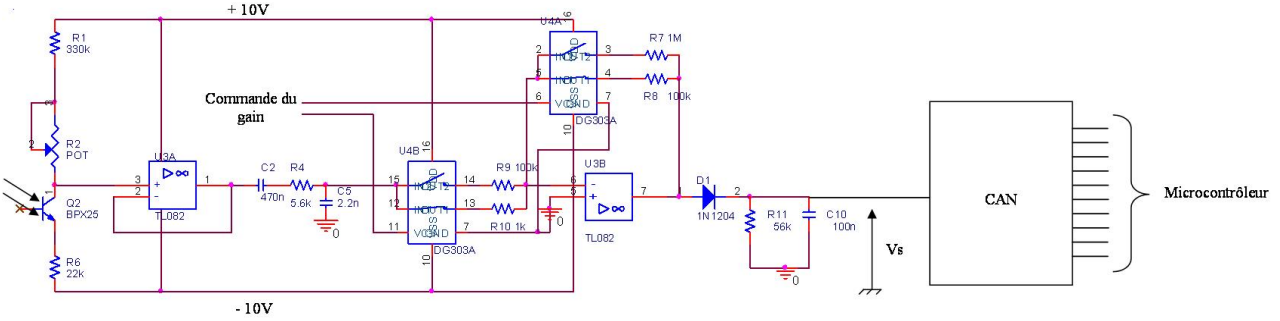


Fig. 6.8 – Schéma complet de la détection de balise

Nous avons réalisé ce montage sur un circuit imprimé. voici le positionnement des composants sur celui-ci ainsi que le typon réalisé :

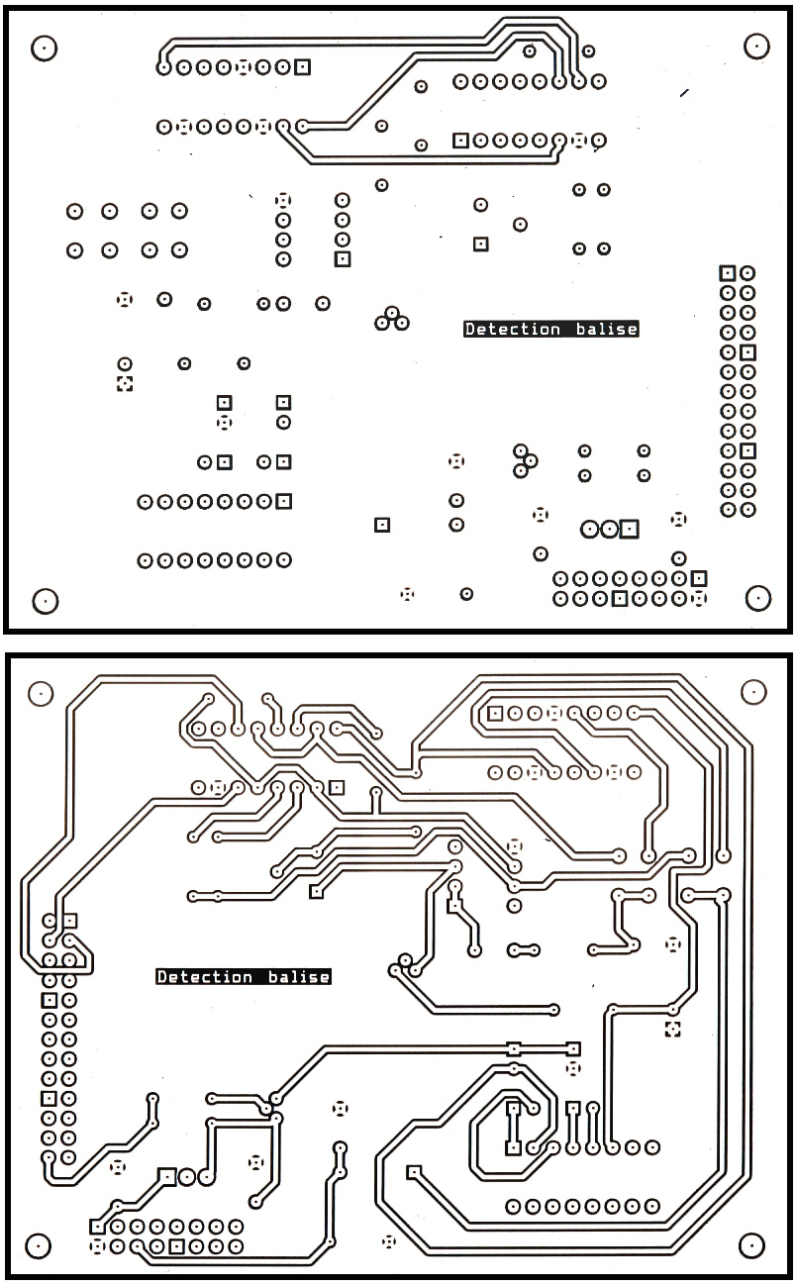


Fig. 6.9 – Typon de la carte destinée à la détection de balise

6.3 Programme de détection de la balise

Dans cette partie nous allons voir le programme réalisé par Loïc SIMON et Guillaume HERAULT. Le programme est commenté et en Annexe D, vous retrouvez le listing complet des programmes utilisés pour la détection de la balise de rechargement. Tout d'abord expliquons succinctement comment le programme détecte la balise.

La détection de la balise repose donc sur le principe suivant :

Le robot étant équipé d'une carte qui délivre, au CAN du microcontrôleur, une tension image de l'éclairement (provenant d'une direction précise), notre programme est sensé :

- dans un premier temps, faire tourner le robot (ceci tout comme les autres commandes de gestion de la carte moteur est le fruit d'un travail antérieur, cela nous a d'ailleurs rendu la tâche plus que difficile : on peut presque parler de retro-engineering).
- simultanément,le microcontrôleur doit lire la valeur convertie par le CAN et détecter l'instant de passage de cette donnée par sa valeur maximale.
- il doit alors immédiatement ordonner au robot d'arrêter sa rotation.
- des lors, le robot est sensé être en direction de la balise, le programme doit alors le faire avancer sur une certaine distance (quelques dizaines de centimètres nous a semblé raisonnable).
- tout ce qui précède doit en fait être réalisé jusqu'à ce que la balise soit atteinte (en fait jusqu'à ce que le robot soit à une distance de moins de trente centimètre de la balise : à cette distance le carte de détection étant non opérationnelle il est facile de détecter cette situation).

Présentons maintenant les différent problèmes que nous avons rencontrés :

– Le problème de l'instabilité de la tension lue sur le CAN :
Lors des premiers test de la fonction detect-max(), nous nous sommes rendus comptes que le programme sortait de cette fonction alors que la valeur de ADCH (celle du CAN) restait constante et même lorsqu'on la faisait diminuer légèrement. La détection de maximum semblait alors bien compromise. Nous n'avons pourtant pas perdu confiance. Et nous nous sommes aperçu que la valeur de ADCH était instable (variation de 10 pour une grandeur codée sur 255 niveaux). Ceci pouvais provenir de deux raisons que sont l'instabilité propre à la tension analogique à convertir et des erreurs de conversion. La première semble tout de même la plus vraisemblable. Ceci étant, quelle qu'en soit la cause la seul façon de parer cette instabilité était de faire une moyenne sur un nombre suffisamment élevé d'échantillons de ADCH (nous avons choisit pour notre part de travailler avec 50 échantillons). Les résultats de cette manipulation ont été plus que probants et ont permis à la fonction detect-max() de fonctionner normalement.

– Le problème du réglage du gain :
La large plage de tensions de détection suivant les distances nous ayant obligé à donner au microcontrôleur le choix d'un calibre sur le système de détection. Le vrai problème que cela a induit était le suivant : après un changement de calibre l'amplitude de ADCH diminuait brutalement, ce qui simulait un passage factice par le maximum. Il a donc fallu faire appel à un système de fonctions imbriquées l'une dans l'autre et ainsi gérer les risques d'un récursivité infinie. Fort heureusement, la condition d'arrêt était évidente et ne laissait aucun doute quant au bon fonctionnement du programme.

– Le problème de l'inertie du robot :
Enfin le dernier problème que nous avions rapidement pressentis fut celui de l'inertie du robot : ce dernier, ayant pourtant détecté la balise avec une bonne précision, dépassait largement la bonne direction (de plus d'une trentaine de degrés sous l'effet de son élan). Nous devions donc compenser cette

erreur d'autant plus que le robot était mal dirigé (en effet, c'est dans ce cas qu'il prend le plus d'élan avant que le maximum ne soit détecté). La première amélioration à laquelle nous avons pensée était d'effectuer une recherche de la balise par tâtonnements : le robot tourne jusqu'à détecter le maximum, il le dépasse sous l'effet de son inertie et avance de 20 à 30 cm. Ceci dure jusqu'à atteindre la balise mais n'était cependant pas assez efficace car le tâtonnement était trop grand (trajectoire très accidentée) et le robot atteignait la balise après un temps trop long (2 mn voire plus).
Nous avons alors pensé a inverser le sens de la rotation à chaque recherche de la direction de la balise. Ainsi le robot prenait moins d'élan à chaque fois. L'amélioration a été assez satisfaisante, cependant une dernière amélioration nous à mener à la solution définitive : On commence par faire deux recherches consécutives de la direction sans avancer et on effectue des rotations pour corriger l'effet de l'inertie, ce qui nous permet de placer le robot dans une direction proche de celle de la balise (moins de 35°). Ainsi on peut relancer la recherche pas à pas présentée précédemment sans que celle ci ne soit laborieuse : en effet le robot étant déjà bien placé il ne prend que peu d'élan et tout se passe très bien.

Voici maintenant les programmes réalisés :

```
/*****************************************************************************
                                    Avril 2005

                                    Loïc SIMON
                                 Guillaume HERAULT
*****************************************************************************/
/*****************************************************************************
programme pricipal de detection de la balise
*****************************************************************************/

/* si on veut executer le programme sur la carte sans tenir compte de la reponse
des LM 629 definir la variable SIMULATION dans def.h */

#include <reg552.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "main.h"
#include "def.h"
#include "variable.h"
#include "conv.h"
#include "serie.h"
#include "utils.h"
#include "robot.h"
#include "can.h"

/*****************************************************************************
variables globales utiles pour la gestion des contrôleurs de moteurs
*****************************************************************************/

u_char tampon[100];
u_char tableau_LM [TAILLE_TABLEAU_COMMANDE];

/*****************************************************************************
programme de détection de la balise
*****************************************************************************/

void detection_balise(void) {

int sens_rotation=1, balise_atteinte=0;

/*****************************************************************************
initialisation relative au contrôleur des moteurs : travail effectué lors d'un
TER précédent
*****************************************************************************/

u_char initialise[] =
{
//        ATTENDRE,
          RESET1,les_2_LM,0x00,
               MSKI,les_2_LM,0x02,0x00,0x04,
          RSTI,les_2_LM,0x02,0x00,0x00,
          RESET2,les_2_LM,0x00,
          RSTI,les_2_LM,0x02,0x00,0x00,
          LFIL,les_2_LM,0x0A,0x02,0x0F,KP_1,KI_1,KD_1,IL_1,
          UDF,les_2_LM,0x00,
          FIN
     };

SP = 0x2F;

nouvelle_commande=1;

#ifndef SIMULATION
     pilote(initialise);
#endif
```

```
/*****************************************************************************
Nos initialisations
*****************************************************************************/
init_detection();
alimenterplaquette();

/*****************************************************************************

Phase d'approche pas à pas de la balise

*****************************************************************************/
while(balise_atteinte == 0 )
     {
     maximum = 0;

/*****************************************************************************
commande des moteurs en pivotement : on commande au robot de tourner plus de
trois tours mais on l'arrêtera dès qu'on aura détecté le maximum
*****************************************************************************/
     sens_rotation*=-1;
     (sens_rotation==1)?strcpy(tampon,"P,0.25,1000,F"):strcpy(tampon,"P,0.25,-
     1000,F");

/*on change de sens de rotation à chaque fois à cause de l'inertie du robot : en
effet on dépasse à chaque fois la bonne direction, il est donc plus judicieux de
repartir en arrière*/

     traduction();
     cv_tab(tableau_LM);
     navigation(tableau_LM);
/*****************************************************************************
Détection du maximum
*****************************************************************************/
          detect_max();

/* arrivé ici on a soit détecté le maximum soit on est assez proche de la balise
pour pouvoir considérer qu'on y est arrivé*/

     if(fintrajectoire(les_2_LM)!=0x04)
/*si cette condition est fausse on a pas pu détecté le maximum et donc on
considère qu'on a atteint la balise*/

          {


          for(k=0;k<0x0fff;k++){}
/*cette boucle d'attente est une sécurite qui nous permet d'être sûr d'avoir au
moins légèrement dépassé le maximum : ainsi lors de la prochaine rotation qui
aura lieu en sens inverse, le robot atteindra rapidement le maximum sans avoir
pris de l'inertie*/

/*****************************************************************************
stopper la rotation
*****************************************************************************/
          strcpy(tampon,"S,F");
          traduction();
          cv_tab(tableau_LM);
          navigation(tableau_LM);
          while(fintrajectoire(les_2_LM)!=0x04){};
```

```
/
*****************************************************************************
Correction de l'inertie lors des deux premiers passages, on ne fait pas avancer
le robot
*****************************************************************************
/
          if(decalage <= 1)
               {
               (sens_rotation==-1)?strcpy(tampon,"P,0.25,35,F"):strcpy
          (tampon,"P,0.25,-35,F");
               decalage = decalage +1;
               traduction();
               cv_tab(tableau_LM);
               navigation(tableau_LM);
               while(fintrajectoire(les_2_LM)!=0x04){};

               }
/
*****************************************************************************
 Par la suite on ne corrige plus le dépassement car le robot ayant peut d'élan
ce dépassement est négligeable. On peut alors faire approcher le robot de la
balise.
*****************************************************************************
/
          else
               {
               strcpy(tampon,"L,0.25,0.2,F");
               traduction();
               cv_tab(tableau_LM);
               navigation(tableau_LM);
               while(fintrajectoire(les_2_LM)!=0x04){};
               }

          }

     else balise_atteinte = 1;
     }
/* si tout c'est bien passé on a atteint la balise on peut donc couper la
plaquette */

couperplaquette();

/
*****************************************************************************
Boucle d'attente
*****************************************************************************
/
while(1)
     {
     }
}
```

```
/*****************************************************************************/

/**************                          CAN                      **************/

/*****************************************************************************/



#include <reg552.h>
#include "can.h"


/* Mes variables globales */

int maximum,valeurADCH;
char Gain;


sbit P3_4= 0xB4;
sbit P3_5= 0xB5;   //fin mes variables globales




/*****************************************************************************
Fonction d'initialisation pour le programme de détecion
*****************************************************************************/
void init_detection()
{

     PWMP = 0xFF;  //initPWM
     init_can();
     maximum = 0;
     valeurADCH = 0;

     Gain = 1;
     P3_4 = (Gain&0x01);
     P3_5 = !((Gain&0x02)>>1);
}

/*****************************************************************************
Fonctions permettant de mettre sous tension ou non l'unité de détection de la
balise
*****************************************************************************/

void alimenterplaquette()
{
     PWM0 = 0xFF;
}

void couperplaquette()
{
     PWM0 = 0;
}
```

```
/*****************************************************************************
Diverses fonctions élémentaires
*****************************************************************************/

void init_can()
{
     ADCON=0x00;       //initialisation du can
//   EAD=1;          //validation de l'interruption du can
}

void lancerconversion()
{
     ADCON = startofconv;
}

void attentefinconv()
{
     while((ADCON & endofconv) != 0x10){}
}

/*****************************************************************************
Fonction de mise à jour de la valeur du maximum d'éclairement
*****************************************************************************/

void traitement()
{
     lecture_can();
     if((valeurADCH>maximum))
     {

          maximum = valeurADCH;

     }
}

/*****************************************************************************
Fonction gérant la lecture de ADCH0 ainsi que le réglage du gain le cas échéant
*****************************************************************************/

int lecture_can()
{
     int i;
     lancerconversion();
     attentefinconv();
     valeurADCH = ADCH;
     if((valeurADCH == 255)&(Gain!=4))
     {
          Gain = (Gain + 1);
          P3_4 = (Gain&0x01);
          P3_5 = !((Gain&0x02)>>1);

          maximum = 0;
          detect_max();
          return 1;

     }
     for(i=0;i<IMAX;i++)
     {
          lancerconversion();
          attentefinconv();
          valeurADCH += ADCH;
     }
     valeurADCH /= IMAX;
     return 0;
}
```

```
/*****************************************************************************
                Fonction de détection du maximum d'éclairement
   on ne sort de cette fonction qu'après avoir détecté une phase de montée de
 l'éclairement puis une phase de diminution, ce qui caractérise bien le passage
  par un maximum. Toutefois si aucun éclairement n'a été détecté on sort quand
           même et on considère que le robot est arrivé a la balise
*****************************************************************************/



void detect_max()
{

     unsigned int i,j;

     traitement();

     //petite boucle d'attente

     for(i=0;i<=0x03ff;i++){for(j=0;j<=0x00ff;j++);}

     do
     {
          lecture_can();
     }while((maximum > valeurADCH + 1)&(fintrajectoire(les_2_LM)!=0x04));
/*on sort de cette première boucle en phase d'éclairement croissant ou si la
valeur lue sur le CAN reste nulle indéfiniment (en fait pendant le temps que le
robot éffectue une rotation de 1000 degrès*/

     do
     {
          traitement();
     }while((maximum <= valeurADCH)&(fintrajectoire(les_2_LM)!=0x04));
/*on sort de cette deuxième boucle dès que l'éclairement se remet a decroître ou
si la valeur lue sur le CAN reste nulle indéfiniment*/
}



/*Attention il est impératif de remarquer que les deux dernières fonctions sont
en fait « imbriquées l'une dans l'autre » ce qui induit une récursivité
complexe. Il a donc fallu faire particulièrement attention à la condition
d'arrêt de cette récursion; en fait cette condition d'arrêt se trouve dans la
fonction lecture_can() : if((valeurADCH == 255)&(gain!=4)).
En fait, il est évident à cause de la condition gain!=4 que l'on ne rentrera
dans la récursion plus de quatre fois.*/
```

Troisième partie

Essais et Performances

Afin de valider nos deux montages, nous avons réalisés une série d'essais pour les deux parties.

Voyons donc dans cette dernière partie ces différents essais, tout d'abord pour le système de contrôle de l'énergie puis pour la partie de la détection de la balise de rechargement.

Chapitre 7

Système de contrôle de l'énergie

Dans ce chapitre nous parlerons du débogage, des essais et de la finalisation du circuit imprimé du système de contrôle de l'énergie.

Comme il l'a été rappelé dans l'introduction du chapitre 5, notre premier montage a été réalisé à partir de composants discrets et non autour d'un PSoC.

Dans un premier temps, nous parlerons donc de ce premier montage, qui n'a par la suite plus été utilisé. Puis dans un deuxième temps du second montage basé sur un PSoC.

7.1 Premier test

Nous avons donc réalisé un circuit (dont le schéma fonctionnel est rappelé en figure 8.3) sur une platine d'essai.

Nous avons obtenu la réalisation suivante :

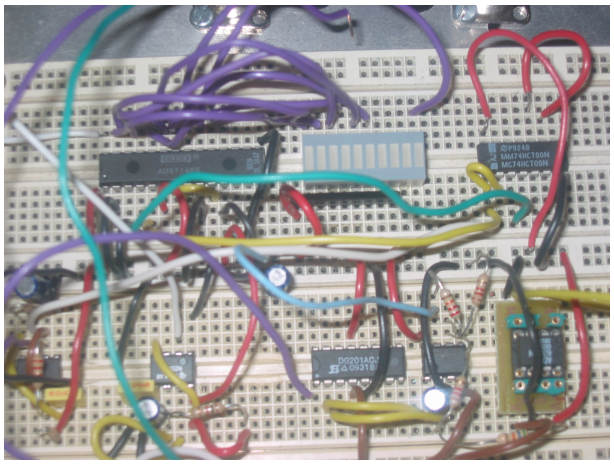
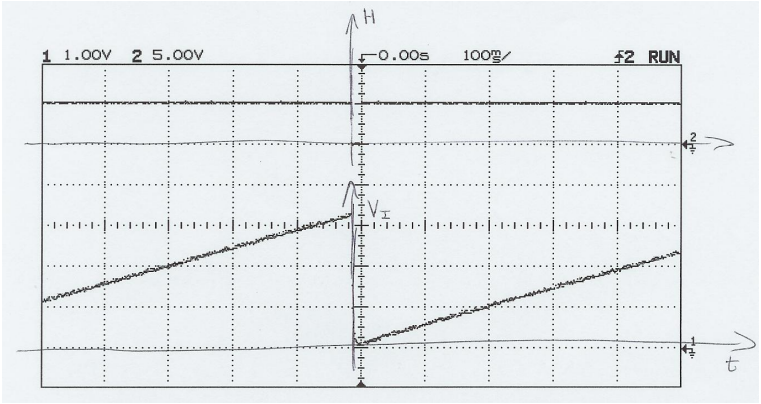


Fig. 7.1 – Montage réalisé sur une platine d'essai

Nous avons alors visualisé les signaux en sortie de l'intégrateur. Nous avons obtenu les signaux suivants :



Cela correspond bien à des courbes d'un intégrateur, dont la pente varie en fonction des tensions mises en entrée du multiplieur.

Afin de remettre à zéro cet intégrateur, nous avons inséré un interrupteur

analogique. Cependant le signal de commande de cet interrupteur qui provenait du CAN, avait un temps à l'état haut trop faible pour être vu par l'interrupteur.

L'intégrateur ne se remettait donc pas à zéro. Et même en rajoutant un monostable, ce dernier ne se déclenchait pas.

Si nous avions poursuivi dans cette voie, il aurait fallu traiter ce signal de conversion à l'aide du microcontrôleur qui a priori était le seul capable de voir le changement d'état sur la patte du CAN.

Voyons à présent le montage qui a été finalement réalisé, autour d'un microcontrôleur.

7.2 Réalisation finale

Avant de programmer le PSoC avec le programme final (présent en **Annexe B**), nous avons fait des essais de ce programme à l'aide de l'émulateur sur une platine d'essai.

Nous avons débogué le programme pas à pas et étape par étape.

Nous avons d'abord eu des problèmes avec la fréquence de l'horloge interne du PSoC et la fréquence d'échantillonnage des CAN. Mais après lecture plus approfondie des "datasheet", nous avons pu résoudre ce problème.

Puis, l'une des boucles principales, celle de décharge, faussait les données recueillies sur les deux CAN. Cela était du aux types de variables utilisées pour stocker toutes ces données, et le fait que les valeurs étaient signées ou non par défaut.

Après avoir effectué les modifications adéquates, il a fallu encore modifier quelques lignes de programme, mais dans l'ensemble, tout fonctionnait comme on le souhaitait.

Nous sommes alors passés à la réalisation du typon et du circuit imprimé, et

parallèlement nous avons chargé notre programme dans l'un des PSoC qui nous était fourni.

Le premier test du circuit imprimé final fût un échec total. Le PSoC n'arrivait pas à s'initialiser correctement, les tensions représentatives du courant et de la tension aux bornes de la batterie étaient totalement différentes de celles souhaitées.

Après un bref contrôle du schéma électrique et du typon, nous nous sommes aperçus que nous avions fait une erreur sur une partie du montage dans le schéma électrique, ce qui avait pour conséquence de rendre une tension négative (au lieu de positive) en entrée du PSoC, d'où le problème d'initialisation de ce dernier.

En effet, durant les phases de débogage à l'aide de l'émulateur puis du PSoC, nous nous sommes rendus compte que la présence d'une tension négative sur l'une des broches du PSoC l'empêchait de s'initialiser.

Après avoir modifié le circuit imprimé en conséquence, il semblait fonctionner comme le montage d'essai que nous avions réalisé.

Nous avons pu le tester dans un premier temps en remplaçant la batterie par une source de tension (alimentation stabilisée) et la charge par une résistance d'environ 25 Ω qui permettait de se rapprocher au mieux des valeurs réelles de consommation du robot (soit environ 0,5A sous 12 V).

Nous avons ensuite fait le test avec la batterie au lithium-polymère et la même charge. Avec une batterie chargée au maximum, nous avons atteint un temps d'utilisation supérieur à 1h.

L'étape suivante aurait été de tester notre montage dans les conditions réelles de fonctionnement, c'est à dire, embarqué sur le robot. Mais les temps de charge des batteries importants et un petit problème sur les deux seuls robots fonctionnant encore dans les locaux, ne nous ont pas permis de le faire.

7.3 Performances

Après avoir réalisé cette série de tests, nous avons pu mesurer les performances et capacités de notre montage.

Pour ce qui est de la consommation du montage final, on arrive à environ 30-35 mA, ce qui par rapport à la consommation totale du robot est faible (0,4 A en moyenne).

On peut ainsi dire que notre montage satisfait le cahier des charges sur ce plan.

Pour la partie de traitement, nous avons dû faire des choix de types de variables.

Pour l'énergie par exemple, nous avons pris un type de variable long et signée, qui nous permet d'aller jusqu'à une valeur de 2^{16} pour la partie positive, et avec les calculs de que nous effectuons pour calculer l'énergie, le calculateur peut fonctionner environ 3h30 à courant maximal et tension nominale (voir calcul dans la section 5.2) sans rencontrer de problème de dépassement.

Ceci nous laisse donc une grande marge, puisque la batterie au lithium-polymère tient 2h dans une telle configuration.

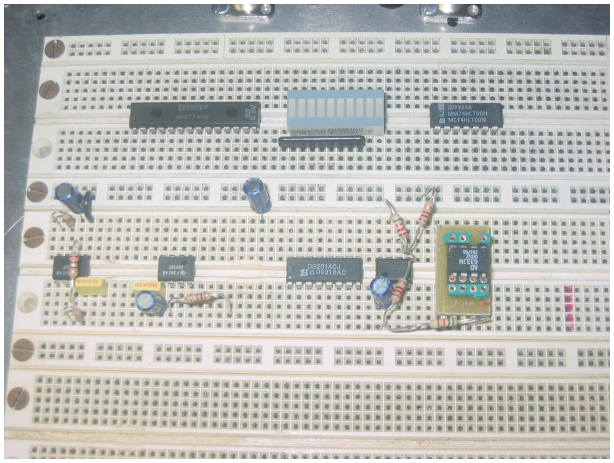
De plus, de part le fait que les calculs d'énergie soient fait logiciellement, on peut facilement modifier les sources du programme afin de rendre compatible ce circuit avec n'importe qu'elle type de batterie.

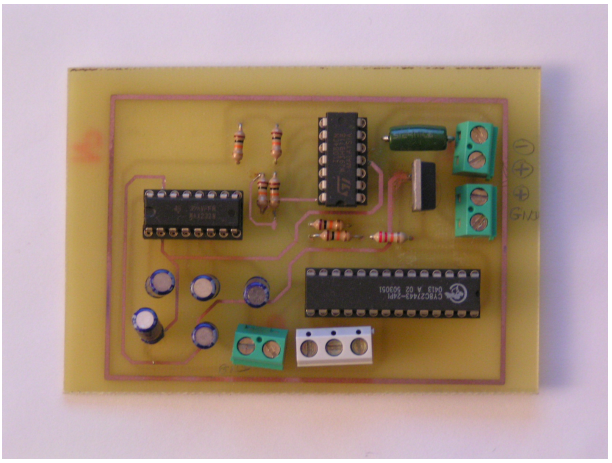
Enfin, afin que notre montage puisse être adaptable à n'importe quel robot, il fallait qu'il soit à la fois compact et autonome.

Grâce au microcontrôleur qu'il embarque, ce module de calcul d'énergie est autonome d'un point de vue calcul.

Pour la compacité, nous avons comparé le montage final à celui que nous avions prévu initialement.

Voici le résultat :





On peut voir à gauche le premier circuit réalisé à partir de composant discret (multiplieur, monostable, interrupteur analogique, CAN...) une fois décablé afin de voir les composants utilisés et à droite le circuit imprimé final.

Sur la photo de gauche on peut compter jusqu'à 7 circuits intégrés, alors que sur le circuit final, il n'y en a que 3.

Le circuit final est donc plus petit et compact que le premier testé.

Notre circuit respecte donc les critères que nous avions défini dans le cahier des charges en début de ce rapport.

Chapitre 8

Système de détection de la balise de rechargement

8.1 Les essais avec la maquette

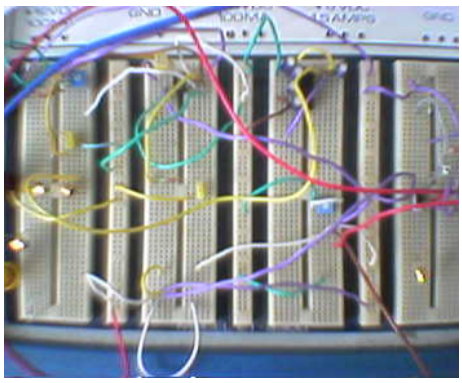


Fig. 8.1 – Plaque de test

Dans un premier temps nous avons fait des essais sur une plaque de test afin de tester la diode émettrice et le phototransistor. Nous avons polarisé le phototransistor de deux manière différentes :

– une avec polarisation de la base

– une autre sans polarisation de la base

Ainsi nous avons remarqué que la polarisation de la base n'était pas nécessaire voire dégradait le signal. D'où le choix de ne pas polariser la base du phototransistor.

Nous avons également déterminé la valeur de la tension continue qu'il fallait appliquer aux bornes de la diode émettrice afin que celle-ci soit toujours passante. Cette tension est de l'ordre de 2 V.

Une fois le photo transistor polarisé et la diode émettant en continu nous avons pu faire des tests avec une distance entre la diode et le phototransistor de 20cm environ. Ces tests nous on permis de constituer le système de détection de balise, c'est-à-dire le besoin de suiveur, d'amplificateur à gain commandable, de filtre et d'alimentation symétrique. Une fois tous ces composants mis en place nous nous sommes occupés de la commande de l'alimentation à l'aide du transistor. Le choix de la résistance de base fut délicat car le courant de la base avait tendance à être trop élevé. Ensuite nous avons fait varier la distance entre la diode émettrice et le phototransistor, ainsi nous avons déterminés les distances extrémales de détection :

- Une distance maximale pour une raison simple : lorsque la balise est éloignée de quelques mètres (3 environ) le signal détecté par le phototransistor a une amplitude du même ordre de grandeur (environs 5 mV) que les bruits d'alimentation. Le rapport signal sur bruit est donc en quelque sorte trop faible. Après étude des signaux en sortie de l'amplificateur on a déduit que le "RSB" minimum (au niveau du collecteur du phototransistor) pour lequel le signal de sortie du système correspond bien au signal informatif est de l'ordre de deux. C'est à dire qu'à partir d'un signal au collecteur d'une amplitude de 10mV le signal de sortie a la même forme (sinusoïde dans notre cas) que le signal d'émission.
- La raison d'une distance minimale de détection ne nous ait apparue qu'au cours de nos expériences : En effet pour la comprendre il faut se rappeler comment est polarisée la diode émettrice et en particulier se souvenir que pour que la diode reste dans un état passant il faut lui imposer une tension de polarisation supérieure à sa tension de seuil sommée à l'amplitude du "petit signal" à émettre. De ce fait la diode émet aussi une composante continue. Fort heureusement cette composante continue s'atténue bien plus rapidement que la composante informative (ie. "petit signal"). Par contre,

pour de courtes distances (de l'ordre de la quinzaine de centimètres) cette composante continue est encore assez importante et modifie la polarisation du transistor. Elle peut même entraîner une saturation du suiveur et rendre ainsi le système inopérant. La distance minimale de détection reste cependant faible (15 cm) et dans ce cas on peut considérer que le robot arrivera sans encombre à la balise.

Finalement, nous avons fait fonctionner le gain contrôlable, ce qui n'a pas posé de problème.

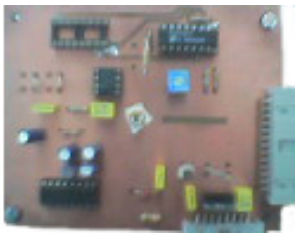


Fig. 8.2 – Carte réalisée

Une fois le montage fonctionnant sur la plaque d'essai nous avons commencé à faire le typon et en parallèle les programmes dédiés à la détection de la balise de rechargement.

8.2 Les essais avec le robot

Fig. 8.3 – Carte montée sur le robot

Après quelques problèmes de réalisation du circuit imprimé (piste manquante, réglage final du choix des résistances) nous avons inséré notre carte sur le robot et ainsi nous avons pu faire des essais avec les programmes réalisés (voir Annexe D).
Lors des essais que nous avons faits l'alimentation de la carte se faisait très bien. Dans un premier temps, nous avons essayé nos programmes sans faire tourner le robot, grâce au debugger nous visualisions l'entrée et la sortie du CAN, c'est ainsi que nous avons remarqués qu'il fallait faire une moyenne du signal entrant dans le CAN afin que les mesures soient les plus précises possible. Nous avons vérifié que lorsque le signal émis pas la diode émettrice variait le programme se déroulait correctement, ensuite nous avons vérifié que le contrôle du gain se faisait aux moments voulus et dans l'ordre voulu. Une fois ces tests réalisés il ne restait plus qu'à voir si le robot s'arrêtait bien au moment de la détection du maximum.

Donc dans un deuxième temps nous avons fait tourner le robot afin de voir si le robot s'arrêtait bien en face de la balise. De gros problèmes de décharge de batterie se sont posées ce qui ne nous permettait pas d'utiliser les contrôleurs de moteur pendant une longue durée.

Lors de ces essais nous avons dû nous plonger dans les programmes déjà fait afin de comprendre comment le programme faisait pour mettre le robot en rotation. Nous ne savions pas si il y avait besoin d'interruptions ou non. Fort heureusement pour nous, le microcontrôleur était disponible pendant la rotation du robot, ainsi les interruptions étaient inutiles. Une fois les programmes liés à la rotation du robot compris nous avons pu faire des essais avec le robot en rotation. Lors de ces essais nous avons remarqués que si la rotation du robot est trop rapide alors la détection ne peut pas se faire, nous avons donc choisi 0,25 m/s.

Ensuite nous avons vu que le robot avait une inertie, il se passe un certain entre la détection effective du maximum et l'arrêt du robot, nous avions donc un décalage angulaire entre la balise et l'axe du robot. Pour remédier à ce problème nous lui avons fait faire une rotation de 35° dans le sens approprié juste après la première détection du maximum, ensuite les problèmes liés à l'inertie n'y sont plus car le robot est quasiment en face de la balise donc il n'a pas le temps d'avoir de l'inertie.

Le problème venait maintenant du fait que si le robot était dès le départ en face de la balise il y aurait quand même cette rotation de 35° ce qui ne résoudrait pas le problème, pour cela nous avon mis une rotation de -35° après la deuxième détection de maximum, ainsi le robot se dirige vers la balise quelque soit sa position initiale. Ensuite nous avons dû, pour des raisons de stabilité de la détection, ajouter une boucle d'attente après la détection du maximum. C'est-à-dire que le robot va mettre plus longtemps avant de s'arrêter. De cette manière le robot passera bien devant la balise en dépassant le maximum de manière certaine.

8.3 Performances

Après ces quelques essais nous avons estimés les performances de notre carte de détection.

La détection de la balise sur le robot fonctionne de 30 cm à 3 m comme sur la maquette, au delà de 3 m le signal qui arrive sur le phototransistor est trop faible pour être détecté et en deçà de 30 cm le montage ne fonctionne plus (voir la section 8.1 les essais avec la maquette). Cela était prévu et remplit le cahier des charges.

Durant les essais, les batteries du robot se déchargeaient en 20 mn comme avant la mise en place de la carte de détection de balise, on peut donc dire que la carte de détection ne consomme rien par rapport à la carte de commande des moteurs. Notre carte est donc conforme au cahier des charges pour ce point.

A une distance de 3 m, le robot met entre 45 s et 1 mn pour arriver à la balise, ce qui est approprié à l'utilisation que l'on veux en faire. Et ce résultat entre bien dans le cahier des charges.

Notre montage respecte bien le cahier des charges défini au début de ce document.

Conclusion

A travers ce TER, nous devions réaliser un circuit à la fois capable de calculer l'énergie emmagasinée dans les batteries d'un robot mobile, destiné au "jeu" avec des enfants autistes, ainsi que de détecter et de se rendre vers une balise de rechargement une fois l'énergie stockée trop faible pour continuer de "jouer".

Ce système, pouvant être embarqué sur n'importe quel type de robot et avec n'importe quel type de batterie, devait à la fois : consommer peu d'énergie, être compatible avec au moins les deux types de batterie utilisée (Plomb et Lithium-polymère), pouvoir être embarqué sur n'importe quel robot utilisé et pouvoir être alimenté en 12 V.

Après avoir effectué des recherches sur les différentes façons de contrôler l'énergie sur une batterie et les différents moyens de détecter une balise, nous nous sommes partagés le travail : un binôme s'est penché sur le problème du calcul de l'énergie et un autre binôme sur la détection de la balise de rechargement.

Chacun des deux binômes a alors décidé d'une structure à réaliser permettant de solutionner les problèmes à traiter.

Pour la partie "calcul de l'énergie embarquée", nous avons choisi un montage basé sur un microcontrôleur intégrant à la fois de l'analogique et du numérique, ce qui nous a permis notamment de rendre le circuit final plus compact, moins "gourmand" en énergie et totalement autonome. Ce circuit pouvant être utilisé lors de l'utilisation ou de la charge de la batterie, il permet de connaître à tout moment l'énergie stockée dans la batterie. Et lorsque cette énergie descend en dessous d'un certain seuil, un signal (une interruption) est envoyé au microcontrô-

leur afin qu'il lance la procédure de recherche développée par le deuxième binôme.

Pour la partie "détection de la balise de rechargement", nous avons choisi une détection par infrarouge, ce qui nous a permis d'être précis dans la détection, rapide pour arriver à la balise et surtout nous a permis d'économiser l'énergie par rapport à une détection ultrasons ou autre. Les composants utilisés sont peu couteux (total de moins de 10€) donc tous les robots peuvent en être équipés. Sur la carte réalisée, nous avons prévu d'utiliser un circuit PGA201 à la place du montage AO + résistances qui sert de gain. Ce circuit est un amplificateur à gain programmable. Plus tard l'AO + les résistances pourront être remplacés par ce circuit afin de gagner de la place et de faire des économies d'énergie. Notre montage utilise le microcontrôleur présent sur le robot ce qui permet de gagner de la place, mais c'est aussi plus pratique afin de récupérer l'interruption venant de la gestion de l'énergie embarquée.

Au final, nous obtenons alors un circuit qui satisfait les exigences du cahier des charges de départ. Ce système final :

- consomme peu par rapport à la consommation du robot (environ 50 mA) ;
- est compact et prend donc peu de place sur le robot ;
- peut être adapté sur tous les robots qui sont utilisés et avec l'une ou l'autre des batteries disponibles (Plomb ou Lithium-polymère) ;
- fonctionne lorsque le robot évolue dans un milieu dégagé ;
- se dirige vers la balise repérée à une distance maximum de 3m ;
- l'arrivée à la balise est assez rapide ($<$ 1mn) ;
- peu coûteux (moins de 25 € pour le système complet).

Grâce à l'intégration des caractéristiques des batteries et des calculs de l'énergie dans le programme principal du "calculateur d'énergie", le système est en plus adaptable à n'importe quel type de batterie.

Et de part son autonomie, le "calculateur" seul peut-être utilisé sur n'importe quel type de circuit embarqué où le contrôle de l'énergie est nécessaire et de plus il gère les interruptions.

Ces deux semaines de réalisation nous aurons donc permis de mener à terme les objectifs que nous nous étions fixés au départ.

Nous avons également appris à travailler en équipe et à réaliser un projet dans un temps limité. Nous avons aussi réalisé ce projet autour d'un tout nouveau composant, ce qui nous a permis d'en voir les différents aspects et les problèmes qui restent liés à ce type de microcontrôleur. Nous pourrons ainsi faire bénéficier de notre expérience à d'autres développeurs dans ce domaine.

De plus pour faire ce rapport nous avons appris à utiliser LaTeX, ce qui est fort utile lorsque nous aurons à rédiger d'autres rapports ou notre thèse par exemple.

Remerciements

Que ce soit pour l'aide qu'ils nous ont apportée durant notre recherche ou durant les deux semaines de réalisation de nos maquettes, nous tenons à remercier les personnes suivantes :

- **Gilbert PRADEL**
- **Pascal VAROQUI**

  nos encadrants.

- **Lionel CIMA** - *Laboratoire SATIE*

  pour sa démonstration du logiciel de développement des PSoC ;

  pour sa disponibilité et son aide lors de la prise en main du kit de développement des PSoC.

- **Christophe** - *Technicien au département EEA*
- **Stéphane** - *Ingénieur au département EEA*

Guillaume HERAULT
Loïc SIMON
Vincent THOMAS
Julien VILLEMEJANE

Bibliographie

[1] G. Pradel
Robot

Batteries au plomb

[2] A. Voir
"Le monde des accus et des batteries rechargeables"
`http ://www.ni-cd.net`

[3] Matagne
"Les batteries au plomb"
`http ://www.lei.ucl.ac.be/ matagne/`

[4] P. Hills,
"Battery Monitoring"
`http ://homepages.which.net/ paul.hills/`

Batteries au polymère

[5] P. Topart et J.-Y. Poinso
"Vers de nouveaux électrolytes de batteries"
`http ://www.cea.fr/fr/Publications/clefs44/clefs4472.html`

[6] Klaudius
"Risques lors de la charge de batterie polymère"
`http ://klaudius.free.fr/lipo.htm`

ACPI

[7] Klaudius
"ACPI"
http ://www.acpi.info/

Ultrasons / Infrarouges

[8] Fribotte
"Distance IR"
http ://fribotte.free.fr/bdtech/distanceIR/distanceIr.html

[9] "Ir"
http ://www.boondog.com/CtutorialsCIrirCmc145026.htm

Constructeurs

[10] **Analog Device**
7805, 78LXX
http ://www.analog.com/

[11] **Dallas Semiconductor**
ICM 7555, DS2751
http ://www.dallassemiconductor.com/

[12] **Kodenshi**
OPE5594A
http ://www.kodenshi.com/

[13] **Maxim**
MAX 232
http ://www.maxim-ic.com/

[14] **Microchip**
MCP79841
http ://www.microchip.com/

[15] **Motorola**
MC145026, MC145027
http ://www.motorola.fr/

[16] **National Semiconductor**
LM3914, LM 3916
http ://www.national.com/

[17] **Sharp**
GPU1U26X
http ://www.sharp.fr//

[18] **Texas Instrument**
CD4011
http ://www.ti.com/

Cypress MicroSystems

[19] **PSoC**
CY8C27443, CY8C27243
http ://www.cypressmicro.com/

Annexes

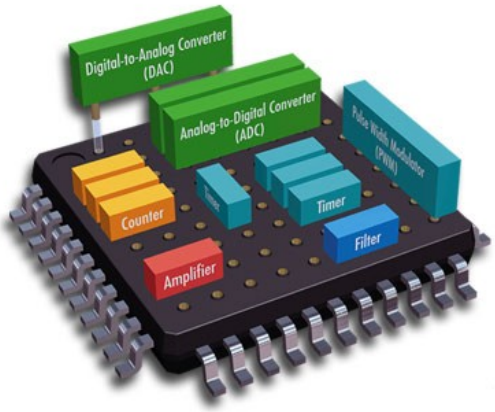
Annexe A
Prise en main des PSoC

Les PSoC, abréviation de *Programmable System-on-Chip*, sont de nouveaux CPLD développés par **Cypress MicroSystems** (19).

CYPRESS MICROSYSTEMS

Ils intègrent une partie numérique composée d'un CPU, d'espace mémoire ROM, RAM, de ports d'entrée/sortie configurables à volonté. Jusque là rien de nouveau par rapport au CPLD ou microcontrôleur classique.

La nouveauté est dans le fait qu'ils intègrent aussi une partie "analogique" composée d'une douzaine de blocs à amplificateurs opérationnels dont il est possible de choisir la fonction dans un catalogue d'une dizaine de fonctions différentes, actuellement.



L'association de fonctions "analogiques" et "numériques" sur un même circuit de silicium fait de *Cypress MicroSystems* un pionnier dans ce domaine.

Voyons plus en détails l'architecture et les fonctions réunies dans un tel circuit.

Structure globale d'un PSoC

Voici l'architecture globale d'un PSoC.

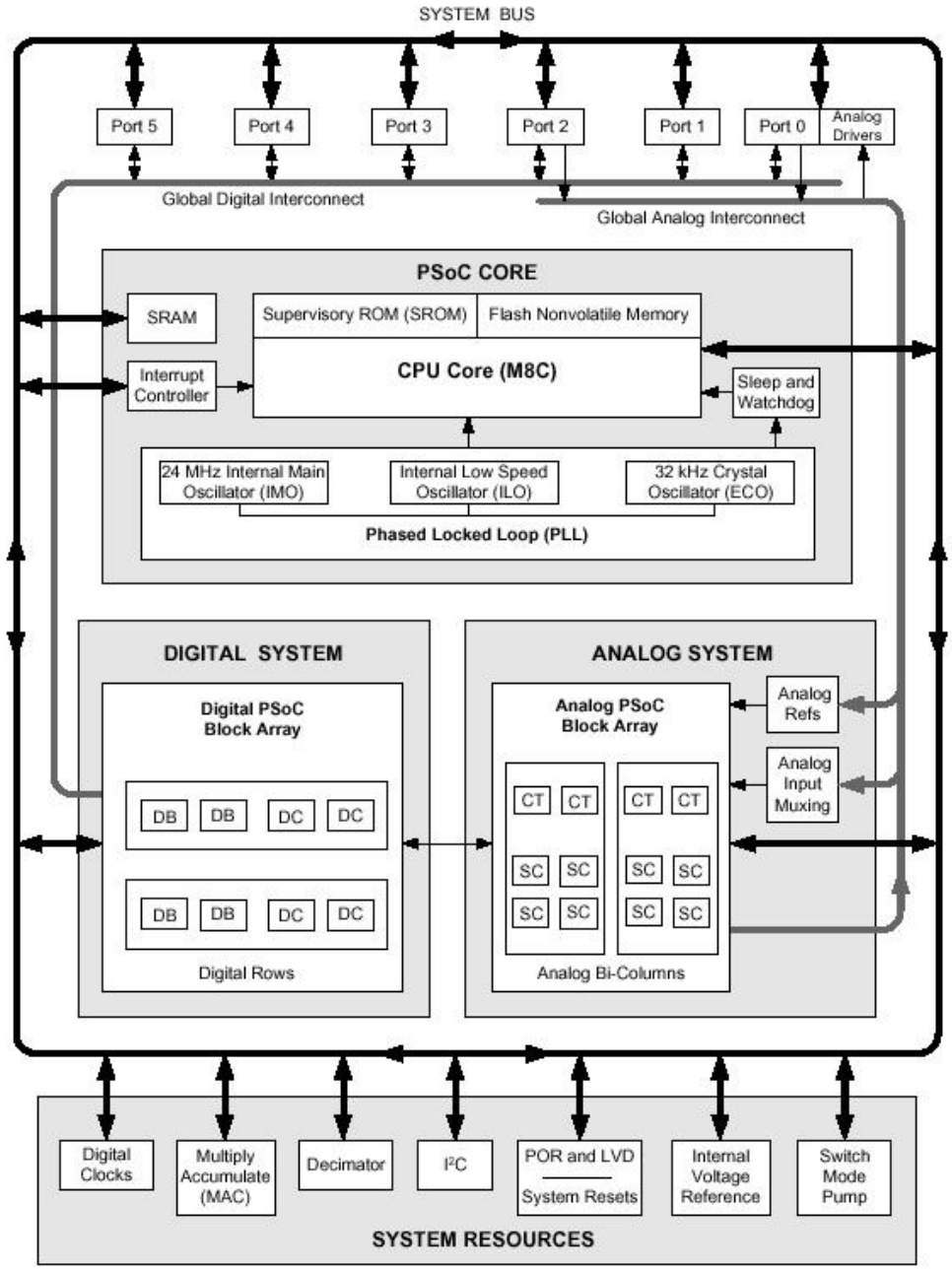


Fig. 8.4 – Architecture globale d'un PSoC

On peut distinguer 4 grands blocs sur un PSoC (voir schéma 8.4) :
- le noyau du PSoC qui regroupe le CPU, la ROM, la RAM et des oscillateurs internes ;
- un bloc “numérique” de 8 sous blocs assignables et configurables ;
- un bloc “analogique” de 12 sous blocs assignables et configurables ;
- un bloc de ressources systèmes (horloges, I2C...) ;
- et jusqu'à six ports d'entrée/sortie analogiques et numériques selon le type de PSoC utilisé.

Ce circuit intègre un oscillateur interne fonctionnant à la fréquence de 24 MHz, qui correspond à la vitesse de calcul du CPU, ce qui est une vitesse correcte pour le développement d'applications temps réel.

Il possède de 2 à 16 Ko de mémoire ROM, utilisable pour l'implantation du programme et 256 octets de RAM, utilisée pour stocker les variables définies dans le programme, ce qui reste un peu juste pour certaines applications.

Un accès en écriture est aussi possible en FlashROM afin de sauvegarder certaines données recueillies de l'extérieur ou calculées par le programme. Cela permet notamment de conserver certaines valeurs en mémoire après une coupure d'alimentation.

Chaque entrée/sortie est configurable.

Certaines sont cependant réservées à la partie numérique et ne peuvent pas être utilisées comme entrée ou sortie analogique, L'inverse étant possible.

Selon aussi le type de boîtier utilisé, il y a plus ou moins d'entrées/sorties analogique et numériques.

Fonctions principales des PSoC

Comme on l'a vu dans la section précédente, les PSoC possèdent 8 blocs numériques et 12 blocs analogiques.

On peut choisir de configurer chacun de ces blocs quasi-indépendants selon une des cinquantaines de fonctions disponibles, qu'elle soit analogique ou numérique.

Chacun des douze blocs "analogiques", par exemple, est configurable : en amplificateur (inverseur, non inverseur...), en étage d'entrée de CAN par exemple ou de sortie de CNA, en filtre passe-bas ou passe-bande.

Les huit blocs "numériques", quant à eux, peuvent être configurés en CAN, en compteurs, en maître de bus I2C, en contrôleur LCD et d'autres fonctions encore.

Cette partie de configuration et de choix des modules qui devront être présents pour l'application que l'on souhaite développer est la première étape à effectuer pour pouvoir ensuite développer sur ce type de microcontrôleur.

Voyons maintenant les autres étapes du développement et de prise en main de ces PSoC.

Prise en main et développement

Le développement de ces contrôleurs se fait à l'aide d'un logiciel : PSoC Designer, développé par *Cypress MicroSystems*, et des outils l'accompagnant : un programmateur de PSoC et un émulateur.

La première étape est, comme dit précédemment, le choix des modules "analogiques" et "numériques" qui nous intéressent pour l'application à développer.

Cette partie est essentiellement graphique.

Puis, une seconde étape consiste à placer ces modules aux emplacements que l'on souhaite sur le PSoC.

Dans un même temps, on peut configurer les entrées/sorties dans le mode que l'on souhaite, et aussi régler les différents paramètres pour le PSoC en général et pour toutes les fonctions ajoutées.

Viens ensuite une première compilation qui permet de rajouter au programme principal les différentes librairies rattachées aux éléments mis en place précédemment.

La troisième partie consiste à la programmation du PSoC, et à l'écriture du programme principal. On peut dans cette partie écrire son propre programme et le faire compiler avec le reste des librairies déjà en place.

Il faut penser dans cette phase à initialiser et démarrer les modules choisis à l'aide des fonctions définies pour cela, sinon les modules ne se lancent pas seuls.

Ces commandes d'initialisation et de démarrage des différentes parties, permettent notamment de ne pas lancer certains modules dès le début si ce module ne sert que pour une petite partie du programme. C'est donc dans un soucis de consommation et d'utilisation des ressources globales du PSoC que ces commandes existent.

Une fois la compilation de son programme réussie, on peut alors procéder au premier test, en utilisant par exemple l'émulateur.

On peut notamment grâce à cet émulateur lancer le programme pas à pas et vérifier à tout moment l'état des différentes variables ou des différents modules.

Cela permet d'effectuer le débogage du programme et ainsi voir les choses qu'il reste à modifier.

Enfin une fois que le programme fonctionne comme on le souhaite, on peut passer à l'étape de programmation de la "puce".

Cette étape ne prend que peu de temps, juste de quoi laisser le temps au logiciel de vérifier les données, de les convertir en binaire et de les envoyer via le port usb de l'ordinateur et à la carte d'interface PSoC dans le composant.

Il reste l'étape ultime qui consiste à tester le montage dans les conditions réelles et avec un PSoC autonome.

Annexe B
Listing du programme PSoC

```
/*****************************************************************************/
/*****************************************************************************/
/**                                                                         **/
/**                                TER 2005                                 **/
/**                  Gestion d'énergie sur un robot mobile                  **/
/**                                                                         **/
/*****************************************************************************/
/*****************************************************************************/
/**                                                                         **/
/**                   Vincent THOMAS - Julien VILLEMEJANE                   **/
/**                                                                         **/
/*****************************************************************************/
/*****************************************************************************/
/**                          ENS Cachan - Département EEA                   **/
/*****************************************************************************/
/*****************************************************************************/


        // Port 0 - Bit 5 --> Sortie analogique Vref=2,5V

        // Port 1 - Bit 0 --> Signal rechargement
        // Port 1 - Bit 1 --> Signal batterie déchargée
        // Port 1 - Bit 2 --< Entrée de réinitialisation
        // Port 1 - Bit 3 --> Signal allumé/éteint
        // Port 1 - Bit 4 --< Entrée de réinitialisation (test)

        // Port 2 - Bit 1 --< Entrée analogique U
        // Port 2 - Bit 2 --< Entrée analogique I


#include "m8c.h"                        // part specific constants and macros
#include "PSoCAPI.h"                    // PSoC API definitions for all User Modules

  // Déclaration des variables

 /* Variables "variables" */
 long U,I;
 long E,Ep;                              // Energie discrête
 long En;                                // Energie "instantanée" discrête
 char Es[4];                             // Energie convertie en char
 long cpt=0,cpts=0;                      // compteur pour l'initialisation et la sauvegarde

 /* Variables "fixes" */
 long Uref=2200;                         // tension de référence (correspond à 10,3 V environ)
 long Ubas=2000;                         // valeur critique de décharge de la batterie (environ U=9V)
 long Eref=100000000;                    // permet de tenir 4-5 min après l'envoi du signal de rech
 long k=4;                               // coefficient multiplicateur pour le calcul de E
 long f=300;                             // frequence pour le calcul de E : E = U.I/(k*f)
```

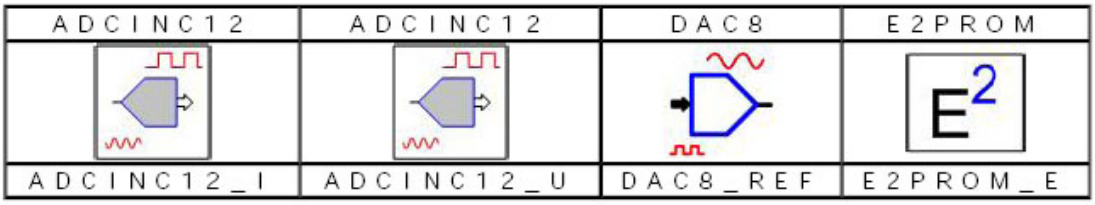
```
/***********************************/
/* Fonction d'acquisition de U et I */
/***********************************/
void Acquis(){
        ADCINC12_U_GetSamples(1);
        ADCINC12_I_GetSamples(1);

        while(ADCINC12_U_fIsDataAvailable() == 0);      // Attente de récupération de U
        U=(ADCINC12_U_iGetData()+2048);                 // Récupération de U
        ADCINC12_U_ClearFlag();

        while(ADCINC12_I_fIsDataAvailable() == 0);      // Attente de récupération de I
        I=ADCINC12_I_iGetData();                        // Récupération de I
        ADCINC12_I_ClearFlag();

}


/***********************************/
/* Fonction de translation E --> Es */
/*          long  -> char          */
/*                                 */
/*      Sauvegarde des données     */
/*    toutes les 30 sec en Flash   */
/***********************************/
void saveE(){
        if(cpts>9000){                          // procedure de memorisation toute les 30s environ
                // MSB
                Ep = E & 0xFF000000;            // Application d'un masque sur les bits de poids fort
                Ep=Ep>>24;                      // Decalage a droite de 24 bits
                Es[0]=(unsigned char)Ep;        // Memorisation
                // 3e octet
                Ep = E & 0x00FF0000;            // Application d'un masque sur les 8 bits suivants
                Ep=Ep>>16;                      // Decalage a droite de 16 bits
                Es[1]=(unsigned char)Ep;                 // Memorisation
                // 2e octet
                Ep = E & 0x0000FF00;            // Application d'un masque sur les 8 bits suivants
                Ep=Ep>>8;                       // Decalage a droite de 8 bits
                Es[2]=(unsigned char)Ep;        // Memorisation
                // LSB
                Ep = E & 0x000000FF;            // Application d'un masque sur les bits de poids
                Es[3]=(unsigned char)Ep;        // Memorisation                                 faible
                // Ecriture en Flash
                E2PROM_E_bE2Write(0,Es,4,25);
                cpts=0;
        }
        else{cpts++;}
}


/***********************************/
/* Fonction de translation Es --> E */
/*           char  -> long         */
/***********************************/
void readE(){
        // Lecture sur Flash
        E2PROM_E_E2Read(0,Es,4);
        //MSB
        Ep=(unsigned long)Es[0];                // Acquisition des 8 premiers bits memorisés
        Ep=Ep<<8;                               // Decalage a gauche de 8 bits
        // 3e octet
        Ep+=(unsigned long)Es[1];               // Acquisition des 8 bits suivants
        Ep=Ep<<8;                               // Decalage a gauche de 8 bits
        // 2e octet
        Ep+=(unsigned long)Es[2];               // Acquisition des 8 bits suivants
        Ep=Ep<<8;                               // Decalage a gauche de 8 bits
        // LSB
        Ep+=(unsigned long)Es[3];               // Acquisition des 8 derniers bits memorisés
        E=Ep;                                   // Stockage de la valeur lue dans E
}
```

```
/*******************************/
/* Fonction de réinitialisation */
/*******************************/
// Fonction permettant de réinitialiser la valeur de E
void EraseE(){
        if((PRT1DR & 0x04)==0){                 // Test le bit 2 du port 1
                E=Eref;
                PRT1DR |= 0x04;                 // Remet à 1 le bit 2 du port 1
                saveE();
        }

// Fonction supplémentaire pour les tests, qui réinitialise E à 2*Eref
        if((PRT1DR & 0x10)==0){                 // Test le bit 4 du port 1
                E=2*Eref;
                PRT1DR |= 0x10;                 // Remet à 1 le bit 4 du port 1
                PRT1DR &= ~0x01; PRT1DR &= ~0x02;
                saveE();
        }
}


/*************************************/
/* Fonction d'initialisation des modules */
/*************************************/
void Init(){
    // Initialisation PSoC
    M8C_EnableGInt;


    // Démarrage des CAN et CNA
    ADCINC12_I_Start(ADCINC12_I_HIGHPOWER);
    ADCINC12_U_Start(ADCINC12_U_HIGHPOWER);
    DAC8_REF_Start(DAC8_REF_HIGHPOWER);

    DAC8_REF_WriteBlind(128);                   // Signal de référence pour le courant


    // EEPROM - Récupération de la valeur de E précédemment stockée
    E2PROM_E_Start();
    readE();


    // Connexion des ports numériques
    PRT1DR &= ~0x0B;                            // mise à zéro des bits utilisés du port 1
    PRT1DR |= 0x04;                             // mise à un du bit 2 du port 1
    PRT1DR |= 0x10;                             // mise à un du bit 4 du port 1


        // Tempo d'initialisation pour la mise sous tension
        while(cpt<300000){cpt++;}
        PRT1DR |= 0x08;                         // Calculateur prêt

        // Récupération de U et I (au démarrage)
        Acquis();
}

/********************/
/* Boucle de Charge */
/********************/
// Si I est positif, alors on rentre dans cette boucle de Charge.
// I étant positif, le résultat E l'est aussi. Le compteur d'énergie E s'incrémente.

void Charge() {

        EraseE();
        if(E>Eref) { PRT1DR &= ~0x01; PRT1DR &= ~0x02;}         // Mise à 1 du bit 0 du port 1

        Acquis();                                               // Acquisition de U et I à une fréquenc

        En=U*I/(k*f);                                           // Calcul de E
        E+=En;

        saveE();                                                // Memorisation de la valeur de l'energ
}
```

```
/**********************/
/* Boucle de Decharge */
/**********************/
// Si I est négatif, alors on rentre dans cette boucle de Decharge.
// I étant négatif, le résultat E l'est aussi. Le compteur d'énergie E décrémente.
// S'il atteind une valeur limite Eref, on envoie alors le signal de rechargement au robot.

void Decharge() {
        EraseE();

                if((E>Eref)&&(U>Uref))PRT1DR &= ~0x01;          // Mise à 1 du bit 0 du port 1 - Pas besoin de
recharger
                Acquis();                                       // Acquisition de U et I à une fréquence de 300 Hz


                if(U>Uref){                                     // Test si U est supérieure à une tension de
référence
                        En=U*I/(k*f);                           // si c'est le cas, calcul de P
                        if(E>En){E+=En;}
                        else {PRT1DR |= 0x02; E=0;}             // Erreur batterie déchargée
                        saveE();

                }

                else{                                           // sinon
                        for(cpt=0;cpt<300;cpt++){               // Attente d'une seconde pour retester U
                                                                // permet de savoir si une chute de tension est
momentanée
                                                                // (due à un appel de courant) ou si la batterie es
dechargee
                                Acquis();
                                En=U*I/(k*f);
                                if(E>En)E+=En;
                                else {PRT1DR |= 0x02; E=0;}     // Erreur batterie déchargée
                                saveE();
                                if(E<Eref){                     // Test sur l'énergie
                                        PRT1DR |= 0x01;         // On envoie l'ordre au robot d'aller se recharger
                                }
                        }
                        if(U<Uref){                             // Test sur U
                                PRT1DR |= 0x01;                 // On envoie l'ordre au robot d'aller se recharger
                }
                if(U<Ubas){
                        PRT1DR |= 0x02;                         // Erreur batterie déchargée : ordre d'arret
d'urgence
                }
                }


                if(E<Eref){                                     // Test sur l'énergie
                        PRT1DR |= 0x01;                         // On envoie l'ordre au robot d'aller se recharger
                }

}


 /**********************/
 /* Programme principal */
 /**********************/
 void main()
 {
        Init();                                         // Initialisation
        while(1){                                       // Gestion de l'utilisation et de la charge
                while(I<=0) Decharge();
                while(I>0) Charge();

        }

 }
```

Annexe C
Fonctions utilisées sur les PSoC et leur implantation

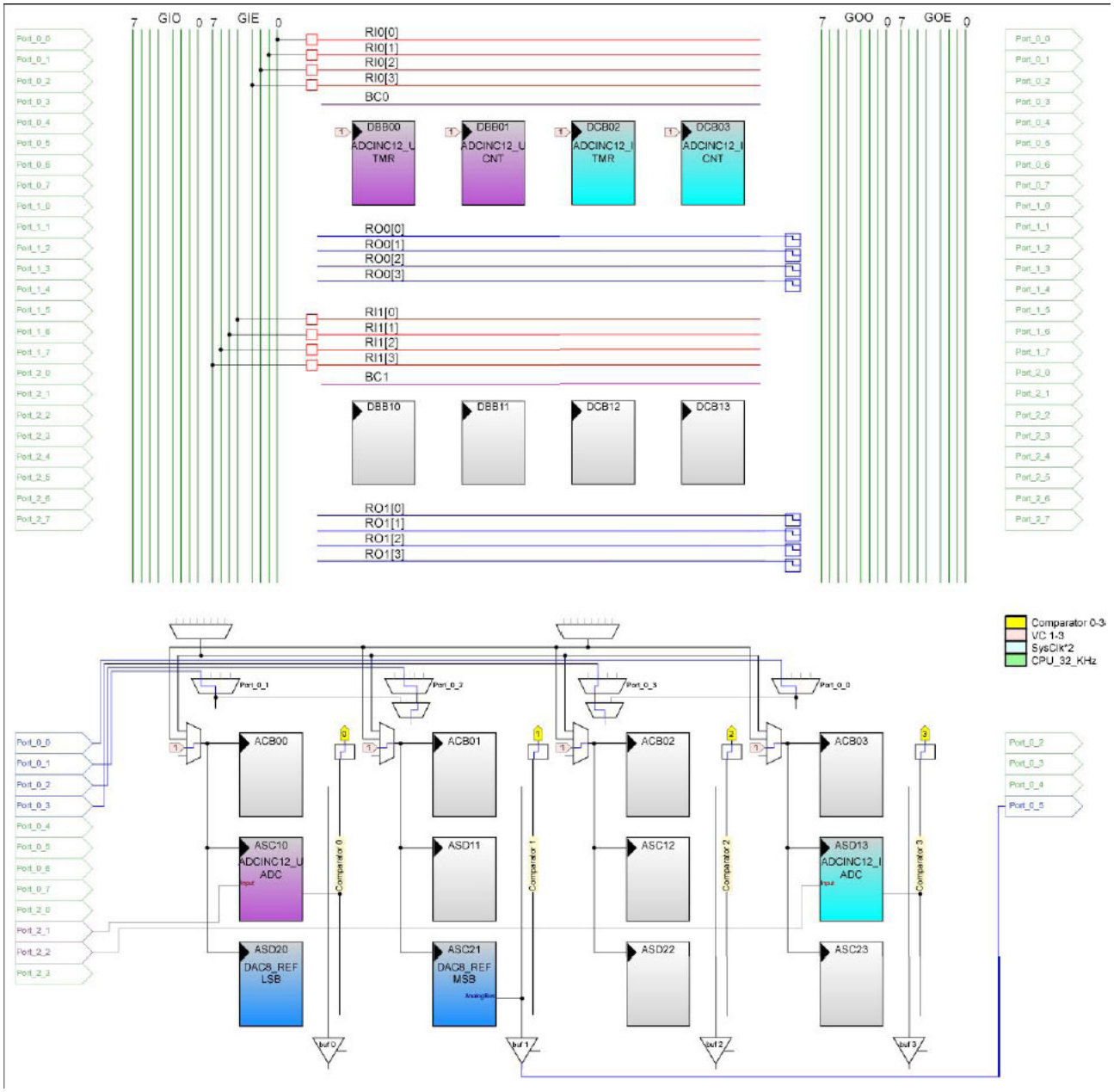
Voici les fonctions utilisées sur le PSoC pour notre application :



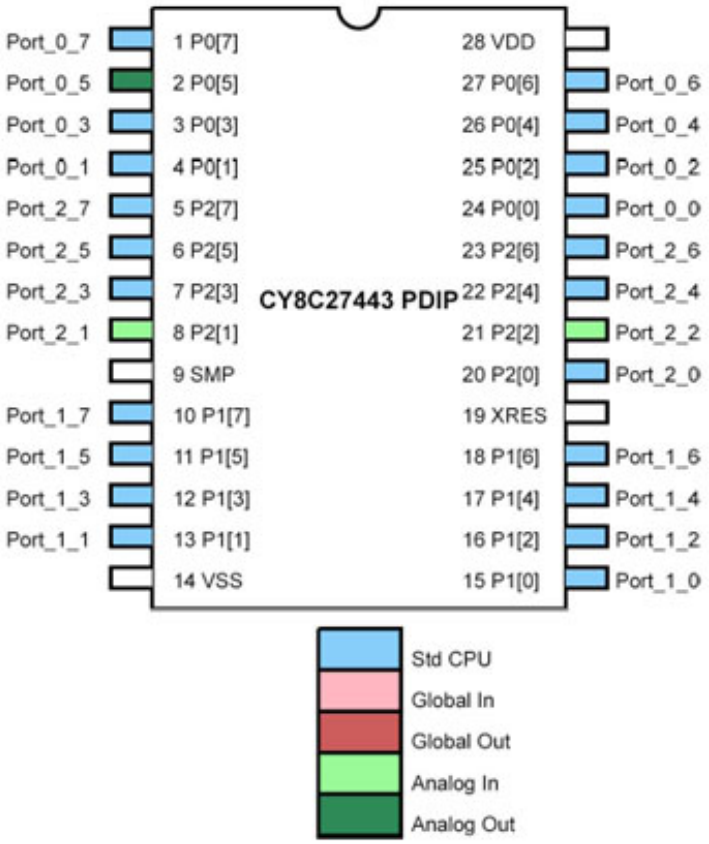
On peut noter la présence de :

- 2 CAN incrémentaux de 12 bits (ADCINC12) servant pour l'acquisition de U et I ;
- 1 CNA de 8 bits servant à réaliser la tension de 2,5V de référence, pour le courant ;
- 1 passerelle EEPROM, qui est un module permettant d'aller écrire en Flash-ROM.

Voici l'implantation de ces fonctions à l'intérieur du microcontrôleur :



Et enfin, voici la correspondance des pattes du PSoC :



Toutes les pattes ne sont pas utilisées.

La patte :

– 2 correspond à la sortie de la tension de référence de 2,5V ;

– 8 est l'entrée du CAN de U ;

– 13 est le signal d'arrêt d'urgence ;

– 15 est le signal de rechargement ;

– 16 est l'entrée de réinitialisation (RAZ) ;

– 21 est l'entrée du CAN de I.

L'alimentation du circuit se fait entre les pattes 14 et 28.

Annexe D
Listing complet des programmes du robot

```
/*****************************************************************************
                                    Avril 2005

                                    Loïc SIMON
                                 Guillaume HERAULT
*****************************************************************************/
/*****************************************************************************
programme pricipal de detection de la balise
*****************************************************************************/

/* si on veut executer le programme sur la carte sans tenir compte de la reponse
des LM 629 definir la variable SIMULATION dans def.h */

#include <reg552.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "main.h"
#include "def.h"
#include "variable.h"
#include "conv.h"
#include "serie.h"
#include "utils.h"
#include "robot.h"
#include "can.h"

/*****************************************************************************
variables globales utiles pour la gestion des contrôleurs de moteurs
*****************************************************************************/

u_char tampon[100];
u_char tableau_LM [TAILLE_TABLEAU_COMMANDE];

/*****************************************************************************
programme de détection de la balise
*****************************************************************************/

void detection_balise(void) {

int sens_rotation=1, balise_atteinte=0;

/*****************************************************************************
initialisation relative au contrôleur des moteurs : travail effectué lors d'un
TER précédent
*****************************************************************************/

u_char initialise[] =
{
//        ATTENDRE,
          RESET1,les_2_LM,0x00,
               MSKI,les_2_LM,0x02,0x00,0x04,
          RSTI,les_2_LM,0x02,0x00,0x00,
          RESET2,les_2_LM,0x00,
          RSTI,les_2_LM,0x02,0x00,0x00,
          LFIL,les_2_LM,0x0A,0x02,0x0F,KP_1,KI_1,KD_1,IL_1,
          UDF,les_2_LM,0x00,
          FIN
     };

SP = 0x2F;

nouvelle_commande=1;

#ifndef SIMULATION
     pilote(initialise);
#endif
```

```
/*****************************************************************************
Nos initialisations
*****************************************************************************/
init_detection();
alimenterplaquette();

/*****************************************************************************

Phase d'approche pas à pas de la balise

*****************************************************************************/
while(balise_atteinte == 0 )
     {
     maximum = 0;

/*****************************************************************************
commande des moteurs en pivotement : on commande au robot de tourner plus de
trois tours mais on l'arrêtera dès qu'on aura détecté le maximum
*****************************************************************************/
     sens_rotation*=-1;
     (sens_rotation==1)?strcpy(tampon,"P,0.25,1000,F"):strcpy(tampon,"P,0.25,-
     1000,F");

/*on change de sens de rotation à chaque fois à cause de l'inertie du robot : en
effet on dépasse à chaque fois la bonne direction, il est donc plus judicieux de
repartir en arrière*/

     traduction();
     cv_tab(tableau_LM);
     navigation(tableau_LM);
/*****************************************************************************
Détection du maximum
*****************************************************************************/
          detect_max();

/* arrivé ici on a soit détecté le maximum soit on est assez proche de la balise
pour pouvoir considérer qu'on y est arrivé*/

     if(fintrajectoire(les_2_LM)!=0x04)
/*si cette condition est fausse on a pas pu détecté le maximum et donc on
considère qu'on a atteint la balise*/

          {


          for(k=0;k<0x0fff;k++){}
/*cette boucle d'attente est une sécurite qui nous permet d'être sûr d'avoir au
moins légèrement dépassé le maximum : ainsi lors de la prochaine rotation qui
aura lieu en sens inverse, le robot atteindra rapidement le maximum sans avoir
pris de l'inertie*/

/*****************************************************************************
stopper la rotation
*****************************************************************************/
          strcpy(tampon,"S,F");
          traduction();
          cv_tab(tableau_LM);
          navigation(tableau_LM);
          while(fintrajectoire(les_2_LM)!=0x04){};
```

```
/
*****************************************************************************
Correction de l'inertie lors des deux premiers passages, on ne fait pas avancer
le robot
*****************************************************************************
/
          if(decalage <= 1)
               {
               (sens_rotation==-1)?strcpy(tampon,"P,0.25,35,F"):strcpy
          (tampon,"P,0.25,-35,F");
               decalage = decalage +1;
               traduction();
               cv_tab(tableau_LM);
               navigation(tableau_LM);
               while(fintrajectoire(les_2_LM)!=0x04){};

               }
/
*****************************************************************************
 Par la suite on ne corrige plus le dépassement car le robot ayant peut d'élan
ce dépassement est négligeable. On peut alors faire approcher le robot de la
balise.
*****************************************************************************
/
          else
               {
               strcpy(tampon,"L,0.25,0.2,F");
               traduction();
               cv_tab(tableau_LM);
               navigation(tableau_LM);
               while(fintrajectoire(les_2_LM)!=0x04){};
               }

          }

     else balise_atteinte = 1;
     }
/* si tout c'est bien passé on a atteint la balise on peut donc couper la
plaquette */

couperplaquette();

/
*****************************************************************************
Boucle d'attente
*****************************************************************************
/
while(1)
     {
     }
}
```

```
/*****************************************************************************/

/**************                          CAN                      **************/

/*****************************************************************************/



#include <reg552.h>
#include "can.h"


/* Mes variables globales */

int maximum,valeurADCH;
char Gain;


sbit P3_4= 0xB4;
sbit P3_5= 0xB5;   //fin mes variables globales









/*****************************************************************************
Fonction d'initialisation pour le programme de détecion
*****************************************************************************/
void init_detection()
{

     PWMP = 0xFF;  //initPWM
     init_can();
     maximum = 0;
     valeurADCH = 0;

     Gain = 1;
     P3_4 = (Gain&0x01);
     P3_5 = !((Gain&0x02)>>1);
}

/*****************************************************************************
Fonctions permettant de mettre sous tension ou non l'unité de détection de la
balise
*****************************************************************************/

void alimenterplaquette()
{
     PWM0 = 0xFF;
}

void couperplaquette()
{
     PWM0 = 0;
}
```

```
/*****************************************************************************
Diverses fonctions élémentaires
*****************************************************************************/

void init_can()
{
     ADCON=0x00;      //initialisation du can
//   EAD=1;         //validation de l'interruption du can
}

void lancerconversion()
{
     ADCON = startofconv;
}

void attentefinconv()
{
     while((ADCON & endofconv) != 0x10){}
}

/*****************************************************************************
Fonction de mise à jour de la valeur du maximum d'éclairement
*****************************************************************************/

void traitement()
{
     lecture_can();
     if((valeurADCH>maximum))
     {

          maximum = valeurADCH;

     }
}

/*****************************************************************************
Fonction gérant la lecture de ADCH0 ainsi que le réglage du gain le cas échéant
*****************************************************************************/

int lecture_can()
{
     int i;
     lancerconversion();
     attentefinconv();
     valeurADCH = ADCH;
     if((valeurADCH == 255)&(Gain!=4))
     {
          Gain = (Gain + 1);
          P3_4 = (Gain&0x01);
          P3_5 = !((Gain&0x02)>>1);

          maximum = 0;
          detect_max();
          return 1;

     }
     for(i=0;i<IMAX;i++)
     {
          lancerconversion();
          attentefinconv();
          valeurADCH += ADCH;
     }
     valeurADCH /= IMAX;
     return 0;
}
```

```
/*****************************************************************************
                Fonction de détection du maximum d'éclairement
   on ne sort de cette fonction qu'après avoir détecté une phase de montée de
 l'éclairement puis une phase de diminution, ce qui caractérise bien le passage
  par un maximum. Toutefois si aucun éclairement n'a été détecté on sort quand
           même et on considère que le robot est arrivé a la balise
*****************************************************************************/



void detect_max()
{

     unsigned int i,j;

     traitement();

     //petite boucle d'attente

     for(i=0;i<=0x03ff;i++){for(j=0;j<=0x00ff;j++);}

     do
     {
          lecture_can();
     }while((maximum > valeurADCH + 1)&(fintrajectoire(les_2_LM)!=0x04));
/*on sort de cette première boucle en phase d'éclairement croissant ou si la
valeur lue sur le CAN reste nulle indéfiniment (en fait pendant le temps que le
robot éffectue une rotation de 1000 degrès*/

     do
     {
          traitement();
     }while((maximum <= valeurADCH)&(fintrajectoire(les_2_LM)!=0x04));
/*on sort de cette deuxième boucle dès que l'éclairement se remet a decroître ou
si la valeur lue sur le CAN reste nulle indéfiniment*/
}



/*Attention il est impératif de remarquer que les deux dernières fonctions sont
en fait « imbriquées l'une dans l'autre » ce qui induit une récursivité
complexe. Il a donc fallu faire particulièrement attention à la condition
d'arrêt de cette récursion; en fait cette condition d'arrêt se trouve dans la
fonction lecture_can() : if((valeurADCH == 255)&(gain!=4)).
En fait, il est évident à cause de la condition gain!=4 que l'on ne rentrera
dans la récursion plus de quatre fois.*/
```

```
/*                                                      */
/*            CONV.C                                    */
/*                                                      */
/*   Jul. 2002                                          */
/*   Mulot Fabien Hessemans Anne-Caroline               */
/*                                                      */
/* conversion de la feuille de route                    */
/* contenant les commandes et les valeurs               */
/* des angles, distances et vitesses en                 */
/* données compréhensibles par les LM629                */
/*                                                      */
/******************************************************/

#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "def.h"
#include "conv.h"

void char_to_double(float tab[], u_char tamp[],u_int *compteur){
int i=0;
int j=0;
char nombre[TAILLE_NOMBRE];
*compteur = 0;

        while( tamp[i] != ',' ) {
      nombre[i]=tamp[i];      //on copie le nombre en ascii dans le tableau nombre
                i++;
                *compteur=*compteur+1;
        }
        nombre[i]='\0';
        tab[0]=atof(nombre);        //conversion du nombre ascii en double
}


long conv_vit(float metre_sec){
        return ( metre_sec * 14564601 );
}

u_char filtre(long x, unsigned long i){
        return ((x / i) );
}


long conv_dist1 (float metre){
        return ( metre * 868120 );
}

long conv_dist2 (float metre){
        return ( metre * -868120 );
}

long conv_ang (float degre){
        return (degre * 1149);
}
```

```
/*****************************************/
/*                                       */
/*      ROBOT.C                          */
/*                                       */
/*   Apr. 2000 :                         */
/*     Vassan Karine, Chapouthier Julien */
/*                                       */
/*   Jul. 2000 :                         */
/*     Salvetti Djoume, Leroy Paul       */
/*                                       */
/*   Nov 2001 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*   Feb 2002 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*****************************************/

#include <reg552.h>
#include <stdio.h>

#include "def.h"
#include "utils.h"
#include "robot.h"
#include "conv.h"
#include "serie.h"

/***********************************************/
/*                                             */
/*            NAVIGATION                       */
/*                                             */
/* Execution de la feuille de route            */
/*                                             */
/* void navigation(u_char tab[])               */
/*                                             */
/* appel : tableau decrivant la trajectoire    */
/*                                             */
/* retour : void                               */
/*                                             */
/***********************************************/


void navigation(u_char tab[]){

        u_char ordre;
   u_int i=0;
        u_int j=0;
```

```
// Commandes a envoyer vers les LM pour effectuer la trajectoire de type STOP

u_char trajectoirestop[]= {
        RSTI,les_2_LM,0x02,0x00,0x00,
                LTRJ,LM_1,0x02,0x04,0x00,
                LTRJ,LM_2,0x02,0x04,0x00,
                STT,les_2_LM,0x00,
        FIN
    };

// Commandes a envoyer vers les LM pour effectuer la trajectoire de type LIGNE DROITE

u_char trajectoirelignedroite[]= {
        RSTI,les_2_LM,0x02,0x00,0x00,
                LTRJ,LM_1,0x0E,0x00,0x2B,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,
        LTRJ,LM_2,0x0E,0x00,0x2B,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,
                STT,les_2_LM,0x00,
        FIN
    };

// Commandes a envoyer vers les LM pour effectuer la trajectoire de type PIVOTEMENT

u_char trajectoirepivoter[] =  {
          RSTI,les_2_LM,0x02,0x00,0x00,
                LTRJ,les_2_LM,0x0E,0x00,0x2B,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,
                        0x00,0x00,0x00,0x00,
                STT,les_2_LM,0x00,
        FIN
        };

// Commandes a envoyer vers les LM pour effectuer la trajectoire de type ARC

u_char trajectoirearc[]= {
        RSTI,les_2_LM,0x02,0x00,0x00,
                LTRJ,LM_1,0x0E,0x00,0x2B,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,
        LTRJ,LM_2,0x0E,0x00,0x2B,0x00,0x00,0x06,0x00,0x00,0x00,0x00,0x00,0x00,
                0x00,0x00,0x00,0x00,
                STT,les_2_LM,0x00,
        FIN
    };

        //while(tab[i]!=(u_char) FIN) {
                ordre=tab[i];
                switch (ordre) {

                        case ST:
                                pilote(trajectoirestop);
                                break;
                case PV:
                for(j=0;j<8;j++) trajectoirepivoter[j+14]=tab[i+1+j];
                i+=9;
                pilote(trajectoirepivoter);
                break;
                        case LD:
                        for(j=0;j<4;j++) {
                        trajectoirelignedroite[j+31]=tab[i+1+j];
                                        trajectoirelignedroite[j+14]=tab[i+1+j];
                        trajectoirelignedroite[j+18]=tab[i+1+j+4];
                        trajectoirelignedroite[j+35]=tab[i+1+j+8];
                }
```

```
                pilote(trajectoirelignedroite);
                i+=13;
                break;
                        case AR:
                trajectoirearc[14]=tab[i+1]; // vitesse du premier LM
                trajectoirearc[15]=tab[i+2];
                trajectoirearc[16]=tab[i+3];
                trajectoirearc[17]=tab[i+4];

                trajectoirearc[31]=tab[i+5]; // vitesse du second LM
                                trajectoirearc[32]=tab[i+6];
                                trajectoirearc[33]=tab[i+7];
                                trajectoirearc[34]=tab[i+8];

                                trajectoirearc[18]=tab[i+9]; // deplacement du premier LM
                                trajectoirearc[19]=tab[i+10];
                                trajectoirearc[20]=tab[i+11];
                                trajectoirearc[21]=tab[i+12];

                                trajectoirearc[35]=tab[i+13];         // deplacement du second LM
                                trajectoirearc[36]=tab[i+14];
                                trajectoirearc[37]=tab[i+15];
                                trajectoirearc[383]=tab[i+16];

                        pilote(trajectoirearc);
                        i+=17;
                        break;
                //}

//              while(fintrajectoire(les_2_LM)!=0x04){};
                i++;
                i--;
        }
}

/*      FIN NAVIGATION      */

/*************************************/
/*                                   */
/*              PILOTAGE             */
/*                                   */
/* pilotage des LM pour la trajectoire */
/*                                   */
/* void pilote(u_char trajectoire[]) */
/*                                   */
/* appel : tableau decrivant la trajectoire */
/*                                   */
/* retour : void                     */
/*                                   */
/*************************************/

void pilote(u_char trajectoire[]){

        u_char LM_occupes;

        u_char etat=lirecommande;
        u_char commande;
        u_int i=0;
        u_char nboctets=0x00;
        u_char quelLM;
        initport1();
        initport4();
```

```
while(etat!=(u_char)FIN) { //il ne faut pas confondre FIN et fintrajectoire
        switch(etat) {
                case (u_char)lirecommande:
                        commande=trajectoire[i];
                i++;
                etat=commande;
                        if( (etat != FIN) && (etat != ATTENDRE))
                                etat=(u_char)lireinstructions;
           break;
        case(u_char)lireinstructions:
                if(commande>0xF7)
                                etat=commande;
                quelLM=trajectoire[i];
      i++;
    nboctets=trajectoire[i];
                i++;
          if((etat!=RESET1)&& (etat!=RESET2))
                                etat=(u_char) envoyercommande;
                break;
        case (u_char)envoyercommande:
                etat=(u_char)envoyerlesdonnees;
        ecrirecommande(commande,quelLM);

                        #ifndef SIMULATION
                                do {
                                        LM_occupes = busy(quelLM);
                                }
                                while(LM_occupes == 0x01);
                        #else
                                LM_occupes = 0;
                        #endif

        break;
case envoyerlesdonnees:
         while(nboctets>0x00) {
        ecriredonnees(trajectoire[i],quelLM);
       i++;nboctets--;
       ecriredonnees(trajectoire[i],quelLM);
       i++;
                                nboctets--;

                        #ifndef SIMULATION
                                do {
                                        LM_occupes = busy(quelLM);
                                }
                                while(LM_occupes == 0x01) ;
                        #else
                                LM_occupes = 0;
                        #endif

  }
   etat=lirecommande;
   break;
        case(u_char)RESET1:

                        #ifndef SIMULATION
/*                      if (reset((u_char)0xC4,(u_char)0x84)==((u_char) VRAI))
                                etat=(u_char)lirecommande;
                        #else*/
                                etat=(u_char)lirecommande;
                        #endif
```

```
                break;
        case(u_char)RESET2:

                                #ifndef SIMULATION
/*                                      if (reset((u_char)0x80,(u_char)0xC0)==((u_char) VRAI))
                                                etat=(u_char)lirecommande;
                                #else*/
                                        etat=(u_char)lirecommande;
                                #endif

                break;
                case (u_char)ATTENDRE:
                fonctionattente();
                        etat=(u_char)lirecommande;
                        break;
                        case (u_char)FIN:
           etat=FIN;
                                break;
                }
        }
}

/* FIN DU PILOTAGE */
```

```
/*****************************************/
/*                                       */
/*          SERIE.C                      */
/*                                       */
/*   Jul. 2002                           */
/*   Mulot Fabien Hessemans Anne-Caroline */
/*                                       */
/* reception de la feuille de route sous  */
/* forme de caracteres ASCII, stockage des*/
/* caracteres  dans un tableau route[],   */
/* puis traduction  en un tableau         */
/* contenant les commandes et les valeurs */
/* des angles, distances et vitesses      */
/*****************************************/

#include <reg552.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

#include "def.h"
#include "conv.h"
#include "serie.h"
#include "utils.h"
#include "robot.h"

//declaration des variables globales
u_int fin_route=0;            // 1: indique que route[] est totalement rempli
u_int nb_virgule=0;
u_char octet;                 // caractere ASCII recu sur le port serie
u_int nouvelle_commande;           // 1:indique que l'octet qui va etre memorise contient une commande(A,P,L,F)
u_char commande_ascii;        // contient  la commande courante (L ou P ou A ou F)
u_int i_route=0;              // indice du tableau route[]
u_int i_tampon=0;             // taille du tableau route
u_int octet_recu=0;
//u_char route[100];    // tableau contenant les caracteres ascii recus sur le port serie
float tableau_commande[TAILLE_TABLEAU_COMMANDE];  // feuille route a convertir en donnees LM629
extern u_char tampon[TAILLE_ROUTE];   // copie de route[] servant a liberer route[] en vue d'une nouvelle
reception

/*
void remplir_tampon(){
u_int l=0;                    //indice de parcours des tableaux
        for(l=0; l<i_tampon+1 ; l++){    //boucle de parcours du tableau route
                tampon[l]=route[l];
        }
}
*/
void recep_octet() interrupt 4 {   // cette fonction ne se déclenche que sous interruption 4
        octet=S0BUF;                //on memorise l'octet lu dans S0BUF dans la variable octet
        octet_recu=1;
        RI=0;                       //l'interruption est remise à l'etat bas
}
/*
void remplir_route(){
        if(nouvelle_commande==1){       //test est valide quand une nouvelle commande arrive
                commande_ascii=octet;   //on place l'octet recu (qui est une commande) dans commande_ascii

          nb_virgule=0;               //afin de passer dans le bon case
        }
        nouvelle_commande=0;              //nouvelle_commande est remis à zero afin d'enregistrer les arguments de
la commande
```

```
        switch (commande_ascii){
                case 'A':
                        route[i_route]=octet; //on enregistre dans la feuille de route l'octet recu
                        if(route[i_route]==',') nb_virgule++;  //on incremente le nb de virgule afin de tester si
                                                //tous les arguments sont memorises
                        i_route++;
                        if(nb_virgule==4) {     //si tous les arguments sont memorises alors on peut passer a la
                              //commande suivante
                                nouvelle_commande=1;
                        }
                        break;

                case 'P':
                   route[i_route]=octet;    //on enregistre dans la feuille de route l'octet recu
                        if(route[i_route]==',') nb_virgule++;  //on incremente le nb de virgule afin de tester si
                                                //tous les arguments sont memorises
                i_route++;
                        if(nb_virgule==3) {     //si tous les arguments sont memorises alors on peut passer a la
                                        //commande suivante
                                nouvelle_commande=1;
                        }
                        break;
                case 'L':
                   route[i_route]=octet;                //on enregistre dans la feuille de route l'octet recu
                        if(route[i_route]==',') nb_virgule++;  //on incremente le nb de virgule afin de tester si
                                                //tous les arguments sont memorises
                        i_route++;
                        if(nb_virgule==3){                  //si tous les arguments sont memorises alors on peut passer a
la
                                                //commande suivante
                                nouvelle_commande=1;
                        }
                        break;
                case 'F':
                        route[i_route]=octet;               //on enregistre dans la feuille de route l'octet recu
                        i_tampon=i_route;                   // on memorise la taille de la feuille de route
                        i_route=0;                        //on reinitialise l'indice de route à 0 pour la prochaine
                                                            //feuille de route a recevoir
                        nouvelle_commande=1;
                        fin_route=1;                       //bit de fin d'aquisition de la feuille de route. Il passe a 1 lorsque
la feuille de route est complete
        break;
        }
}
*/

void traduction(){

u_int continuer = 0;
u_int compteur=0;
int i=0;
int j=0;
int k=0;
u_char ordre;
```

```
        while (continuer == 0){
                ordre=(u_int)tampon[i];
                switch (ordre){
                        case 'A':
                                tableau_commande[j]=(float) A;
                        for(k=0; k<3 ; k++) {     //boucle de conversion de tous les arguments d'une commande
                                        j++;
                                i++;
                                        char_to_double(tableau_commande+j,tampon+i+1,&compteur);  //conversion
d'un argument code en
                                                                                        //ascci en float
                                        i=i+compteur;    //compteur qui permettra d'incrementer correctement le tableau
route
                                }
                j++;
                        i=i+2;                  //permet de pointer sur la commande suivante
                                break;
                        case 'P':
                                tableau_commande[j]=P;
                                for(k=0; k<2 ; k++) {    //boucle de conversion de tous les arguments d'une commande
                                        j++;
                                i++;
                                        char_to_double(tableau_commande+j,tampon+i+1,&compteur);//conversion
d'un argument code en
                                                                                //ascii en float
                                        i=i+compteur;     //compteur qui permettra d'incrementer correctement le
tableau route
                                }
                j++;
                                i=i+2;
                                break;
                        case 'L':
                                tableau_commande[j]=L;
                                for(k=0; k<2 ; k++) {     //boucle de conversion de tous les arguments d'une commande
                                        j++;
                                i++;
                                        char_to_double(tableau_commande+j,tampon+(i+1),&compteur);//conversion
d'un argument code en
                                                                                //ascci en float
                                        i=i+compteur;     //compteur qui permettra d'incrementer correctement le
tableau route
                                }
                j++;
                                i=i+2;
                                break;
                case 'S':
                                tableau_commande[j]= S;
                                j++;
                                i+=2;
                                break;
                        case 'F':
                                tableau_commande[j]= F;
                                continuer=1;
                                break;
                }
        }
}
```

```
void cv_tab(u_char tab_out[]){
int continuer;
long ordre;
float v1;
float v2;
float d;
int j;
long i_commande;

        j = 0;
        i_commande = 0;
        continuer = 0;
        ordre = 0;

        while ( continuer==0 ){
                ordre = (long) tableau_commande[i_commande];
                switch (ordre){
                        case L:
                                tab_out[j] = L ;
                                //conversion de la vitesse

                                tab_out[j+1] = filtre(( conv_vit( tableau_commande[i_commande+1]) & 0xff000000l),
0x01000000l);
                                tab_out[j+2] = filtre(( conv_vit( tableau_commande[i_commande+1]) & 0x00ff0000l),
0x00010000l);
                                tab_out[j+3] = filtre(( conv_vit( tableau_commande[i_commande+1]) & 0x0000ff00l),
0x00001100l);
                                tab_out[j+4] = filtre(( conv_vit( tableau_commande[i_commande+1]) & 0x000000ffl),
0x00000011l);
                                //conversion des 2 octets de la distance
                        tab_out[j+5] = filtre(( conv_dist1( tableau_commande[i_commande+(2)]) & 0xff000000l),
0x01000000l);
                                tab_out[j+6] = filtre(( conv_dist1( tableau_commande[i_commande+(2)]) &
0x00ff0000l), 0x00010000l);
                                tab_out[j+7] = filtre(( conv_dist1( tableau_commande [i_commande+(2)]) &
0x0000ff00l), 0x00001100l);
                                tab_out[j+8] = filtre(( conv_dist1( tableau_commande [i_commande+(2)]) &
0x000000ffl), 0x00000011l);

                tab_out[j+9] = filtre(( conv_dist2( tableau_commande [i_commande+(2)]) & 0xff000000l),
0x01000000l);
                                tab_out[j+10] = filtre(( conv_dist2( tableau_commande [i_commande+(2)]) &
0x00ff0000l), 0x00010000l);
                                tab_out[j+11] = filtre(( conv_dist2( tableau_commande [i_commande+(2)]) &
0x0000ff00l), 0x00001100l);
                                tab_out[j+12] = filtre(( conv_dist2( tableau_commande [i_commande+(2)]) &
0x000000ffl), 0x00000011l);
```

```
                        i_commande = i_commande+3;
                        j = j+13;
                        break;
                case P:
                        tab_out[j] = P;
                        //conversion de la vitesse
                        tab_out[j+1] = filtre(( conv_vit( tableau_commande [i_commande+(1)]) &
0xff000000l), 0x01000000l);
                        tab_out[j+2] = filtre(( conv_vit( tableau_commande [i_commande+(1)]) &
0x00ff0000l), 0x00010000l);
                        tab_out[j+3] = filtre(( conv_vit( tableau_commande [i_commande+(1)]) &
0x0000ff00l), 0x00000100l);
                        tab_out[j+4] = filtre(( conv_vit( tableau_commande [i_commande+(1)]) &
0x000000ffl), 0x00000001l);
                        //conversion de l'angle
                        tab_out[j+5] = filtre(( conv_ang( tableau_commande [i_commande+(2)]) &
0xff000000l), 0x01000000l);
                        tab_out[j+6] = filtre(( conv_ang( tableau_commande [i_commande+(2)]) &
0x00ff0000l), 0x00010000l);
                        tab_out[j+7] = filtre(( conv_ang( tableau_commande [i_commande+(2)]) &
0x0000ff00l), 0x00000100l);
                        tab_out[j+8] = filtre(( conv_ang( tableau_commande [i_commande+(2)]) &
0x000000ffl), 0x00000001l);

                        i_commande = i_commande+ 3 ;
                        j = j+9;
                        break;
                case A:
                        tab_out[j] = A;
        v1=(tableau_commande[1]);
                        v1 = (1 - (largeur / (2 * tableau_commande[i_commande+2]))) * tableau_commande
[i_commande+1];
                        v2 = (1 + (largeur / (2 * tableau_commande[i_commande+2]))) * tableau_commande
[i_commande+1];
                        d = (tableau_commande[i_commande+2] - (largeur / 2)) * (3.146 / 180) *
tableau_commande[i_commande+3];
        //conversion vitesse
                        tab_out[j+1] = filtre(( conv_vit( v1) & 0xff000000l), 0x01000000l);
                        tab_out[j+2] = filtre(( conv_vit( v1) & 0x00ff0000l), 0x00010000l);
                        tab_out[j+3] = filtre(( conv_vit( v1) & 0x0000ff00l), 0x00000100l);
                        tab_out[j+4] = filtre(( conv_vit( v1) & 0x000000ffl), 0x00000001l);

                        tab_out[j+5] = filtre(( conv_vit( v2) & 0xff000000l), 0x01000000l);
                        tab_out[j+6] = filtre(( conv_vit( v2) & 0x00ff0000l), 0x00010000l);
                        tab_out[j+7] = filtre(( conv_vit( v2) & 0x0000ff00l), 0x00000100l);
                        tab_out[j+8] = filtre(( conv_vit( v2) & 0x000000ffl), 0x00000001l);

                        //conversion des distances
                tab_out[j+9] = filtre(( conv_dist1( d) & 0xff000000l), 0x01000000l);
                        tab_out[j+10] = filtre(( conv_dist1( d) & 0x00ff0000l), 0x00010000l);
                        tab_out[j+11] = filtre(( conv_dist1( d) & 0x0000ff00l), 0x00000100l);
                        tab_out[j+12] = filtre(( conv_dist1( d) & 0x000000ffl), 0x00000001l);

        tab_out[j+13] = filtre(( conv_dist2( d) & 0xff000000l), 0x01000000l);
                        tab_out[j+14] = filtre(( conv_dist2( d) & 0x00ff0000l), 0x00010000l);
                        tab_out[j+15] = filtre(( conv_dist2( d) & 0x0000ff00l), 0x00000100l);
                        tab_out[j+16] = filtre(( conv_dist2( d) & 0x000000ffl), 0x00000001l);

                        i_commande = i_commande+4;
                        j = j+17;
                        break;
```

```
                    case S:
                        tab_out[j] = S;
                        i_commande = i_commande+1;
                        j++;
                        break;

                    case F:
                        tab_out[j] = F;
                        continuer=1;
                        break;
                }
            }
        }
```

```
/************************************/
/*                                  */
/*      UTILS.C                     */
/*                                  */
/*   Apr. 2000 :                    */
/*     Vassan Karine, Chapouthier Julien */
/*                                  */
/*   Jul. 2000 :                    */
/*     Salvetti Djoume, Leroy Paul  */
/*                                  */
/*   Nov 2001 :                     */
/*     Pradel Gilbert               */
/*                                  */
/*   Feb 2002 :                     */
/*     Pradel Gilbert               */
/*                                  */
/************************************/

/*******************************/
/*                             */
/* fonctions diverses          */
/*                             */
/*******************************/


#include <reg552.h>
#include <stdio.h>

#include "def.h"

#include "utils.h"
#include "robot.h"
#include "conv.h"
#include "serie.h"

/*******************************/
/*                             */
/* fonction d'attente          */
/*                             */
/*******************************/

void fonctionattente(void){
        u_int j;
        for(j=0;j<61000;j++){};
}

/*********************************************/
/* fonction de mise à 1 des bits du port 1 */
/*********************************************/
void initport1(void)
{
        P1 = (P1&MSKbits05)|BitsPortA1;
}

/*********************************************/
/* fonction de mise à 1 des bits du port 4 */
/*********************************************/
void initport4(void){
        P4 = BitsPortP4A1;
}
```

```
/*****************************************/
/* fonction de lecture du status byte    */
/*                                       */
/* u_char lirestatusbyte(u_char quelLM)  */
/*                                       */
/* appel : identite du LM concerne       */
/*                                       */
/* retour : status byte du LM concerne   */
/*                                       */
/*****************************************/
u_char lirestatusbyte(u_char quelLM){

        u_char donnees=0;
        initport1();
        switch (quelLM){
                case LM_1:
                        P1 = PSactif; // genere signal espace commande
                        P1 = CS1_PS_RD;   // genere signal de lecture espace commande
                        donnees = P4;   // lecture LM629_1
                        P1 = PSactif; // genere signal espace commande
                        initport1();    //P1 = inactif;
                        break;

                case LM_2:
                        P1 = PSactif; // genere signal espace commande
                        P1 = CS2_PS_RD;   // genere signal de lecture espace commande
                        donnees = P4;   // lecture LM629_2
                        P1 = PSactif; // genere signal de lecture espace commande
                        initport1();    //P1 = inactif;
                        break;
        }
        return (u_char)donnees;
}

/*************************************************/
/* fonction d'ecriture d'une commande dans un LM */
/*                                               */
/* ecrirecommande(u_char commande,u_char quelLM) */
/*                                               */
/* appel : commande a ecrire                     */
/*       identite du LM concerne                 */
/*                                               */
/*       Attention :                             */
/*               si quelLM ==les_2_LM alors      */
/*               la meme commande est            */
/*               appliquee sur les 2 LM          */
/*                                               */
/* retour : void                                 */
/*                                               */
/*************************************************/
```

```
void ecrirecommande(u_char commande,u_char quelLM){
        switch (quelLM)  {
        case LM_1:
                        P1 = PSactif;
                        P1 = CS1_PS_WR;
                        P4 = commande;
                        P1 = PSactif;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
            case LM_2:
                        P1 = PSactif;
                        P1 = CS2_PS_WR;
                        P4 = commande;
                        P1 = PSactif;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
            case les_2_LM:
                        P1 = PSactif;
                        P1 = CS1_PS_WR;
                        P4 = commande;
                        P1 = PSactif;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        P1 = PSactif;
                        P1 = CS2_PS_WR;
                        P4=commande;
                        P1 = PSactif;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
      }
}

/*************************************************/
/*                                               */
/* fonction d'ecriture des donneees dans un LM   */
/*                                               */
/* appel : donnee a ecrire                       */
/*        identite du LM concerne                */
/*                                               */
/*       Attention :                             */
/*                si quelLM ==les_2_LM alors     */
/*                la meme donnee est             */
/*                appliquee sur les 2 LM         */
/*                                               */
/* retour : void                                 */
/*                                               */
/*************************************************/
```

```
void ecriredonnees(u_char donnees,u_char quelLM){
        switch (quelLM)  {
       case LM_1:
         P4=donnees;
                        P1 = CS1_WR;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
                case LM_2:
         P4=donnees;
                        P1 = CS2_WR;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
                case les_2_LM:
                        P4=donnees;
                        P1 = CS1_WR;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
         P4=donnees;
                        P1 = CS2_WR;
                        initport1();    //P1 = inactif;
                        initport4();    //P4 = inactif;
                        break;
        }
}

/*************************************************************/
/* fonction d'interrogation du busy_bit pour le LM concerne  */
/*                                                           */
/* u_char busy(u_char quelLM)                                */
/*                                                           */
/* appel : identite du LM concerne                           */
/*                                                           */
/* retourne VRAI si occupé                                   */
/*         FAUX si non                                       */
/*                                                           */
/*************************************************************/

u_char busy(u_char quelLM){
u_char result;

        switch (quelLM){
                case (u_char) les_2_LM:

                        result = ((lirestatusbyte(LM_1)) & ((u_char) BusyBitMask))
                                ||
                        ((lirestatusbyte(LM_2)) & ((u_char) BusyBitMask)) ;
                        return (result);
                        break;
                default:
                        result = (lirestatusbyte(quelLM)) & ((u_char) BusyBitMask);
                        return (result);
                break;
        }
}
```

```
/*************************************************/
/* fonction d'interrogation de fin de trajectoire */
/*                                               */
/* u_char fintrajectoire(u_char quelLM)          */
/*                                               */
/* appel : identite du LM  concerne              */
/*                                               */
/* retour : VRAI si fini                         */
/*          FAUX sinon                           */
/*                                               */
/*************************************************/

u_char fintrajectoire(u_char quelLM) {
        switch (quelLM) {
        case(u_char) les_2_LM:
                return  ((lirestatusbyte(LM_1) & ((u_char) FinTrajectoireBitMask))
                                        &
                                (lirestatusbyte(LM_2) & (u_char) FinTrajectoireBitMask));
                        break;
                case(u_char) LM_1:
                        return (lirestatusbyte(LM_1) & (u_char) FinTrajectoireBitMask);
                break;
                case(u_char) LM_2:
                        return (lirestatusbyte(LM_2) & (u_char) FinTrajectoireBitMask);
                break;
                default:
                        return (lirestatusbyte(LM_1) & (u_char) FinTrajectoireBitMask);
                break;
        }
}

/*************************************************/
/* fonction d'interrogation du reset             */
/*                                               */
/* u_char reset(u_char reset1,u_char reset2)     */
/*                                               */
/* appel :                                       */
/*                                               */
/* retour :                                      */
/* FAUX =0                                       */
/*   VRAI =1                                     */
/*                                               */
/*************************************************/

u_char reset(u_char reset1,u_char reset2){
        u_char reset_LM1 = (u_char) 0x00;
        u_char reset_LM2 = (u_char) 0x00;

        reset_LM1 = lirestatusbyte(LM_1);
        reset_LM2 = lirestatusbyte(LM_2);
        if((reset_LM1==reset1)||(reset_LM1==reset2)
                &&
          (reset_LM2==reset1)||(reset_LM2==reset2))
                return (u_char) VRAI;
        else
                return (u_char) FAUX;
}
```

```
/*******************************/
/**            CAN.h          *******/
/*******************************/


#ifndef __CAN_H__
#define __CAN_H__


#define startofconv 0x08
#define endofconv 0x10
#define poidsfaibles 0xc0
#define IMAX 50



/*prototypes*/



void detect_max();
void init_detection();
void init_can();
void lancerconversion();
void attentefinconv();
void traitement();
int lecture_can();
void alimenterplaquette();
void couperplaquette();

#endif
```

```
/*****************************************/
/*                                       */
/*      CONV.H                           */
/*                                       */
/*   JUL. 2002 :                         */
/*                                       */
/*    A.Caroline Hessemans               */
/*    Fabien Mulot                       */
/*                                       */
/*****************************************/

/*****************************************/
/* conv_vit                              */
/* convertit la vitesse de metre par     */
/* seconde en increment                  */
/*****************************************/

long conv_vit(float metre_sec);

/*****************************************/
/* filtre                                */
/* filtre permettant de transmettre      */
/* les differents octets d'une valeur    */
/*****************************************/

u_char filtre(long x, unsigned long i);

/*****************************************/
/* conv_dist1                            */
/* convertit une distance en metre       */
/* en increment        pour une des roues du */
/* robot                                 */
/*****************************************/

long conv_dist1 (float metre);

/*****************************************/
/* conv_dist2                            */
/* convertit une distance en metre       */
/* en increment        pour l'autre roue du */
/* robot                                 */
/*****************************************/

long conv_dist2 (float metre);

/*****************************************/
/* conv_ang                              */
/* convertit un angle en degre           */
/* robot                                 */
/*****************************************/

long conv_ang (float degre);

/*********************************/
/* char_to_double                */
/* permet de convertir une suite */
/* de nombre ASCII en un nombre  */
/* reel                          */
/*********************************/

void char_to_double(float tab[], u_char tamp[],u_int *compteur);
```

```
/*****************************************/
/*                                       */
/*    DEF.H                              */
/*                                       */
/*  Apr. 2000 :                          */
/*    Vassan Karine, Chapouthier Julien  */
/*                                       */
/*  Jul. 2000 :                          */
/*    Salvetti Djoume, Leroy Paul        */
/*                                       */
/*  Nov. 2001 :                          */
/*    Pradel Gilbert                     */
/*                                       */
/*  Jul. 2002 :                          */
/*        Hessemans Anne-Caroline,       */
/*    Mulot Fabien                       */
/*                                       */
/*  Feb. 2003 :                          */
/*    Pradel Gilbert                     */
/*                                       */
/*****************************************/

//#define SIMULATION    // si on veut executer le programme sur la carte
                        // sans tenir compte de la reponse des LM 629


/*****************************/
/* DEFINITIONS DU PROGRAMME */
/*****************************/

#define u_char unsigned char
#define u_int unsigned int

#define FAUX 0
#define VRAI 1

#define MSKbits05       0xC0

#define BitsPortA1      0x3F
#define BitsPortP4A1    0x3F
#define BitsPortA0      0x00
#define BitsPortAA      0xAA
#define BitsPortA5      0x55

#define BitsPortAEA             0x2A
#define BitsPortAE9     0x29
#define BitsPortAF2     0x32
#define BitsPortAF1     0x31
#define BitsPortAE7     0x27
#define BitsPortAEB             0x2B
#define BitsPortAF5     0x25
#define BitsPortAF6     0x36

#define BusyBitMask             0x01
#define FinTrajectoireBitMask   0x04

/**********************/
/* MACROS DU PILOTAGE */
/**********************/

/* Caractéristiques du robot */
```

```
#define largeur 0.15
#define Reducteur1 0.052083 /* 1/19.2 */
#define Reducteur2 0.1875  /* 6/32 */
#define DiametreRoue 0.08  /* 8 cm */
#define VitesseMax 0.25    /* 25cm/s */
#define MasseRobot 2.750   /* 2.75 kg */


/* macros de commande de l'automate de pilotage */

#define ATTENDRE          0xFA
#define lirecommande      0xEA
#define envoyercommande          0xEB
#define envoyerlesdonnees 0xEC
#define RESET1            0xF8
#define RESET2            0xF9
#define FIN               0xFB
#define lireinstructions  0xED
#define trajectoireterminee 0xEE

/* adresse de base */
#define adresse_de_base 0x6000

#define les_2_LM    0x03
#define LM_1              0x01
#define LM_2              0x02

/*  LD signifie ligne droite */
/*  PV signifie pivoter */
/*  AR signifie arc */

#define LD 0x00
#define PV 0x01
#define AR 0x02
#define ST 0x06

/* macros de selection des ports */
/* CSactif correspond à /CS à 0  */

#define CS1actif     0xFE
#define CS2actif     0xFD
#define PSactif      0xFB
#define RDactif      0xF7
#define WRactif      0xEF
#define CS1inactif   0x01
#define CS2inactif   0x02
#define PSinactif   0x04
#define RDinactif   0x08
#define WRinactif   0x10

/*declaration des ordres*/

#define CS1_PS_WR        (CS1actif & PSactif & WRactif)  //  Ecriture commande
LM629_1

#define CS2_PS_WR        (CS2actif & PSactif & WRactif)  //  Ecriture commande
LM629_2

#define CS1_PS_RD        (CS1actif & PSactif & RDactif)     //  Lecture commande
LM629_1
```

```
#define CS2_PS_RD       (CS2actif & PSactif & RDactif)  //  Lecture commande
LM629_2

#define CS1_WR          (CS1actif & WRactif)        //  Ecriture data LM629_1
#define CS2_WR          (CS2actif & WRactif)        //  Ecriture data LM629_2

#define CS1_RD          (CS1actif & RDactif)        //  Lecture data LM629_1
#define CS2_RD          (CS2actif & RDactif)        //  Lecture data LM629_2

/*************************************************************************/
/* définition des constantes et des instructions LM629                   */
/*************************************************************************/
/* Les noms des commandes sont celles utilisées dans la documentation */
/* de National Semiconductor intitulée "LM628 Programming Guide"     */

/* COMMANDES D'INITIALISATION */
#define RESET 0x00  /* Reset du LM629                                */
#define PORT8 0x05  /* Met la taille du port de sortie à 8 bits      */
#define DFH   0x02  /* La position courante devient la position absolue */

/* COMMANDES des LM */
/********************/
/* COMMANDES DE CONTROLE D'INTERRUPTION */
#define SIP   0x03  /* Set index position                            */
#define SBPA  0x20  /* Charge un breakpoint  (position absolue)      */
#define SBPR  0x21  /* Charge un breakpoint  (pôsition relative)     */
#define MSKI  0x1C  /* Masque les interruptions LM629 ---> 80C552    */
#define RSTI  0x1D  /* Rest les interruptions                        */
/* commandes de lecture */
#define RDRP        0x0A  /* lire la position reelle */
#define RDSIGS      0x0C /* lire le signal register */
#define RDIP        0x09 /* lire la distance enregistree */

/* COMMANDES DE CONTROLE DU FILTRE PID */

#define LFIL  0x1E  /* Charge les coef. du filtre (KP,KI,KD,IL)      */
#define UDF   0x04  /* Mise à jour du filtre                         */

#define KP_1  0x00,0x80
#define KI_1  0x00,0x02
#define KD_1  0x02,0x00
#define IL_1  0x7F,0xFF

#define KP_2  0x00,0x20
#define KI_2  0x00,0x01
#define KD_2  0x00,0x00
#define IL_2  0x00,0x00

/* COMMANDES DE CONTROLE DES TRAJECTOIRES */

#define LTRJ  0x1F  // Charge les paramètres de trajectoire
#define STT   0x01  // Commence le mouvement

/* F: fin de la feuille de route */
#define L 0x00        /* ligne droite */
#define P 0x01  /* pivoter */
#define A 0x02  /* arc */
#define F 0xFB  /* fin */
#define S 0x06




/* definition des dimensions des tableaux : route[] et tableau_commande[] */

#define TAILLE_ROUTE 100
#define TAILLE_TABLEAU_COMMANDE 100
#define TAILLE_NOMBRE   10
```

```
/*****************************************/
/*                                       */
/*      MAIN.H                           */
/*                                       */
/*   Apr. 2000 :                         */
/*     Vassan Karine, Chapouthier Julien */
/*                                       */
/*   Jul. 2000 :                         */
/*     Salvetti Djoume, Leroy Paul       */
/*                                       */
/*   Nov 2001 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*   Feb 2002 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*   Jul 2002:                           */
/*     Hessemans Anne-Caroline Mulot     */
/*       Fabien                          */
/*****************************************/

#ifndef __MAIN_H__
#define __MAIN_H__

/* variables globales extern*/
extern index;

extern maximum;
extern decalage;
extern valeurADCH;
extern Gain;

extern TAB_GAIN[4];//entré dans l'ordre decroissant de gain

extern P3_4;
extern P3_5;

extern trajectoirepivoter[];
extern trajectoirelignedroite[];
extern trajectoirearc[];
#endif
```

```
/*****************************************/
/*                                       */
/*     ROBOT.H                           */
/*                                       */
/*   Apr. 2000 :                         */
/*     Vassan Karine, Chapouthier Julien */
/*                                       */
/*   Jul. 2000 :                         */
/*     Salvetti Djoume, Leroy Paul       */
/*                                       */
/*   Nov 2001 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*   Feb 2003 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*****************************************/

#ifndef __ROBOT_H__
#define __ROBOT_H__

/***********************************************/
/*                                             */
/*            NAVIGATION                       */
/*                                             */
/* Execution de la feuille de route            */
/*                                             */
/* void navigation(u_char tab[])               */
/*                                             */
/* appel : tableau decrivant la trajectoire    */
/*                                             */
/* retour : void                               */
/*                                             */
/***********************************************/

void navigation(u_char tab[]);

/***********************************************/
/*                                             */
/*                PILOTAGE                     */
/*                                             */
/* pilotage des LM pour la trajectoire         */
/*                                             */
/* void pilote(u_char trajectoire[])           */
/*                                             */
/* appel : tableau decrivant la trajectoire    */
/*                                             */
/* retour : void                               */
/*                                             */
/***********************************************/

void pilote(u_char trajectoire[]);

#endif
```

```
/*****************************************/
/*                                       */
/*      SERIE.H                          */
/*                                       */
/*   JUILL. 2002 :                       */
/* A.Caroline Hessemans, Fabien Mulot    */
/*                                       */
/*****************************************/

/*prototypes des fonctions utilisees dans serie.c*/

/*********************************/
/* remplir_tampon()              */
/* libere le tableau route[]     */
/* en le memorisant dans un      */
/* tableau tampon                */
/*********************************/
void remplir_tampon(u_char route);

/*********************************/
/* recep_octet()                 */
/* lit le caracatere ASCII recu  */
/* sur le port série             */
/*                               */
/*********************************/

void recep_octet();

/*********************************/
/* remplir_route()               */
/* met en memoire dans route[]   */
/* le nouvel octet recu          */
/*                               */
/*********************************/
void remplir_route(u_char route);


/*********************************/
/* traduction                    */
/* permet de convertir le        */
/* tableau de ASCII en tableau   */
/* de double contenant les       */
/* parametres de la feuille      */
/* de route                      */
/*********************************/

void traduction();


/*****************************************/
/* cv_tab                                */
/* fonction traduisant la feuille de     */
/* route du robot en unités comprises    */
/* par le LM                             */
/*****************************************/

void cv_tab(u_char tab_out[]);
```

```
/*****************************************/
/*                                       */
/*      UTILS.H                          */
/*                                       */
/*   Apr. 2000 :                         */
/*     Vassan Karine, Chapouthier Julien */
/*                                       */
/*   Jul. 2000 :                         */
/*     Salvetti Djoume, Leroy Paul       */
/*                                       */
/*   Nov 2001 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*   Feb 2002 :                          */
/*     Pradel Gilbert                    */
/*                                       */
/*****************************************/

#ifndef __UTILS_H__
#define __UTILS_H__

/*********************************/
/*                               */
/* fonctions diverses            */
/*                               */
/*********************************/

/*********************************/
/*                               */
/* fonction d'attente            */
/*                               */
/*********************************/

void fonctionattente(void);

/***********************************************/
/* fonction de mise à 1 des bits du port 1     */
/***********************************************/
void initport1(void);

/***********************************************/
/* fonction de mise à 1 des bits du port 4     */
/***********************************************/
void initport4(void);

/*******************************************/
/* fonction de lecture du status byte      */
/*                                         */
/* u_char lirestatusbyte(u_char quelLM)    */
/*                                         */
/* appel : identite du LM concerne         */
/*                                         */
/* retour : status byte du LM concerne     */
/*                                         */
/*******************************************/

u_char lirestatusbyte(u_char quelLM);
```

```
/*****************************************************/
/* fonction d'ecriture d'une commande dans un LM     */
/*                                                   */
/* ecrirecommande(u_char commande,u_char quelLM)     */
/*                                                   */
/* appel : commande a ecrire                         */
/*         identite du LM concerne                   */
/*                                                   */
/*         Attention :                               */
/*                 si quelLM ==les_2_LM alors        */
/*                 la meme commande est              */
/*                 appliquee sur les 2 LM            */
/*                                                   */
/* retour : void                                     */
/*                                                   */
/*****************************************************/

void ecrirecommande(u_char commande,u_char quelLM);

/*****************************************************/
/* fonction de lecture des donnees dans un LM        */
/*                                                   */
/* liredonnees(u_char quelLM)                        */
/*                                                   */
/* appel : identite du LM concerne                   */
/*                                                   */
/*         Attention :                               */
/*                 si quelLM ==les_2_LM alors        */
/*                 la meme commande est              */
/*                 appliquee sur les 2 LM            */
/*                                                   */
/* retour : donnee issue du LM                       */
/*                                                   */
/*****************************************************/

u_char liredonnees(u_char quelLM);

/*****************************************************/
/*                                                   */
/* fonction d'ecriture des donneees dans un LM       */
/*                                                   */
/* appel : donnee a ecrire                           */
/*         identite du LM concerne                   */
/*                                                   */
/*         Attention :                               */
/*                 si quelLM ==les_2_LM alors        */
/*                 la meme donnee est                */
/*                 appliquee sur les 2 LM            */
/*                                                   */
/* retour : void                                     */
/*                                                   */
/*****************************************************/

void ecriredonnees(u_char donnees,u_char quelLM);
```

```
/*****************************************************/
/* fonction d'interrogation du busy_bit pour le LM concerne     */
/*                                                   */
/* u_char busy(u_char quelLM)                        */
/*                                                   */
/* appel : identite du LM concerne                   */
/*                                                   */
/* retourne VRAI si occupé                           */
/*          FAUX si non                              */
/*                                                   */
/*****************************************************/

u_char busy(u_char quelLM);

/*****************************************************/
/* fonction d'interrogation de fin de trajectoire    */
/*                                                   */
/* u_char fintrajectoire(u_char quelLM)              */
/*                                                   */
/* appel : identite du LM  concerne                  */
/*                                                   */
/* retour : VRAI si fini                             */
/*           FAUX sinon                              */
/*                                                   */
/*****************************************************/

u_char fintrajectoire(u_char quelLM);

/*****************************************************/
/* fonction d'interrogation du reset                 */
/*                                                   */
/* u_char reset(u_char reset1,u_char reset2)         */
/*                                                   */
/* appel :                                           */
/*                                                   */
/* retour :                                          */
/* FAUX=0                                            */
/*   VRAI=1                                          */
/*                                                   */
/*****************************************************/

u_char reset(u_char reset1,u_char reset2);

#endif
```

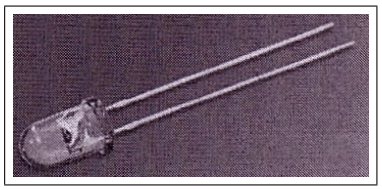
```
/*******************************************/
/*                                         */
/*                                         */
/*                                         */
/*   Jul. 2002                             */
/*   Mulot Fabien Hessemans Anne-Caroline  */
/*                                         */
/*                                         */
/*******************************************/
#ifndef _var_h_
#define _var_h_

extern u_int fin_route;              // 1: indique que route[] est totalement rempli
extern u_int fin_message;                    // 1: indique que message[] est plein
extern u_int nb_virgule;
extern u_char octet;                 // caractere ASCII recu sur le port serie
extern u_int nouvelle_commande;   // 1:indique que l'octet qui va etre memorise contient une
commande(A,P,L,F)
extern u_char commande_ascii;          // contient  la commande courante (L ou P ou A ou F)
extern u_int i_route;               // indice du tableau route[] OS=windows
extern u_int i_message;                                 // indice du tableau message[] OS=linux
extern u_int i_tampon;                                      // taille du tableau route
extern u_int octet_recu;
extern u_char route[TAILLE_ROUTE];      // tableau contenant les caracteres ascii recus sur le port
serie
extern float tableau_commande[TAILLE_TABLEAU_COMMANDE];  // feuille route a convertir
en donnees LM629
extern u_char tampon[TAILLE_ROUTE];      // copie de route[] servant a liberer route[] en vue
d'une nouvelle reception
extern char message[3];

#endif
```

Annexe E
Composants utilisés : Doc. Technique

Diode émettrice

SFH485

Features

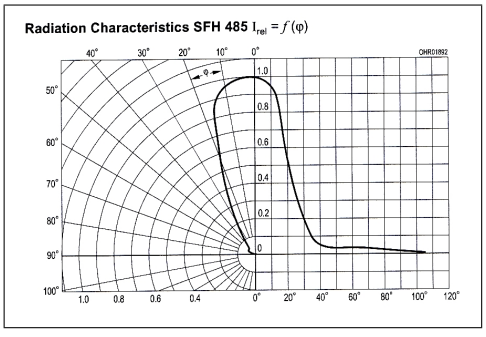


- Very highly efficient GaAlAs-LED
- High reliability
- Spectral match with silicon photodetectors
- Available on tape and reel
- Available in bins

Applications

- IR remote control of Hi-Fi and TV-sets, video tape recorders, dimmers
- Remote control for steady and varying intensity
- Smoke detectors (UL-approval)
- Sensor technology
- Discrete interrupters

Parameter	Symbol	Value	Unit
Reverse voltage	V_r	5	V
Forward current	I_F	100	mA
Surge current	I_{FSM}	2,5	A
Power dissipation	P_{tot}	200	mW
Wavelength at peak emission	λ_{peak}	880	nm
Half angle	φ	20	deg.



Phototransistor

phototransistors au silicium NPN

BPX 25

Phototransistors planar épitaxiés, en boîtier métallique SOT-29/1 avec lentille frontale pour le BPX 25 ou SOT-29/2 avec fenêtre plane pour le BPX 29.

Leur grande sensibilité les destine aux applications de détection à faible niveau et l'herméticité du boîtier les recommande plus spécialement pour des environnements difficiles dans les domaines professionnel et militaire associés à leurs homologues CQY 49B et CQY 49C.

CARACTERISTIQUES PRINCIPALES

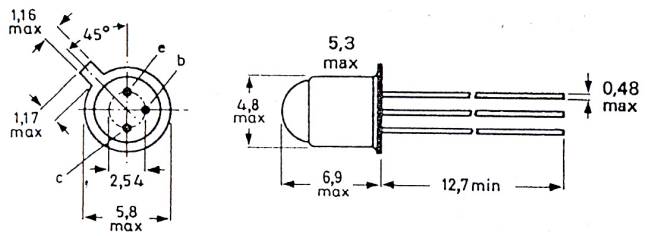
Tension collecteur-émetteur (base ouverte)		V_{CEO}	max	32	V
Courant collecteur en continu		I_C	max	100	mA
Courant d'obscurité $V_{CE} = 24$ V; $E = 0$		I_{CEO}	max	100	nA
Courant collecteur en éclairement à 1000 lux $V_{CE} = 6$ V	BPX 25	$I_{C(L)}$	min	4	mA
	BPX 29	$I_{C(L)}$	min	0,2	mA
Température de jonction		T_j	max	150	°C
Longueur d'onde du pic de réponse spectrale		λ_p	typ	800	nm
Angle de réceptivité	BPX 25	θ	typ	±7,5	°
	BPX 29	θ	typ	±30	°

DONNEES MECANIQUES

Dimensions en mm

BOITIER SOT 29/2
avec lentille

BPX 25



BOITIER SOT 29/1
avec fenêtre plane

BPX 29

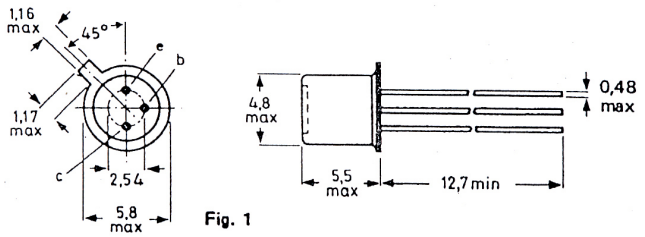


Fig. 1

VALEURS A NE PAS DEPASSER (limites absolues selon publication CEI 134)

Tensions				
Tension collecteur-base (émetteur ouvert)	V_{CBO}	max	32	V
Tension collecteur-émetteur (base ouverte)	V_{CEO}	max	32	V
Tension émetteur-base (collecteur ouvert)	V_{EBO}	max	5	V
Courants				
Courant collecteur en continu	I_C	max	100	mA
Courant collecteur (valeur crête)	I_{CM}	max	200	mA
Puissance				
Puissance totale dissipée ($T_{amb} \leqslant 25°C$)	P_{tot}	max	300	mW
Températures				
Température de stockage	T_{stg}		− 65 à + 150	°C
Température de jonction	T_j	max	150	°C

RESISTANCES THERMIQUES

Jonction-ambiance	$R_{th\ j\text{-}a}$	400	K/W
Jonction-boîtier	$R_{th\ j\text{-}c}$	150	K/W

CARACTERISTIQUES

$T_{amb} = 25°C$ sauf indication contraire

			BPX 25	BPX 29	
Courant d'obscurité collecteur					
$V_{CE} = 24$ V; $E = 0$	I_{CEO}	typ	10	10	nA
		max	100	100	nA
$V_{CE} = 24$ V; $E = 0$; $T_j = 100°C$	I_{CEO}	typ	10	10	µA
		max	100	100	µA
Courant collecteur en éclairement de 1000 lx[1]					
$V_{CE} = 6$ V	$I_{C\ (L)}$	min	4	0,2	mA
		typ	10	0,6	mA
Gain en courant continu					
$I_C = 2$ mA; $V_{CE} = 6$ V	h_{FE}	typ	500	500	
Fréquence de coupure [2]	f_{co}	typ	220	200	kHz
Temps de commutation[2]					
Temps de retard à la montée	t_d	typ	1	2,5	µs
		max	3	5	µs
Temps de montée	t_r	typ	1,5	2,5	µs
		max	3	5	µs
Temps de stockage	t_s	typ	0,2	0,2	µs
		max	0,5	0,5	µs
Temps de décroissance	t_f	typ	1,5	3,5	µs
		max	4	8	µs
Angle de demi-sensibilité	θ	typ	± 15	± 40	°
Longueur d'onde du pic de réponse	λ_p	typ	800	800	nm

1) Lampe à filament de tungstène $T_c = 2856$ K
2) Source au GaAs modulée : 0,4 mW/cm² : Résistance de charge optimale : 50 Ω; $V_{CE} = 24$ V

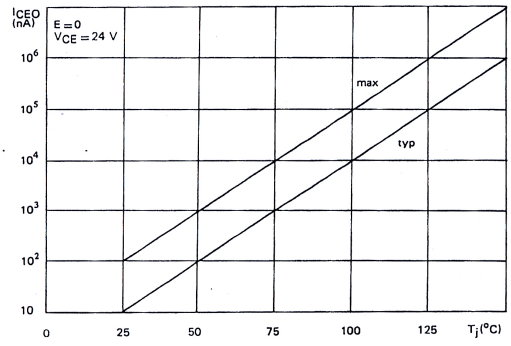


Fig. 8.5 – Caractéristique du courant en fonction de la température

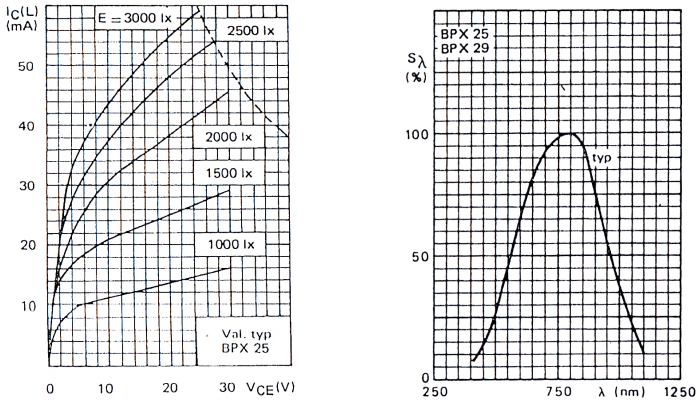


Fig. 8.6 – Caractéristiques courant-tension du phototransistor et sensibilité spectrale

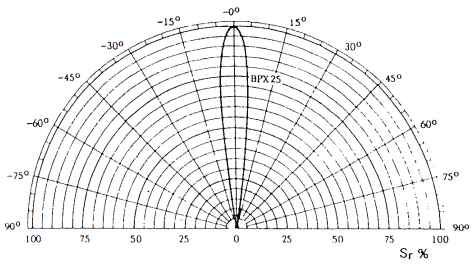


Fig. 8.7 – Réceptivité du phototransistor

MAX232

MAXIM

+5V-Powered, Multichannel RS-232 Drivers/Receivers

General Description

The MAX220-MAX249 family of line drivers/receivers is intended for all EIA/TIA-232E and V.28/V.24 communications interfaces, particularly applications where ±12V is not available.

These parts are especially useful in battery-powered systems, since their low-power shutdown mode reduces power dissipation to less than 5μW. The MAX225, MAX233, MAX235, and MAX245/MAX246/MAX247 use no external components and are recommended for applications where printed circuit board space is critical.

Applications

Portable Computers
Low-Power Modems
Interface Translation
Battery-Powered RS-232 Systems
Multi-Drop RS-232 Networks

Features

Superior to Bipolar

- **Operate from Single +5V Power Supply (+5V and +12V-MAX231/MAX239)**
- **Low-Power Receive Mode in Shutdown (MAX223/MAX242)**
- **Meet All EIA/TIA-232E and V.28 Specifications**
- **Multiple Drivers and Receivers**
- **3-State Driver and Receiver Outputs**
- **Open-Line Detection (MAX243)**

+5V-Powered, Multichannel RS-232 Drivers/Receivers

ELECTRICAL CHARACTERISTICS MAX220/222/232A/233A/242/243

(V_{cc}=+5V ±10%, C1-C4=0.1μF, MAX220, C1=0.047μF, C2-C4=0.33μF, T_a=T_{min} to T_{max}, unless otherwise noted.)

Parameter	Conditions	Min	Typ	Max	Units
RS-232 Transmitters					
Output voltage Swing	All transmitter outputs loaded with 3kΩ to GND	±5	±8		V
Input Logic Threshold Low Input			1.4	0.8	V
Logic Threshold High		2	1.4		V
Logic Pull-Up/lnput Current			5	40	μA
Transmitter Output Resistance	V_{cc}=V+=V-=0V, V_{out}=±2V	300	10M		Ω
Output Short-Circuit Current	V_{out}=0V	±7	±22		mA
RS-232 Receivers					
RS-232 Input Voltage Operating Range				±30	V
Input Threshold Low	V_{cc}=5V	0.8	1.3		V
Input Threshold High	V_{cc}=5V		1.8	2.4	V
RS-232 Input Hysteresis	V_{cc}=5V	0.2	0.5	1	V
RS-232 Input Resistance		3	5	7	kΩ
TTL/CMOS Output Voltage Low	I_{out}=3.2mA		0.2	0.4	V
TTL/CMOS Output Voltage High	I_{out}=-1.0mA	3.5	VCC- 0.2		V
TTL/CMOS Output Short-Circuit Current	Sourcing V_{out}=GND Shrinking V_{out}=V_{cc}	-2 10	-10 30		mA

Package

MAX220/232/232A

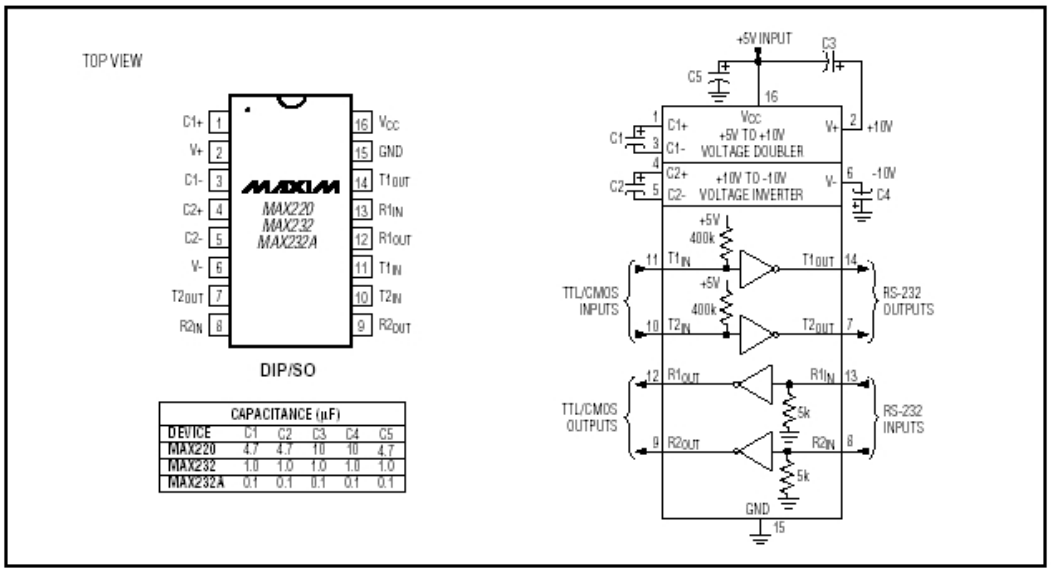


Fig. 8.8 – MAX220/MAX232/MAX232A Pin Configuration and Typical Operating Circuit