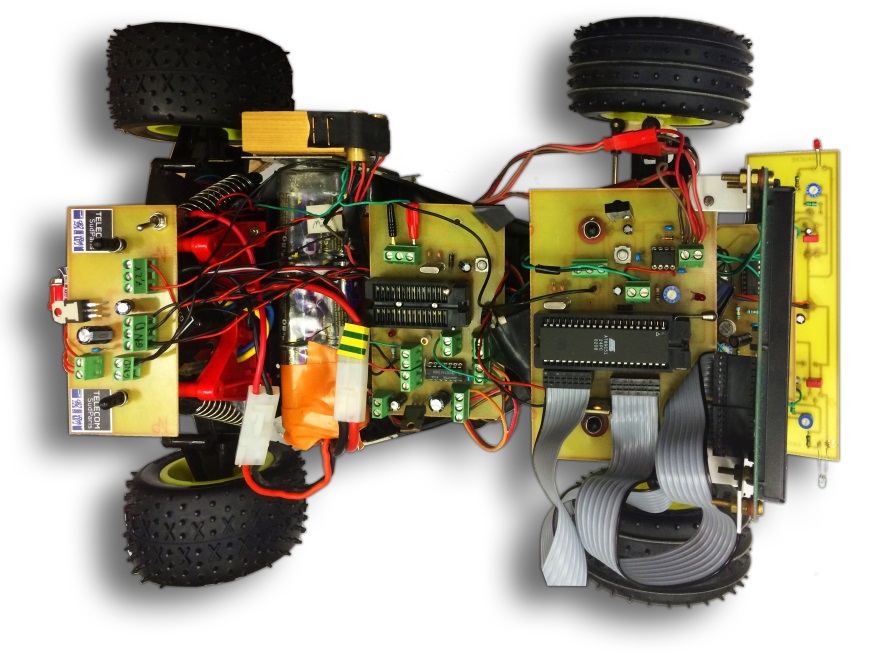
sudparis_cont_pant.eps

P1 : INTEGRATION DES SYSTEMES ELECTRONIQUES

COMPTE RENDU TECHNIQUE

EQUIPE N°1 : « VROUM »

PROJET EFFECTUE DANS LE CADRE DU CURSUS INGENIEUR DE 2EME ANNEE DE TELECOM SUDPARIS



|  |  |
| --- | --- |
| Pierre-François Gimenez | Pierre Bourré |
|  |  |
| François Boutigny | Clément Christomanos |
|  |  |

Table des matières

aperçu du projet

Présentation

Objectif

L’objectif de ce projet est l’élaboration d’un mobile autonome, capable de suivre une piste tout en s’en écartant le moins possible.

Il doit effectuer trois tours chronométrés, ajoutant ainsi un objectif de vitesse au projet.

Par ailleurs, il est muni d’un dispositif de réception infrarouge qui va lui permettre de communiquer avec deux balises situées sur la piste.

Présentation du circuit

Le circuit, large de 14 cm, est composé de trois bandes : une bande noire centrale bordée de deux bandes blanches, toutes les bandes étant de largeur identique. La ligne de départ est figurée par une bande noire transversale. À côté de cette ligne de départ se trouve une balise (dite balise de départ) émettant un signal infrarouge permettant de repérer le nombre de tours effectués et d’arrêter le véhicule lorsque l’objectif de trois tours est atteint.

La piste en elle-même présente des tournants, et se situe sur un milieu plat. Une de ses portions est droite. Sur cette portion se situe une autre balise infrarouge (dite balise cible ou balise caractérielle), qui va émettre des signaux variés selon une interaction avec le véhicule que nous allons décrire ci-après. Ces signaux doivent être traités par le système.

Le véhicule

Le véhicule est piloté par l’intermédiaire de quatre cartes électroniques :

– Une carte de détection de piste. Il s’agit de manière basique d’une diode infrarouge émettant un signal vers la piste. Un capteur infrarouge analyse la réflexion du signal sur la piste afin de déterminer si la portion de piste détectée est blanche ou noire. La carte de détection est munie de deux capteurs de ce type (à gauche et à droite). La détection du noir et du blanc renvoient un état logique bas pour noir et haut pour blanc. Nous avons choisi de détecter l'extérieur de la piste (sol) comme du blanc, car on peut modifier cette couleur par le réglage de potentiomètres associés aux capteurs.

– Une carte d’alimentation. Son rôle est de réguler la tension délivrée par l’une des deux batteries du véhicule (l’autre alimentant le moteur), et de la répartir vers les autres cartes, en 7.2V et en 5V régulé.

– Une carte maître, qui gère la vitesse et la direction. Son fonctionnement détaillé sera abordé dans une partie dédiée de ce rapport.

– Une carte esclave, qui s’occupe de fonctionnalités annexes : la gestion d’un laser, d’une sirène et d’un écran LCD. Nous détaillerons également son fonctionnement dans une partie dédiée.

Par ailleurs, le mobile comprend :

– Un moteur de propulsion à courant continu. Ce moteur est piloté par un variateur de puissance. La consigne en vitesse est donnée par un signal particulier, le PWM (*Pulse Width Modulation*, Modulation de la largeur d’impulsion en français). Il s’agit d’un signal d'une période de 20 ms, dont environ 1,5 ms à l'état haut et le reste à l'état bas. De cette manière, on définit le rapport cyclique comme étant le rapport, au sein d’une même période du signal, entre le temps passé à l’état haut et le temps passé à l’état bas. Ce rapport cyclique permet de piloter la consigne en vitesse du variateur de puissance.

– D’un servomoteur permettant de piloter la direction (roues avant). De manière analogue au moteur, la consigne en direction est pilotée par un signal PWM de même période (20 ms).

– D’un capteur de vitesse : Il s’agit d’un ensemble composé d’une diode infrarouge et d’un récepteur analysant la réflexion du signal sur une roue. Cette roue est solidaire du moyeu des roues de propulsion, et possède des bandes alternant entre le blanc et le noir. Il s’agit du principe de la roue codeuse : en comptant chaque seconde le nombre de passages de noir à blanc (front montant), on obtient une mesure de vitesse.

communication entre les éléments du circuit et le véhicule

Comme nous l’évoquions, le véhicule dispose d’un émetteur laser.

Ce laser ne doit pas être mis en route pendant toute la durée du circuit, mais seulement en présence de la balise. Ce laser va interagir avec la balise cible.

Lorsque le laser émis par le véhicule frappe la balise cible, celle-ci retourne une lettre de l’alphabet, envoyée sous forme de caractère ASCII.

– Tir au centre (C)

– Tir à gauche (G)

– Tir à droite (D)

– Aucun signal reçu (Caractère 4)

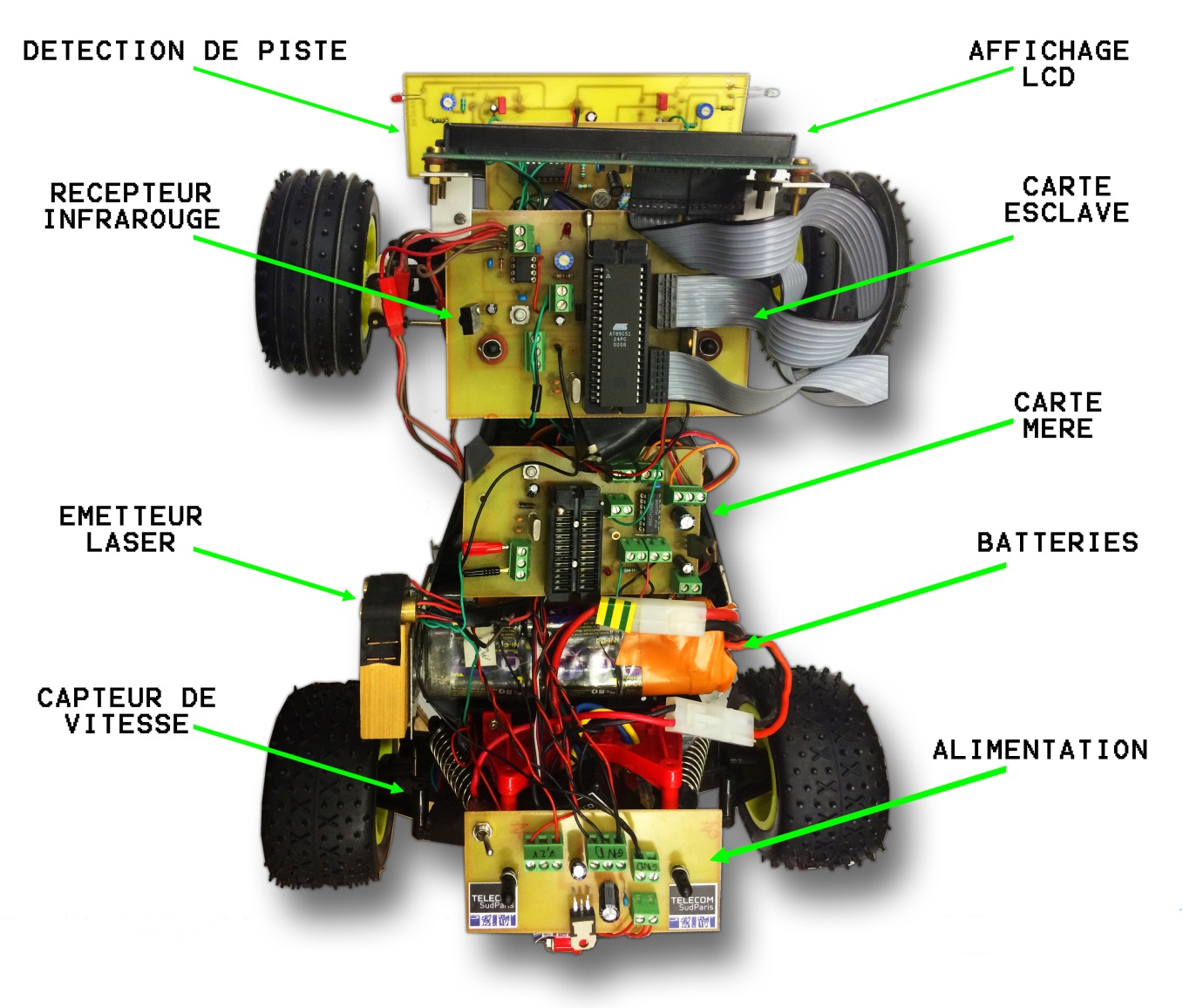
Un capteur infrarouge situé sur le véhicule permet de recevoir le signal laser émis par la balise. Un écran LCD situé sur le véhicule permet d’afficher le résultat du tir, en fonction de la lettre reçue.

Il faut noter la présence d’une sirène, qui permet d’annoncer l’émission laser et le fait que celui-ci est en cours de fonctionnement, dans le but d’avertir les personnes aux alentours.

aspects techniques

Nous avons précédemment donné un premier aperçu du projet, en donnant les objectifs. Nous allons à présent détailler notre travail : l’évolution du fonctionnement du véhicule jusqu’à son état actuel, ainsi que les différents problèmes ou contraintes qui se sont présentés avec les solutions que nous avons choisies.

Notre groupe a décidé d'exploiter au maximum les ressources à disposition. A ce titre, quelques points techniques peuvent être considérés plus comme des présentations de faisabilité technique que d'aide véritable au véhicule (multithreading et affichage animé d'un pacman par exemple).



Carte Maître

Fonctions réalisées

* Gestion de la consigne en direction et en vitesse (génération du signal PWM)
* Pilotage en direction et en vitesse en fonction de la détection de la piste
* Stratégie de changement de vitesse
* Asservissement en vitesse grâce à la roue codeuse
* Gestion des boutons poussoirs (augmentation et diminution de vitesse)
* Mode d'économie d'énergie

contraintes

La carte maître prend la décision de la direction et de la vitesse. Elle élabore pour cela une consigne.

Nous pouvons régler la consigne en vitesse par appui sur les boutons poussoirs situés sur les côtés du véhicule.

Le moteur de la voiture est capable de l’entraîner extrêmement rapidement. Nous sommes donc dans l’obligation de le brider électroniquement et de régler précisément la consigne en vitesse via le signal de PWM, afin d’éviter un emballement incontrôlé du véhicule.

Deux signaux PWM sont à générer simultanément : l’un pour la direction, l’autre pour la vitesse.

Il faut faire attention à ne pas faire travailler trop longtemps les servomoteurs en butée.

PWM de direction et de vitesse

La boucle principale du code, qui couvre presque tout le code (hormis l'initialisation) dure 2,5ms. Elle est donc exécutée 8 fois par cycle PWM (qui dure 20ms). C'est la valeur pointée par le registre R0 qui détermine la valeur du PWM (R0 pointera sur la direction une fois sur huit, sur la vitesse une fois sur huit, et sur une valeur nulle six fois sur huit).

Le fait d'avoir une boucle de faible durée permet d'augmenter la précision temporelle de la détection de la ligne noire. Il ne serait pas impossible de réduire encore cette boucle, mais ceci n'a pas été jugé rentable, au vu de la complexité de cette opération et au vu des maigres avantages qu'elle pouvait apporter (la fréquence de rafraîchissement des capteurs, qui est de 400 Hz, est bien suffisante).

Une attention particulière a été apportée à la durée du PWM. Celui-ci est affecté d'une durée précise, entre 1ms et 2ms, avec un pas de 4µs, ce qui est largement suffisant pour cette application.

Les timers mis à disposition par le microcontrôleur ne sont jamais arrêtés ou relancés, et aucun calcul n'est effectué pour leur affecter une valeur. Le programme a été calibré de manière à ce que la gestion du temps soit très précise. Des commentaires dans les sources précisent ce point.

PILOTAGE DE LA DIRECTION EN FONCTION DE LA DETECTION DE PISTE

L’algorithme de pilotage de la direction fonctionne de la manière suivante :

On se dirige toujours vers le dernier endroit où on a vu du noir. Toutes les 50ms, on tourne un peu plus. Si on n'a pas vu de ligne noire depuis 250ms, on considère qu'on l'a perdue et on braque dans la direction dans laquelle on l'a vue pour la dernière fois.

Dès qu'on détecte une ligne noire, on met les roues droites.

Voici comment se comporte la voiture en ligne droite et en virage.

* En ligne droite : le véhicule suit la ligne noire. S'il en écarte, il modifie légèrement sa direction et retrouve rapidement la ligne noire. Ainsi, il y a peu d'oscillations.
* En virage : le virage est détecté un peu tard car le système ne braque qu'au bout de 250ms. Ainsi, il peut sortir un peu de piste.

Ces deux comportements sont à équilibrer. Si on attend plus avant de braquer, alors il n'y aura presque pas d'oscillations en ligne droite et on pourra aller plus vite. Par contre, on ne pourra une sortie de route avant de braquer car la détection du virage sera tardive. Si on attend moins avant de braquer, il prendra très bien les virages mais risque, en ligne droite, de braquer à tort et à travers et d'osciller.

Les valeurs obtenues sont donc expérimentales et nous semblaient être un bon compromis.

Stratégie de changement de vitesse

La stratégie de changement de vitesse est simple : quand on est sur du noir, c'est-à-dire a priori en ligne droite, on accélère jusqu'à atteindre une valeur maximale, et on retrouve une vitesse de base dès qu'on perd la ligne.

Le fait d'accélérer progressivement permet de ne pas avoir une sorte d'impulsion lorsque le mobile sort traverse rapidement la ligne. Ainsi, on est relativement sûr que l'accélération se fait en ligne droite.

Asservissement en vitesse grâce à la codeuse

L'asservissement en vitesse prévu est assez simple, puisqu'il se contente d'accélérer si le véhicule s'arrête. Plus exactement, le timer1 est utilisé en compteur sur une entrée extérieure (T1) qui est relié au circuit auxiliaire. Sa valeur est lue toutes les 500ms. Ce laps de temps assez grand permet au timer1 d'avoir des valeurs peu bruitées. Cela signifie également qu'entre deux changements de vitesse s'écoulent au moins 500ms, ce qui nous paraissait raisonnable.

Malheureusement, nous avons remarqué que le système de codeuse était assez peu fiable. En effet, les fixations des éléments formant le capteur (c'est-à-dire la diode d'émission et le récepteur) devaient être positionnées de manière précise. Or, lors des aléas mécaniques inhérents à un véhicule, le système pouvait ne plus détecter la vitesse du mobile. Or, dans ce cas, la stratégie dit qu'il faut accélérer. Autant d'un point de vue sécurité que performance, cela nous semblait être un trop grand risque. C'est pour cela que nous n'avons finalement pas utilisé l'asservissement en vitesse dans le rendu final.

Gestion des boutons poussoirs

Comme proposé, un appui du bouton poussoir de droite accélère tandit que le bouton poussoir de gauche ralentit la voiture. Le code détecte un front descendant avec des opérations logiques en se souvenant de l'état des boutons à l'état précédent, sans utiliser d'interruptions.

Ces boutons poussoirs modifient la vitesse basique du véhicule ; par contre, la vitesse maximale que le véhicule pouvait atteindre en ligne droite n'est pas modifiée par les boutons poussoirs pour des raisons de sécurité.

Mode économie d'énergie

Dès que c'est possible, le microcontrôleur passe en mode d'économie d'énergie (idle). Des tests ont été effectués : sur RIDE, on peut estimer qu'environ 95 % du temps, le microcontrôleur est en mode d'économie d'énergie.

Conclusion de la carte maître

Le fonctionnement de la carte maître tel quel nous paraît satisfaisant. Il existe toujours des voies d'améliorations, notamment pour l'asservissement en vitesse qui est resté basique, par manque de temps.

Nous avions également eu des idées plus complexes concernant le traitement des capteurs de ligne, mais elles s'avérèrent peu efficaces au final.

Carte Esclave

Fonctions réalisées

* Répartition du programme sur deux threads
* Communication série avec la balise
* Activation des lasers et de la sirène
* Affichage LCD d'un PAC-MAN animé
* Mode économie d'énergie et utilisation de la ROM

principe de fonctionnement

La carte esclave a pour fonction dans ce système de :

 – Recevoir le signal IR envoyé par la balise

 – Commander l’émission laser et la sirène

 – Commander l’écran LCD

Au moment où l’on détecte la présence de la balise (détection d’un caractère ‘4’ sur la liaison série), on active la sirène, puis après une temporisation, le laser, qui en touchant la cible entraîne un changement du message de la balise. Celle-ci indique si : on a tiré à gauche, au centre, à droite, ou nulle part sur la balise. Le signal IR de la balise varie en conséquence et c’est un message équivalent que nous devons afficher sur l’écran LCD.

Notre première approche a été d’attendre la réception du signal IR sur le récepteur (liaison série à 1200 bauds). Alors, nous entrions dans un système de comparaison de l’octet reçu avec les valeurs prévues : ‘4’, ‘C’, ‘D’, ‘G’, en prenant en compte les bits de parité inhérents au fonctionnement de la balise.

Alors le système activait le laser s’il recevait le caractère ‘4’. À chaque fois, il demandait l’émission d’un message sur l’écran LCD, correspondant au cas de comparaison. Quant à l’extinction du laser, nous avons eu recours à une temporisation, qui s’activait au moment de l’émission. Nous attendions une certaine durée, au bout de laquelle (par interruption), nous demandions l’extinction du laser. En effet, comme la voiture peut être amenée à capter le signal ‘4’ à n’importe quel moment, nous ne pouvions donc pas nous reposer simplement sur l’idée qu’on attendait un deuxième ‘4’ pour signifier au système qu’il avait dépassé la balise.

Ce principe de fonctionnement a toutefois changé avec le recours au multithreading.

Threads sommaires

Un mécanisme de threads sommaire a été mis en place. D'une part parce que c'était un défi technique, et d'autre part parce que le développement de la partie LCD et de la partie balise + laser + sirène a été mené de manière indépendante. Fusionner les deux aurait pu être long et fastidieux, alors que les threads permettaient de simplifier la tâche.

Chaque thread possède sa propre pile (de 16 mots chacune, ce qui est largement suffisant) et sa propre banque. Le passage d'un thread à l'autre passe par une interruption de timer1, qui overflow toutes les 25ms. De plus, ces threads offrent deux outils : un mutex et une procédure « sleep ».

Le mutex est nécessaire car deux octets sont partagées entre les deux threads, et qu'il faut attendre l'écriture complète avant de les utiliser. Les mutex sont possibles grâce à l'instruction « jbc » qui permet à la fois de tester un bit et de le modifier de manière atomique. Le mutex de ce programme est utilisé via les procédures « lock » et « unlock ». Un mécanisme de protection a été mis en place et empêche par exemple de déverrouiller une ressource utilisée par l'autre thread ou de s'attendre soi-même (en cas de deux « lock » consécutifs dans un même thread, par exemple).

Le fait de proposer une procédure « sleep » est important pour deux raisons. La première est qu'il n'y a plus de timer disponible pour les threads, qui ne peuvent donc les utiliser pour temporiser. La seconde qu'il aurait été possible que, pendant qu'un thread attend, l'autre travaille. Ce dernier point n'a pas été mis en place car trop peu rentable. En effet, les seules temporisations nécessaires sont des temporisations de 50ms. Il a donc été décidé que pour faire dormir un thread, on mettrait le microcontrôleur en mode idle pendant le reste de son cycle (près de 25ms) et qu'on laisserait l'autre thread agir (encore 25ms). Cette temporisation n'est pas précise mais est suffisante dans le cas des threads de la carte esclave.

Commentaire sur l'absence de communication série esclave-maître

La série entre l'esclave et le maître nous paraissait à la fois peu utile, au vu des données échangées, et encombrante, car elle nécessite un timer dédié de chaque côté. Or, la carte maître avait déjà ses deux timers pris (un pour compter le PWM, l'autre pour l'asservissement en vitesse), sans possibilité d'en passer un en mode 3 (deux timers en un).

Finalement, il y a bien une communication entre les deux cartes, mais il s'agit seulement de deux pins reliées ensemble afin de faire passer un bit d'information. Ce bit est le bit de démarrage : la carte esclave, recevant le signal de la balise de départ, prévient la carte maître qu'il faut partir, et lui ordonne de s'arrêter une fois les trois tours effectués.

Mode économie d'énergie et utilisation de la ROM

De même que la carte maître, le mode d'économie d'énergie est utilisée, notamment dans la procédure « sleep » définie pour les threads. Plus de 90 % du temps le microcontrôleur est en mode « idle ».

Le code de la carte esclave est assez conséquent et utilise une grande partie de la ROM (mémoire programme), en partie dû aux chaînes de caractères affichées. Plus de 95 % des 1024 octets de la ROM sont utilisés.

affichage LCD d'un PAC-MAN animé

L'affichage d'un pac-man animé relevait plus d'une preuve de la faisabilité technique que d'un réel besoin du véhicule.

Conclusion de la carte esclave

Nous sommes particulièrement satisfaits du système de threads mis en place sur les programmes de cette carte. Ce système a en effet permis de simplifier de manière importante le travail en équipe sur cette carte, en séparant clairement l’affichage LCD et la gestion de l’émission / réception laser.

Conclusion

RESULTATS DE LA COURSE

Le véhicule a correctement suivi la ligne durant le premier tour. Au passage devant la balise cible, le véhicule a correctement tiré au laser. Cependant, un bug (apparemment dans la gestion du multi-threading) sur la carte esclave est apparu, et a empêché le fonctionnement ultérieur de la carte esclave. Le laser est donc resté bloqué en émission, et l’écran LCD est resté sur son dernier affichage, ce jusqu’à la fin de l’expérience.

Par la suite, le véhicule est sorti de la piste. Un rayon de soleil a en effet perturbé la détection de la piste. Malgré la modulation du signal infrarouge, la puissance du signal est trop faible face aux rayons du soleil. Il est néanmoins re-rentré seul sur le circuit et a poursuivi comme prévu ses trois tours. De par le dysfonctionnement de la carte esclave, le mobile n’a pas été arrêté automatiquement au bout des trois tours comme convenu.

Durant notre seconde tentative, le système a parfaitement fonctionné, du départ automatique jusqu’à l’arrêt au bout des trois tours (effectués de manière plutôt rapide). La cible a systématiquement été atteinte par le laser. La sirène est manifestement endommagée car elle n’a fonctionné qu’à un volume sonore très faible (il s’agit peut-être d’un problème de placement). De même, le rétro-éclairage de notre écran LCD semble avoir souffert de quelques problèmes indépendants du programme.

Ce résultat a été d’abord source de déception – il a été rageant de constater qu’un dysfonctionnement jusqu’alors inconnu puisse apparaître lors de l’évaluation finale. Néanmoins, nous n’avions pas pu tester la configuration « réelle » avant cette évaluation. Une part d’incertitude est inexorablement introduite par la position variable des différentes balises, les reflets du soleil sur le sol, voire même les imperfections dans le tracé de la piste, âgée de plusieurs années.

Une fois cette déception passée, nous avons été particulièrement fiers de constater le fonctionnement de notre système et l’accomplissement du challenge que représentait ce projet, compte-tenu des contraintes temporelles.

Il reste un enseignement que nous avons acquis au cours de cette dernière expérience : le système le plus fiable et le plus robuste est parfois le plus simple. Nous avions envisagé la complexité technique comme un challenge, et l’accomplissement de tâches techniquement complexes était source de motivation au sein du projet. Cependant, cette motivation nous a parfois fait oublier le vieil adage selon lequel « le mieux est l’ennemi du bien », ce qui a pu conduire à un système moins fiable que ce que nous aurions voulu. Toutefois, il est également possible de se dire que ces perturbations extérieures sont inévitables et qu’une légère part de la réussite des objectifs du véhicule est laissée au hasard.

BILAN DU PROJET

A l’aube de notre spécialisation d’ingénieurs, nous avons tous été ravis de participer à cette période P1 « Intégration des systèmes électroniques ». Elle nous a permis de toucher à des problèmes pratiques, là où le tronc commun du cursus ingénieur a pu nous confiner trop longtemps à la théorie. Nous possédions tous les quatre une appétence particulière pour l’électronique, ce qui a été à l’origine de notre choix pour cette période P1. Celle-ci n’a fait que confirmer notre goût pour ce domaine de l’ingénierie.

ANNEXES

Codes sources

Code de la carte maître

; VROUM PROJECT / CARTE MAITRE

; Le code est fait de manière à ce que la partie pour commander la direction soit la même que pour commander le moteur.

; La différenciation se fait au niveau de la valeur de R0 (utilisé en adressage indirect)

; Utilisations des registres:

; R0: pointe vers un registe contenant (temps d'état haut en microsecondes - 1000) / 4

; Les autres registres sont des labels explicites

; Mémo des valeurs de directions:

; Entre 28 (droite) et 222 (gauche). Tout droit: 125

; La diode est éteinte quand le mobile soit être arrêté (avant le début et après la fin des trois tours), allumée sinon.

; Asservissement:

; Un asservissement en vitesse est effectué. Pour cela, le timer 1 compte sur P3.5, qui est relié à la diode.

; Régulièrement, la valeur de ce timer est vérifiée. La décision d'augmenter la vitesse ou de la diminuer est alors prise selon cette valeur. Enfin, le timer est remis à zero.

PIN\_DIR bit P1.4 ; pin PWM de la direction

PIN\_MOTEUR bit P1.5 ; pin PWM du moteur

CAPT\_D bit P1.6 ; le capteur gauche

CAPT\_G bit P1.7 ; le capteur droit

DIODE bit P1.0 ; la diode

ATTEND bit P3.0 ; ce bit est partagé avec la carte esclave

BP0 bit P1.1 ; le bouton poussoir gauche

BP1 bit P1.2 ; le bouton poussoir droit

NOIR\_DROIT bit 00h ; vrai si on a vu la ligne noire pour la dernière fois à droite, faux si c'était à gauche

VITESSE equ 01b ; la vitesse actuelle du mobile

DIRECTION equ 10b ; la direction actuelle du mobile

VITESSE\_NORMALE equ R2 ; la vitesse hors accélération

ATTENTE\_ASSERV equ R3 ; compteur

BP\_MEM equ R4 ; sauvegarde de l'état des boutons poussoirs pour détecter un front descendant

BP\_MEM\_ADR equ 0Ch ; utilisé pour POP (F4 de la banque 1)

DIVERGE equ R5 ; de combien en tourne, en valeur absolue

COMPTE\_FIN equ R6 ; temporisateur d'arrêt final

org 0000h

jmp init

; ne pas mettre d'autres interruptions qui pourraient

; perturber le fonctionnement du mode idle

; Réveil par cette interruption (mode idle)

org 000Bh

clr TF0

reti

org 0030h

;---------------------------

; Initialisation

init:

clr PIN\_DIR ; on éteint la pin du moteur et de la direction

clr PIN\_MOTEUR

setb DIODE

boucle\_debut: ; on attend la feu vert de la carte esclave pour partir

; on peut bypasser cette attente avec

; les boutons poussoirs

jnb BP0, bypass\_debut

jnb BP1, bypass\_debut

jb ATTEND, boucle\_debut

jmp go

bypass\_debut:

clr ATTEND

go:

clr DIODE ; on allume la diode de la carte

clr NOIR\_DROIT ; nécessaire si reset

setb RS0 ; utilisation de la banque 1

; ASSERVISSEMENT

mov DIRECTION, #125 ; roues droites

mov VITESSE\_NORMALE, #174

mov VITESSE, VITESSE\_NORMALE

mov ATTENTE\_ASSERV, #20 ;2.5ms \* 20 = 50ms

; entre deux changements de puissance délivrée au moteur

mov COMPTE\_FIN, #40

; TIMER

mov TMOD, #01010001b ; initialisation du timer 0 (mode 16bits)

; et du timer 1 (mode 16bits, avec horloge externe) et démarrage

setb TR0

setb TR1

setb EA ; activation de l'interruption de timer0

setb ET0

mov SP, #0Fh ; afin de ne pas écraser la banque 1

nop ; équilibrage de timer 0

nop

; ne rien écrire ici

demiBouclePWM:

; --------------------------

; PWM

; on s'occupe du moteur si R0 vaut 01b, de la direction si R0 vaut 10b

; il ne faut pas faire d'embranchement conditionnels sinon timer 0 n'aura pas toujours la même valeur

mov A, R0

mov C, ACC.0 ; on allume le moteur si R0 vaut 001b

anl C, /ACC.1

anl C, /ACC.2

mov PIN\_MOTEUR, C

mov C, ACC.1 ; on change de direction si R0 vaut 010b

anl C, /ACC.0

anl C, /ACC.2

mov PIN\_DIR, C

; attente de 1ms.

mov TL0, #2Eh

mov TH0, #0FCh

inc PCON ; DODO

; attente de 1ms encore.

mov TL0, #00Bh

mov TH0, #0FCh

; R0 pointe soit vers vitesse, soit vers direction, soit vers 0

mov A, @R0

jz etat\_bas

; ne rien écrire ici, sinon le cas où @R0=0 serait décalé de quelques microsecondes, ce qui pourrait faire exploser le véhicule.

boucleElementaire: ; durée d'une boucle élémentaire: 4microsecondes

nop

nop

djnz ACC, boucleElementaire

etat\_bas:

clr PIN\_MOTEUR

nop ; afin que l'écart temporel entre les deux clr soit le même qu'en les deux mov

nop

nop

nop

nop

nop

clr PIN\_DIR

inc PCON ; DODO

; on lance le timer pour 0.5ms. Pendant ce temps,

; on fait le reste (asservissement, boutons, capteurs, ...).

mov TL0, #026h

mov TH0, #0FEh

; ----------------------

; gestion des capteurs et de la direction

mov C, NOIR\_DROIT

anl C, /CAPT\_G ;si on capte à gauche

; on met NOIR\_DROIT à 0. Sinon, on ne change rien.

orl C, CAPT\_D ;si on capte à droite

;on met NOIR\_DROIT à 1. Sinon, on ne change rien.

mov NOIR\_DROIT, C

jb CAPT\_G, gaucheOuDroite

jb CAPT\_D, gaucheOuDroite

jmp niGaucheNiDroite

gaucheOuDroite:

; on a détecté d'un côté : on va tout droit et on accélère

mov DIRECTION, #125

mov DIVERGE, #0

mov A, VITESSE\_NORMALE

add A, #5

mov VITESSE, A

jmp fin\_direction

niGaucheNiDroite:

tourne:

; on a pas détecté de bande noire ? On tourne

jb NOIR\_DROIT, tourne\_droite

; on tourne à gauche

mov A, #125

add A, DIVERGE

mov DIRECTION, A

jmp fin\_direction

tourne\_droite:

; on tourne à droite

mov A, #125

clr C

subb A, DIVERGE

mov DIRECTION, A

fin\_direction:

; ----------------------

; gestion des boutons poussoirs

mov C, BP0

mov ACC.1, C

mov C, BP1

mov ACC.2, C

push ACC

xrl A, BP\_MEM ; on vérifie que l'état a changé (qu'on soit sur un front)

anl A, BP\_MEM ; on vérifie que la précédente valeur est 1. Ces deux lignes détectent donc un front descendant.

rr A

rrc A

jnc pas\_bp0

dec VITESSE\_NORMALE ; si BP0 est enfoncé, on ralentit

jmp pas\_bp1

pas\_bp0:

rrc A

jnc pas\_bp1

inc VITESSE\_NORMALE ; si BP1 est enfoncé, on accélère

pas\_bp1:

pop BP\_MEM\_ADR ; on actualise la mémoire

fin\_bp:

; --------------------

; Asservissement vitesse et direction

djnz ATTENTE\_ASSERV, pasEncore

mov ATTENTE\_ASSERV, #20

mov A, DIVERGE

cjne A, #80, change\_diverge

jmp dejaTrop

change\_diverge:

inc A

inc A

cjne A, #10, dejaTrop ; on augmente l'angle petit à petit

; (de 2 en 2 jusqu'à 10), puis on braque

mov A, #80 ; on braque

mov VITESSE, VITESSE\_NORMALE

dejaTrop:

mov DIVERGE, A

; asservissement vitesse

mov A, TL1 ; on récupère le nombre de fragments de tours de roues, qu'on réinitialise au passage

mov TL1, #0

clr C

subb A, #3d

jmp pas\_bloque ; l'asservissement est désactivé par ce JMP

jnb ACC.7, pas\_bloque

; TL1 < 3

mov VITESSE, #180d ; BOOST!

jmp pasEncore

pas\_bloque:

mov VITESSE, VITESSE\_NORMALE

pasEncore:

; ----------------------

; C'est la fin! On attend une seconde avant de s'arrêter.

jnb ATTEND, pasFin

djnz COMPTE\_FIN, pasFin

mov COMPTE\_FIN, #1

mov VITESSE, #125

pasFin:

; ----------------------

; fin gestion du PWM

inc R0

anl 08h, #111b ; on tourne sur 8 registres

inc PCON

; DODO

jmp demiBouclePWM

end

Code de la carte ESCLAVE

;-----------------------------------------------

; VROUM PROJECT / CARTE ESCLAVE

;-----------------------------------------------

; Deux proto-threads sont utilisés sur cette carte :

; Le thread 0 s’occupe de la communication avec le capteur infrarouge et de commander les lasers. Sa pile commence à 60h.

; Le thread 1 s’occupe de l’affichage LCD. Sa pile commence à 40h.

; Le passage de l’un à l’autre se fait de la manière suivante :

; Partant de l’un des deux threads : à chaque fois que le timer1 met à 1 le drapeau d’overflow (toutes les 25ms), son interruption effectue un push du contexte (de l’ancien thread), modifie le SP (vers l’autre thread) puis effectue un pop du contexte (de l’autre thread). Ainsi, tout se passe comme si les deux threads étaient exécutés en parallèle.

; Un mécanisme de mutex est mis en place pour les données partagées (ADRESSE\_H et ADRESSE\_L) entre les deux threads. Ce mécanisme permet de contrôler l’accès à ces données (en vérifiant qu’un thread ne la modifie pas alors que l’autre l’utilise).

;-----------------------------------------------

; VARIABLES :

;-----------------------------------------------

; Variables de gestion des threads :

DIODE bit P1.0 ; diode de contrôle

ATTEND bit P3.2 ; bit partagé avec la carte maître

MUTEX bit 00h ; 0 si la variable est occupée, 1 sinon. Le fonctionnement est détaillé plus tard.

DODO\_FINI bit 02h ; Utilisé pour éviter à PCON de se réveiller non pas à cause d'un overflow de timer1 (comme on aimerait) mais par une autre interruption

COMPTEUR\_ON bit 03h ; Compteur supplémentaire qui permet d'attendre 6s entre deux changements de tour.

AUTRE\_SP equ 2h ; Emplacement du SP pour l’autre thread

NB\_TOURS equ 33h ; Nombre de tours effectués.

QUI\_UTILISE equ 34h ; Identification de celui qui a utilisé la procédure lock

COMPTEUR equ 35h ; Registre de comptage.

; Variables de gestion du laser :

Laser bit p1.2 ; Demande de l’émission laser si 1.

Sirene bit p1.3 ; Enclenchement de la sirène si 1.

; Variables de l’affichage LCD :

RS bit p0.5 ; Indique le type de données échangées :

; RS=0 : instruction.

; Rs=1 : donnée.

RW bit p0.6 ; bit qui indique:

; RW=1 : lecture (read).

; RW=0 : écriture (write).

E bit p0.7 ; validation des données en entrée. Actif sur front descendant.

LCD equ p2 ; Bus de données de l’afficheur.

Busy bit p2.7 ; Drapeau de fin d’exécution

; D’une commande :

; BUSY=0 terminé.

; BUSY=1 en cours.

; Variables partagées :

; Ces variables vont contenir l’adresse référençant les différents messages à afficher sur le LCD (première et deuxième ligne).

ADRESSE\_H equ 30h

ADRESSE\_L equ 31h

;--------------------------------------------------------------------------------------------------------------------------

; INTERRUPTION : à exécuter au reset.

;--------------------------------------------------------------------------------------------------------------------------

org 0000h

mov ADRESSE\_H, #03h ; Adresse 338h. Regarder à la toute fin.

mov ADRESSE\_L, #38h

setb PT0 ; Priorité au timer0 (qui gère les threads) sur la série en l’occurrence.

jmp init

;-----------------------------------------------

; INTERRUPTION : gestion des threads (par interruption sur TIMER0).

;-----------------------------------------------

org 000Bh ; programme d’interruption TIMER0

jmp gestion\_threads

;-----------------------------------------------

; INTERRUPTION : gestion de la liaison série

;-----------------------------------------------

org 0023h ; interruption série.

interrupt\_serie\_laser :

; /!\ Dans cette interruption, le contexte est inconnu.

push PSW ; sauvegarde.

push ACC ; sauvegarde.

mov PSW, #10h ; Sélection de banque 2. Ce changement de banque permet surtout d'avoir son propre identifiant dans le système de mutex, qui se base sur le numéro de banque

clr RI ; raz obligatoire.

mov A, SBUF

call lock ; programme de verrouillage, car on va utiliser les variables communes.

cjne A, #047h, pas47 ; a-t-on détecté "G"? Sinon, on saute.

mov 01h, #20 ; On relance l’attente de fin de laser pour 1s. 01h : R1 du thread 0.

mov adresse\_H, #03h ; Adresse 310h pour « tir à gauche. »

mov adresse\_L, #10h

jmp fin\_interrupt\_serie

pas47:

cjne A, #044h, pas44 ; a-t-on détecté "D"? Sinon, on saute.

mov 01h, #20 ; on relance l’attente de fin de laser pour 1s. 01h: R1 du thread 0

mov adresse\_H, #02h ; Adresse 2E8h pour « tir à droite. »

mov adresse\_L, #0E8h

jmp fin\_interrupt\_serie

pas44:

cjne A, #0C3h, pas43 ; a-t-on détecté "C"? Sinon, on saute.

mov 01h, #20 ; On relance l’attente de fin de laser pour 1s. 01h: R1 du thread 0

mov adresse\_H, #02h ; Adresse 2C0h pour « tir au centre. »

mov adresse\_L, #0C0h

jmp fin\_interrupt\_serie

pas43:

cjne A, #0B4h, pas34 ; a-t-on détecté "4"? Sinon, on saute.

mov 01h, #20 ; On relance l’attente de fin de laser pour 1s. 01h: R1 du thread 0

mov adresse\_H, #02h ; Adresse 298h pour « Détection. »

mov adresse\_L, #98h

jmp fin\_interrupt\_serie

pas34:

cjne A, #030h, pas30 ; a-t-on détecté "0"?

jb COMPTEUR\_ON, fin\_interrupt\_serie ; si compteur actif, on passe ce qui suit.

setb COMPTEUR\_ON ; on a détecté ‘0’, on active un compteur.

dec NB\_TOURS ; on décrémente les tours.

mov ADRESSE\_H, #03h ; cf. messages à la fin pour le nombre de tours.

mov A, NB\_TOURS ; NB\_TOURS = 4, 3, 2, 1 ou 0 (cf. initialisation du programme principal).

jnz pasFini ; A non nul : la course n’est pas finie !

setb ATTEND ; Sinon, on prévient la carte maître qu'il faut s'arrêter

jmp debute

pasFini:

clr DIODE

clr ATTEND ; La course n'est pas finie, on prévient la carte maître

debute:

; L’adresse pour NB\_TOURS = 0 est 0360h

; L’adresse pour NB\_TOURS = 1 est 0360h + 28h = 0388h

; L’adresse pour NB\_TOURS = 2 est 0360h + 2\*28h = 03B0h

; L’adresse pour NB\_TOURS = 3 est 0360h + 3\*28h = 03D8h

mov B, #28h

mul AB

add A, #60h ; on ajoute au produit 60h, ce qui donne l’octet de poids faible de l’adresse du message (puisqu’il n’y a pas de retenue)

mov ADRESSE\_L, A

pas30: ; par défaut on ne change pas le message

fin\_interrupt\_serie:

call unlock ; programme de déverrouillage.

pop ACC

pop PSW

reti ; retour d’interruption et se trouve APRÈS la boucle d’attente : attente\_laser.

;-----------------------------------------------

; PROGRAMME PRINCIPAL : initialisation.

;-----------------------------------------------

init:

; Pas besoin de mutex car les threads ne sont pas encore lancés.

mov NB\_TOURS, #4 ; nous utilisons un système de démarrage à distance. Le bolide détecte donc au départ la balise, et passe ensuite cette valeur à 3

clr COMPTEUR\_ON ; Désactivation du compteur.

mov DPTR, #debut\_lcd ; On appelle le thread1 pour activer le LCD.

; Enregistrement dans la pile à 40h :

mov 40h, DPL ; octet de poids faible

mov 41h, DPH ; octet de poids fort

; Enregistrement pour le thread1 de l’utilisation de la banque 1 :

mov 42h, #08h ; là où est censé se trouver PSW, disant qu’il utilise la banque 1.

; Remplissage de la case censée contenir A :

;mov 43h, #00h

; Remplissage du SP actuel avec la valeur 5Fh. Il passe ainsi à 60h à l’utilisation.

mov SP, #5Fh

; AUTRE\_SP référence la dernière case de la pile remplie par l’autre thread, ici thread1 :

mov AUTRE\_SP, #43h

; Mutex, au début disponible

setb MUTEX

; Initialisation de timer0 (mode 1, 16 bits) :

mov TMOD, #1b

mov TL0, #058h

mov TH0, #09Eh

; Autorisation des interruptions et lancement du timer0 :

setb EA

setb ET0

setb TR0

;-----------------------------------------------

; PROGRAMME PRINCIPAL : le code du THREAD0 (utilisant la banque 0).

; GESTION DE L’ÉMISSION LASER

;-----------------------------------------------

init\_laser:

; Extinction générale de sécurité :

clr laser ; car ports mis à un par défaut.

clr sirene

; Réglages des timers :

orl tmod, #100000b ; timer 1 : auto-remplissage 8 bits.

mov scon, #01010000b ; Mode 1, Ren a 1, ri=0, ti=0.

setb ES ; Autorisation de l’interruption série.

mov th1, #230d ; Réglage pour 1200 bauds.

mov tl1, #230d ; Car mode de rechargement.

setb tr1 ; Enclenchement timer.

setb REN ; Validation de réception.

mov R1, #0 ; R1 : compteur d’attente. Il sera mis à #20 par l'interruption de série

attente\_laser: ; 20x50ms=1s.

lcall sleep\_50ms

cjne R1, #20, attente\_laser

; /!\ s’il y a une interruption série concernant le laser, due à la réception d’un caractère, on sautera à l’instruction qui suit :

setb sirene ; activation de la sirène.

lcall sleep\_50ms ; la tempo de 50ms.

setb laser ; enclenchement du laser.

boucle\_attente\_laser:

lcall sleep\_50ms

djnz R1, boucle\_attente\_laser ; tant qu'on reçoit des caractères pour le laser, R1 est remis à #20 et le boucle continue

clr laser ; extinction du laser.

call lock

mov ADRESSE\_H, #03h ; Adresse 338h. Message par défaut.

mov ADRESSE\_L, #38h

call unlock

lcall sleep\_50ms ; Tempo avant extinction sirène.

clr sirene

jmp attente\_laser

;-----------------------------------------------

; PROGRAMME PRINCIPAL : le code du THREAD1 (utilisant la banque 1).

; GESTION DE L’ÉCRAN LCD

;-----------------------------------------------

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : initialisation du LCD :

;-----------------------------------------------

init\_lcd:

lcall sleep\_50ms

mov lcd,#38h ; Affiche sur 2 lignes en 5x8 points

lcall en\_lcd\_code ; Sous-programme de validation d’une commande

lcall sleep\_50ms

mov lcd,#0Ch ; Allumage de l’afficheur

lcall en\_lcd\_code ; Sous-programme de validation d’une commande

lcall sleep\_50ms

mov lcd,#01h ; Effacement de l’affichage

lcall en\_lcd\_code ; Sous-programme de validation d’une commande

lcall sleep\_50ms

mov lcd,#06h ; Incrémente le curseur

lcall en\_lcd\_code ; Sous-programme de validation d’une commande

mov lcd,#38h ; Affiche sur 2 lignes en 5x8 points (sécurité)

lcall en\_lcd\_code ; Sous-programme de validation d’une commande

ret

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : validation de l’envoi d’une instruction, avec vérification de l’état du BUSY FLAG

;-----------------------------------------------

en\_lcd\_code: ; Sous-programme de validation d’une instruction

clr rs ; 5 lignes = séquence permettant de valider l’envoi d’une instruction au LCD/

clr rw

clr E

setb E

clr E

lcall test\_busy\_lcd ; Appel au Sous-programme de test de l’état d’occupation du LCD

ret

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : test du busy flag pour envoi d’autres instructions ou données.

;-----------------------------------------------

test\_busy\_lcd: ; Test de la valeur du BUSY FLAG renvoyée sur DB7 par le LCD.

mov lcd,#0ffh ; Déclaration du port de communication avec LCD en lecture

setb rw ; 2 lignes pour autoriser la lecture de BF.

clr rs

setb E ; Bf doit être lu entre un front montant et un front descendant de E

check\_busy:

jb busy,check\_busy ; BF = 1 LCD occupé, BF = 0 LCD libre

clr E

ret

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : validation de l’envoi d’un caractère avec vérification de l’état du BUSY FLAG.

;-----------------------------------------------

en\_lcd\_data:

setb rs ; 5 lignes = séquence permettant de valider l’envoi d’une instruction au LCD.

clr rw

clr E

setb E

clr E

lcall test\_busy\_lcd ; Appel au sous-programme de test de l’état d’occupation du LCD.

ret

;--------------------------------------------------------------------------------------------------------------------------

; PROGRAMME PRINCIPAL THREAD1 :

;--------------------------------------------------------------------------------------------------------------------------

debut\_lcd:

lcall init\_lcd ; Initialisation du LCD.

mov R1,#18h

lcall CGRAM\_lcd ; Ajout des caractères de Pacman dans la CGRAM.

restart\_lcd:

mov lcd, #02h ; Se place à l’adresse 00h de la DDRAM et remet le curseur à son origine.

lcall en\_lcd\_code

mov R3, #13h ; On recommence la boucle mange\_lcd 19 fois avant de recommencer.

mange\_lcd:

mov R1, #28h ; Registre permettant de sélectionner le nombre de caractère à rentrer en mémoire pour la 1ère ligne.

lcall LINE1\_lcd ; Donne l’état de la détection de la balise.

mov R1, #28h

mov DPTR, #map\_lcd ; Copie de la carte pour le pacman dans DPTR.

lcall LINE2\_lcd ; Met la carte du pacman sur la ligne 2.

mov DPTR,#pacman\_open\_lcd ; Copie le caractère du pacman avec la bouche ouverte dans DPTR.

mov R1,#02h

lcall LINE2\_lcd\_pacman ; Affiche le pacman avec la bouche ouverte sur la ligne 2.

lcall sleep\_50ms

lcall sleep\_50ms

mov R1, #28h

lcall LINE1\_lcd ; Donne l’état de la détection de la balise.

lcall sleep\_50ms

lcall sleep\_50ms

mov R1, #28h

lcall LINE1\_lcd ; Donne l’état de la détection de la balise.

mov DPTR,#pacman\_close\_lcd ; Copie le caractère du pacman avec la bouche fermée dans DPTR.

mov R1,#02h

lcall LINE2\_lcd\_pacman ; Affiche le pacman avec la bouche ouverte sur la ligne 2.

lcall sleep\_50ms

lcall sleep\_50ms

mov R1, #28h

lcall LINE1\_lcd ; Donne l’état de la détection de la balise.

lcall sleep\_50ms

lcall sleep\_50ms

lcall decale\_lcd ; Décale l’affichage vers la gauche.

djnz R3, mange\_lcd ; On boucle pour animer le pacman.

jmp restart\_lcd ; Quand le pacman arrive à la fin, on recommence l’animation.

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : affichage sur la première ligne.

;-----------------------------------------------

LINE1\_lcd:

call lock ; on affiche le message décidé par le thread 0. On n'oublie pas de lock et d'unlock avant et après utilisation des variables communes

mov A, ADRESSE\_H

mov DPH, A

mov A, ADRESSE\_L

mov DPL, A

call unlock

mov lcd,#80h ; Se place au début de la première ligne.

lcall en\_lcd\_code

lcall boucle\_ligne\_lcd

ret

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : affichage sur la deuxième ligne.

;-----------------------------------------------

LINE2\_lcd:

mov lcd,#0C0h ; Se place au début de la seconde ligne

lcall en\_lcd\_code

lcall boucle\_ligne\_lcd

ret

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : affichage sur la deuxième ligne (bis).

;-----------------------------------------------

LINE2\_lcd\_pacman:

mov lcd,#0D0h ; Se place à la fin de la seconde ligne.

lcall en\_lcd\_code

lcall boucle\_ligne\_lcd

ret

decale\_lcd:

mov lcd,#18h ; Incrémente le curseur.

lcall en\_lcd\_code

;-----------------------------------------------

; SOUS-PROGRAMME THREAD1 : boucle.

;-----------------------------------------------

boucle\_ligne\_lcd: ; Permet d’envoyer le code des caractères

clr A

movc A, @A + DPTR

mov lcd, A

lcall en\_lcd\_data

inc DPTR

djnz R1, boucle\_ligne\_lcd

ret

;--------------------------------------------------------------------------------------------------------------------------

; SOUS-PROGRAMME THREAD1 : CGRAM.

;--------------------------------------------------------------------------------------------------------------------------

CGRAM\_lcd:

mov lcd,#40h ; Se place à l’adresse 00h de la CGRAM

lcall en\_lcd\_code

mov DPTR,#pacman\_skin\_lcd

lcall boucle\_ligne\_lcd

ret

;--------------------------------------------------------------------------------------------------------------------------

; LES TEXTES À ENVOYER (THREAD1) :

;-----------------------------------------------

map\_lcd:

db 0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,0A5h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h,20h

pacman\_skin\_lcd:

db 18h,1Ch,1Eh,1Fh,1Fh,1Eh,1Ch,18h,03h,07h,06h,03h,01h,03h,07h,03h,03h,07h,0Eh,1Fh,1Fh,0Fh,07h,03h

pacman\_open\_lcd: db 01h, 00h

pacman\_close\_lcd: db 02h, 00h

;-----------------------------------------------

; SOUS-PROGRAMME DE GESTION : procédure lock.

;-----------------------------------------------

lock:

push ACC

boucle\_lock:

mov A, PSW

anl A, #00011000b ; on ne garde que la banque, qui sert de fiche d’identité à un thread

jbc MUTEX, fin\_lock ; le choix de cette instruction n’est pas innocent. C’est une des rares instructions à fournir à la fois un test et une affectation. De ce fait, ces deux étapes d’acquisition du mutex sont atomiques et ne peuvent être interrompues par un changement de thread qui pourrait tout ruiner.

; Si on est ici, c’est que la variable n’est pas disponible

clr C

subb A, QUI\_UTILISE

jz fin\_lock ; Dans ce cas, c’est que c’est ce même thread qui utilise déjà la ressource

call sleep\_50ms

jmp boucle\_lock

fin\_lock:

mov QUI\_UTILISE, A

pop ACC

ret

;-----------------------------------------------

; SOUS-PROGRAMME DE GESTION : procédure unlock.

;-----------------------------------------------

unlock:

push ACC

mov A, PSW

anl A, #00011000b

clr C

subb A, QUI\_UTILISE ; QUI\_UTILISE identifie le propriétaire du mutex

jnz fin\_unlock ; C’est l’autre qui l’utilise, on ne peut pas libérer la ressource

setb MUTEX

fin\_unlock:

pop ACC

ret

;-----------------------------------------------

; SOUS-PROGRAMME DE GESTION : procédure sleep.

;-----------------------------------------------

sleep\_50ms:

clr DODO\_FINI

boucle\_dodo:

inc PCON

jnb DODO\_FINI, boucle\_dodo ; afin d’être réveillé par la bonne interruption

ret

;-----------------------------------------------

; PROGRAMME PRINCIPAL : gestion des threads.

;-----------------------------------------------

gestion\_threads:

push PSW

push ACC

mov TL0, #058h ; on repart pour 25ms

mov TH0, #09Eh

mov A, SP

xch A, AUTRE\_SP ; on échange les SP entre les deux threads

mov SP, A

jnb COMPTEUR\_ON, pasCompteur ; gestion du compteur pour la balise de départ

inc COMPTEUR

mov A, COMPTEUR

jnz pasCompteur

clr COMPTEUR\_ON

pasCompteur:

setb DODO\_FINI

pop ACC

pop PSW ; le changement de banque se fait au chargement de PSW

reti

;-----------------------------------------------

; MESSAGES À L’ÉCRAN LCD :

;-----------------------------------------------

org 298h

db '.: DETECTION! :.: DETECTION! :.: DETECTI'

org 2C0h

db '.: TIR CENTRE :.: TIR CENTRE :.: TIR CEN'

org 2E8h

db '.: TIR DROITE :.: TIR DROITE :.: TIR DRO'

org 310h

db '.: TIR GAUCHE :.: TIR GAUCHE :.: TIR GAU'

org 338h

db 'SUPER VROUM! SUPER VROUM! SUPER VROUM!! '

org 360h

db 'C', 27h,'EST FINI! C', 27h, 'EST FINI! C', 27h, 'EST FINI! C', 27h, 'ES'

org 388h

db 'PLUS QUE 1 TOUR! PLUS QUE 1 TOUR! PLUS Q'

org 3B0h

db 'ENCORE 2 TOURS! ENCORE 2 TOURS! ENCORE 2'

org 3D8h

db 'DEBUT DES 3 TOURS! DEBUT DES 3 TOURS! DE'

end