Rapport de stage

DESS Isi filière Itra
Essi
Sophia-Antipolis

Programmation d'une centrale de contrôle-commande à base de Power-PC

Inria Rhône-Alpes
Service Robotique, Vision et Réalité Virtuelle

Fabien Lydoire

Mai - Septembre 2000

Table des matières

Table des figures

Liste des tableaux

Remerciements

Je tiens tout d'abord à remercier Gérard Baille, mon responsable de stage, pour ses conseils et le le temps qu'il a bien voulu me consacrer tout au long de ce stage. Il m'a patiemment expliqué les concepts et les techniques rencontrés que je ne connaissais pas. C'est grâce à lui et au travail proposé que ce stage a été si formateur.

Je souhaite aussi remercier toute l'équipe du Service Robotique, Vision et Réalité Virtuelle de l'Inria Rhône-Alpes: Hervé Mathieu, Pascal Di-Giacomo et Soraya Arias pour leur aide lorsque certains problèmes se sont présentés.

Première partie

Le stage

Chapitre 1

Introduction

Ce stage intitulé "Programmation d'une centrale de contrôle-commande à base de Power-PC" s'est déroulé au sein du Service Robotique, Vision et Réalité Virtuelle de l'Institut National de Recherche en Informatique et en Automatique (Inria) Rhône-Alpes.

Il s'agit d'étudier les différentes fonctionnalités d'une carte électronique basée sur un micro-contrôleur MPC555 de Motorola. Pour cela, il faudra étudier le fonctionnement du micro-contrôleur, mettre en place les outils logiciels de développement, puis réaliser une application afin d'illustrer le fonctionnement de cette carte.

Ce document rapporte dans la première partie mon expérience acquise lors de ce stage, que ce soit dans la mise en place d'outils logiciels, ou dans l'interfaçage des capteurs et actionneurs avec la carte.

La seconde partie, que l'on peut considérer comme un rapport technique, est consacrée à une description détaillée de mon travail.

Chapitre 2

Description du travail proposé

Le rôle du Service Robotique est de mettre en œuvre des outils matériels et logiciels pour les expérimentations robotiques des projets de recherche du site. En particulier, l'Inria dispose d'un véhicule électrique.

2.1 Le CyCab

FIG. 2.1: *le CyCab*

Les chercheurs de l'Inria et de l'Inrets (Institut National de Recherche sur les Transports et leur Sécurité) travaillent depuis 1991 sur de nouveaux moyens de transport intelligent pour la ville. Ils étudient en particulier le concept du libre-service et celui de la voiture automatique. Les premiers résultats de recherche ont débouché sur le projet Praxitèle (1993-1999), qui était en exploitation à Saint-Quentin-en-Yvelines. Les partenaires industriels du projet étaient CGFTE (la filiale transports publics de Vivendi), Dassault Electronique, EDF et Renault.

Dans le cadre du projet Praxitèle l'Inria a démontré la faisabilité de la conduite automatique sous certaines conditions : créneau et train de véhicule expérimentés sur une Ligier électrique instrumentée à cet effet. Pour des raisons

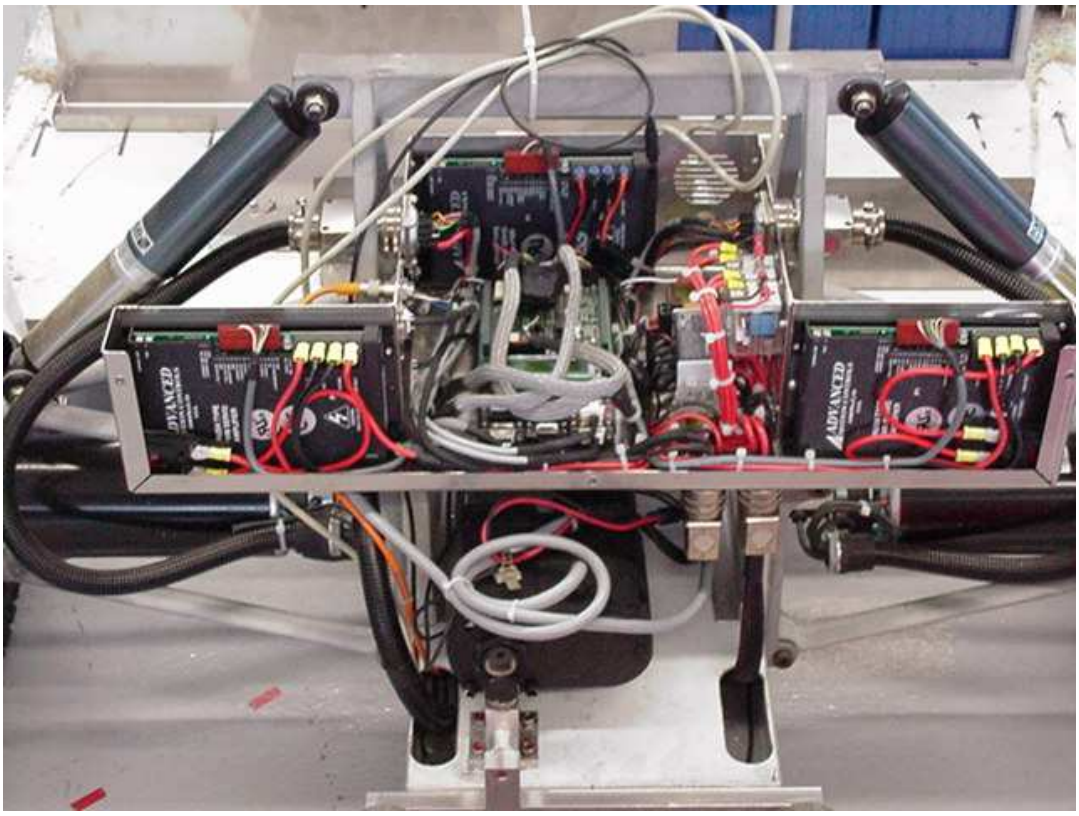

FIG. 2.2: *Nœud à base de MPC555 : vue d'ensemble dans le CyCab*

de législation et de responsabilité ces automatismes de conduite n'ont pas pu être implémentés sur les Clio électriques de Saint-Quentin-en-Yvelines.

Le CyCab (figure 2.1) a ensuite été développé par l'Inria avec l'aide de l'Inrets, de EDF, de la RATP et de la société Andruet S.A. pour montrer le potentiel de l'informatique dans la conduite de véhicules.

Le CyCab est un véhicule électrique à quatre roues motrices et directrices avec une motorisation indépendante pour chacune des roues et pour la direction. Pour contrôler et commander les différents moteurs du CyCab, l'architecture est répartie autour d'un bus de terrain CAN[1]. Dans une première version, des nœuds basés sur le micro-contrôleur MC68332 (cœur 68020) étaient connectés sur ce bus. Le CyCab a ensuite évolué, et les nœuds ont été remplacés par de nouvelles cartes à base de MPC555 (cœur Power-PC) contruites par la société Virtual Micro Design. Le CyCab est maintenant commercialisé par la société Robosoft.

On peut voir sur la figure 2.2 le nœud placé à l'avant du CyCab autour de trois amplificateurs de puissance (commande de deux moteurs de traction et d'un moteur de direction). La figure 2.3 présente un agrandissement de la partie "intelligence".

1. Controller Area Network

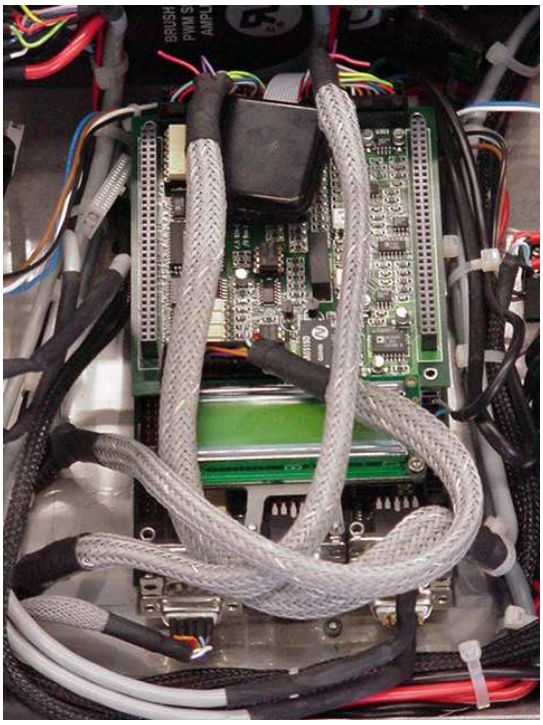

FIG. 2.3: *Nœud à base de MPC555 : vue détaillée*

Dans la suite de ce document on appellera cette carte "nœud Robosoft".

2.2 Le stage

L'objet du stage est d'évaluer le choix technologique concernant l'évolution du CyCab en réalisant une application autour du micro-contrôleur MPC555 pour faire l'acquisition de données capteurs (proximètres à ultrasons, microphone, codeurs de position moteur ...) et commander les 2 moteurs d'une tourelle pan/tilt. Le Service Robotique dispose d'une carte d'évaluation pour MPC555 et des outils de développement et de *debug* pour le 555 de SDS[2]. Ce travail se décompose en plusieurs étapes:

- Installation et prise en main de la carte d'évaluation MPC555 et des outils de développement logiciel associés ;
- Prise en main des différents capteurs et actionneurs ;
- Mise en place des interfaces (matérielle et logicielle) entre les capteurs/actionneurs et la carte d'évaluation ;
- Écriture des programmes de base d'entrées/sorties,
- Réalisation d'une application pour illustrer et évaluer l'utilisation du MPC555.

2. Software Development Systems

L'application visée est l'asservissement de l'orientation de la tourelle en fonction de la détection d'un son (localisation angulaire) afin de mesurer la distance entre émetteur et récepteur.

Au niveau du micro-contrôleur, cette application permettra de tester les acquisitions analogiques, les bus CAN, les liaisons séries et la conversion numérique analogique.

Ce stage implique le suivi d'un projet de sa conception à son implémentation, et l'intégration des composants matériels et logiciels complexes.

Mon choix s'est porté sur ce stage car il intègre toutes les étapes de développement d'une application de contrôle-commande à base de micro-contrôleur. Le fait que ce stage s'inscrive dans un projet de robotique mobile au sein de l'Inria m'a aussi beaucoup attiré car je souhaite poursuivre mes études par un DEA dans ce domaine de recherche.

Chapitre 3

Description du travail réalisé

On peut distinguer deux phases dans le déroulement de ce stage (figure 3.1).

La première, après une période de familiarisation avec le matériel, est la mise en place des outils de développement pour le nœud Robosoft et plus particulièrement pour le micro-contrôleur MPC555.

Le seconde phase est l'étude et la validation de la carte ainsi que de la bibliothèque fournie. L'étude de l'interface avec le monde extérieur a permis d'envisager des applications en dehors du CyCab. Pour cela, il s'est avéré nécessaire de compléter la bibliothèque d'origine.

3.1 Familiarisation avec le matériel

Lors de mon arrivée à l'Inria, le Service Robotique disposait d'une carte d'évaluation pour le MPC555, ainsi que des outils logiciels SDS pour ce micro-contrôleur et quelques CD de démonstration. Tout ceci était encore dans sa boite et avait à peine été déballé.

Je me suis tout d'abord familiarisé avec le MPC555 en parcourant le manuel utilisateur. Le MPC555 est un micro-contrôleur 32 bits à base de Power-PC. Il possède de nombreux modules périphériques notamment deux liaisons séries, deux unités TPU[1], deux contrôleurs CAN, des modules d'entrée/sortie. Pour utiliser chacun de ces modules, il faut remplir convenablement les registres associés.

Je me suis aussi familiarisé avec l'approche Syndex développé dans l'unité de recherche de l'Inria Rocquencourt. Syndex est un outil logiciel qui permet l'implémentation d'applications temps-réel parallèles sur une architecture multi plate-forme.

On m'a fourni un PC sous Windows NT afin d'y installer SDS. Ce logiciel permet de charger un programme sur la carte par le port BDM[2] connecté au

1. Time Processor Unit
2. Background Debug Mode

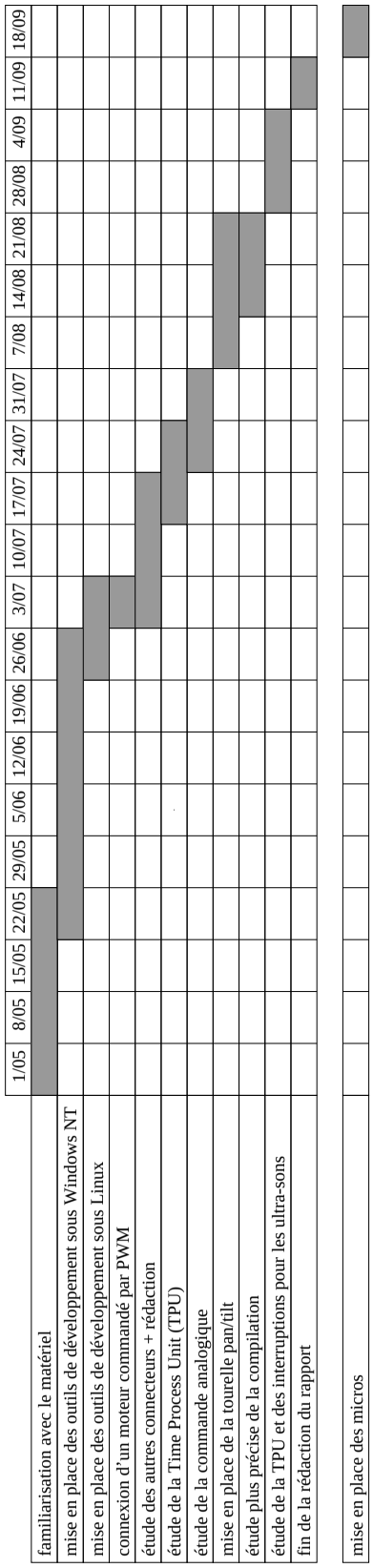

FIG. 3.1: *le planning*

port parallèle du PC. Il permet également l'exécution pas à pas d'un programme ainsi que la visualisation de la valeur des registres.

SDS est livré avec un exemple déjà compilé. La première étape a donc été de charger ce code sur la carte afin de le faire exécuter par le MPC555.

J'ai étudié la disposition de la mémoire sur la carte ainsi que les différents registres du MPC555 gérant l'accès à la mémoire.

Le code exemple a été compilé en spécifiant précisément les zones mémoires dans lequel il doit être chargé. Or, par défaut SDS ne configurait pas le MPC555 afin qu'il puisse accéder à la portion de mémoire dans lequel se situait le code. J'ai donc dû configurer SDS pour qu'il active convenablement le mapping de la mémoire interne au MPC555.

3.2 Mise en place des outils de développement

3.2.1 Mise en place sous Windows NT

Le compilateur

Après l'étape de chargement d'un exemple exécutable pré-compilé, venait l'étape de compilation. SDS n'est pas livré avec des outils de compilation. Nous avions donc le choix des outils : soit le compilateur commercial proposé par Diab Data, soit GCC (the GNU Compiler Collection). Notre choix s'est porté sur GCC puisque c'est un compilateur GNU et qu'il représente un standard dans le monde UNIX.

Il a été décidé d'essayer de compiler GCC sur la plate-forme Windows NT. Dans le même temps, s'est effectuée une demande d'informations à l'unité de recherche de l'Inria Rocquencourt où avait déjà été utilisé le MPC555 et donc un compilateur.

Cygnus Cygwin qui fournit un environnement UNIX/Linux sous Windows NT et qui permet de porter des programmes sous cette plate-forme était déjà installé sur mon PC. J'ai donc compilé Cross GCC sous Windows NT avec le support Cygwin. Bien entendu, porter GCC ne s'est pas fait automatiquement. J'ai dû ajuster de nombreux paramètres en fonction de la configuration de ma machine. Ceci a représenté un bon exercice de transposition d'un logiciel du monde Unix au monde Windows.

La compilation

Cette étape devait aboutir à la génération de code chargeable et exécutable sur la carte d'évaluation. J'ai donc dû étudier les étapes de la compilation croisée pour le MPC555 : les différents fichiers d'initialisation et les paramètres passés à l'éditeur de lien pour le placement des différentes sections du code dans la mémoire. Après plusieurs tâtonnements sur les bibliothèques à utiliser et les

différents paramètres pour l'éditeur de lien, j'ai chargé un exemple de base sur la carte.

J'ai ensuite travaillé sur un exemple plus complexe mais qui paraissait l'un des plus simple à mettre en œuvre sur la carte : la liaison série. En effet, la validation de la bonne utilisation de la liaison série a pu être réalisée en connectant une console (VT100) à la carte. La simplicité de la connexion et du matériel connecté permet de situer les éventuels problèmes au niveau de la carte d'évaluation.

J'ai donc étudié le fonctionnement et la configuration de la liaison série sur le MPC555. Ceci comprend les paramètres de la transmission (bits de données, bits de stop, vitesse de transmission, parité), ainsi que la position en mémoire où l'on doit placer les données à transmettre et où l'on doit lire les données reçues.

Après avoir assimilé ces informations, je les ai validées en manipulant directement les registres sur la carte à l'aide de SDS.

A ce moment là, le Service Robotique a reçu un nœud Robosoft commandé précédemment. Ce nœud était livré avec une bibliothèque C écrite en assembleur (qu'on appellera "bibliothèque Robosoft") permettant son utilisation et différents outils logiciels pour le chargement de programmes par le bus CAN.

Les connecteurs présents sur la carte d'évaluation du MPC555 n'étaient pas de type courant, contrairement à ceux du nœud Robosoft. Il a donc été rapidement décidé d'abandonner la carte d'évaluation pour la nouvelle carte. L'utilisation, dans un premier temps, de la carte d'évaluation m'a cependant permis de me familiariser avec le MPC555 et l'outil SDS.

Avant d'utiliser la nouvelle carte, j'ai étudié le code de la bibliothèque Robosoft concernant la liaison série. Ce fut mon premier contact avec de l'assembleur Power-PC.

L'étude des fonctions manipulant la liaison série m'a permis de vérifier ma bonne compréhension de ce module. De plus, connaissant les opérations à effectuer, j'ai compris plus facilement le code assembleur.

J'ai écrit ensuite un premier programme C d'écho utilisant la liaison série, avant de connecter le nœud Robosoft. Le programme d'écho consiste à lire les caractères envoyés sur la liaison série par la console, puis de les renvoyer sur l'écran.

L'organisation de la mémoire diffère entre les deux cartes. Les paramètres du mapping ne convenaient plus. De plus, dans le CyCab, le BDM par lequel SDS charge le programme, n'est pas accessible. Il faut charger le programme par le bus CAN. S'est alors posé le problème de l'état initial de la carte. En effet, SDS, avant de charger un programme, initialise de nombreux registres. Or cette initialisation ne sera pas faite lorsque l'on chargera un programme par le bus CAN. Je devais donc empêcher SDS de modifier l'état initial du nœud Robosoft.

Cet état initial est déterminé par une portion de code placé dans la mémoire flash de la carte (déjà écrit). J'ai donc relevé à l'aide de SDS les valeurs des registres à l'état initial, et j'ai reporté ces valeurs dans le script d'initialisation de SDS. J'ai ensuite étudié le MPC555 afin de connaître la signification de ces valeurs.

Je ne pouvais pas changer l'initialisation de la carte. C'est donc la compilation qu'il fallait adapter, c'est à dire la disposition des sections de code dans la mémoire. Après une étude précise des exemples fournis avec la carte et surtout de leurs paramètres de compilation tels que les scripts pour l'éditeur de lien, j'ai réussi à compiler mon programme d'écho pour le nœud Robosoft.

Restait maintenant à charger le code par le bus CAN ce que permettent les programmes fournis par Robosoft. Robosoft utilise ses logiciels de chargement sous Linux. Le PC embarqué sur le CyCab à partir duquel on chargera le code sur les nœuds fonctionne sous Linux. J'ai donc trouvé plus commode de travailler sous cette plate-forme et j'ai alors demandé un second PC sous Linux, le premier étant réservé à SDS.

3.2.2 Mise en place sous Linux

La première étape a été la compilation de Cross GCC sous Linux. J'ai rencontré quelques difficultés dues à un problème de patch. Au bout de deux jours, j'ai décidé d'abandonner la compilation de GCC et d'utiliser la version disponible sur le réseau AFS de l'Inria, mise à disposition par les personnes du site de Rocquencourt. La compilation des outils Robosoft n'a pas posé de problème. J'ai donc réussi ensuite la compilation et le chargement d'un programme par le CAN.

3.3 Étude du nœud Robosoft

Pour étudier précisément le nœud, on m'a fourni les plans de la carte électronique.

J'ai commencé par regarder à quelle broche du MPC555 est reliée chaque broche de chaque connecteur. Les broches ne sont pas reliées directement au MPC555. Le plus souvent, les entrées/sorties du MPC555 sont opto-isolées du monde extérieur. Le sens de la transmission de l'information sur les broches est donc figé, ce qui limite les possibilités au niveau des applications. De plus, certaines broches du MPC555 ne sont pas câblées. A cette étape, j'ai commencé à avoir une idée précise de ce qui était possible avec cette carte et de ce qui ne l'était pas.

Les broches sont associées à un ou plusieurs modules du MPC555 comme la liaison série ou la TPU. L'étude de la carte a pour but d'extraire les possibilités

de manipulation des broches disponibles sur les connecteurs. Elle doit permettre d'acquérir le savoir-faire nécessaire à la maintenance du CyCab mais aussi d'envisager le développement de "nœuds" capteurs à insérer dans la boucle CAN du CyCab ainsi que le développement de toute application sur cette plate-forme.

Afin de valider à la fois mes explications sur le fonctionnement de la carte et le code fourni, j'ai fait fonctionner différents capteurs et actionneurs. Ceci est expliqué dans les sections suivantes.

La lecture approfondie des m'a permis d'en apprendre un peu plus sur les signaux logiques et sur différents composants que l'on retrouve fréquemment dans les applications de contrôle-commande comme les convertisseurs analogique numérique et numérique analogique.

3.3.1 Mise en œuvre d'un moteur à courant continu et d'un codeur incrémental

Mon responsable de stage Gérard Baille a construit un câble afin de connecter la carte à un amplificateur PWM[3] de puissance lui même relié à un moteur à courant continu. En plus de la commande, cet amplificateur permet le contrôle du moteur en renvoyant une tension proportionelle à la valeur du couple exercé sur celui-ci. Un codeur optique monté sur l'axe du moteur donne une information de vitesse.

Étude du PWM

Le PWM est un signal carré périodique dont on peut faire varier la fréquence et le rapport cyclique. Il est utilisé dans la commande de moteur en faisant varier la largeur de l'impulsion.

J'ai commencé par étudier le module du MPC555 permettant la génération de signal PWM. J'ai ensuite analysé le code assembleur de la bibliothèque Robosoft, pour terminer par l'écriture d'un programme C devant faire tourner le moteur.

L'exécution de ce programme n'a pas permis de faire tourner le moteur. J'ai donc étudié à nouveau le module PWM du MPC555, puis validé mon code C au niveau des registres à l'aide de SDS. N'ayant pas détecté d'erreur, j'ai consulté mon responsable de stage. Nous avons alors vérifié les différents signaux émis par la carte à l'aide d'un oscilloscope : le signal PWM n'était pas émis.

En mesurant la tension sur la carte d'abord près du MPC555 nous avons pu lire le signal. Puis en piquant de plus en plus loin, nous avons détecté l'endroit où nous le perdions. Il s'agit en fait d'un système de chien de garde qui nous avait échappé. Ceci est une bonne mesure de sécurité puisque dans le CyCab, le signal PWM commande les moteurs des roues du véhicule.

3. Pulse Width Modulator

Le chien de garde est un monostable dont la sortie conditionne l'émission du PWM. Ce monostable est réarmé par l'émission su signal $\overline{CS2}$ qui sert habituellement à la gestion de la mémoire extérieure au MPC555. Ce signal est émis lorsque l'on fait un accès à une zone mémoire particulière qui dans notre cas n'est pas utilisée. On fait donc une lecture "à vide" à cet emplacement mémoire uniquement pour émettre le signal.

Pour permettre l'émission du PWM, il faut donc réarmer un monostable régulièrement en faisant un accès mémoire. Ceci est réalisé en utilisant le PIT[4] qui permet de lever une interruption à une fréquence définie.

La vérification des signaux à l'aide d'un oscilloscope représente une étape supplémentaire dans ma formation au cours de ce stage. En effet je me suis placé à un niveau inférieur au des registres et des valeurs binaires pour descendre au niveau des tensions des signaux électriques.

Étude du convertisseur analogique numérique

Afin de faire l'acquisition des données renvoyées par l'amplificateur PWM de puissance, j'ai étudié le module convertisseur analogique numérique du MPC555.

J'ai ensuite analysé puis validé le code de la bibliothèque Robosoft, pour aboutir à la lecture des informations en utilisant un programme C.

La lecture et la conversion des valeurs lues ne m'a pas posé de problème.

Étude de la TPU

La TPU est une unité indépendante du CPU qui permet de compter de manière fiable. De nombreuses fonctions de la TPU sont câblées dans le MPC555 comme par exemple une fonction permettant de compter le temps pendant lequel un signal reste haut durant un nombre de périodes précises, ou encore d'incrémenter un compteur à chaque impulsion d'un signal. C'est cette dernière fonction qui est utilisée dans le cadre de l'utilisation d'un codeur optique.

Pour configurer la TPU, il faut notamment définir la fonction que l'on veut utiliser et ses différents paramètres. Dans le CyCab, on utilise la TPU pour lire les codeurs optiques associés aux axes commandés. J'ai donc analysé le code de la bibliothèque Robosoft qui correspondait à ce que je recherchais, puis implémenté le comptage. Là encore, aucun résultat.

La vérification des signaux à l'aide d'un oscilloscope apermis de déceler un problème matériel.

Dans l'application CyCab, les codeurs incrémentaux sont reliés au nœud par des liaisons différentielles RS-485 pour une meilleure immunité aux bruits. Le codeur dont je disposais fournissait des sorties TTL[5] incompatibles avec les récepteurs de ligne de la carte.

4. Periodic Interrupt Timer
5. Transistor Transistor Logic

3.3.2 Mise en œuvre d'un moteur à courant continu brushless

Un moteur brushless et son variateur analogique étaient disponibles dans le Service Robotique. Le variateur permet de commander le moteur en couple ou en vitesse à partir d'une consigne donnée par une tension analogique variant entre 0 et 10 Volts. Le moteur est équipé d'un synchro-résolveur et le variateur d'une carte émulateur codeur optique qui à partir des informations du résolveur délivre des signaux différentiels que l'on trouve sur les codeurs. Ce système devait permettre de tester les convertisseurs numérique analogique du nœud et les fonctions de comptage de la TPU.

Étude des convertisseurs numérique analogique

La génération du signal analogique pour la commande du moteur se fait en utilisant un convertisseur numérique analogique placé sur le nœud Robosoft.

L'étude de la carte m'a indiqué les broches du MPC555 permettant de contrôler le convertisseur numérique analogique. J'ai ensuite étudié le fonctionnement du convertisseur, celui du MPC555 concernant les broches que je souhaitais utiliser, pour valider ensuite le code de la bibliothèque Robosoft. J'ai écrit un programme C dont l'exécution n'a donné aucun résultat. Après avoir de nouveau vérifié l'utilisation du convertisseur dans les fonctions Robosoft, sans détecter d'erreur, j'ai observé les signaux avec l'oscilloscope.

Le signal n'était pas du tout ce qu'il devait être et je ne trouvais pas d'explication. J'ai donc soumis le problème à Gérard Baille qui en a déduit que certaines résistances n'avaient pas les bonnes valeurs. Après remplacement de ces résistances, la commande analogique a fonctionné parfaitement, de même que l'utilisation de la TPU pour la lecture du codeur.

3.3.3 Mise en œuvre de la tourelle pan/tilt

Le Service Robotique m'a fourni une tourelle pan/tilt commandée par liaison série ainsi que des bibliothèque pour cette tourelle et la liaison série. Il m'a été demandé de porter ces bibliothèques sur le MPC555.

Mon premier réflexe a été de tester le bon fonctionnement de ces tourelles en utilisant mon PC sous Linux et le code fourni. L'exécution du code ne donnait aucun résultat. J'ai donc d'abord vérifié le câble de la liaison série. J'ai ensuite vérifié le code pour la liaison série.

La bibliothèque pour la liaison série était sensée fonctionner sous VxWorks, Solaris et Linux. Après mes quelques remarques et recherches à propos de cette bibliothèque, il m'a été confirmé qu'elle ne fonctionnait pas bien sous Linux. Je me suis donc placé une étape en dessous et j'ai testé la tourelle avec un

terminal. Après avoir validé son bon fonctionnement, j'ai écrit une bibliothèque pour la liaison série pour le MPC555 en utilisant les fonctions Robosoft. Ceci n'a pas présenté de difficultés puisque j'avais déjà étudié la liaison série sur le MPC555. La bibliothèque pour la gestion de la tourelle n'a nécessité que des modifications mineures (plus de `printf` par exemple).

Ceci a représenté un travail formateur puisque il a fallu comprendre et adapter du code que je n'avais pas écrit. Ce travail a aussi présenté des contraintes car il m'a fallu respecter les signatures des fonctions afin de conserver la compatibilité.

Au cours de ce travail, a été mis en évidence un problème au niveau de la compilation non apparu jusque alors car je n'avais compilé que des programmes assez simples. En effectuant une dé-compilation de mon code exécutable en assembleur, j'ai observé que l'appel à la fonction `sprintf` réalisait un branchement dans une zone mémoire inappropriée.

J'ai donc repris mon travail sur la mise en place de Cross GCC sous Linux que j'avais abandonné précédemment, ceci jusqu'à sa compilation avec succès. Ne constatant pas de différence malgré l'utilisation d'un compilateur compilé différemment, je me suis replongé dans l'étude de la compilation et en particulier de l'éditeur de lien, du script pour celui-ci et du code d'initialisation, délivrés par Robosoft.

Le script définit de manière précise le placement des sections du code dans la mémoire. Le code d'initialisation, appelé crt0 (C run time 0) détermine le déroulement du programme (appel à une fonction d'initialisation puis à la fonction `main` puis à la fonction `exit`).

Après une analyse précise de ces différents éléments, j'ai modifié le script pour l'éditeur de lien ce qui a permis de placer la fonction `sprintf` là ou elle était demandée.

Cette étape a approfondi mes connaissances sur les mécanismes de compilation, et particulièrement sur les différentes sections d'un code ainsi que leur disposition en mémoire.

3.3.4 Mesure de distances à l'aide de capteurs à ultrasons

Un capteur à ultrasons (transceiver polaroid ainsi que sa carte analogique) résultant d'un stage IUT précédemment effectué au sein du Service Robotique était disponible.

Principe de la mesure

La carte analogique dispose d'un cristal piézo-électrique que l'on fait entrer en résonnance afin de faire vibrer une membrane. Cette membrane produit une

onde de résonnance. La carte signale la capture de l'écho, ce qui permet de mesurer le temps de vol et d'en déduire ainsi une distance.

Mise en œuvre

Pour utiliser l'ultrason, on doit envoyer un signal *INIT* afin de provoquer l'émission d'une salve d'ultrasons et l'initialisation d'un compteur. La réception du signal *ECHO* doit provoquer l'arrêt du comptage. La valeur du compteur permet de déduire la distance.

Pour mesurer le temps, on utilise la TPU, mais d'une façon différente de celle de l'application CyCab. J'ai donc écrit d'autres fonctions assembleur pour appeler et paramètrer une nouvelle fonction de la TPU. J'ai choisi une fonction permettant de détecter un front montant sur une broche (réception du signal *ECHO*), de capturer la valeur d'un compteur interne et de lever une interruption.

La TPU permet de régler la valeur du compteur interne dans une certaine mesure. J'ai choisi d'avoir une valeur qui corresponde à une lecture directe en millimètres afin de limiter le temps de traitement. Pour initialiser le compteur, j'effectue un *reset* de la TPU, puis je l'initialise avant de lancer le signal *INIT*.

La mesure d'un temps m'a montré les différentes contraintes liés à un problème temps-réel. En effet, un délai de quelques dizaines de micro-secondes aboutit à une incertitude de quelques centimètres sur la mesure d'une distance. Après une série de mesure, j'ai observé que le temps écoulé entre le *reset* et le signal *INIT* correspondait à 40 μs. De plus, si une interruption survient entre ces deux moments, ce temps est susceptible de s'allonger. L'idéal serait de câbler le signal *INIT* également sur l'unité TPU afin d'avoir des mesures de temps fiables.

La programmation des nouvelles fonctions pour la TPU n'a pas posé de problème. Cependant, il m'a fallu étudier précisément le mécanisme des interruptions dans le MPC555. Ceci a permis de révéler un problème matériel au niveau de la carte, plus précisément au niveau du *reset*.

C'est arrivé à cette étape que j'ai interrompu mon travail afin de préparer ma soutenance. Après celle-ci, je disposerai encore d'une semaine et demie pour achever mon travail.

Chapitre 4

Conclusion

Ce stage m'a donné l'occasion de mettre en œuvre toutes les étapes de développement d'une application de contrôle-commande à base de micro-contrôleur en partant de la carte électronique, en passant par l'assembleur, pour arriver au langage C.

J'ai mis en place les outils nécessaires à la compilation pour la cible MPC555, ainsi que les outils de *debug*. J'ai ensuite étudié le fonctionnement de ce micro-contrôleur pour aboutir à l'utilisation de différents moteurs et capteurs (codeur incrémental, ultrasons).

Ce stage m'a permis de faire face à des problèmes matériels qui ne manquent pas de se poser dès que l'on met en jeu différents capteurs et actionneurs. Face aux difficultés rencontrées, j'ai pu instancier une méthode de travail :

1. étude d'un point de fonctionnement du MPC555
2. étude et validation du code assembleur délivré
3. validation du code généré par une analyse au niveau des registres
4. *debug* au niveau matériel à l'aide d'un oscilloscope

Ce stage m'a donné l'occasion d'étudier précisément la compilation croisée, et de m'investir dans l'étude d'un micro-contrôleur complexe.

La documentation technique que j'ai écrite sera reprise et complétée dans un rapport technique de l'Inria. Elle permettra la poursuite et l'utilisation de mon travail ainsi que le développement d'application fonctionnant sur le nœud Robosoft dans le cas du CyCab ou dans un autre contexte.

Ce stage achève ma formation dans la filière Itra à l'Essi que j'ai choisie justement pour sortir de l'informatique pure. Je voulais écrire des programmes qui agissent sur des objets physiques, chose que ce stage m'a donné l'occasion de faire.

L'étude des ultrasons n'est pas encore achevée. Il reste aussi à étudier l'utilisation de microphones pour détecter une source sonore. Enfin, on pourra assembler les différentes parties (tourelle, ultrasons, micros) pour construire une application. Ceci sera effectué lors de ma dernière semaine de stage, jusqu'au 26 Septembre.

Deuxième partie

Description en détail du travail réalisé

Chapitre 1

Mise en place des outils de développement pour le nœud Robosoft à base de MPC555

1.1 Mise en place sous Windows NT

1.1.1 Outils utilisés

On utilise

- Single Step On Chip de SDS pour la connection sur la carte par le BDM (Background Debug Mode)
- Cross-GCC compilé pour la cible powerpc-eabi

Cross-GCC est utilisé avec le support Cygwin.

1.1.2 Compilation de Cross-GCC

Il est conseillé de se référer à la FAQ de Cross-GCC[1].

On utilisera

- binutils-2.9.1
- gcc-2.95.2
- newlib-1.8.2

Tout d'abord placer `sh.exe` dans `/bin` afin que les scripts standards le trouvent. On a décompressé les sources dans `/crossgcctmp` et que l'on veut installer dans `/crossgcc`.

1. http://www.objsw.com/CrossGCC/

Cygwin est installé dans `Program~Files/cygnus/cygwin-b20/`
Voici la marche à suivre:

```
PATH=$PATH:/crossgcc/bin:/crossgcc/powerpc-eabi/bin

cd /Crossgcctmp/
target=powerpc-eabi
prefix=/crossgcc
i=$prefix/bin

mkdir build-binutils build-gcc build-newlib build-gdb

cd build-binutils
../binutils-2.9.1/configure --target=$target --prefix=$prefix
--with-headers=/Progra~1/cygnus/cygwin-b20/H-i586-cygwin32/i586-cygwin32/include
--with-newlib
make all install

cd ../build-gcc
../gcc-2.95.2/configure --target=$target --prefix=$prefix
--with-headers=/Progra~1/cygnus/cygwin-b20/H-i586-cygwin32/i586-cygwin32/include
--with-newlib
make all install

cd ../build-newlib
../newlib-1.8.2/configure --target=$target --prefix=$prefix
--with-headers=/Progra~1/cygnus/cygwin-b20/H-i586-cygwin32/i586-cygwin32/include
--with-newlib
make all install
```

1.1.3 Single Step On Chip

Installation “classique” d'un logiciel sous Windows.

Configuration pour la carte MPC555 et Cross-GCC

On configure SDS de façon à ce que l'état initial lors du chargement du programme ne soit pas modifié par rapport à celui définit par la portion de code placée dans la mémoire flash (fichier de configuration disponible dans l'annexe A).

On veut aussi ajouter les outils GNU dans le path.
Sous le répertoire SDS75 créer un répertoire `mpc555`. Placer dans ce répertoire des fichiers d'initialisation de SDS (ici seulement sstep.ini).
On lance SDS à l'aide d'un raccourci dans lequel on spécifie
"`démarrer en C:\sds75\mpc555`".

Dans le fichier sstep.ini on ajoute un chemin dans le path :
`setenv PATH "$PATH;C:\Crossgcc\bin"`

1.1.4 Compilation d'un exemple simple

Voici l'exemple `simpl.c` :

```
#include <stdlib.h>

int main(){
  int a = 10;
  a = 25;
  return EXIT_SUCCESS;
}
```

On utilise la "C run time 0" (crt0).
On positionne le symbole `_exit` à `_start` ainsi le programme boucle.
Les appels systèmes ne sont pas définis dans la bibliothèque libc. Ils sont définis dans les libgloss très dépendantes de la cible. Il n'y a pas de bibliothèque spécifique à notre carte alors on utilise la libnosys.
Il faut aussi définir le symbole `__stack`.
On utilise aussi `ecrti` (initialisation) et `ecrtn` (cleanup).

Voici un `Makefile` qui permet de compiler cet exemple :

```
CC = c:/crossgcc/bin/powerpc-eabi-gcc
LIBS = -lc -lnosys -lnode555
LOADLIBS = -Lc:/robosoft/lib/
LDFLAGSADD = --defsym=__stack=0xfff20000,-Ttext=0xfff00000,-Tdata=0x3f9f58
LDFLAGS = -Wl,-Map=map,--defsym=_exit=_start,$(LDFLAGSADD)
CFLAGS = -mcpu=505 -mbig -gdwarf
OBJ = "c:\crossgcc\lib\gcc-lib\powerpc-eabi\2.95.2\ecrti.o" \
"c:\crossgcc\powerpc-eabi\lib\crt0.o" \
$(TARGET).o "c:\crossgcc\lib\gcc-lib\powerpc-eabi\2.95.2\ecrtn.o"
TARGET = simpl

all:
make $(TARGET).elf

$(TARGET).o: $(TARGET).c
$(CC) $(CFLAGS) -c $?

$(TARGET).elf: $(TARGET).o
$(CC) $(CFLAGS) $(OBJ) -o $@ $(LDFLAGS) $(LIBS) $(LOADLIBS)

clean:
rm $(TARGET).elf
```

1.2 Mise en place sous Linux

1.2.1 Outils utilisées

On utilise les logiciels délivrés par Robosoft avec la carte MPC555 sur le CD "MPC555 Toolchain - Release 17 March 2000". Ce CD contient

- Cygnus GNUPro pour MPC555 avec un patch pour son utilisation avec la RedHat 6.0
- les outils Robosoft
- libelf-0.7

Les outils Robosoft consistent en deux programmes. L'un (`elf2sdxbin`) permet de transformer le code compilé (format elf) en binaire Syndex. C'est ce format que l'on peut charger sur la carte en utilisant le second programme : `dwnbin`.

Au niveau matériel, on utilise la carte CAN NuDAQ PCI-7841 identique à celle présente dans le PC embarqué du CyCab.

1.2.2 Compilation des outils

On décide d'installer l'ensemble des logiciels en local (répertoire `/home/ciney/lydoire/local/mpc555`). Cela nécessite de modifier le script d'installation, mais permet la mise en place des logiciels sans droits privilégiés. Cependant, le programme `dwnbin` des outils Robosoft utilise directement les ports d'entrée/sortie de la carte CAN. Il est donc nécessaire d'être *root* pour l'utiliser.

On copie l'ensemble du CD dans un répertoire temporaire (ici `~/tmp/cdrom`).

On spécifie le répertoire local dans lequel on veut installer la bibliothèque elf dans le script d'installation. On remplace

```
# Compile and install libelf
cd libelf-0.7.0; \ ./configure --prefix=/usr; \
make; \
make install; \
cd ..
```

par

```
# Compile and install libelf
cd libelf-0.7.0; \
./configure --prefix=$installdir/usr; \
make; \
```

```
make install; \
cd ..
```

Il faut donc maintenant spécifier l'emplacement de la bibliothèque elf pour la compilation de `elf2sdxbin`. On remplace donc dans le `Makefile` de `elf2sdxbin`

```
LIBS = -lelf
INC_DIR = -I /usr/include/libelf
```

par

```
LIBS = -lelf -L /home/ciney/lydoire/local/mpc555/usr/lib
INC_DIR = -I /home/ciney/lydoire/local/mpc555/usr/include/libelf \
-I /home/ciney/lydoire/local/mpc555/usr/include/
```

Il faut adapter le programme dwnbin au ports d'entrées/sorties de la carte CAN, valeurs que l'on obtient en effectuant la commande `cat /proc/pci`. Dans notre cas on obtient

```
Bus  0, device  13, function  0:
    Network controller: Unknown vendor Unknown device (rev 1).
      Vendor id=144a. Device id=7841.
      Medium devsel.  Fast back-to-back capable.  IRQ 11.
      I/O at 0xdc00 [0xdc01].
      I/O at 0xd800 [0xd801].
      I/O at 0xd400 [0xd401].
```

On remplace donc dans le code de `dwnbin.c`

```
#define CAN0 0xE400
```

par

```
#define CAN0 0xD800
```

On se place maintenant dans le répertoire `GNUProToolkit-990319`. On doit ensuite changer les droits de certains fichiers qu'il est nécessaire de patcher lorsque on utilise la RedHat 6.0. On lance ensuite la compilation. Voici la marche à suivre.

```
cd GNUProToolkit-990319
chmod 755 src
cd src
chmod 755 tcl expect make gdb tcl/generic
chmod 644 tcl/generic/tclPosixStr.c expect/exp_strf.c \
make/arscan.c gdb/gdb_string.h
patch -p0 < ~/tmp/cdrom/GNUProToolkit-990319/ecos-rh6.patch
```

Il ne reste plus qu'à compiler en spécifiant le répertoire d'installation.

```
cd ..
./Install --installdir=/home/ciney/lydoire/local/mpc555
```

1.2.3 Compilation d'un exemple simple

On reprend l'exemple simpl.c utilisé dans la compilation sous Windows NT. On utilise la C run time 0 et le script pour l'éditeur de lien légèrement modifié fournis par Robosoft (disponibles dans les annexes B et C).

Voici un `Makefile` qui permet de compiler cet exemple :

```
SHELL = /bin/sh
.SUFFIXES = .s .c .o

RM = rm -f

# Specify MPC555 cross GCC compiler.
XGCC = ppc-elf32-gcc
# Set libraries path and the list of libraries to be passed
# to the linker.
LIBDIR = -L./
LIB     = -lm -lc -lnode555

# Linker options passed from compiler to linker.
LDSCRIPT = script.ld
LDFLAGS = -Wl,-T,$(LDSCRIPT),-Map,$@.map


# Hardware configurations. Target related options.
CFLAGS = -mpowerpc -mhard-float -ansi

# SRC (source filess) and OBJ (object files) variables
# list the files needed to generate the 'main.elf' executive.
SRC = simpl.c

TMP = $(SRC:.c=.o)
OBJ = $(TMP:.s=.o)

# Makefile commands.
all :
make main.elf
make download

%.o : %.c
$(XGCC) -O2 -c $< -o $@ $(INCDIR)

%.o : %.s
$(XGCC) -O2 -c $< -o $@
```

```
main.elf : $(OBJ) crt0.o $(LDSCRIPT)
$(XGCC) $(LDFLAGS) $(CFLAGS) $(OBJ) -o $@ $(LIBDIR) $(LIB)

clean :
$(RM) *.o crt0.o main.elf main.elf.map main.bin *~

download :
elf2sdxbin main.elf main.bin
```

Il suffit alors de télécharger le fichier `main.bin` sur la carte en utilisant le programme `dwnbin`. Afin d'identifier le destinataire du code dans le réseau CAN, la carte possède un identifiant en hexadécimal. Ici 4014 qui correspond à 16404 en binaire. On télécharge donc le programme à l'aide de la commande suivante

```
dwnbin main.bin 16384
```

Chapitre 2

Connexions sur la carte VMD MPC555

2.1 Connecteur JAXE1 : PWM, entrées signaux différentiels pour TPU, entrées/sorties générales, entrées analogiques, commande analogique

On peut voir dans le tableau 2.1 les broches du MPC555 reliées au connecteur JAXE1. On dispose sur la carte de trois autres connecteurs identiques nommés JAXE[2:3].

Les broches 1,2,3,4,5,6 acceptent en entrée des signaux différentiels. Ces signaux sont ensuite transformés par un receveur de ligne différentielle. Le signal différentiel sur les broches 5,6 peut entrer ensuite en tant qu'interruption ou comme une entrée générale.

Les broches MPIO[0:15] du MPC555 peuvent êtres configurées comme des entrées ou des sorties mais l'électronique sur la carte impose l'utilisation les broches MPIO[4:11] en sortie et MPIO[12:15] en entrée. Les broches 11 et 15 sont donc des sorties et la broche 23 une entrée.

La broche 16 est une entrée logique.

La broche 12 peut être utilisée comme une entrée logique ou analogique (montage émetteur/suiveur).

La broche 10 correspond à une commande générée grâce à un convertisseur numérique analogique.

La broche 18 est une entrée analogique. Un potentiomètre numérique permet de régler la plage de variation du signal entrant entre ± 10 Volts et ± 12 Volts.

JAXE1	Carte	MPC555	
1,2	A-1	A_TPUCH0	récepteur de ligne différentiel
3,4	B-1	A_TPUCH1	récepteur de ligne différentiel
5,6	IDX-1	$\overline{\text{IRQ4}}$/SGPIO4	récepteur de ligne différentiel
11	DIR1	VFLS1/MPIO4	sortie opto-découplée
15	INH1	MPIO8	sortie opto-découplée
23	Fault1	MPIO12	entrée opto-découplée
19	PWM1	MPWM0	sortie opto-découplée
12	Temp1-in	B_AN48/PQB4	entrée opto-découplée
18	V1+	B_AN0/PQB0	entrée opto-découplée
10	Vout-1		sortie analogique
16	Break	B_AN56/PQA4	
14	Loop		
24	Break (idem INH1)		
8	+5V		
7	Masse		
20	+5v-Iso		
22	+5v-Iso		
9	0V-Iso		
13	0V-Iso		
17	0V-Iso		
21	0V-Iso		
25	0V-Iso		
26	0V-Iso		

TAB. 2.1: *broches reliées au connecteur JAXE1*

2.1.1 Utilisation d'un moteur commandé par un signal modulé en amplitude

On branche le moteur sur un amplificateur PWM de puissance pour moteur à courant continu à balai lui même relié à la carte par le connecteur JAXE1 (broches 18, 15, 11 et 23).

Le signal PWM

Pour le signal modulé en amplitude il faut définir sa période (registre MPWMSPERRX) et la largeur de l'impulsion (registre MPWMSPULRX). Il faut aussi lancer le PWM à l'aide du bit 5 du registre MPWMSMSCRX.

Tout ceci peut être fait à l'aide de la bibliothèque C libnode555 fournie par Robosoft.

- la fonction `PWMInit()` met la fréquence à 1000Hz (0x3E8), soit une période de 1μs et lance le PWM.

– la fonction `PWMWrite (int axis, short int *value)` permet de définir le rapport cyclique et donc la largeur de l'impulsion. L'axe correspond au générateur PWM que l'on utilise (0 à 3).

Signaux inhibit, direction et fault

Pour que le moteur tourne, il faut aussi mettre à 0 le signal inhibit. Ceci est fait en spécifiant le sens des informations sur la broche MPIO8 (sortie) et en lui affectant la valeur 0 (registres MPIOSMDDR et MPIOSMDR). Nous devons aussi spécifier le sens de rotation du moteur avec le signal direction (broche MPIO4) et lire un éventuel signal fault envoyé par l'amplificateur de puissance (broche MPIO12).

On peut utiliser la fonction `int WritePWMIO (int value)` qui définit le sens des broches MPIO (MPIO[0:11] en sortie et MPIO[12:15] en entrée). La valeur passée en argument correspond à la valeur des broches MPIO[4:12]. Par exemple la valeur 1 mettra la broche MPIO4 à 1, et la valeur 72 (1001000 en binaire) mettra à 1 les broches MPIO7 et MPIO10. Cette fonction renvoie la valeur précédente des broches.

Pour inhiber ou non le moteur sur l'axe 0 il faut modifier le signal INH1. On inhibe en envoyant `valeur_precedente | 0x10` et on desinhibe en envoyant `valeur_precedente & 0xef`.

Chien de garde

Sur la carte sont cablés des OU entre les signaux INH[1:4] et un signal venant d'un monostable réarmable avec le signal CS2 (Chip Select 2) de période 15ms : c'est un chien de garde (watchdog). Pour réarmer le monostable, il faut donc lire ou écrire à un emplacement mémoire en utilisant le CS2. Cet emplacement mémoire a été paramètré dans la crt0 (C run time 0) en utilisant les registres BR2 et OR2 pour adresser à l'emplacement 0x0FF00000. La fonction `WatchDog()` fait une lecture à vide à cet emplacement.

Il faut donc appeler la fonction `WatchDog()` régulièrement, ce qui peut être fait en utilisant le PIT (Periodic Interrupt Timer) qui génère des interruptions périodiques. On définit donc le handler d'interruption à l'aide de la fonction `InterruptHandlerPIT()` afin qu'il appelle la fonction `WatchDog()` et on lance le PIT avec la fonction `InitInterrupt(int PITValue)` en choisissant `PITValue` à l'aide de la formule :

$$\texttt{PITPeriod} = \frac{\texttt{PITValue} + 1}{15625}$$

où `PITPeridod` est exprimée en secondes.

2.1.2 Utilisation d'un convertisseur analogique numérique

Le MPC555 possède deux convertisseurs analogique numérique (QADC) qui eux mêmes sont constitués de deux ports A et B. Les broches du port A sont préfixées par A et celles du port B par B. Lorsque elles sont utilisées comme des entrées analogiques, les broches sont référencées AN. Ces mêmes broches sont référencées PQ lorsque elles sont utilisées de façon numérique.

On veut lire le signal analogique V1+ envoyé sur la broche 18 (B_AN0/PQB0) par l'amplificateur PWM de puissance utilisé pour commander le moteur dans la section 2.1.1. Ce signal est l'image du courant circulant dans le moteur, il est proportionnel à la valeur du couple.

Pour cela, il faut tout d'abord initialiser le convertisseur analogique numérique (QADC). On se propose d'utiliser la fonction `ADCInit()` fournie par Robosoft.

Initialisation avec la fonction `ADCInit()`

Configuration du sens des broches (registres DDRQA_X_DDRQB_X) Les broches du port B ne peuvent être que des entrées. On peut voir dans le tableau 2.2 le sens des broches du port A initialisé par `ADCInit()`.

broches	sens
A_PQA[0:6]	Sortie
A_PQA7	Entrée
B_PQA[0:3]	Sortie
B_PQA[4:7]	Entrée

TAB. 2.2: *sens des broches du Port A du QADC initialisées par* `ADCInit()`

Les broches A_PQA[0:7] concernent l'écran LCD.

Ecriture sur les ports (registres PORTQA_X_PORTQB_X) On envoie des signaux pour

- le LCD en écrivant 1 sur la broche A_PQA0
- les potentiomètres qui règlent les plages de variations des signaux V[0:3]+ (on met les Chip Select à 1)

Configuration des modules QADC (registres QADCMCR_X)

- horloge du QADC activée
- signal IMB3 FREEZE ignoré
- espace de données en mode non restreint (unrestricted data space)

Contrôle du QADC (registres QACR[0:3]_X) Réglage de la fréquence de l'horloge (comme dans l'exemple 1 section 13.10.4 du manuel du MPC555) :

- high clock time = 300ns
- low clock time = 200ns
- QADC clock freq = 2Mhz

On se place en mode scan continu et on désactive la queue 2. L'espace mémoire est donc entièrement pour la queue 1.

Convertion Command Word Table (registres CCW_X_[0:64]) On sélectionne les entrées analogiques à traiter : X_AN[0:3] et X_AN[48:51].

Il faut maintenant lire la valeur envoyé sur la broche 18. On utilise la fonction Robosoft
`ADCRead (int channel, int *value)`.

Lecture avec `ADCRead (int channel, int *value)`

On lit les valeurs dans le tableau de résultats justifiés à droite, non signés du MPC555 (registres RJURR_X_[0:63]).

La valeur de `channel` permet de choisir la broche que l'on veut lire (voir le tableau 2.3).

channel	broche	channel	broche
0	A_AN0	8	B_AN0
1	A_AN1	9	B_AN1
2	A_AN2	10	B_AN2
3	A_AN3	11	B_AN3
4	A_AN48	12	B_AN48
5	A_AN49	13	B_AN49
6	A_AN50	14	B_AN50
7	A_AN51	15	B_AN51

TAB. 2.3: *sélection de la broche pour* `ADCRead(int channel, int *value)`

On peut aussi régler l'offset avec la fonction `PotSet (int PotCh, int Value)` qui incrémente (`Value` ≥ 0) ou décrémente (`Value` < 0) le potentiomètre de la valeur $|$`Value`$|$. `PotCh` correspond au signal que l'on veut régler (valeurs de 0 à 3 pour les signaux V[1:4]+).

2.1.3 Utilisation d'un moteur commandé en tension analogique

On souhaite utiliser un moteur commandé en tension. On utilise donc la broche 10 qui correspond à une commande générée grâce à un quadruple convertisseur numérique analogique à entrée numérique série contrôlé par les broches MPIO[0:2] du MPC555. On envoie 16 bits en série sur la broche MPIO0 (2 pour adresser démultiplexer un des 4 convertisseurs, 12 pour coder la valeur de la tension de sortie et 2 indifférents). La broche MPIO1 sert d'horloge pour le registre à décalage. La broche MPIO2 permet de lancer la conversion.

Pour cela on utilise la fonction Robosoft `DACWrite (int Ch, int Value)`.

Cette fonction spécifie le sens des broches MPIO (MPIO[0:11] en sortie et MPIO[12:15] en entrée). Elle construit la trame de 16 bits à partir des valeurs de `Ch` (entre 0 et 3) et de `Value` (entre 0 et 4095). Elle envoie ensuite cette trame.

La fonction `DACWrite` ne modifie pas les autres signaux sur les broches MPIO.

2.1.4 Utilisation de la Time Processor Unit (TPU) avec un codeur optique

On souhaite utiliser la TPU pour compter les tours d'un bouton de façade avec codeur optique. Pour cela, on utilise la fonction FQD (Fast Quadrature Decode) câblée dans l'unité TPU. La fonction Robosoft `int TPU3Init(void)` permet cette utilisation.

Cette fonction initialise la TPU en spécifiant le numéro de la fonction à utiliser (ici le numéro 6 pour la fonction FQD) et les broches à utiliser pour cette fonction. Elle initialise ensuite les différents paramètres de FQD notamment l'initialisation du compteur. Elle assigne ensuite une priorité afin de lancer la fonction.

La fonction `int EncRead (int axis, int *value)` permet de lire la valeur du compteur utilisé par la fonction FQD. `axis` correspond au numéro du connecteur JAXE utilisé, et `value` est la valeur lue.

2.1.5 Utilisation de la TPU avec des ultra-sons

On souhaite utiliser la TPU pour mesurer le temps de vol d'un train d'ultrasons, c'est à dire le temps entre l'envoi un signal *INIT* et la réception d'un signal *ECHO*. Cependant, seules les broches portant le signal *ECHO* sont connectées à l'unité TPU. On décide donc d'utiliser la fonction NITC de la TPU qui permet de capturer la valeur interne d'un compteur de la TPU et de lever une

interruption. La démarche est donc la suivante : initialisation des interruptions, *reset* de l'unité TPU (remise à zero des compteurs), puis envoi du signal *INIT*. Le traitement lors de la réception du signal *ECHO* sera effectué dans le *handler* d'interruption.

Afin de ne pas attendre indéfiniment la capture du signal *ECHO*, on met en place un système de *timer*. Celui-ci lit la valeur du compteur interne de la TPU et arrête l'attente du signal *ECHO* lorsque le temps écoulé correspond à la distance maximale couverte par le capteur à ultra-sons.

Tout ceci est réalise grace aux fonctions :

- `void TPU3InitUltra()` qui initialise la fonction NITC, ainsi que le coefficient diviseur du compteur interne TCR1 de la TPU.
- `void TPU3Stop()` nécessaire avant de faire un *reset*
- `void TPU3Reset()`
- `void TPU3Inittcr1()` qui associe la fonction FQD sur des broches non câblées sur la carte de l'unité TPU, uniquement pour lire la valeur du compteur TCR1 pour la gestion du *timer*.
- `void TCR1Read(int *val)` poyr lire la valeur du compteur TCR1.

Afin d'utiliser au mieu l'unité TPU, il sera nécessaire de câbler le signal *INIT* sur la TPU. Ceci permettra de mesurer exactement le temps entre ce signal et le signal *ECHO*.

2.2 Connecteur J201 : entrées des convertisseurs analogique numérique

On peut voir dans le tableau 2.4 les broches du MPC555 reliées au connecteur J201.

Les broches 1,3,4,5,6,7,8,9 sont connectées aux broches du port B du QADC. Les huit broches du port B peuvent être utilisées comme des entrées analogiques, ou comme une entrée numérique huit bits. L'interface entre le connecteur J201 et les broches du MPC555 (ampli. opérationnel monté en émetteur suiveur) permet ces deux utilisations.

Lorsque elles sont utilisées comme des entrées analogiques, les huit broches du port B sont référencées AN[51:48]/AN[3:0].

Ces mêmes broches sont référencées PQB[7:0] lorsque elles sont utilisées en entrée numérique.

On peut utiliser la fonction Robosoft `ADCRead(int channel, int *value)` en choisissant la valeur de `channel` comme spécifié dans le tableau 2.3.

J201	Carte	MPC555
4		A_AN0/ANW/PQB0
6		A_AN1/ANX/PQB1
8		A_AN2/ANY/PQB3
9		A_AN3/ANZ/PQB3
7		A_AN48/PQB4
5		A_AN49/PQB5
3		A_AN50/PQB6
1		A_AN51/PQB7
2	+5V	
10	Masse	

TAB. 2.4: *broches reliées au connecteur J201*

2.3 Connecteur J601 : entrées/sorties générales

On peut voir dans le tableau 2.5 les broches du MPC555 reliées au connecteur J601.

A cause de l'électronique sur la carte, nous ne pouvons qu'utiliser les broches SGPIOD[24:31] en sortie et SGPIOD[16:23] en entrée. En effet toutes ces entrées/sorties sont opto-découplées : sorties entre collecteur et émetteur du transistor de sortie de l'optocoupleur et entrées entre anode et catode de la diode d'entrée de l'optocoupleur.

On souhaite utiliser ces broches en tant que SGPIOD (general purpose inputs/outputs). Pour cela on paramètre les bits SC du registre SIUMCR de la façon suivante :

– SC = 01 (multiple chip, 16-bit port size)

2.4 Connecteur P232-1 : liaison série RS232

Le MPC555 possède deux liaisons série accessibles sur les connecteurs P232-1 et P232-2.

On peut voir dans le tableau 2.6 les broches du MPC555 reliées au connecteur P232-1.

Les broches 2 et 3 peuvent aussi être utilisées comme des entrées sorties générales mais l'électronique sur la carte impose leur utilisation pour la liaison RS232.

Pour transmettre sur le port série X il faut écrire le bit 3 du registre SCCXR1 (0x8). On configure la vitesse de transmission à l'aide de la valeur de SCCXBR

J601	Carte	MPC555
1,3	OUT0-C,OUT0-E	DATA31/SGPIOD31
5,7	OUT1-C,OUT1-E	DATA30/SGPIOD30
9,11	OUT2-C,OUT2-E	DATA29/SGPIOD29
13,15	OUT3-C,OUT3-E	DATA28/SGPIOD28
17,19	OUT4-C,OUT4-E	DATA27/SGPIOD27
21,23	OUT5-C,OUT5-E	DATA26/SGPIOD26
25,27	OUT6-C,OUT6-E	DATA25/SGPIOD25
29,31	OUT7-C,OUT7-E	DATA24/SGPIOD24
2,4	IN0-A,IN0-K	DATA23/SGPIOD23
6,8	IN1-A,IN1-K	DATA22/SGPIOD22
10,12	IN2-A,IN2-K	DATA21/SGPIOD21
14,16	IN3-A,IN3-K	DATA20/SGPIOD20
18,20	IN4-A,IN4-K	DATA19/SGPIOD19
22,24	IN5-A,IN5-K	DATA18/SGPIOD18
26,28	IN6-A,IN6-K	DATA17/SGPIOD17
30,32	IN7-A,IN7-K	DATA16/SGPIOD16
33		
34		

TAB. 2.5: *broches reliées au connecteur J601*

P232-1	Carte	MPC555
2	TX	RXD1/QGPI1
3	RX	TXD1/QGPO1
5		Masse

TAB. 2.6: *broches reliées au connecteur P232-1*

(bits 0 à 12 dans le registre SCCXR0) définie par la formule suivante:

$$\text{vitesse de transmission} = \frac{\text{fréquence du système}}{32 \times \text{SCCXBR}}$$

Pour écrire sur le port série, il faut successivement :

1. écrire dans le registre SCCXR0
2. écrire dans le registre SCCXR1
3. lire le registre SCXSR
4. écrire dans le registre SCXDR

Pour lire on lit au lieu d'écrire à l'étape 4.

On peut aussi utiliser les fonctions Robosoft.

- `int UARTInit (int UartNb, int Baud, int nbit, int nstop, int parity, int BufPtr, int BufSize)` renvoie 0 si l'initialisation s'est bien déroulée. Le buffer sert pour la réception.
- `int UARTClose (int UartNb)`
- `int UARTGetChar (int UartNb, char *C)`
- `int UARTGetWaitChar (int UartNb, char *C)`
- `int UARTGetNbChar (int UartNb)`
- `int UARTPutChar (int UartNb, char x)`

`UartNb` est le numéro de la liaison série (0 ou 1).

2.5 Connecteur P501

On peut voir dans le tableau 2.7 les broches du MPC555 reliées au connecteur P501.

P501	Carte	MPC555
5,4	DATA-C1	MISO/GPIO4
2,1	HCK-C1	SCK/GPIO6
3	ZERO-C1	VFLS0/MPIO3
9		+5V
8		+5V
7		Masse
6		Masse

TAB. 2.7: *broches reliées au connecteur P501*

Les broches 5 et 4 acceptent en entrée un signal différentiel qui est ensuite transformé en signal logique pour le MPC555. Les broches 2 et 1 émettent un signal différentiel à partir d'un signal logique émit par le MPC555. Ces broches peuvent être utilisées comme entrées sorties générales (GPIO) ou en utilisant le QSPI (Queud Serial Peripheral Interface).

Lorsque on utilise le QSPI, il faut le placer en master mode pour que la broche MISO (Master In Slave Out) soit une entrée. Dans ce mode, la broche SCK délivre un signal d'horloge depuis le QSPI.

La broche 3 peut être utilisée comme une sortie pour voir le "Visible History Buffer Flush Status (VFLS)" ou comme une entrée sortie générale.

2.5.1 Utilisation d'un codeur absolu

On veut lire les données renvoyées par un codeur absolu connecté au connecteur P501. Pour cela, on se propose d'utiliser les fonctions fournies par Robosoft.

Initialisation avec la fonction `int InitSPICoder (int Speed)`

Cette fonction détermine la valeur des broches (PORTQS) lorsque elles sont définies comme des entrées sorties générales (tableau 2.8).

Broches	MISO	MOSI	SCK	PCS0	PCS1	PCS2
Valeur	0	0	1	1	1	0

Broches	PCS3	TXD1	RXD1	TXD2	RXD2
Valeur	0	1	0	1	0

TAB. 2.8: *valeur des broches définie par* `InitSPICoder`

Elle détermine ensuite quelles broches sont utilisées comme entrées sorties générales et quelles broches sont utilisées par le QSPI (PQSPAR), ainsi que leur sens avec le registre DDRQS (tableaux 2.9 et 2.10).

Broches	MISO	MOSI	PCS0	PCS1	PCS2	PCS3
Assignation	QSPI	QSPI	I/O	I/O	I/O	I/O

TAB. 2.9: *assignation des broches définie par* `InitSPICoder`

Broches	MISO	MOSI	SCK	PCS0	PCS1	PCS2	PCS3
Sens	I	O	O	O	O	O	O

TAB. 2.10: *sens des broches défini par* `InitSPICoder`

Ensuite elle configure le QSPI à l'aide des registres SPCR[0:3] : master mode, 16 bits par transfert, donnée capturée sur front montant, pas de queue, vitesse déterminée par la formule

$$\text{vitesse de l'horloge} = \frac{\text{fréquence du système}}{2 \times \texttt{Speed}}$$

Ensuite on active le QSPI, avec un délai après transmission de 25,6μs.

La valeur de retour de la fonction est 0 si tout s'est bien passé.

Mise à zéro avec la fonction `int SetZeroSPICoder (void)`

Cette fonction définit le sens des broches MPIO (MPIO[0:11] en sortie et MPIO[12:15] en entrée), puis écrit 1 sur la broche MPIO3 (qui correspond au signal ZERO).

Lecture avec la fonction `int ReadSPICoder (int *Value)`

Cette fonction construit la donnée à envoyer (0x000055AA) puis la place dans la queue d'émission.

Elle place la commande 0x7f dans la queue : met les 4 Chip Select(PCS[0:3]) à 1, détermine les délais de transmission d'après ceux définis par les SPCR, revient à l'état définit par PORTQS après la transmission.

Puisqu'il n'y a qu'une commande, il n'y a qu'un seul transfert de données, et donc on n'envoie que les 16 premiers bits de 0x000055AAA c'est à dire seize 0.

Ensuite on active le QSPI ce qui génère le signal d'horloge et permet la réception sur la liaison série. On décale ensuite la valeur de 3 bits vers la droite pour récupérer les 13 bits du codeur absolu.

Désactiver le QSPI avec `int CloseSPICoder (void)`

Cette fonction permet de désactiver le QSPI, mais le QSPI est désactivé automatiquement lorsque il a terminé sa transmission.

2.6 Connecteur PSCI-1

On peut voir dans le tableau 2.11 les broches du MPC555 reliées au connecteur PSCI-1.

PSCI-1	Carte	MPC555
3	MISO	MISO/GPIO4
2	MOSI	MOSI/GPIO5
1		SCK/GPIO6
4		PCS0/$\overline{\text{SS}}$/QGPIO0
6		+5V
5		Masse

TAB. 2.11: *broches reliées au connecteur PSCI-1*

Les broches 1, 2, 3, 4 peuvent être utilisées par le QSPI ou comme des entrées sorties générales. Le signal $\overline{\text{SS}}$ est utilisé pour placer le QSPI en mode slave.

2.7 Connecteur PCAN1 : bus CAN

La carte possède un second CAN sur le connecteur PCAN2.

Le MPC555 possède deux broches pour émission et réception (CNTX et CNRX). Les signaux sont ensuite transformés à travers un circuit PCA82C250 pour être transmis sur deux fils CANH et CANL (tableau 2.12).

PCAN1	Carte
7	CANH
2	CANL
5	Masse
3	Masse
6	Masse

TAB. 2.12: *broches reliées au connecteur PCAN1*

2.7.1 Utilisation du bus CAN

On se propose d'utiliser le bus CAN à l'aide des fonctions Robosoft.

```
int CANOpen (int nCan, int CanDiv)
```

Cette fonction vérifie que la valeur de `nCan` est valide (0 ou 1). Si elle ne l'est pas, elle renvoie un message d'erreur. Elle met ensuite tous les buffers de transmission en mode non prêt. Elle initialise ensuite le reset du CAN et se place en mode HALT. Suivant le CAN choisi (0 ou 1) place l'interrupt request level à 0 ou 1. On choisit ensuite le buffer 0 qui générera une interruption après une transmission/réception réussie. On configure ensuite à l'aide des registres CANCTRL[0:2]: 0 logique est un bit dominant pour le réception et la transmission, l'interruption sur une erreur est désactivée. Ensuite avec le registre PRESDIV on détermine la fréquence du CAN avec la formule

$$\text{fréquence du CAN} = \frac{\text{fréquence du système}}{\texttt{CanDiv} + 1}$$

On termine le reset. On initialise ensuite les buffers. Retourne 0 si tout s'est bien passé.

```
int CANClose (int nCan)
```

Cette fonction arrête l'horloge du CAN.

Emission et réception de messages avec `int CANSendMsg (byte nCAN, TCanMsgStruct *MsgPtr)` **et** `int CANRecvMsg (byte nCAN, TCanMsgStruct *MsgPtr)`

On utilise la structure TCanMsgStruct décrite ci-après.

```
typedef struct
{
      word id         //  + 0x00
      byte rtr        //  + 0x02
      byte dlen       //  + 0x03
      byte data[8]    //  + 0x04
}TCanMsgStruct
```

Bibliographie

[1] La page de l'Inria :
http://www.inria.fr

[2] La page de l'Inria Rhône-Alpes :
http://www.inrialpes.fr

[3] La page du Service Robotique, Vision et Réalité Virtuelle de l'Inria Rhône-Alpes :
http://www.inrialpes.fr/iramr

[4] La page de Motorola :
http://www.motorola.com

[5] Motorola, "MPC555 User's Manual"

[6] Motorola, "MPC555 Evaluation Board, Quick Reference"

[7] Motorola, "Time Processor Unit Reference Manual"

[8] La page de Virtual Micro Design :
http://www.vmdesign.com/

[9] La page de GCC :
http://www.gnu.org/software/gcc/gcc.html

[10] La FAQ de Cross-GCC :
http://www.objsw.com/CrossGCC/

[11] Utilisation des outils GNU pour les systèmes embarqués :
http://www.redhat.com/support/manuals/gnupro98r2/7_embed/emb.html

[12] Advanced Motion Controls, "Series 50A-DD PWM brush type servo amplifiers"

[13] Parvex, "Servo-amplificateur SBS $2^{ème}$ génération, Notice d'utilisation"

[14] Agilent Technologies, "Panel Mount Optical Encoders, Technical Data"

[15] Directed Perception, "Computer Controlled Pan Tilt Unit User's Manual"

[16] J-M Bourgeat, P-A Degaches et C. Despinasse, “Ceinture à ultrasons”, IUT GEII1, 99/00

[17] Polaroid, “6500 Series Sonar Ranging Module”

[18] Analog Devices, “DAC8420”

[19] X. Margueron, “Capteur pour la localisation d’une source sonore”, Rapport de Stage, IUT GEII1, 2000

Annexe A

Fichier de configuration SDS pour le nœud Robosoft

```
[MPC555]
CFGFILE=C:\sds75\cmd\mpc555.cfg
FILETIME=961764546
MSR=0x0
IMMR=0x30100800
BBCMCR=0x0
SIUMCR=0x400000
SYPCR=0xFFFFFF00
BR0=0x2
OR0=0xF6
BR1=0xFFF00823
OR1=0xFFFE0000
BR2=0xFF00823
OR2=0xFF00000
BR3=0x0
OR3=0x0
DMBR=0x80000
DMOR=0x80000
SCCR=0x81210100
PLPRCR=0x915000
DPTMCR=0x100
RAMBAR=0x1
PORTQS=0xF6E
PQSPAR_DDRQST=0x0
MPIOSMDR=0xFFEF
MPIOSMDDR=0x0
MIOS1TPCR=0x0
SRAMMCR=0x0
SGPIODT1=0x0
SGPIODT2=0x0
```

```
SGPIOCR=0x0
EMCR=0x126C
UMCR=0x0
PQSPAR/DDRQS=0x0
VECTADDR=0x0
STRAYINT=0x0
[CONFIGURATION]
PROCESSORS=MPC555
LASTPROC=MPC555
HEX=TRUE
```

Annexe B

Fichier d'initialisation "C run time 0"

```
#*****************************************************************************
#* File:  CRT0.S
#* Version: 1.00,   01/10/1999
#* Author:  Philippe TECHER,  Virtual Micro Degin (ptecher@vmdesign.com)
#* Modified by: M. GHRISSI 04/11/99
#-----------------------------------------------------------------------------
#* Modification:
#*   1.00 - Base version
#-----------------------------------------------------------------------------

#*****************************************************************************

        .file           "crt0.s"
        .text
        .globl          start
        .align          2

.include  "555regsdef.s"

#*****************************************************************************
.macro  Set32 AdRgHi,AdRgLo,ValHi,ValLo
addis r11,r0,\AdRgHi
ori r11,11,\AdRgLo
addis r12,r0,\ValHi
ori r12,r12,\ValLo
stw r12,0(r11)
.endm

.macro  Set16 AdRgHi,AdRgLo,Value
addis r11,r0,\AdRgHi
ori r11,r11,\AdRgLo
li r12,\Value
```

```
sth r12,0(r11)
.endm

#*****************************************************************************
#*****************************************************************************

        addi            r0,r0,0         # Debuggers may object to starting at 0.

start:
addis r11,r0,__SP_INIT@ha # Initialize stack pointer r1 to
addi r1,r11,__SP_INIT@l # value in linker command file.
addis r13,r0,_SDA_BASE_@ha # Initialize r13 to sdata base
addi r13,r13,_SDA_BASE_@l # (provided by linker).
addis r2,r0,_SDA2_BASE_@ha # Initialize r2 to sdata2 base
addi r2,r2,_SDA2_BASE_@l # (provided by linker).
addi r0,r0,0 # Clear r0.
stwu r0,-64(r1) # Terminate stack.

#*****************************************************************************
# Insert other initialize code here.
mfmsr r0
ori r0,r0,0x2002
mtmsr r0

li r0,0x07
mtspr 158,r0

# set16 0x002F, 0xC288, -0x0001 # clear reset status

li r0,0

#
# Finished user initialisation


bl __init_main     # Finishes initialization (copies .data
                                        # ROM to RAM, clears .bss), then calls
                                        # example main(), which calls exit(),
                                        # which halts.
b exit # Never returns.
# bl main # Dummy to pull in main() as soon as
# possible.
#------------------------------------------------------------ .init section --
        .section        .init
        .globl          __init
        .align          2
__init:                                 # Entry to __init, called by
```

```
        mfspr           r0,8            # __init_main called above.
        stwu            r1,-64(r1)
        stw             r0,68(r1)

        # Linker places .init sections from other modules, containing
        # calls to initialize global objects, here.

        .section        .init
        lwz             r0,68(r1)       # Return from __init.
        addi            r1,r1,64
        mtspr           8,r0
        blr

#---------------------------------------------------------- .fini section --
        .section        .fini
        .globl          __fini
        .align          2
__fini:                                 # Entry to __fini, called by exit().
        mfspr           r0,8
        stwu            r1,-64(r1)
        stw             r0,68(r1)

        # Linker places .fini sections from other modules, containing
        # calls to destroy global objects, here.

        .section        .fini
        lwz             r0,68(r1)       # Return from __fini.
        addi            r1,r1,64
        mtspr           8,r0
        blr
```

Annexe C

Script pour l'éditeur de lien

```
/*****************************************************************
 * (c) ROBOSOFT (pierre@robosoft.fr)
 *
 * Example of linker script for non-SynDEx generated code. Need to
 * be modified according to your own application.
 ----------------------------------------------------------------
 * Warning: this linker script is expected to link programs with
 * 'libc.a' and 'libm.a' built using Diab Data libraries!
 *****************************************************************/

STARTUP(crt0.o)

MEMORY {
  ram1  : org = 0x3F9800, len = 26K /* MPC555 internal RAM */
  ram   : org = 0xFFF00000, len = 128K /* ROBOSOFT external RAM */
  flash : org = 0x00000000, len = 448K /* mpc555 internal FLASH */
}
ENTRY(start) /* first instruction to be executed after load completes */

/*****************************************************************
 * If you intend to create a SynDEx binary compatible file to allow
 * CAN bus download into MPC555 target, you have to tag, here,
 * the segments you need to include in this binary file. PT_LOAD is
 * the Program Header flag that will indicate this. If segments are
 * not needed you just have to omit them from the list.
 *****************************************************************/
PHDRS {
  ithandler   PT_LOAD;
  fpu         PT_LOAD;
  dec         PT_LOAD;
  text        PT_LOAD;
  data        PT_LOAD;
  sdata2      PT_LOAD;
}
```

```
/*****************************************************************
 * Note: in order to simplify the download process, each loadable
 * section is 8 bytes aligned. By this way, at download time, we will
 * send completly filled CAN frames (8 data bytes). It principally
 * avoid us to deal with bytes padding.
 *****************************************************************/

SECTIONS {
  /*******************************************************
   * Some sections provided by '.org' directives after
   * compilations under Diab Data compiler
   *******************************************************/

  /*******************************************************
   * Section containing interrupt handler code
   *******************************************************/
  .ithandler 0xFFF00500 : {
    *(.ithandler)
    . = ALIGN(8) ;
  } >ram :ithandler

  /*******************************************************
   * Section containing fpu exeption handler code
   *******************************************************/
  .fpu 0xFFF00800 : {
    *(.fpu)
    . = ALIGN(8) ;
  } >ram :fpu

  /*******************************************************
   * Section containing decrementer exception handler code
   *******************************************************/
  .dec 0xFFF00900 : {
    *(.dec)
    . = ALIGN(8) ;
  } >ram :dec

  /*************************************************
   * .text section for code, constants, and init data
   *************************************************/
  .text 0xFFF02000 : {
    __IT_END = ABSOLUTE(.);
    *(.text)   /* code */
    *(.rodata) /* des commentaires seraient bienvenus */
    *(.init)   /* pour expliquer le role de ces 4 sections */
    *(.fini)
```

```
  *(.eini)
  _etext = ALIGN(8) ; /* cleanup alignment for following ROMed data */
  . = ALIGN(8) ;
} >ram :text /* > flash */

/*****************************************************
 * .data section for initialized data
 * linked in ram1, but loaded after .text if ROMable
 *****************************************************/
.data 0x3F9800 : { /* AT(ABSOLUTE(_etext)) if ROMable) */
  __DATA_ROM = ABSOLUTE(.) ;   /* Needed for Diab Data library linking */
  __DATA_START = ABSOLUTE(.) ;  /* start of ROM>RAM transfer at startup */
  __DATA_RAM = __DATA_START ;   /* Needed for Diab Data library linking */
  *(.data)
  . = ALIGN(8) ;                /* cleanup alignment for following sdata */
  __SDATA_START = ABSOLUTE(.) ; /* init value for r13, see gcc -mdata-sysv */
  *(.sdata)                     /* "short" initialized data */
  . = ALIGN(4) ;
  __DATA_END = ABSOLUTE(.) ;    /* end of ROM>RAM transfer at startup */
  __DATA_SIZE = ( __DATA_END - __DATA_START ) >> 2 ; /* words to transfer */
                /* PP: use >>2 in spite of /4 cause of a ld possible bug. */
  . = ALIGN(8) ;
} >ram1 :data

/*********************************************************
 * .bss section for uninitialized data, cleared at startup
 * WARNING: "short" sections .sdata and .sbss must be close
 *          to each other; DO NOT change the relative order
 *          of the .data and .bss sections in this file
 *********************************************************/
.bss (NOLOAD) : {
  . = ALIGN(8) ;
  __BSS_START = ABSOLUTE(.) ;
  *(.sbss)                      /* "short" uninitialized data */
  *(.bss) *(COMMON)             /* all other uninitialized data */
  . = ALIGN(4) ;
  __BSS_END = ABSOLUTE(.) ;
  __HEAP_START = __BSS_END;     /* Needed for Diab Data library linking */
  __BSS_SIZE = ( __BSS_END - __BSS_START ) >> 2 ; /* words to clear */
       /* PP: use >>2 in spite of /4 cause of a ld possible bug. */
} >ram1

/*****************************************************
 * .stack section for uninitialized stack of call-frames
 * Note: the stack is aligned on cache lines
 *****************************************************/
.stack (NOLOAD) : {
```

```
    __SP_END = ABSOLUTE(.) ;       /* pour check_stack_overflow ? */
    . += 6144 ;                    /* 192 stack frames of 32 bits each */
    . = ALIGN(32);                 /* align stack-frames on cache lines */
    __SP_INIT = ABSOLUTE(.) ;
  } >ram1

  .sdata2 : {
*(.sdata2)
  } >ram1 :sdata2

  .got (NOLOAD) : {
    *(.got)
    __HEAP_END = ABSOLUTE(.) ;     /* Needed for Diab Data library linking */
  } >ram1
}
```

Résumé

LYDOIRE Fabien (Filière : Itra) DUREE : 5 mois

Programmation d'une centrale de contrôle-commande à base de Power-PC

Inria Rhône-Alpes - Montbonnot

L'Inria possède un véhicule électrique (le CyCab) construit par Robosoft doté d'une centrale de contrôle-commande à base de MPC555. Il s'agit d'étudier la carte (nœud Robosoft) sur laquelle est placé ce micro-contrôleur afin d'en déterminer ses possibilités.

Dans un premier temps je me suis familiarisé avec le micro-contrôleur et l'outil de *debug* SDS avec une carte d'évaluation. Ensuite, un nœud Robosoft ainsi qu'une bibliothèque permettant son utilisation dans le cadre du CyCab ont été fournis. J'ai mis en place des outils de développement pour le MPC555, étudié et validé le code fourni par la société Robosoft. L'étude de l'interface avec le monde extérieur (connecteurs présents sur la carte) ouvre des perspectives d'utilisation de la carte dans d'autres applications en complétant la bibliothèque d'origine.

Ce stage a présenté toutes les étapes de développement d'une application de contrôle-commande à base de micro-contrôleur en partant de la carte électronique, en passant par l'assembleur, pour arriver au langage C.