ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MATRÎSE EN GÉNIE ÉLECTRIQUE
M.Ing

PAR
PIERRE GRONDIN

GÉNÉRATION DE CODE CIBLANT UN MICROCONTRÔLEUR MPC555 À PARTIR D'UN MODÈLE SIMULINK

MONTRÉAL, LE 7 MAI 2003

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE:

M. Maarouf Saad, directeur de mémoire
Département de génie électrique à l'École de technologie supérieure

M. Roger Champagne, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Jean Bélanger, jury externe
Opal-RT Technologies

IL A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 19 MARS 2003

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

GÉNÉRATION DE CODE CIBLANT UN MICROCONTRÔLEUR MPC555 À PARTIR D'UN MODÈLE SIMULINK

Pierre Grondin

SOMMAIRE

Le but de ce projet est de permettre la génération d'un modèle Simulink en code C afin de l'exécuter sur une carte de développement avec un MPC555. L'hypothèse de base est que les générateurs de code actuels sont suffisamment configurables pour le ciblage de processeurs embarqués. Une des applications futures visées par ce projet est le contrôle d'un robot à cinq degrés de liberté.

Le générateur de code Real-Time Workshop (RTW) de MathWorks a été sélectionné pour ce projet. Lors du développement, on doit d'abord mettre en place un modèle de génération de code et un séquenceur. Ensuite, on développe des blocs Simulink utilisant les entrées-sorties du MPC555.

Une application de contrôle en position d'un moteur DC et une interface graphique Labview permettent de confirmer le bon fonctionnement du projet. On arrive à la conclusion que cette solution est très intéressante pour le prototypage rapide d'applications de contrôle.

SIMULINK CODE GENERATION TARGETTING A MPC555 MICROCONTROLLER

Pierre Grondin

ABSTRACT

The objective of this project is to execute Simulink generated code on a development board with a MPC555. The main hypothesis is that standard code generator performance is good enough to target embedded processors. The MPC555 microcontroller is used in many control applications including combustion control in cars. One future application for this project is to control a five degrees of freedom robot.

Development starts with the analysis of different code generation solutions. Real-Time Workshop from MathWorks is selected for its availability and price. A code generation template and scheduler are then developed. Once generated, the code is compiled and executed in the CodeWarrior environment from Metrowerks. A synchronization feature was then added to the scheduler. Two Simulink blocks have been developed for interaction with the simulation through a terminal software connected to an RS-232 port. Two other blocks allow using the Quadrature Decoding and the Pulse Width Modulation output of the MPC555. All blocks follow Simulink's S-function format.

A position controlled DC motor application and a Labview graphical interface were created to assess the overall solution. The interface allows changing model parameters and visualizing motor position on a scope indicator. Model time step is 1 ms. After performing different tests, it was established that the computation time required for each block was less than 10 µs with the exception of the terminal print block where the `sprintf` function takes about 120 µs per variable. The application requires 150 KB of Flash and 36 KB of RAM which is reasonable for the MPC555. With all the results obtained, it was concluded that this solution is very interesting for doing rapid control prototyping.

As a recommendation, high speed data acquisition feature using the CAN or the second RS-232 port would constitute an interesting addition.

REMERCIEMENTS

Ce projet n'aurait pas été possible sans l'appui de ma famille, ma conjointe et mes amis qui m'ont encouragé tout au long de mes travaux. J'en profite pour souligner la patience de Florence, ma petite fille, qui aurait bien aimé jouer avec papa plutôt que de le voir travailler sur l'ordinateur.

Je remercie aussi Maarouf Saad, mon directeur de mémoire, qui m'a appuyé et conseillé dans mes différentes démarches. La planification et les rencontres que nous avons eues m'ont permis de me diriger à coup sûr vers une solution gagnante.

Finalement, je remercie mon employeur Opal-RT pour m'avoir permis d'évoluer dans le milieu de la simulation temps réel pendant les cinq dernières années. Je remercie aussi Jean Bélanger, mon supérieur, pour sa compréhension lors des nombreuses absences requises afin de travailler sur ce projet.

TABLES DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS ET SIGLES

ADI	Applied Dynamics International
ASAM	Association pour la standardisation et l'automatisation des systèmes de mesures (Association for Standardisation of Automation and Measuring system)
BDM	Port de débogage (Background Debug Mode)
CAN	Control Area Network
CISC	Ordinateur avec jeu d'instructions complexes (Complex Instruction Set Computer)
CW	Code Warrior
DEL	Diode électro-luminescente
DLL	Lirairie dynamique (Dynamic Link Library)
E/S	Entrée-sortie
ECU	Unité électrique de contrôle (Electric Control Unit)
FAA	Administration fédérale de l'aviation (Federal Aviation Administration)
FPGA	Matrice de portes logiques à programmation rapide (Fast Programmable Gate Array)
ISI	Integrated Systems, Inc
PIT	Compteur avec interruption périodique (Periodic Interrupt Timer)
PWM	Modulation de largeur d'impulsion (Pulse Width Modulation)
RISC	Ordinateur avec jeu d'intructions simples (Resumed Instruction Set Computer)
RTW	Real-Time Workshop
RTWEC	Real-Time Workshop Embedded Coder

TPU	Unité de traitement du temps (Time Processing Unit)
UML	Langage de modélisation unifié (Unified modeling language)

INTRODUCTION

La mondialisation des marchés amène une concurrence très forte obligeant les milieux industriels à trouver de nouvelles méthodes pour développer leurs produits rapidement tout en réduisant leur coût de production. De ce besoin naissent les outils de prototypage rapide basés sur la modélisation et la simulation. Ces outils permettent déjà de réduire considérablement le temps de développement en générant automatiquement des programmes correspondant à la modélisation effectuée dans Simulink ou autres logiciels similaires. Les programmes produits sont ensuite utilisés avec différentes cartes d'acquisition pour tester les algorithmes finaux sur des bancs d'essai. Cette technique a cependant une limite ; les simulations sont généralement effectuées sur un PC et non sur les processeurs embarqués utilisés pour le produit fini. Depuis un peu moins de trois ans, des logiciels de génération de code pour processeurs embarqués ont fait leur apparition. Ceux-ci sont presque exclusivement employés par les constructeurs d'automobiles et sont très dispendieux.

Le projet qui suit vise à utiliser un générateur de code Simulink standard et de l'adapter afin de cibler un microcontrôleur MPC555 de Motorola. Ce dernier contient des ports de communication ainsi que plusieurs fonctionnalités d'entrées-sorties ce qui en fait un processeur polyvalent. Ce projet repose sur l'hypothèse suivante : les générateurs de code standard actuels sont suffisamment configurables et optimisés pour le ciblage de processeurs embarqués.

Une des applications futures visée par ce projet est le contrôle d'un robot à cinq degrés de liberté. On peut cependant affirmer que cette solution sera utile dans plusieurs applications à faible déploiement nécessitant un microcontrôleur.

Les objectifs généraux sont les suivants :

- évaluer les options de génération de code permettant un code de taille réduite;

- développer un modèle de génération de code permettant une utilisation des entrées-sorties du MPC555 ainsi qu'une interaction avec le programme à partir du terminal d'un PC externe;
- tester le code généré avec une application physique (banc d'essai avec un moteur DC);
- tester les performances du code généré.

Le document débute avec un bref historique de l'évolution des solutions de génération de code ainsi qu'une évaluation des solutions actuelles sur le marché. On y présente différentes publications scientifiques sur les technologies relatives au projet. À la suite de cette étude, un générateur de code est sélectionné et le plan de développement est établi. Suivent les différentes étapes de développement dont une évaluation du MPC555 et le développement d'une application de test. Celle-ci permet de se familiariser avec l'environnement de développement et les fonctions disponibles. Une deuxième étape consiste à programmer et configurer la génération code afin d'exécuter un modèle Simulink très simple sur le MPC555. Suite à cette première génération complète, le développement se complète en ajoutant le support des fonctions d'entrées-sorties et d'interaction à l'aide d'un PC branché sur le port série. L'application finale utilise les entrées-sorties du MPC555 pour créer un contrôleur de moteur en position.

Cette méthode de développement graduelle permet d'isoler facilement les erreurs et de développer le tout plus efficacement. Les schémas présentés dans le document seront, lorsqu'approprié, faits selon le langage de modélisation objet unifié (UML).

CHAPITRE 1

REVUE DE LA LITTÉRATURE

1.1 Introduction

La section qui suit présente un bref historique des générateurs de code pour Simulink ou autres outils similaires. On y retrouve aussi une description des principaux générateurs de code disponibles sur le marché. Dans une seconde partie, on fait la revue de certaines publications scientifiques relatives à la génération de code.

1.2 Historique des générateurs de code

Bien que plusieurs ingénieurs ne connaissent que l'environnement de modélisation Simulink, il est intéressant de savoir que le précurseur en ce domaine était Integrated Systems, Inc (ISI). Cette compagnie, achetée par Wind River en 1999, a développé une suite d'outils nommés MATRIXx, SystemBuild et Autocode qui est l'équivalent de MATLAB, Simulink et Real-Time Workshop (RTW) de MathWorks. Au niveau de la génération de code, la première version d'Autocode a été produite en 1987 [Wind River, 2002] alors que l'équivalent MathWorks, RTW, n'a été mis en marché qu'en 1994 soit environ sept ans plus tard [MathWorks, 2002].

En 1995, la compagnie Applied Dynamics produisait Beacon, un générateur de code utilisé avec leur propre logiciel de simulation. Depuis 1997, ce générateur permet aussi la génération de code pour les modèles Simulink. [ADI, 2002]

Finalement, TargetLink de dSpace est apparu à la fin de 1999. Ce générateur de code Simulink est possiblement le premier générateur destiné à créer du code de production pour les "Electric Control Unit" (ECU) des automobiles [dSpace, 2002]. Un ECU est un module contenant un microcontrôleur qui est utilisé pour gérer différents éléments de la voiture tels que la combustion du moteur et la transmission. Ici, le terme code de production a beaucoup d'importance puisqu'il signifie que le code ne doit pas être modifié par des programmeurs avant d'être transféré dans les ECU des voitures. Ce

nouveau paramètre requiert une production de code sécuritaire dont la taille doit être réduite afin de minimiser les coûts liés à la mémoire flash contenant le programme.

En septembre 2000, MathWorks réplique en produisant Real-Time Workshop Embedded Coder, une extension de RTW permettant de générer du code de production pour des processeurs embarqués [MathWorks, 2002].

Finalement, en mai 2001, on annonce que The MathWorks prend en charge le support technique de MATRIXx suite à un accord avec Wind River. Cet évènement marque très certainement la fin de MATRIXx, SymtemBuild et AutoCode, les plus proches compétiteurs de la famille de produits MathWorks [MathWorks, 2001]. Un dernier évènement survient en août 2002 alors que le département de la justice américaine annonce que MathWorks devra vendre MATRIXx en vertu de la loi antitrust américaine [Département de justice américaine, 2002]. Finalement, le 20 février 2003, National Instruments achète MATRIXx et devrait relancer le développement du produit sous peu [National Instruments, 2002].

1.3 Évaluation des différents générateurs de code

1.3.1 Beacon

Selon les informations obtenues sur leur site Internet, BEACON vise principalement les applications critiques des milieux militaires et aéronautiques. Leur générateur de code respecte la norme de développement de logiciel RTCA/DO-178B établie par le Federal Aviation Administration (FAA).

Le générateur vérifie le code généré pour s'assurer qu'il n'y ait pas de code non-sécuritaire. Un code sécuritaire doit remplir les critères suivants:

- code structuré (aucun goto, une entrée et une sortie);
- mémoire statique (pas de récursion, aucune mémoire allouée sur la pile);
- "strong typing" (utilisation des variables toujours dans le même format);

- usage rigoureux des données;
- les décisions de cas ont un chemin par défaut;
- pas d'utilisation variable des paramètres de fonction;
- pas de réassignation des entrées.

Le code peut être généré en langage Ansi-C et ADA-83/95

ADI offre aussi Beacon Tester, un logiciel permettant de générer des séquences de tests pour le programme créé. Chaque séquence permet de tester rapidement plusieurs éléments du code dont la variation des paramètres d'entrées et le test de toutes les décisions d'un diagramme d'états.

Le prix de vente de ADI Beacon pour Simulink est de plus de USD$ 10,000.

1.3.2 Real-Time Workshop

Real-Time Workshop est de loin le plus populaire de tous les générateurs de code. Bien qu'il n'ait pas été créé pour le code de production, bien des applications de production à bas volume reposent sur son code. La taille et la sécurité moins élevée du code n'ont d'ailleurs pas beaucoup d'importance pour la majorité des applications de contrôle et de prototypage rapide.

Beaucoup de logiciels de prototypage rapide pour Simulink utilisent RTW. Voici une liste non exhaustive des solutions les plus populaires :

- RT-LAB de Opal-RT Technologies (Montréal, Canada) : supporte SystemBuild - Autocode et Simulink - RTW sur des PC avec cartes d'entrées-sorties de plusieurs manufacturiers dont National Instruments. Son utilisation se fait dans plusieurs domaines dont la robotique, l'aérospatiale et les simulations distribuées;

- dSpace Simulator de dSpace (Paderborn, Allemagne) : supporte Simulink-RTW et Simulink-TargetLink. Utilisé principalement dans le milieu de l'automobile et de la robotique, le logiciel supporte une gamme de cartes processeurs et d'entrées-sorties fabriquée par dSpace;
- xPC de The MathWorks (Natick , États-Unis) : supporte Simulink - RTW. Cet outil est populaire dans les milieux éducationnels et dans certaines applications de contrôle. Il supporte des cartes d'entrées-sorties de plusieurs manufacturiers.

RTW génère le code en fonction de modèles ("templates") qui peuvent être modifiés selon le système d'exploitation et le processeur utilisé. RTW fonctionne directement sur les systèmes d'exploitation suivants :

- AIX;
- Digital Unix;
- HP-UX 10 et 11;
- IRIX/IRIX64;
- Linux;
- Solaris;
- SunOS 4;
- Windows.

Pour tout autre système d'exploitation, l'usager doit modifier le modèle de génération de code.

RTW supporte les variables entières de 8,16 et 32 bits ainsi que les nombres à point flottant de précision simple (32 bits) et double (64 bits).

Le prix de vente de RTW est de USD$ 7,500 .

1.3.3 Real-Time Workshop Embbedded coder

Real-time Workshop Embedded Coder (RTWEC), dont la version 3.0 est parue à l'automne 2002, est un ajout à Real-Time Workshop qui permet la génération de code de production. Ses modèles de génération de code sont beaucoup plus évolués ce qui constitue sa principale différence par rapport à RTW. On peut aussi définir les classes de données qui sont créées ainsi que leur présentation dans le code généré. MathWorks affirme que la qualité du code produit par RTWEC équivaut à celle d'un programmeur tel que demontré par la figure 1.

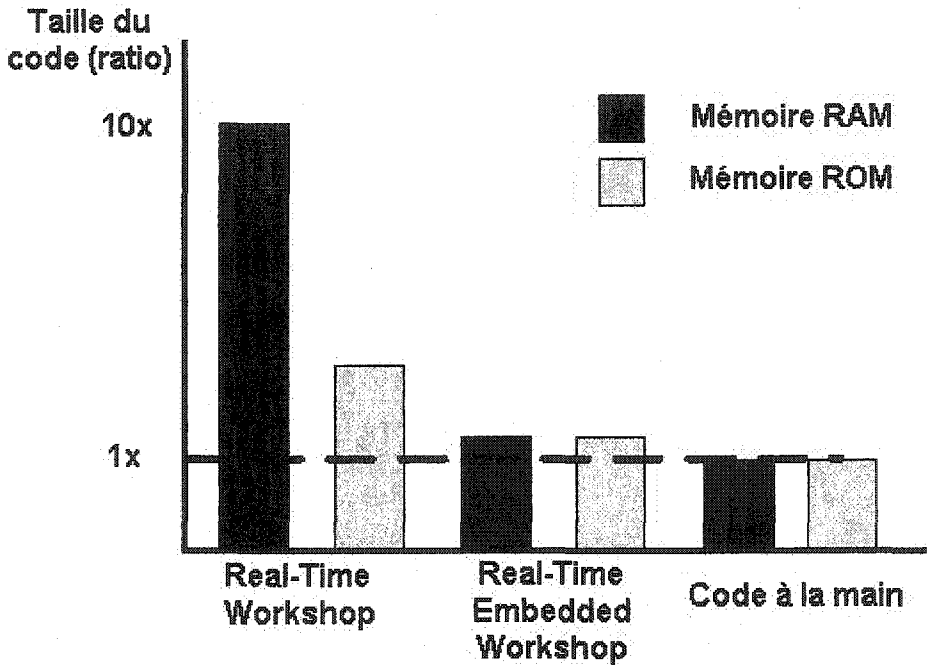

Figure 1 Comparaison de la mémoire RAM/ROM en fonction de la méthode.

Une fonction d'analyse de code produit un rapport en format HTML incluant des liens sur le modèle généré.

Il est aussi possible d'exporter des données de simulation en format ASAM/ASAP2. Ce dernier format a été créé par L'Association for Standardisation of Automation and Measuring Systems (ASAM), un regroupement de producteurs d'automobiles et de

plusieurs de leurs partenaires. Ce groupe a pour but de standardiser les processus de tests, d'instrumentation et de diagnostic requis pour la production d'automobiles. Ce dernier point confirme le marché visé par RTWEC [Automotive Test Report, 2002].

Le prix de vente de RTWEC V3.0 est de USD$ 5,000. Si l'on ajoute à cela le prix de RTW qui est requis, le total est de USD$ 12,500. On peut aussi acheter séparément des modules spécifiques pour certains processeurs comme le MPC555.

1.3.4 TargetLink

dSpace produit depuis environ 10 ans des solutions de prototypage rapide. L'arrivée de TargetLink à la fin de 1999 leur a permis de supporter à la fois RTW pour les applications de contrôle standard et TargetLink pour les applications de production. Tout comme RTWEC dont la version 3.0 vient tout juste de paraître, TargetLink 1.3 est aussi un logiciel très jeune.

TargetLink possède des fonctionnalités semblables à celle de RTW et RTWEC. Les modèles de génération de code sont déjà disponibles pour plusieurs processeurs utilisés dans le milieu automobile. Voici une liste des processeurs supportés :

- Hitachi SH-2 avec compilateur Hitachi;
- Hitachi H8S avec compilateur Hitachi;
- Infineon C16x avec compilateur Tasking;
- Infineon TriCore avec compilateur Tasking;
- Mitsubishi M32R avec compilateur GAIO;
- Motorola 683xx avec compilateur Microtec;
- Motorola HC12 avec compilateur Cosmic;
- Motorola MPC555 avec compilateur Diab-SDS et Green Hills;
- Texas Instruments TMS470 avec compilateur TI.

Tout comme RTWEC, la génération de fichiers de données de type ASAM/ASAP2 est possible.

La solution TargetLink comporte plusieurs options ce qui complique le calcul du prix de la solution. L'usager doit d'abord acheter le TargetLink Base Suite. Ensuite, on doit se procurer les modules d'optimisation et de simulation pour le processeur choisi. La génération du module ASAM/ASAP2 est aussi une option.

1.4 Interface graphique

Les logiciels mentionnés ci-haut proposent différentes options pour la création d'interface graphique. dSpace reste fidèle à leur architecture relativement fermée. Ils ont développé Cockpit, un outil similaire à Labview de National Instrument. Il est aussi possible d'utiliser des bibliothèques de fonctions en format C ou m (format utilisé dans MATLAB) pour s'interfacer avec une application de visualisation externe.

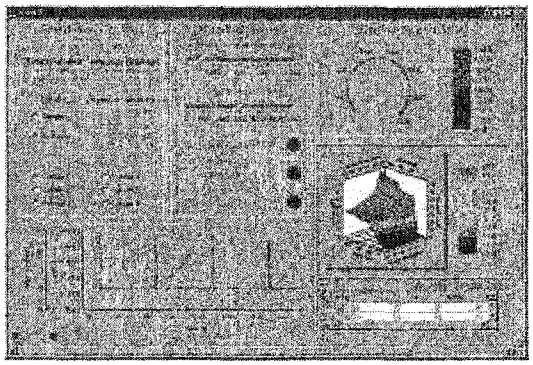

Figure 2 Exemple d'interface avec Cockpit de dSpace

Les solutions de ADI s'interfacent avec Altia de la compagnie du même nom. Altia est un outil de développement d'interface haut de gamme permettant la génération du code de l'interface graphique ce qui n'est pas le cas avec Cockpit [Altia, 2002]

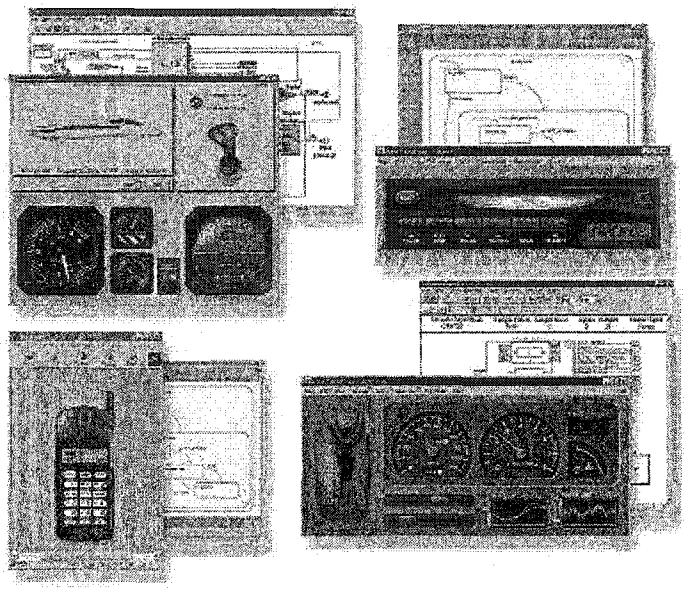

Figure 3 Exemple d'interface Altia

Les interfaces graphiques pour les générateurs RTW et RTWEC sont disponibles à travers les différents logiciels de prototypage mentionnés précédemment (RT-LAB, dSpace et xPC).

RT-LAB supporte une connexion à des interfaces Altia, Labview et optionnellement Sense8 WorldUp viewer pour les environnements de réalité virtuelle. Tout comme Altia, l'environnement Labview permet la génération de l'interface sous forme d'une application Windows. Le code source n'est cependant pas fournit à l'usager. RT-LAB

permet aussi de s'interfacer avec des applications externes à l'aide de bibliothèques de fonctions C et m.

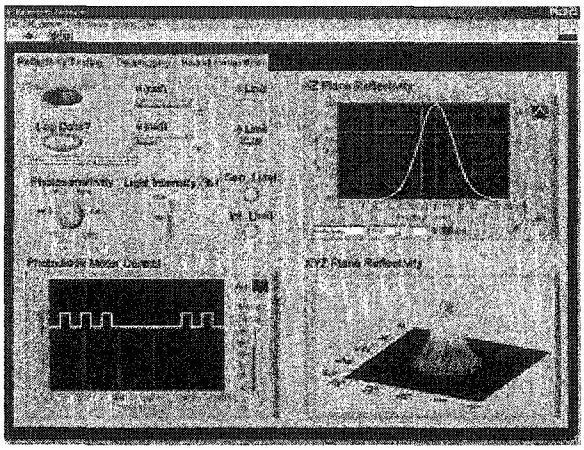

Figure 4 Exemple d'interface Labview

xPC offre de son coté une interface minimale composée d'oscilloscopes, d'affichages et de contrôles [MathWorks, 2002].

1.5 Publications scientifiques

Les recherches sur la génération de code permettent de constater que les articles scientifiques ne sont pas très nombreux et proviennent principalement du milieu de l'automobile et des universités. Curieusement, le domaine de l'aérospaciale est presque absent de celles-ci. Les sous-sections suivantes décrivent certaines publications et permettent d'établir l'état actuel de la technologie.

1.5.1 Analyse de génération de code embarqué

La publication *Embedded Software Analysis and Generation* de l'université de Californie à Berkeley [Bostic et al, 2001] évalue l'état de la technologie et la direction à suivre afin de permettre l'utilisation de la modélisation et de la génération de code pour la création de logiciels embarqués pour les voitures. D'une manière générale, plusieurs recherches réfèrent à l'expression Model-Based Integration of Embedded Software

(MoBIES) pour représenter l'ensemble des recherches dans ce domaine. Plusieurs entreprises dont GM Motor Company, Ford et Motorola Automotive Group participent à l'étude de l'université Californie de Berkeley. La présence de ces entreprises dans la recherche permet l'échange d'informations sur les problèmes rencontrés lors d'expériences pratiques sur le sujet. L'étude est divisée en quatre sections représentant les éléments critiques du processus de développement. Les problèmes et objectifs cités dans la publication sont liés à l'utilisation des outils Matlab / Simulink et Real-Time Workshop. On note cependant qu'on demeure ouvert à l'évaluation d'autres solutions.

1.5.1.1 Génération de vecteurs de tests

Cette étape du développement consiste à définir tous les tests qui devront être faits afin d'assurer un programme final robuste et sécuritaire. Au niveau des diagrammes d'états faits avec la boîte à outils StateFlow de MathWorks, on devra s'assurer de tester chaque état et chaque transition au moins une fois. On doit aussi tester plusieurs autres points comme les différents cas de logique conditionnelle ainsi que les minimums, maximum et le fonctionnement de chaque bloc. Comme on utilise un environnement de modélisation, les tests sont définis dans une partie du modèle Simulink représentant le programme embarqué.

1.5.1.2 Gestion des tâches

On indique ici qu'un outil devra être développé afin de permettre d'analyser les performances d'exécution des différentes tâches du programme embarqué. Dans un premier temps, on devra mesurer le pire cas de temps d'exécution en forçant l'exécution de toutes les boucles de calcul conditionnelles dans un même pas de calcul. Lors d'une deuxième étape, on doit confirmer que toutes contraintes de temps des requis sont respectées. Ces tests doivent tenir compte de l'impact de la préemption d'une tâche de plus haute priorité sur une autres tâche, de l'utilisation d'antémémoire et du temps d'accès aux différentes régions de mémoire ou entrées-sorties. Outre ces informations

sur les fonctionnalités de l'application d'analyse, aucune mention n'est faite sur l'implémentation future de celle-ci.

1.5.1.3 Génération de code

Dans cette section, on discute des limites actuelles de la technique de génération de code et de ses avantages par rapport à la méthode actuelle de développement de code à la main. Tout d'abord, on note que la technique actuelle (à la main) est sujette aux erreurs et à l'inefficacité humaine. Aussi, les styles différents de chaque programmeur rendent l'analyse du code difficile. La technique de génération de code permet un code cohérent, un temps de génération plus court et l'assurance que l'interprétation du modèle de référence est exacte. On note que les générateurs de code actuels tels Real-Time Workshop de MathWorks et TargetLink de dSpace ne produisent pas un code suffisamment petit pour les ressources limitées des solutions utilisées dans le milieu de l'automobile.

1.5.1.4 Système d'exploitation temps-réel et séquenceur

Le dernier point mentionné dans cet article est l'importance du séquenceur de tâche et le désir des manufacturiers d'investir un minimum de ressources et de temps pour celui-ci. On note que le besoin typique des applications automobiles est de séquencer environ dix tâches avec un pas de calcul minimum de 1ms. Il est évidemment important de minimiser l'utilisation de mémoire RAM et ROM. À titre d'exemple, on indique que le séquenceur utilisé pour un PowerPc 5XX devrait utiliser un maximum de 19 Ko de ROM et 6 Ko de RAM.

1.5.2 Technique de séquençage dynamique

La publication *Dynamic Run-Time HW/SW Scheduling Technique* [Noguera et al, 2002] du Technical University of Catalonia (UPC) discute de la technologie de séquençage dynamique pouvant être utilisée dans des systèmes embarqués sur puce ou System On a Chip (SOC). On y mentionne que le séquençage des tâches était généralement statique

mais que l'augmentation de complexité des programmes justifie maintenant l'utilisation de séquenceurs dynamiques. Bien qu'un peu trop évolué pour l'application désirée de ce projet, il est intéressant de constater l'importance de l'architecture du séquenceur. Ce point est discuté au chapitre 5 lors de la mise en place d'un premier modèle de génération de code.

1.5.3 Impact de la génération de code sur le cycle de développement

Dans *Code generation speeds debug cycle* du EETimes [McElroy, 2002], on discute de l'impact de l'utilisation de la technique de génération de code à partir de schéma UML sur le temps de développement et de débogage d'une application. On mentionne d'abord que l'arrivée des générateurs de code a remodelé la séquence traditionnelle de développement qui était composée dans l'ordre de l'analyse, du design, de l'implémentation et finalement des tests. Avec la technique de génération de code, on peut vérifier et corriger plusieurs parties du design avant d'avoir complété le design final ce qui permet de réduire considérablement le temps de développement lié aux erreurs non détectées lors du design. Au niveau de la compréhension et de la modification du code, on considère qu'il est beaucoup plus facile de modifier un schéma graphique que de corriger une ligne de code source du programme. En conclusion, l'auteur indique que ces techniques réduisent le temps de boucle entre le plan de design et la confirmation de la faisabilité ce qui constitue le plus grand avantage de la technique. Malgré le fait que cet article réfère à à la génération de représentations UML, on peut très bien associer les mêmes avantages à un outil comme Simulink.

1.6 Conclusion

Ce chapitre nous a permis de constater que les solutions pour la génération de code à partir de modèle Simulink sont nombreuses et offrent différentes options selon le marché visé.

Au niveau des recherches sur la technologie de génération de code, on constate que celles-ci sont récentes et souvent liées à des entreprises du milieu de l'automobile. Parmi

les avantages de la technologie, on note la réduction du temps de développement, la production d'un code concis et une conversion précise du modèle de rérérence.

Le prochain chapitre présente l'évaluation des fonctionnalités du processeur MPC555. Comme suite à cette analyse, le plan de développement sera établi.

CHAPITRE 2

Le MPC555

2.1 Introduction

Ce chapitre présente les différentes caractéristiques du MPC555. On y retrouve une description de l'unité TPU et du PIT qui constituent deux éléments importants dans le développement de l'interface d'entrée-sortie prévue.

2.2 Justification du choix de processeur

Afin de choisir un processseur qui permet d'accomplir le projet tout en prenant en compte les incertitudes comme la taille du programme final, les critères suivants ont été établis :

- processeur de 32 bits avec unité de calcul à point flottant;
- disponibilité de fonctions d'entrées-sorties numériques complexes telles la génération de signaux à modulation de largeur d'impulsion (PWM) et la lecture de signaux d'encodeurs afin de pouvoir s'interfacer à un robot;
- port de communication permettant une interface avec le PC;
- taille de mémoire flash et RAM elevée afin de permettre le fonctionnement d'un programme en C;
- disponibilité d'une carte de développement.

Face à ces critères, les principaux candidats sont le MPC555 ou le 68332 de Motorola, le SH7058 de Hitachi et le TC1775 de Infineon. En faisant des recherches sur internet, on remarque que les processeurs de Hitachi et Infineon semblent beaucoup moins populaires que leur équivalent Motorola. Dans le cas de Hitachi, on note principalement des références à des sites asiatiques ce qui permet de déduire le principal marché de ces processeurs. Pour les processeurs Motorola, on retrouve beaucoup de sites informatifs sur les outils et plusieurs choix de cartes de développement. Face à la disponibilité de

ressources et au fait que le MPC555 est la dernière génération de microcontrôleur 32 bits de Motorola, celui-ci a été préféré aux autres choix.

2.3 Historique des processeurs Motorola

Motorola fabrique des processeurs pour diverses applications depuis le tout début de cette industrie. En 1974, Motorola met en marché un premier processeur de 8 bits, le MC6800. De ce processeur initial naît en 1979 une version 16 bits, le MC68000. Cette version et ses successeurs 32 bits (68020,68030 et 68040) ont connu un succès énorme. Ils seront utilisés pour de multiples applications y compris dans plusieurs ordinateurs de types MAC et AMIGA. En 1991, IBM, Motorola et Apple font une alliance afin de créer le premier processeur RISC PowerPC alors que les processeurs de Intel et Motorola utilisaient tous une architecture CISC. Les différences entre une architecture RISC (Reduced Intruction Set Computer) et CISC (Complex Intruction Set Computer) sont décrites dans la section suivante de ce chapitre. Depuis le PowerPC G1 en 1991, Motorola améliore constamment les processeurs PowerPC qui en sont à leur quatrième génération avec le PowerPC G4 [White, 2001].

Un microcontrôleur est légèrement différent d'un processeur simple puisqu'il contient à la fois un processeur, un ensemble de fonctions d'entrées-sorties ainsi qu'une combinaison de RAM, ROM, EPROM, EEPROM ou FLASH. Dans le domaine des microcontrôleurs, Motorola a pris d'assaut le marché de l'automobile en 1984 avec le 68HC11 qui est encore utilisé aujourd'hui pour le contrôle de moteur et de transmission. De plus, le 68HC11 est très populaire dans le domaine du contrôle puisque qu'il intègre beaucoup de fonctionnalités à peu de frais [Motorola, 2002].

Le MPC555, introduit en 1998, est le premier d'une nouvelle génération de microcontrôleur basé sur un processeur de type PowerPC MPC550. Il vise encore une fois le marché lucratif de l'automobile. On y a d'ailleurs intégré deux port de communication Control Area Network (CAN) qui est le standard le plus populaire pour le réseautage de processeurs, de capteurs et d'actuateurs dans les voitures. Avec cette

nouvelle gamme de processeurs, Motorola estime que sa part de marché pour les ECU devrait passer de 10 à 25% d'ici 2005 [Screaming Media, 2001].

2.4 RISC et CISC

Un processeur RISC ne contient que des instructions simples qui sont executées en un cycle d'horloge. Par opposition, un processeur de type CISC contient un nombre élevé d'instructions dont le temps d'exécution varie de un à plusieurs cycles d'horloge. Les architectures CISC ont le désavantage de posséder plus de transistors que les RISC étant donné la plus grande quantité d'instructions qui doivent être intégrées dans le processeur. Leur consommation énergétique est donc généralement plus elevée. Cependant, un programme compilé pour une architecture CISC requiert moins de mémoire que pour un processeur RISC puisque moins d'instructions sont nécessaires pour faire un même travail. L'exemple qui suit démontre clairement ces différences.

Supposons le cas d'une multiplication de deux nombres, l'un situé à l'addrese 0x20 et l'autre à 0x30. On veut calculer le produit et écrire le résultat à l'addresse 0x20. Sur un processeur CISC, on utilise la commande assembleur MULT qui fait tout le travail.

```
MULT 0x20, 0x30
```

Au niveau du compilateur, la conversion est simple. Elle se réduit à la création d'une seule instruction. Pour une architecture RISC, il n'y a pas d'instruction permettant à la fois de charger les données, multiplier, et écrire le résultat dans la mémoire. On obtient le code suivant :

```
LOAD A, 0x20
LOAD B, 0x30
PROD A, B
STORE 0x20, A
```

Il est évidemment un peu plus complexe pour un compilateur de générer cette séquence. Il faut cependant considérer l'impact de ces deux architectures sur le pipeline

d'instruction qui permet à plusieurs instructions de s'exécuter en même temps en divisant celles-ci en sous-tâches. Avec des instructions de taille égale (RISC), il sera beaucoup plus facile d'optimiser la compilation en fonction du pipeline d'instruction.

Depuis leur introduction, les processeurs RISC ont eu un peu de difficulté à percer le marché. Ils étaient alors plus dispendieux qu'un équivalent CISC dû à un plus faible niveau de production. De plus, le prix de la mémoire était élevé et les compilateurs n'étaient pas très évolués. Aujourd'hui, de bons compilateurs RISC sont disponibles et le prix de la mémoire est passé de USD$ 5,000 pour 1MB de DRAM en 1977 à USD$ 6 en 1994 (en tenant compte de l'inflation). [Chen et al, 2002]

2.5 Un premier aperçu

Le MPC555 constitue une solution évoluée des microcontrôleurs actuels avec son juste équilibre entre une bonne puissance de calcul, une consommation raisonnable et une liste impressionnante de fonctions d'entrées-sorties. Au coeur de celui-ci, on retrouve un processeur PowerPC 32 bits de la famille MPC550 ainsi qu'une unité arithmétique à point flottant. Au niveau des fonctions d'entrées-sorties, le MPC555 possède deux TPUs (Time Processing Unit) pour les fonctions de compteurs, deux ports CAN, deux ports SCI (RS-232), deux convertisseurs analogique à numérique et beaucoup d'autres fonctions. La liste qui suit résume les différentes fonctionnalités du MPC555 :

- processeur 32 bits cadencé à 40MHz (Compatible PowerPC);
- opération à 3.3 V et/ou 5 V;
- unité arithmétique à point flottant;
- performance mesurée de 52.7-KB sur l'outil Dhrystones (v2.1);
- 26 KB de RAM statique;
- 448 KB de mémoire Flash EEPROM;
- unité de protection de mémoire;

- communication Série: Queued Serial Multi-Channel Module (QSMCM), deux contrôleurs CAN 2.0B (TouCAN);
- deux TPU3 pour un total de 50 canaux et un système d'E/S modulaires (MIOS1);
- deux convertisseurs analogique à numérique de 16 canaux (QADC64);
- technologie Submicron (CDR1);
- mode d'opération basse-puissance;
- modèle d'interruption précis;
- gestion d'exécution via BDM (points d'arrêt, exécution du programme et émulation).

2.6 PIT

Lors du développement du projet, il sera nécessaire de synchroniser notre programme afin de respecter le pas de calcul du modèle Simulink. Pour ce faire, le PIT (Periodic Interrupt Timer) sera utilisé. Ce module permet la génération d'interruptions à des périodes allant de 200 ns à 0.8 s lorsque qu'utilisé avec une horloge externe de 20 Mhz.

2.7 Time Processing Unit

Le TPU3 (TPU version 3) est une unité permettant de réaliser une multitude de fonctions reliées à l'usage de compteurs. Ce module est semi-autonome, il contient son propre microprocesseur qui fonctionne en parallèle avec le processeur principal. Il utilise des micro-instructions qui sont principalement des commandes pour lire les évènements et écrire des valeurs dans les compteurs. Le fait d'utiliser un processeur dédié plutôt que le processeur principal élimine les délais entre un évènement et la lecture du registre contenant l'information sur celui-ci.

Le lien entre le TPU et le processeur principal se fait par l'entremise du port IBM3 qui est relié à une mémoire partagée (DPRAM) de 6KB utilisée à la fois par les deux TPU et le processeur principal.

2.7.1 Fonction par défauts du TPU

Une série de micro-programmes pour le TPU est disponible dans une section de ROM du MPC555. Ces micro-programmes permettent de subvenir à la plupart des besoins sans avoir à écrire de micro-code. Il existe un total de 32 fonctions prédéfinies divisées en deux banques de 16. Les tableaux I et II décrivent les fonctions disponibles ainsi que leur positions en mémoire.

Tableau I

Liste des fonctions prédéfinies du TPU3 (Bank 0)

Numéro de fonction	Nom de la fonction	Description sommaire
0xF	PTA	Accumulateur de temps programmable
0xE	QOM	Comparaison de sortie avec queue
0xD	TSM	Table de moteur pas à pas
0xC	FQM	Mesure de fréquence
0xB	UART	Transmetteur/Récepteur asynchrone universel
0xA	NITC	Nouveau: Capture d'entrée/compteur de transition d'entrée
0x09	COMM	Commutation de moteur multiphase
0x08	HALLD	Décodage d'effet Hall
0x07	MCPWM	Signal à largeur d'impulsion variable multi-canaux
0x06	FQD	Décodage de quadrature rapide
0x05	PPWA	Accumulateur de période/impulsion
0x04	OC	Comparaison de sortie
0x03	PWM	Signal à largeur d'impulsion variable
0x02	DIO	Sortie discrète entrée-sortie
0x01	SPWM	Signal à largeur d'impulsion variable synchronisé
0x00	SIOP	Port d'entrée-sortie série

Tableau II

Liste des fonctions prédéfinies du TPU3 (Bank 1)

Numéro de fonction	Nom de la fonction	Description sommaire
0xF	PTA	Accumulateur de temps programmable
0xE	QOM	Comparaison de sortie avec queue
0xD	TSM	Table de moteur pas à pas
0xC	FQM	Mesure de fréquence
0xB	UART	Transmetteur/Récepteur Asynchrone universel
0xA	NITC	Nouveau: Capture d'entrée/compteur de transition d'entrée
0x09	COMM	Commutation de moteur multiphase
0x08	HALLD	Décodage d'effet Hall
0x07	Réservé	
0x06	FQD	Décodage de quadrature rapide
0x05	ID	Identification
0x04	OC	Comparaison de Sortie
0x03	PWM	Signal à largeur d'impulsion variable
0x02	DIO	Sortie discrete (entrée-sortie)
0x01	RWTPIN	Lecture/écriture compteur et connexion
0x00	SIOP	Port d'entrée-sortie série

De la liste de fonctions ci-haut, on utilisera la fonction de FQD (Décodage de quadrature rapide) pour la lecture d'encodeur et la fonction PWM pour la génération de signaux à largeur d'impulsion variable.

2.8 Ports de communication

Le MPC555 inclut différents ports de communication dont les principaux sont les deux ports CANbus 2.0B (touCAN) qui est un bus série différentiel ainsi que les ports asynchrones du SCI (Serial Communication Interface) qui seront utilisés pour ce projet afin de se brancher à un terminal RS-232. Il contient aussi un QSPI (Queued Serial

Peripheral Interface) qui permet une communication série synchrone allant jusqu'à 4 Mbits/s.

2.9 Conclusion

Bien que le processeur n'ait pas été évalué dans les détails, ce chapitre nous a permis de constater la grande flexibilité du MPC555. En fonction de l'information disponible, nous avons établi que la fonction de synchronisation utilisera le PIT. Le décodage de signaux en quadrature ainsi que la génération de signaux à largeur d'impulsion variable seront effectués avec les fonctions prédéfinies du TPU. Dans le prochain chapitre, nous établirons le plan de développement.

CHAPITRE 3

PLAN DE DÉVELOPPEMENT

3.1 Introduction

Le chapitre qui suit présente les différentes étapes menant à l'accomplissement de ce projet. On y choisit d'abord le générateur de code , la carte de développement et les outils qui seront utilisés. Dans une seconde étape, on fait les principaux choix de design en décrivant les options disponibles pour chaque composante du projet. À travers toutes ces étapes, il faut garder la vision du projet qui est au départ un système de génération de code pour MPC555 mais qui devrait pouvoir facilement s'adapter à un nouveau processeur. On doit favoriser les architectures flexibles et ouvertes qui s'adaptent plus facilement aux nouvelles technologies.

3.2 Sélection d'un générateur de code

Selon les données recueillies lors de la revue de littérature, on a établi que les principaux choix de générateurs étaient ceux produit par Applied Dynamics International, MathWorks ou dSpace. Comme les générateurs de dSpace et ADI ont été développés en fonction de leurs logiciels propriétaires de développement, on peut douter de la facilité à s'interfacer avec ceux-ci. Les deux autres options sont Real-Time Workshop (RTW) ou Real-Time Workshop Embedded Coder (RTWEC). RTW est sûrement plus simple et moins coûteux alors que RTWEC génère un code source plus compact. Comme RTWEC en est tout juste à sa troisième version et que la taille du code n'est pas le premier objectif du projet, on choisis d'utiliser RTW. Le grand nombre de logiciels basés sur cette solution démontrent bien les possibilités de ce générateur.

3.3 Carte de prototypage

Le nombre de solutions pour le développement pour un MPC555 est encore relativement réduit étant donné la courte existence du processeur. Une recherche sur internet a cependant permis d'identifier les cartes suivantes.

3.3.1 phyCORE MPC555 de Phytec

La carte phyCORE MPC555 de la compagnie Phytec (www.phytec.com) est un module de la taille d'une carte de crédit incluant à la fois un MPC555, de 256KB à 4MB de mémoire SRAM ainsi que de 256KB à 4MB de FLASH. Le module phyCORE se branche optionnellement dans une carte de développement fournissant les interfaces physiques pour les ports CAN et RS-232 (DB9), le port JTAG avec un connecteur de port parallèle de type DB25 ainsi que des boutons de Reset, IRQ, WakeUP. Un avantage du module phyCORE est qu'il peut, une fois le projet terminé, être séparé de la carte de développement afin de s'intégrer facilement à un design matériel personnalisé. La compagnie Phytec produit d'ailleurs des modules phyCORE pour plusieurs autres processeurs de Motorola, Infineon, Philips et AMD. L'ensemble de développement Phytec est fourni avec une version d'essai de l'environnement Code Warrior pour PowerPC [Phytec, 2002]. .

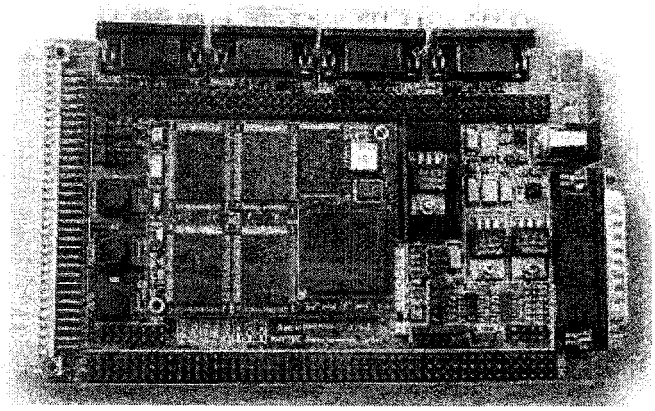

Figure 5 Carte de développement avec phyCORE MPC555 de Phytec

3.3.2 EVB555 de Etas

L'EVB555 est la carte de développement officielle du MPC555 selon le site Internet de Motorola. Cette carte a été développée en partenariat direct entre Motorola et Etas (www.etasinc.com) qui est un manufacturier de simulateurs d'ingénierie pour le milieu de l'automobile. En plus du processeur, la carte EVB555 contient 1MB de RAM et

512KB de Flash. Bien que la carte ait été conçue pour répondre à tous les besoins, seul un connecteur RS-232 de type DB9 est présent sur la carte. Les autres connecteurs sont disponibles à travers différents connecteurs de cables plats ce qui rend la tâche de branchement un peu plus difficile.

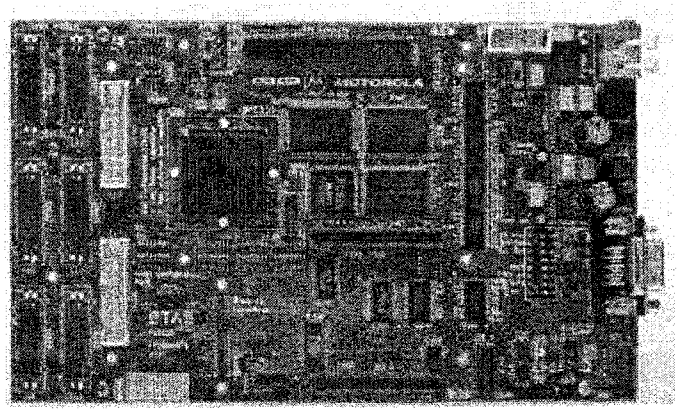

Figure 6 Carte EVB555 de Etas

3.3.3 CME0555 de Axiom

Axiom (www.axman.com) offre une solution similaire aux produits précédents. La CME0555 est une carte de développement avec un MPC555 contenant deux ports DB9 pour les SCI (RS-232). Tout comme la carte EVB555, les autres ports sont disponibles à partir de connexions pour lesquelles un câble doit être fabriqué. La CME0555 offre un

espace pour le développement de circuit personnalisé ce qui constitue un avantage intéressant.

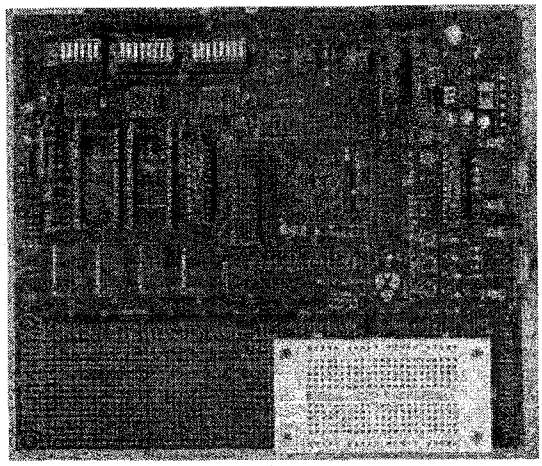

Figure 7 CME0555 de Axiom

3.3.4 Choix de la carte

La carte phyCORE MPC555 de Phytec a été sélectionnée pour ce projet. Ce choix repose principalement sur le fait que tous les ports requis (SCI sur DB9 et JTAG sur DB25) sont déjà sur la carte et que celle-ci peut inclure jusqu'à 4MB de Flash et de RAM. De plus, l'environnement de développement Code Warrior a une très bonne réputation. Le projet se basera donc sur ce dernier pour la compilation et l'exécution du code.

3.4 Définition du projet

La section précédente a permis de sélectionner les outils de développement et la carte utilisée. Cette seconde étape vise à définir le projet et à le séparer en différentes tâches. La figure 8 présente le modèle conceptuel du projet.

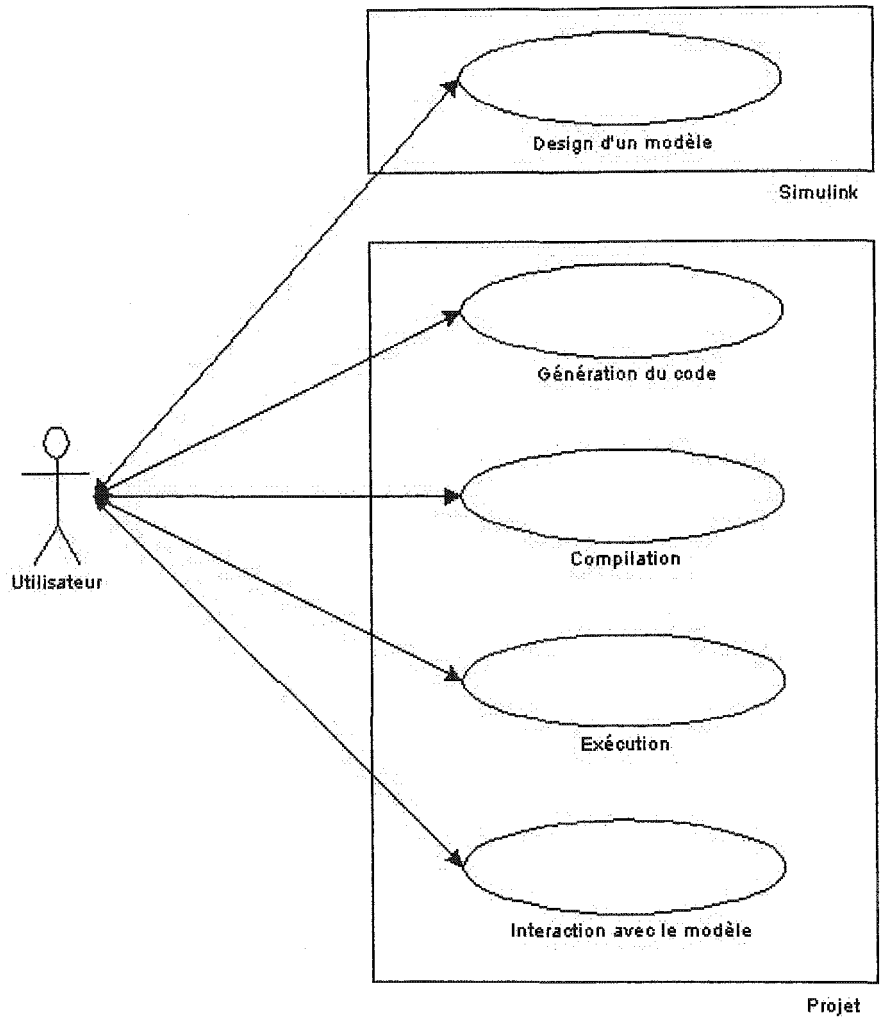

Figure 8 Modèle conceptuel du projet

L'usager crée d'abord un modèle Simulink en utilisant au besoin les blocs d'E/S qui seront disponibles pour le MPC555. Ensuite, dans le panneau de paramètres de Simulink,

le gabarit de génération du MPC555 doit être choisi. Le code sera généré en appuyant sur le bouton Build. La compilation et l'exécution sont ensuite effectuées avec l'environnement de développement Code Warrior. Finalement, l'usager peut interagir avec le modèle en utilisant une application de terminal sur un PC. À ce jour, la conversion du modèle en mode autonome sur le MPC555 n'est pas prévue. Ce mode permettrait de copier l'application de manière permanente sur le processeur et d'exécuter celle-ci dès la mise sous tension de la carte.

Les sous-sections suivantes décrivent les étapes de développement.

3.5 Étape 1 : Programme de test

Avant de générer et compiler un premier modèle Simulink, il est prévu de créer un programme de test. Ceci a pour but de se familiariser avec l'environnement de programmation mais aussi d'apprendre les différentes fonctions nécessaires à la communication sur le port série. Sans cette communication, il serait difficile d'interagir et de confirmer la bonne exécution du code à moins d'être en mode de débogage sur l'environnement de développement. L'application de test devrait effectuer le travail suivant:

- réception de caractères provenant d'un terminal;
- envoi sur ce même terminal des valeurs reçues en format hexadécimal;
- envoi d'autres messages informatifs indiquant le début et la fin du programme;
- utilisation des deux diodes électro-luminescentes (DEL) disponibles sur la carte de développement.

3.6 Étape 2 : Génération d'un modèle simple

Cette étape consiste à étudier le fonctionnement du générateur de code RTW et de créer un gabarit (template) de génération de code pour le MPC555. Une fois le gabarit complété, un modèle Simulink sera généré en code C. Ensuite, un projet Code Warrior

sera développé afin de compiler et d'exécuter le code. Ce projet Code Warrior devrait idéalement être suffisamment flexible pour être utilisé pour tous les modèles générés. Si ce requis est difficile à respecter, il faudra évaluer la possibilité de créer automatiquement un nouveau projet lors de la génération de code.

3.7 Étape 3 : Ajout de la synchronisation

Bien que l'étape précédente permette l'exécution d'un modèle Simulink sur le MPC555, le fonctionnement n'est pas en temps réel. En effet, les pas de calcul du modèle sont exécutés les uns après les autres le plus rapidement possible alors qu'ils devraient l'être à chaque période spécifiée dans le modèle Simulink (FixedStepSize). Comme le MPC555 interagira avec des équipements externes, il est requis de supporter la synchronisation temps réel. Pour ce faire, le Programmable Interrupt Timer (PIT) du MPC555 pourra être utilisé. On devra la programmer afin d'obtenir une interruption à chaque période du modèle. La séquence d'exécution du modèle sera la suivante:

1. initialisation du modèle;
2. attente de l'interruption du PIT;
3. exécution d'un pas de calcul;
4. retour à l'étape 2 jusqu'à ce que la condition de fin (temps final) soit atteinte;
5. procédure de fermeture du modèle;
6. fin.

Le fait d'attendre pour le PIT à l'étape 2 assure la synchronisation du modèle.

3.8 Étape 4 : Ajout des blocs d'E/S

Cette étape permet de compléter la solution développée en y ajoutant une bibliothèque de blocs Simulink. Ceux-ci pourront être insérés dans le modèle afin d'effectuer les tâches suivantes:

- entrée de données à partir du terminal;
- affichage de résultats sur le terminal;

- sortie numérique à largeur d'impulsion variable (fonction PWM du TPU);
- entrée numérique pour décoder les signaux en quadrature (fonction FQD du TPU).

Bien que le principe soit toujours le même, il existe à ce jour plusieurs philosophies quant à la création de bloc représentant des E/S. Les différences se retrouvent au niveau de la création d'un bloc pour plusieurs usages ou de plusieurs blocs. Aussi, la présentation des paramètres diffère. Les sous-sections qui suivent décrivent les méthodes les plus populaires desquelles on choisira celle qui sera utilisée.

3.8.1 Real-Time Windows Target

Real-Time Windows Target est un logiciel de prototypage rapide utilisant Simulink et RTW. Le logiciel comporte un bloc Simulink pour chaque fonction. Il y a un bloc d'entrée analogique, un de sortie analogique, un d'entrée numérique, et ainsi de suite. La figure 9 représente le formulaire d'entrée ("masque" dans le jargon de Simulink) du bloc d'entrée analogique. On y retrouve un paramètre pour le choix de la carte et un autre qui est un vecteur correspondant aux canaux désirés.

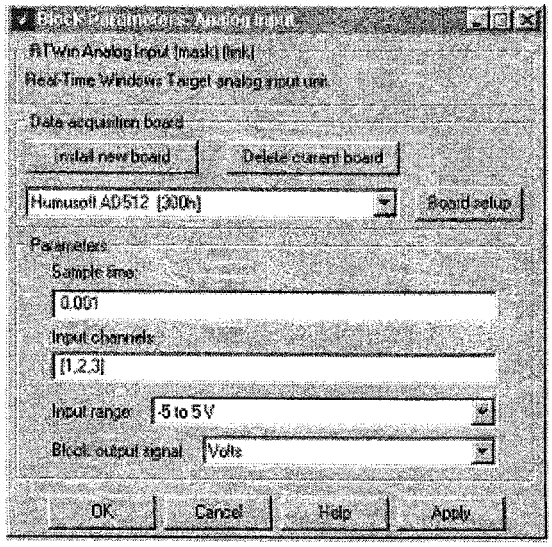

Figure 9 Masque du bloc Analog Input de Real-Time Windows Target

3.8.2 RT-LAB

Dans RT-LAB, un logiciel de prototypage rapide temps réel, on retrouve aussi un bloc par carte par fonctionnalité tel que montré par la figure 10. Le masque contient des paramètres pour définir le type de carte et son modèle. Le nombre de canaux est défini dynamiquement selon la largeur de l'entrée du bloc. Ainsi, si l'on veut utiliser trois canaux, on devra avoir un vecteur d'entrée d'une largeur de trois. Pour d'autres blocs de la bibliothèque d'E/S de RT-LAB, on utilise des vecteurs tout comme les blocs de Real-Time Windows Target.

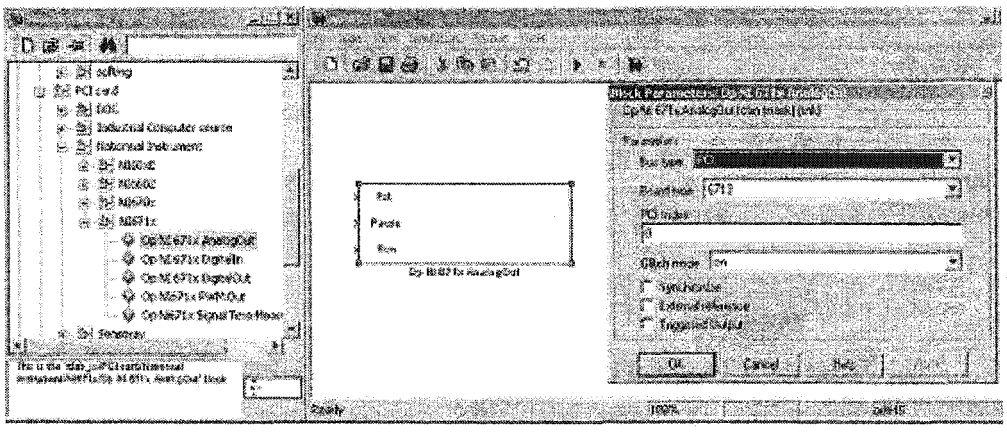

Figure 10 Bibliothèque, Bloc Analog In et masque dans RT-LAB

3.8.3 Labview

Bien qu'il ne soit pas directement lié à Simulink, Labview est un environnement de modélisation par schéma bloc très populaire pour l'acquisition de données. Dans ce logiciel, on utilise un bloc générique appelé DAQ. Ce bloc peut ensuite être configuré

pour la carte désirée à travers plusieurs menus. Dans Labview, le bloc DAQ correspond à un canal.

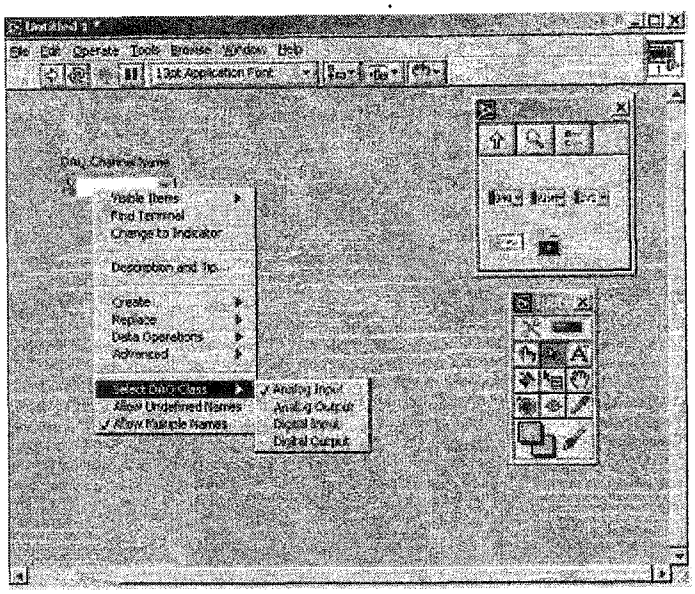

Figure 11 Bloc DAQ de Labview

3.8.4 Choix de la configuration des blocs

Les méthodes énoncées ci-haut diffèrent principalement au niveau du regroupement de canaux par bloc et de la création d'un ou plusieurs blocs par carte ou par fonction. Après évaluation des options, on choisi d'utiliser un bloc par fonction. Il y aura donc deux blocs pour la lecture de données et l'affichage sur le terminal et deux autres blocs pour les fonctions du TPU (PWM et FQD). Un mécanisme de protection devra détecter une erreur si plusieurs blocs TPU tentent d'accéder au même canal. Il sera possible d'utiliser plusieurs blocs du même type pour chaque fonctionnalité. Ce choix est justifié par la simplicité du design et l'avantage de pouvoir insérer un même bloc Simulink à plusieurs niveaux du diagramme Simulink.

3.9 Étape 5 : Modèle de démonstration

Après avoir complété les étapes précédentes, il ne restera qu'à construire une application confirmant le bon fonctionnement de tous les modules. Le modèle proposé permettra de contrôler un moteur en position. La période de la boucle de contrôle sera établie en fonction des performances du code généré. On peut cependant estimer qu'il ne devrait pas y avoir de problème pour fonctionner à 1 ms. La figure 12 présente le schéma conceptuel du modèle désiré. On constate que tous les blocs et toutes les fonctions seront utilisés. Le terminal est utilisé pour envoyer la consigne et recevoir la position actuelle du moteur. Le bloc du décodage en quadrature sera utilisé pour lire la position du moteur (fonction FQD) et le bloc PWM pour fournir la commande. Il est important de noter que l'interaction avec le terminal devra se faire à une vitesse beaucoup plus lente que la boucle de contrôle. Ainsi, le bloc d'entrée sur le terminal retournera constamment la dernière commande reçue jusqu'à la réception d'une nouvelle commande. Au niveau de l'affichage, celui-ci se fera environ une fois par seconde.

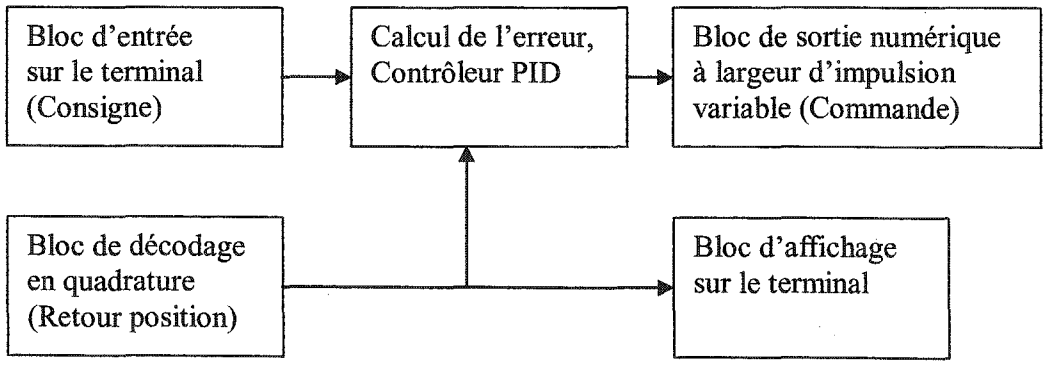

Figure 12 Modèle conceptuel du contrôleur en position

3.10 Conclusion

Bien que le projet soit relativement simple, le plan nous a permis de constater que les étapes menant à compléter celui-ci sont nombreuses. En débutant par la création d'une application simple, on réduit les risques d'erreurs lors des étapes suivantes du développement.

Les chapitres suivants présentent le développement effectué pour chacune des étapes.

CHAPITRE 4

PROGRAMME DE TEST

4.1 Introduction

Ce chapitre décrit les différentes étapes menant à la réalisation d'un programme de tests fonctionnant sur le MPC555. L'environnement de développement CodeWarrior (CW) pour Power PC ainsi que les étapes de création et d'exécution d'un projet y sont présentés. On établit aussi les fonctions C qui doivent être utilisées pour la communication série avec le PC ainsi que l'utilisation des DEL d'état de la carte phyCORE MPC555.

4.2 Code Warrior pour PowerPC

CodeWarrior est un outil de développement pour la programmation en langage C, C++ et Java fonctionnant sous Windows, Linux , Mac OsX ou Solaris. Plusieurs versions existent selon le langage, le système d'exploitation et le type de processeur. À ce niveau, il est important de noter que CodeWarrior supporte un grand nombre de processeurs embarqués incluant la plupart des 68K (série 68XXX de Motorola) ainsi que les Arm, ColdFire, MIPS, Power PC et StarCore DSP. Dans le cadre du projet, CodeWarior for PowerPC Embedded Systems version 5.01 pour Windows 98/ME/2000/XP a été utilisé.

4.2.1 Présentation de l'interface

L'interface CodeWarrior de la figure 13 est semblable à d'autres outils tels Microsoft Visual C++. La gestion des fichiers s'effectue dans une fenêtre de type explorateur Windows dans laquelle on ajoute les différents fichiers sources (.c,.h) et les bibliothèques (.a). De cette fenêtre, on peut double-cliquer sur un fichier afin d'ouvrir celui-ci dans une fenêtre d'édition. Tout comme la plupart des logiciels de

développement, l'éditeur ajuste la couleur du texte selon des regroupements tels les commentaires, les noms de fonctions, les chaînes de caractères et autres.

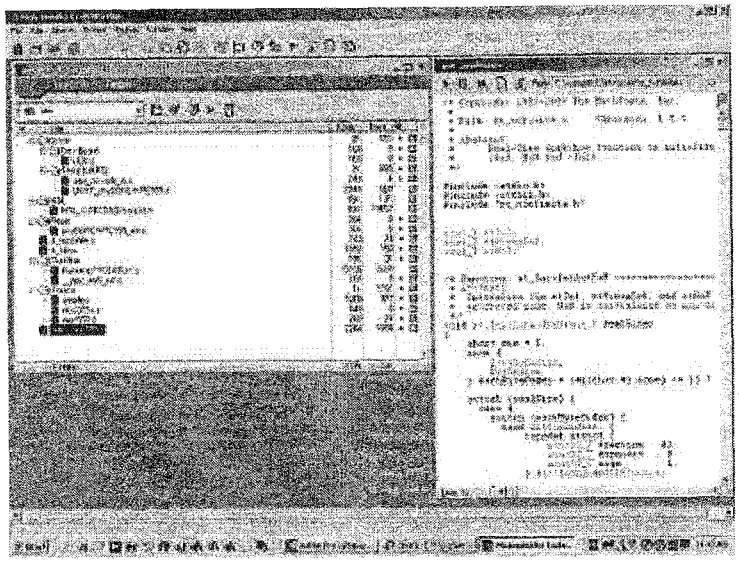

Figure 13 Interface Code Warrior

4.2.2 Branchement de la carte de développement au PC

4.2.2.1 Port BDM

CodeWarrior supporte pleinement la connexion BDM disponible sur la carte de développement PhyCORE MPC555. Cette dernière permet de télécharger, d'exécuter et de déboguer les programmes directement dans l'interface graphique. La connexion du port BDM au PC se fait à l'aide d'un câble DB-25 standard qui relie le connecteur P1 de la carte et le port LPT1 (parallèle) du PC.

4.2.2.2 Port Série

Pour cette connexion, on relie le port P2 (Port 1 du SCI) de la carte de développement au port COM1 du PC à l'aide d'un câble DB-9 NULL MODEM.

4.3 Création d'un premier projet

Étant donné les nombreux processeurs et les cartes de développement supportées par CW, la création d'un projet pour la carte Phytec n'est pas une entreprise triviale. Dans l'exemple qui suit, on décrit les différentes étapes permettant de créer un projet pour la carte phyCORE MPC555.

4.3.1 Configuration du projet

L'usager doit d'abord créer un nouveau projet en sélectionnant `File/New/Empty Project`. Ensuite, on doit entrer les informations concernant la carte phyCORE MPC555. La figure 14 et le tableau III indiquent les différents paramètres à modifier dans le panneau de configuration accessible en sélectionnant `Edit/NomduProjet Settings` du menu de CW.

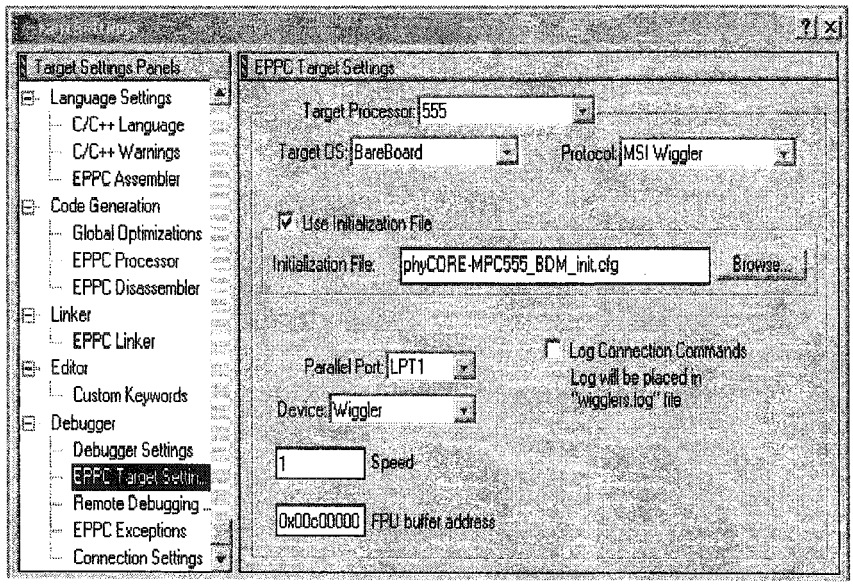

Figure 14 Panneau de configuration de projet CodeWarrior

Tableau III

Configuration de CodeWarrior pour la carte phyCORE MPC555

Rubrique	Valeur
Target / Target Settings / Linker	Embedded PPC Linker
Target / Access Paths / system Paths	{Compiler} PowerPC_EABI_Support\Runtime
Target / Access Paths / system Paths	{Compiler} PowerPC_EABI_Support\MSL\MSL_C
Target / FileName	Nom de l'éxécutable
Language Settings / C/C++ Language/Enable C++ exceptions	Non sélectionné
Language Settings / C/C++ Language / Enable RTII	Non sélectionné (Run-Time Type Information, non requis)
Language Settings / C/C++ Language / Prefix File	ansi_prefix.PPCEABI.bare.h
Language Settings / C/C++ Warning	Tous sélectionnés sauf "Treat All Warning as Errors"
Code Generation / EPPC Processor / Struct Alignment	PowerPC
Code Generation / EPPC Processor / Processor	555
Code Generation / EPPC Processor / Floating Point	Hardware
Linker / EPPC Linker / Code Address	Sélectionné, valeur = 0x00c02000
Linker / EPPC Linker / Data Address	Non Sélectionné
Debugger / EPPC Target Settings / Target proceesor	555
Debugger / EPPC Target Settings / Target OS	Bare board
Debugger / EPPC Target Settings / Protocol	MSI Wiggler
Debugger / EPPC Target Settings / Use Initialization File	Sélectionné, fichier = phyCORE-MPC555_BDM_init.cfg
Debugger / EPPC Target Settings / Device	Wriggler
Debugger / EPPC Target Settings / FPU buffer address	0x00c00000

Ces données ont été extraites de certains exemples fournis avec la carte Phytec mais aussi par essais-erreurs.

4.3.2 Ajout de fichiers

Cette étape consiste à ajouter les fichiers sources qui doivent être compilés avec le projet en utilisant la commande `Project/Add Files` du menu. Avant toutes choses, les fichiers `Runtime.PPCEABI.H.a` et `__ppc_eabi_init.c` relatifs à la carte de developpement doivent être ajoutés. Ceux-ci contiennent toutes les fonctions d'initialisation. On ajoute ensuite les autres fichiers .c du projet un à la suite de l'autre. L'ordre de compilation des fichiers peut être modifié en choisissant l'onglet `Link Order` de l'explorateur de fichier. Afin de maintenir une classification de tous les fichiers sources, il est possible de créer des groupements de fichiers à l'aide de fonction `Project/ Create Group`

4.3.3 Compilation, exécution

Suite à l'ajout de tous les fichiers, on compile avec `Project/Make` et on démarre le tout en appuyant sur le bouton d'exécution ou en sélectionnant `Project/Run`. Lors de l'exécution, le code est d'abord transféré sur la carte de développement à l'aide du port BDM puis le programme est démarré. Il est possible de différencier ces deux étapes à l'aide de la DEL D6 de la carte phyCORE. Celle-ci est éteinte lors de l'exécution et active lors du transfert ou d'un arrêt dû à une commande de fin ou à un point d'arrêt.

4.3.4 Débogage

Comme un programme fonctionne rarement du premier coup, il est très utile d'utiliser les fonctions de débogage de CW. Pour ce faire, on active tout d'abord le débogueur en sélectionnant l'item `Project/Enable Debugger` du menu. Ensuite, il ne reste qu'à insérer des points d'arrêts (avec le bouton droit de la souris) et de démarrer le programme. Le fonctionnement du débogueur est le même que pour la plupart des autres logiciels, on peut vérifier la valeur de certaines variables et entrer ou non dans les fonctions tracées.

4.4 Programme de test

Tel que défini précédemment, la première étape du projet consiste à développer un programme de test permettant à la fois de tester les communications séries avec le PC ainsi que l'utilisation des DEL d'état de la carte phyCORE MPC555. Un projet a été créé tel que décrit dans la section précédente. Le projet contenait initialement un fichier vide avec une fonction principale `void main()`. On y a ensuite ajouté les fonctionnalités décrites dans les sous-sections suivantes.

4.4.1 DEL d'état

Afin d'avoir l'assurance de la bonne exécution d'un programme, il sera intéressant d'avoir accès au DEL D4 et D5 de la carte phyCORE et de les utiliser pour visuellement connaître l'état du système sans avoir à accéder le débogueur.

Les deux DEL sont reliées aux bits 0 et 1 du port de 16 bits du MIOS1 (Modular Input / Output subsystem). Afin de contrôler ceux-ci, on doit d'abord configurer les bits en sortie dans le registre MPIOSM Data Direction Register (MPIOSMDDR) pour ensuite fixer la valeur des bits à l'aide du registre MPIOSM Data Register (MPIOSMDR). Le tableau IV présente le pseudo code pour les différentes fonctions.

Tableau IV

Pseudo code du contrôle des DEL

Séq.	Fonction	Pseudo code
1	Déclarer un pointeur sur le registre de direction MPIOSMDDR	MPIOSMDDR = Pointeur sur registre de direction MPIOSMDDR
2	Déclarer un pointeur sur le registre du port MPIOSMDR	MPIOSMDR = Pointeur sur le registre du port MPIOSMDR
3	Initialiser les bits 0 et 1 du MPIOSSM en sorties	MPIOSMDDR |= 0x0003
4	Mettre bit 0 à 0	MPIOSMDR &= 0xFFFE
5	Mettre bit 0 à 1	MPIOSMDR |= 0x0001

Séq.	Fonction	Pseudo code
6	Mettre bit 1 à 0	MPIOSMDR &= 0xFFFD
7	Mettre bit 1 à 1	MPIOSMDR |= 0x0002

Il est à noter que la logique entre les bits et l'activité des DEL est inversée. On doit donc mettre le bit à 0 pour que la DEL soit allumée ou à 1 pour qu'elle soit éteinte. Ceci est dû au type de circuit d'entrée-sortie utilisé qui permet de drainer plus de courant (bit à 0) qu'il ne peut en fournir (bit à 1). Il est donc plus avantageux de relier une DEL à la sortie numérique et à une source de 5 volts plutôt que de relier cette diode à la sortie numérique et à la masse de cette même source. Dans le premier cas, le courant de la diode est fourni par la source alors que dans le second c'est le circuit intégré qui doit fournir le courant.

4.4.2 Communication série

Le MPC555 possède une interface de communication série appelée Serial Communication Interface (SCI) qui contient deux ports de communications série RS-232. L'application de test doit utiliser l'un d'eux (port1) qui est relié au port COM1 du PC.

Il existe deux principales utilisations pour le port série. La première consiste à envoyer et/ou recevoir des trames à un autre appareil ou processeur selon un protocole distinct. C'est le cas de l'utilisation de la plupart des systèmes de positionnement GPS pour lesquels on doit envoyer une requête avec le taux de rafraîchissement ainsi que certains autres paramètres pour ensuite recevoir continuellement notre position du GPS. Le format de ces trames est défini par le manufacturier du GPS. La communication est généralement effectuée par une seule section de programme. Les étapes de communication sont les suivantes :

1. initialiser le port;
2. préparer une requête en remplissant une structure de données respectant le format du manufacturier;
3. envoyer la requête;

4. attendre les données jusqu'à la réception d'un caractère ou d'une condition de fin de trame;
5. vérifier l'intégrité de la trame;
6. extraire les données utiles (position actuelle, vitesse ...);
7. transmettre les données à la section du programme concernée;
8. boucler à l'étape de réception de données.

Dans un deuxième cas, on peut utiliser le port comme un terminal d'usager. Dans cette configuration, on désire simplement envoyer des chaînes de caractères provenant de un ou de plusieurs modules du logiciel. La plupart du temps, il n'y a pas de format de paquets prédéfini, on envoie tous les caractères un à la suite de l'autre et les caractères spéciaux tels le retour de chariot et l'indicateur de nouvelle ligne seront utilisés pour le formatage de l'affichage. Il est aussi possible de recevoir des commandes qui sont entrées par l'usager. Ce projet requiert l'utilisation d'un mode terminal afin de pouvoir afficher des données à partir de plusieurs sections du logiciel. Par exemple, il y a la section principale qui se charge de l'exécution du modèle en temps réel. Cette section doit envoyer de l'information sur l'exécution du modèle telles les indications de départ et de fin ainsi que les statistiques d'exécution. Sur le même port, on veut aussi pouvoir afficher l'évolution de certaines valeurs de notre contrôleur. Cette dernière tâche sera effectuée par un bloc Simulink d'affichage.

La bibliothèque MetroTRK UART développée par Metrowerks permet l'utilisation d'un port dans les deux modes. En mode terminal, on doit d'abord déclarer le port qui sera utilisé comme sortie terminal standard (`STDOUT`) et configurer la vitesse de transmission et les autres paramètres. Une fois ces deux étapes complétées, on peut utiliser les fonctions ANSI C d'interaction de terminal tels les `printf`, `sprintf`, `gets`, `getch` et autres. Ces dernières peuvent être insérées dans plusieurs modules du logiciel. Le tableau V présente le pseudo code pour l'utilisation du port série en mode terminal.

Tableau V

Pseudo code de la communication série en mode terminal

Séq.	Fonction	Pseudo code
1	Déclarer le port à utiliser en sortie terminal standard	STDOUT = port1
2	Initialiser le port	Port1.BaudRate = 9600 Bauds
3	Utiliser le port	printf(...), getch(...)

4.4.3 Résultats

Suite à la conception préliminaire, un programme de tests a été écrit. Celui-ci repose sur des exemples obtenus avec la carte Phytec MPC555.

Lors du démarrage, le programme affiche tout d'abord un message confirmant son exécution. Ensuite, celui-ci bloque jusqu'à la réception de caractères provenant du terminal et les retourne sur le port. À chaque réception de caractère, les valeurs des DEL D5 et D6 sont inversés ce qui donne une alternance entre la première DEL qui est rouge et la deuxième qui est verte. Le logiciel HyperTerminal pour Windows a été utilisé. Celui-ci était configuré avec les mêmes paramètres que le port 1 du MPC555 soit 9600 bauds, 8 bits, aucune parité et 1 bit de fin.

4.4.4 Problèmes d'initialisation du port série

Bien que le programme de test ait bien fonctionné, il serait important de noter un problème qui est survenu pendant les tests initiaux. Lors des premières exécutions du programme, le terminal n'affichait qu'un ou deux caractères incohérents sur le terminal. Ensuite, les DEL commencaient à scintiller très rapidement dès la transmission d'un premier caractère.

Suite à la vérification des paramètres du port dans HyperTerminal, le débogueur de CodeWarrior a été utilisé afin de tracer le code du programme. Après quelques répétitions du problème, il a été confirmé que celui-ci se produisait au niveau de la fonction `getchar()` qui doit normalement bloquer sur la réception d'un caractère et le

retourner. Lors du premier appel de la fonction, celle-ci bloquait correctement jusqu'à l'envoi d'un caractère du PC mais la valeur retournée ne correspondait pas au code ASCII du caractère envoyé. Pour tous les appels subséquents. la fonction `getchar()` ne bloquait plus et retournait directement la même valeur erronée. Les sous-sections qui suivent présentent le développement qui a mené à la résolution de ce problème.

4.4.4.1 Modification du Baud Rate

Mon expérience avec les communications série m'a amenée à supposer dès le départ que le Baud Rate du PC et celui du MPC555 n'étaient pas les mêmes. La vérification de toutes les fonctions bas niveau du MPC555 n'a pas permis de déceler d'erreur de la configuration du port. De plus, la variation du Baud Rate dans le logiciel HyperTerminal et dans le programme de test n'a eu aucun effet sur le problème.

4.4.4.2 Bibliothèque de communication

Puisqu'une partie des bibliothèques de communication bas niveau est compilée donc non accessible, le support à la clientèle de Phytec a été contacté. Après les quelques semaines requises pour contacter l'expert MPC555 de Phytec en Allemagne, nous avons procédé à une vérification du programme et des bibliothèques. Suite à cette vérification, nous avons confirmé que mon programme fonctionnait correctement sur le MPC555 de l'expert Phytec.

4.4.4.3 Initialisation de l'horloge

Mon contact avec le département de support de Phytec n'ayant pas été fructueux, j'ai tenté d'exécuter un programme sans communication série dont le seul but était de faire scintiller les DEL d'état. Ce programme avait auparavant toujours bien fonctionné. Une première exécution m'a permis de constater que le programme fonctionnait toujours et que la fréquence de scintillement était constante. Cependant, lors d'une seconde exécution après avoir enlever et remis l'alimentation de la carte MPC555, j'ai constaté que la fréquence du scintillement avait augmenté significativement. D'autres tests ont

finalement permis de constater que la fréquence de l'horloge de la carte MPC555 pouvait être inexacte lors de la première mise sous tension après quelques jours d'inactivité. À ce jour, Phytec n'a pu confirmé ce problème. La technique de débranchement et rebranchement de l'alimentation fonctionne tout de même à coup sur.

4.5 Conclusion

Le développement de l'application de test permet de réduire le risque des étapes de développement suivantes en établissant clairement les fonctions qui seront utilisées dans la solution finale. Un problème d'initialisation d'horloge sur la carte phyCORE MPC555 a aussi été identifié, mais celui-ci est facilement réglé avec un débranchement et rebranchement de l'alimentation.

CHAPITRE 5

GÉNÉRATION D'UN MODÈLE SIMPLE

5.1 Introduction

Le but de cette section est d'utiliser le générateur de code Real-Time Workshop afin de créer, compiler et exécuter le code d'un modèle Simulink simple. On y présentera d'abord une vue d'ensemble de RTW et de ses composantes. Dans la seconde partie, on décrit le modèle de génération de code RTW pour le MPC555 et on présente un modèle Simulink simple ainsi que les différentes étapes menant à l'exécution de ce dernier sur le MPC555.

5.2 Real-Time Workshop

Tel que vu précédemment, Real-Time Workshop permet la génération de code en fonction d'un modèle Simulink. Le processus de génération de code est configuré par un fichier de traduction des blocs pour l'environnement ciblé (Target Language Compiler), un fichier modèle de compilation (Template Makefile) et un programme qui sera utilisé pour contrôler l'exécution. Il est possible de définir ces items individuellement ou simplement de choisir parmi les modèles par défaut de RTW. Selon le système cible choisi, il sera possible de compiler manuellement sur le système cible ou sur le système hôte qui est le PC où MATLAB, Simulink et RTW sont installés.

À ce jour, RTW génère du code pour pratiquement tous les blocs Simulink à l'exception des fonctions Matlab et des S-function en m et en Fortran.

5.2.1 Formats de code

RTW supporte différents formats de code qui seront utilisés selon le type d'application. Le tableau VI présente les cinq formats de code et leurs caractéristiques:

Tableau VI

Formats de code de Real-Time Workshop

Format de code	Caractéristiques
Real-Time	- format par défaut; - mémoire statique; - supporte les états continus; - supporte les CMEX S-Functions; - excellent pour les applications de prototypage rapide;
Real-Time malloc	Tel que Real-Time plus - allocation dynamique de mémoire; - permet d'inclure plusieurs fois le même modèle ou plusieurs modèles avec leurs propres données; - excellent pour les applications de prototypage rapide;
S-function	- génère le modèle dans une CMEX S-function qui peut être insérée dans un autre modèle Simulink;
Embedded C code	- requiert l'utilisation de Real-Time Workshop Embedded Coder - génère un code avec usage de RAM et ROM minimum; - utilisé pour les applications de production;
Ada	- similaire au mode real-time. La génération produit du code ADA et non du code C comme tous les formats précédents

5.2.2 Fichier Target Language Compiler

RTW utilise un fichier ASCII appelé Target Language Compiler (TLC) afin de définir comment les blocs Simulink doivent être convertis en code. Il est possible d'utiliser le TLC par défaut de RTW qui propose un format standard pour la génération de code ANSI C.

Un exemple d'utilisation du TLC serait dans le cas où l'on veut utiliser un bloc Simulink tel le WriteFile dont l'exécution n'est pas temps réel. En remplaçant le code par défaut par une version optimisée, on peut obtenir un WriteFile temps réel sans recourir à l'écriture complète d'un nouveau bloc.

5.2.3 Fichier Template Makefile

Le fichier ASCII Template Makefile (TMF) permet à RTW de générer automatiquement un fichier de compilation (Makefile). Le fichier de compilation obtenu après la

génération est utilisé par le compilateur afin de convertir le code source en exécutable. Le fichier TMF contient la définition de tous éléments de compilation telle la liste de fichiers sources et bibliothèques ainsi que les répertoires et les options d'optimisation.

Comme beaucoup d'éléments de la compilation dépendent du contenu de modèle, il est possible d'insérer des mots clés qui seront automatiquement remplacés par RTW lors de la génération. Ainsi le mot clé |>SOLVER<| sera remplacé par l'algorithme d'intégration choisi et le mot |>MODEL_NAME<| sera remplacé par le nom du modèle Simulink.

5.2.4 Programme du séquenceur

En fonction du modèle Simulink utilisé, RTW crée plusieurs fonctions permettant d'initialiser, de compléter ou d'exécuter un pas de tous les blocs contenus dans le modèle. Ces fonctions n'ont cependant aucune utilité sans le séquenceur, un programme qui se charge d'appeler les fonctions dans le bon ordre et de synchroniser l'exécution. La séquence qui suit représente un cas typique d'exécution :

1. initialisation de tous les blocs Simulink;
2. exécution d'un pas de calcul de tous les blocs;
3. vérifier si l'exécution du modèle est complétée, si oui exécuter étape 5);
4. synchronisation selon le temps d'échantillonnage, attente puis retour en 2);
5. appel de la fonction de fin de tous les blocs Simulink;
6. fin de l'exécution.

Cette séquence simple permet de comprendre l'idée générale du programme, mais il reste beaucoup de points qui doivent être considérés selon l'application et le modèle. Par exemple, pour un modèle à plusieurs temps d'échantillonnage (Multirate), le séquenceur devra s'assurer de synchroniser toutes les tâches et idéalement de les exécuter en parallèle. Aussi, la plupart des outils de prototypage rapide possèdent plusieurs fonctionnalités telles l'acquisition, la transmission et le changement de paramètres en cours d'exécution ce qui augmente encore la complexité du programme. Face à toutes ses fonctionnalités, on comprend à quel point l'architecture du séquenceur est

importante. Le tableau VII présente une liste sommaire des fonctions Simulink-RTW qui sont appelées par le séquenceur.

Tableau VII

Liste sommaire des fonctions Simulink-RTW

Fonction	Description
`rt_InitInfAndNaN`	Initialise les valeurs de l'infini et de Not A Number
`rt_OneStep`	Exécute un pas du modèle (tous les temps d'échantillonnage)
`MdlOutputs`	Exécute un pas de tous les blocs avec un même temps d'échantillonnage
`MdlInitializeSizes`	Initialise les vecteurs d'entrée et de sortie de tous les blocs
`ssGetTFinal(S)`	Obtient le temps de fin de la simulation
`ssGetT`	Obtient le temps actuel de la simulation
`ssGetErrorStatus`	Obtient la dernière erreur de la simulation
`ssGetStopRequested`	Vérifie si un bloc a demandé la fin de la simulation
`MdlTerminate(`	Fonction de fin de tous les blocs

5.2.5 Génération du code

Pour activer la génération de code, l'usager doit d'abord ouvrir le modèle désiré dans l'environnement Simulink. Ensuite, il faut sélectionner l'item `Simulation Parameters` sous la rubrique `Simulation` du menu. Dans le panneau de la figure 15, l'usager choisit les fichiers TLC et TMF qui seront utilisés pour la génération de code. Il est aussi possible d'ajuster certaines options tel le nom de la commande de compilation et les paramètres spécifiques au modèle de génération choisi. Une fois la configuration complétée, on peut enclencher le processus de génération en appuyant sur

le bouton `Build`. La fenêtre de commande MATLAB affiche alors le progrès de la génération et de la compilation du code.

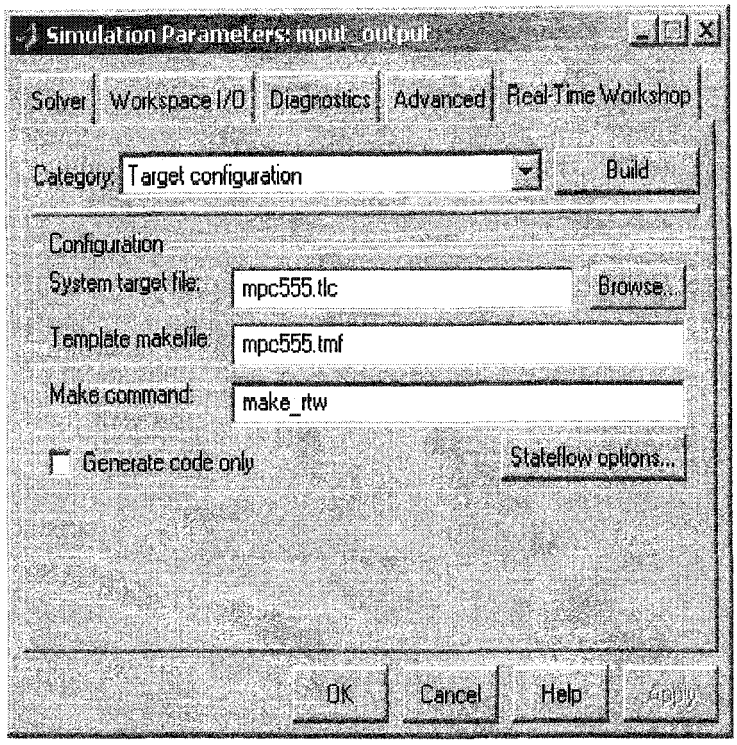

Figure 15 Panneau de configuration RTW

5.3 Modèle pour le MPC555

Cette partie vise à expliquer les différentes composantes qui ont été créées afin d'obtenir un modèle RTW pour la génération ciblant le MPC555.

Tout d'abord, selon les informations obtenues précédemment, il a été choisi de démarrer avec un des modèles par défaut de RTW soit le modèle real-time. Celui-ci alloue la mémoire de manière statique et permet l'utilisation de S-functions qui seront utilisées pour implémenter les fonctions d'entrées-sorties lors du chapitre suivant.

Les sous-sections suivantes décrivent les différents changements et ajouts qui ont été effectués ainsi que le processus de développement.

5.3.1 Modèles à pas d'intégration multiple

Les modèles à pas d'intégration multiple (multirate) ne pourront être complètement supportés à l'aide de notre modèle de génération pour le MPC555. Ceci est dû au choix de ne pas utiliser de système d'exploitation sur le MPC555. Sans mécanisme de gestion de tâches, on ne peut assurer la gestion de tous les pas d'intégration en parallèle, on fonctionne dans un mode appelé pseudo multitâche. Dans ce mode, tous les pas d'intégration sont exécutés à l'intérieur du pas de calcul le plus rapide. Ceci amène la limite de ne pouvoir répartir le calcul des tâches lentes sur tout le temps de libre de la tâche la plus rapide.

Les figures 16 et 17 illustrent le concept de multitâche et pseudo multitâche à l'aide de deux exemples. On considère ici un modèle avec une tâche à 1 ms et une autre à 2 ms. Le temps de calcul de la tâche à 1 ms est fixé à 500us alors que le temps de calcul de la tâche à 2 ms varie de 300 (cas 1) à 800 µs (cas 2). En mode multitâche, la fin de la tâche à 1 ms entraîne automatiquement l'exécution de la tâche à 2ms jusqu'à que la temps soit venu de réexécuter la tâche à 1 ms. Dans le mode multitâche, le temps disponible pour exécuter la tâche à 2ms est de 1 ms. Ce nombre est calculé en multipliant le temps non utilisé de la tâche à 1ms (500us) par l'intervalle entre les pas à 2 ms et ceux à 1 ms (2). Dans les deux cas présentés, on constate que la synchronisation en mode multitâche respecte le critère temps réel.

En mode pseudo multitâche, tout le calcul de la tâche à 2 ms doit être exécuté dans le temps non utilisé de la tâche à 1ms soit 500 µs. On se retrouve donc avec une erreur de synchronisation dans le second cas puisque le temps de calcul de la tâche est supérieur à 500 µs. Cette limitation spécifique au mode pseudo multitâche s'accentue avec l'augmentation de l'intervalle entre le pas le plus rapide et les autres pas de calcul. Ainsi, si la boucle à 2ms fonctionnait plutôt à 10ms, le temps disponible pour cette tâche en

mode multitâche serait de 5 ms (500 µs * intervalle de 10) alors que le mode pseudo multitâche serait toujours limité à 500 µs.

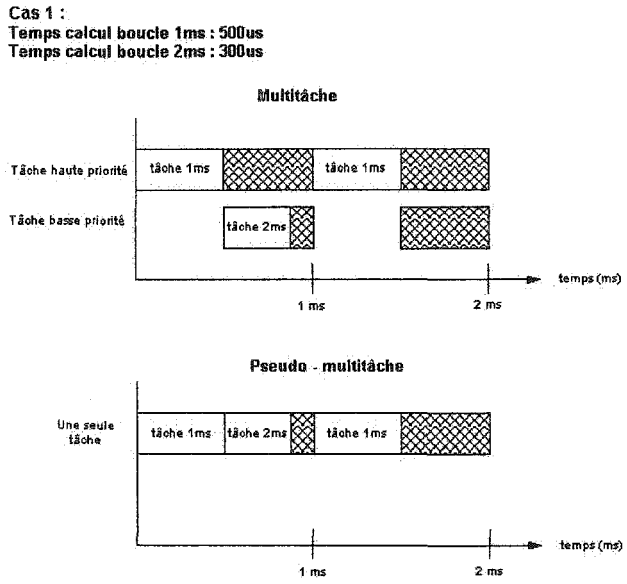

Figure 16 Exemple multitâche, cas 1

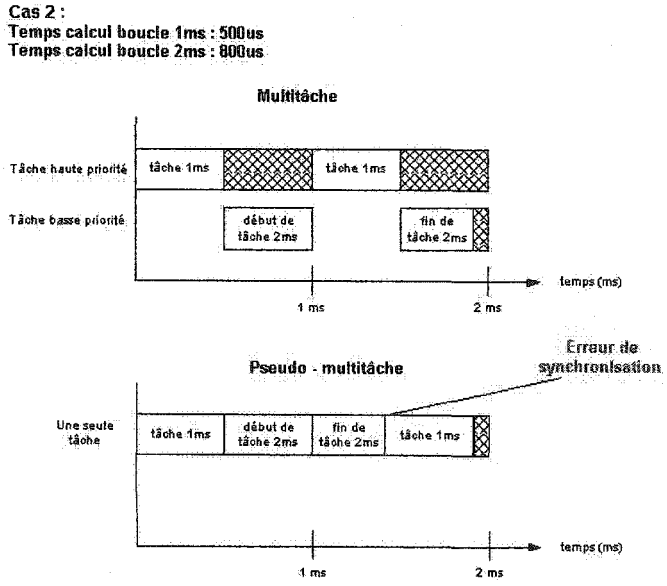

Figure 17 Exemple multitâche, cas 2

5.3.2 Fichier Target Language Compiler

Nous n'avons à première vue aucun besoin d'automatiquement remplacer certains blocs par un équivalent pour le MPC555. Il est considéré que la majorité des blocs Simulink sauf ceux nécessitant un système de gestions des fichiers devraient fonctionner sur le microcontrôleur tels quels.

Les seuls changements effectués sont donc le nom du répertoire par défaut qui contient maintenant le terme MPC555 ainsi que les options de génération telles `External Mode` et `MAT-file variable name modifier` qui ont été supprimées.

5.3.3 Fichier Template Makefile

Suite à une évaluation du fonctionnement de la commande `Build` de RTW, il a été établi que RTW génère d'abord le code C et selon la configuration du fichier Template Makefile (TMF) crée un fichier Makefile (.mk) et exécute au besoin une ligne de commande permettant la compilation de code C en exécutable. Par exemple, en supposant l'utilisation de l'environnement Microsoft Visual C++, la commande de compilation pourrait être "cl -f nom_du_makefile.mk".

De notre côté, l'environnement CodeWarrior pour PowerPC n'offre pas suffisamment de flexibilité pour utiliser le fichier Makefile et compiler un projet simplement à l'aide d'une ligne de commande. On devra donc utiliser RTW uniquement pour la génération alors que la compilation et l'exécution prendront place dans l'environnement CW.

Certaines définitions spécifiques au modèle sont écrites dans le fichier Makefile (.mk) qui est créé en fonction du fichier Template Makefile. Ces définitions doivent être utilisées pour la compilation du code source. Parmi celles-ci on retrouve les éléments SOLVER, NUMST, NCSTATES qui définissent le type d'intégrateur utilisé ainsi que le nombre d'états (continus ou non) du modèle. Une solution temporaire consistait à manuellement copier les informations du fichier Makefile généré dans un fichier d'entête (.h) du projet CW. Comme cette modification n'était pas très pratique, le fichier TMF a

été modifié afin de générer un fichier d'entête (.h) au lieu d'un Makefile (.mk). Le fichier d'entête créé peut être utilisé directement lors de compilation. Un problème rencontré lors de cette étape est que les scripts MATLAB nomment toujours le fichier selon le format `nom_du_modèle.mk`. Afin de convertir l'extension .mk en .h, une commande de compilation a été créée et le fichier TMF a été modifié afin d'appeler celle-ci lors de la génération de code. La commande de compilation dont la seule fonction est de convertir le fichier .mk en .h respecte le format des scripts DOS (fichier .bat). La commande a été copiée dans le répertoire `'C:\Matlab6p1\bin\win32'` afin d'être retrouvée par MATLAB. Tout autre répertoire défini dans la variable d'environnement PATH de Windows aurait aussi pu être utilisé.

5.3.4 Programme du séquenceur

La documentation de RTW à ce sujet n'est pas mauvaise, mais puisque le tout doit être compilé dans l'environnement CodeWarrior pour PowerPC, plusieurs étapes ont dû être effectuées de manière itérative.

Dans un premier temps, le modèle Simulink suivant a été créé.

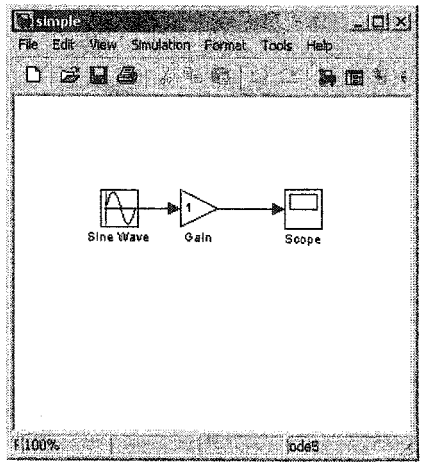

Figure 18 Modèle Simulink simple

Le code C du modèle a ensuite été généré dans RTW en choisissant les fichiers TLC et TMF pour le MPC555 et en appuyant sur le bouton `Build`. On constate alors que RTW produit les fichiers simple.c, simple.mk ainsi que plusieurs fichiers .h dans le sous-répertoire simple_mpc555_rtw.

Dans un deuxième temps, il faut compiler le tout dans CW. Pour ce faire, le projet créé lors du chapitre précédent a été utilisé comme référence. Ce projet comporte l'avantage d'avoir tous les bons paramètres concernant le processeur utilisé ainsi que l'interface de débogage JTAG sur le port parallèle. Le fichier principal du programme de test a été remplacé par les fichiers simple.c générés par RTW et le séquenceur RTW fournit en exemple (renommé mpc555_main.c).

Lors d'une première analyse du fichier séquenceur fourni en exemple dans RTW, on constate que celui-ci exécute simplement la simulation pendant le nombre de pas spécifié dans le modèle, puis quitte. Les données recueillies par la simulation sont sauvegardées dans un fichier binaire MATLAB. Puisque nous n'avons pas de système de gestion de fichiers sur le MPC555, toute la partie d'écriture de données dans un fichier a dû être supprimée. La plupart des autres fonctions ont pu être compilées sans problèmes. CW semble toutefois plus strict que d'autres compilateurs en ce qui a trait au transtypage (casting). Ainsi, le fichier d'entête `rt_matrx.c` de MATLAB provoque des avertissements de compilations. Il a du être modifié afin de définir explicitement le transtypage.

Parmi les autres modifications effectuées, on note l'ajout des répertoires de fichiers d'entêtes MATLAB (sous `Simulink/extern/include` et `Simulink/include`) ainsi que des fichiers `rt-sim.c` et `rt_nonfinite.c` (sous `rtw/src` et `rtw/libsrc`). Finalement, on a ajouté quelques impressions redirigées sur le terminal ainsi que l'utilisation des DEL rouges et vertes qui alternent à chaque seconde d'exécution du modèle.

L'organigramme de la figure 19 présente l'algorithme du programme séquenceur.

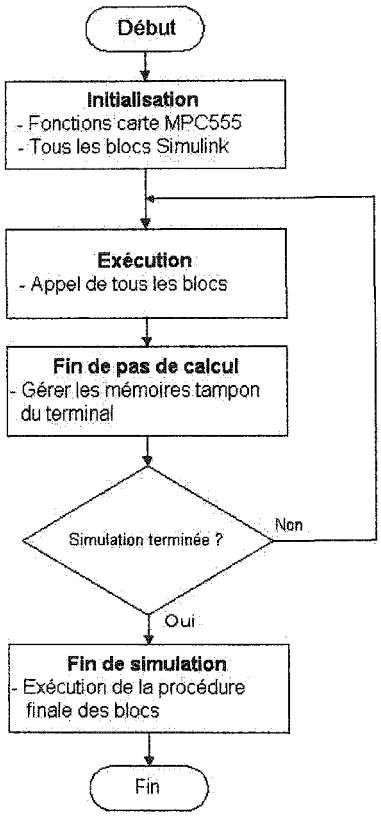

Figure 19 Organigramme du séquenceur initial

Avec ce premier modèle de génération, il a été possible d'exécuter un modèle simple sur le MPC555. Afin de simplifier la tâche pour la génération de nouveaux modèles, un projet CW auquel il ne manque que les fichiers provenant de RTW a été créé. La procédure complète pour la génération du code et l'exécution d'un modèle sur le MPC555 se retrouve à l'annexe A.

5.4 Conclusion

Cette première étape confirme qu'il est possible d'exécuter le code généré par RTW sur un MPC555 en mode non synchronisé. Des tests plus approfondis nous permettront d'évaluer le bon fonctionnement et la performance des blocs Simulink. Dans la

prochaine étape, nous allons ajouter la possibilité de synchroniser le modèle Simulink afin que le temps d'exécution sur le MPC555 reflète celui spécifié dans le modèle Simulink. De plus, cette étape nous permettra d'ajouter une fonction de mesure du temps de calcul qui sera très utile afin de s'assurer du respect des contraintes temps réel.

CHAPITRE 6

SYNCHRONISATION

6.1 Introduction

Le but de cette étape est d'ajouter la synchronisation au séquenceur d'exécution. Pour ce faire, nous utiliserons le module PIT (Programmable Interrupt Timer) du MPC555. À la fin de l'étape, nous obtiendrons le séquenceur final.

6.2 PIT

Le PIT est un compteur de 16 bits dont la sortie peut générer une interruption. La source en fréquence du PIT provient du cristal externe du MPC555 qui est divisé par 4 ou 256. Le fonctionnement du PIT est simple, on calcule d'abord le nombre de coups d'horloge pour obtenir la fréquence désirée. Une fois ce nombre programmé dans le PIT, celui-ci décrémente jusqu'à zéro et une interruption est générée. Ensuite, le PIT se réarme automatiquement et la séquence recommence.

6.3 Sélection du diviseur d'horloge

La carte MPC555 de Phytec possède un cristal de 20Mhz. Les deux fréquences d'horloge disponible pour le PIT sont donc 5Mhz (diviseur de 4) ou 78.125 Khz (diviseur de 256). Étant donné que la plupart des algorithmes de contrôle fonctionnent à des vitesses plus lentes que 1ms, la fréquence de 78.125 Khz a été sélectionnée. Considérant que le compteur du PIT permet des valeurs allant de 2 à 65535, la plage de pas de calcul sera de 25 µs à 131 ms. Il est important de noter que d'autres modules du MPC555 tel le MIOS (Modular Input/output subsystem) peuvent utiliser une fréquence de 40Mhz qui est synthétisée à partir du cristal externe.

6.4 Programmation du PIT

La programmation du PIT nécessite l'utilisation des registres suivants :

- PISCR : PIT Status and Control;

SCCR : System Clock Control Register;

PITC : PIT Count;

PITR : PIT Register.

La séquence d'initialisation du PIT est :

1. remettre à zéro le PIT Interrupt Status Flag (PISCR bit PS = 0);
2. choisir le cristal externe comme source du PIT (SCCR bits RTSEL = 0x0);
3. calculer et programmer le nombre de comptes du PIT (PITC = 20000000 * 0.001 = horloge * pas de calcul);
4. mise en route du PIT (PISCR bit PTE = 1).

Puisque l'architecture utilisée pour ce projet ne comporte qu'une seule tâche, l'utilisation de l'interruption du PIT s'avère inutile. On détectera donc la fin de période par scrutation du registre de status (PISCR).

6.5 Analyse du temps de calcul

En plus de la mise en place de la fonction de synchronisation, une fonction d'analyse de temps de calcul a aussi été ajoutée. Cette fonction utilise le compteur PIT afin de calculer le temps que prend le modèle pour exécuter tous les blocs Simulink du modèle. Les temps maximum, minimum et moyens sont comptabilisés à chaque pas et les résultats sont présentés à l'usager à la fin de l'exécution du modèle.

En analysant ces résultats, l'usager peut établir le pas de calcul minimum auquel le modèle peut fonctionner. Idéalement, on espère avoir des temps maximum, minimum et moyens qui sont semblables. Le fonctionnement de la fonction de mesure est simple, on prend une lecture du PIT au début et à la fin du pas de calcul. On soustrait ensuite les deux lectures afin d'obtenir la période en nombre de coups d'horloge. Dans le cas où la valeur obtenue est inférieure à zéro, cela signifie qu'il y a eu un roulement de compteur, on doit y additionner le nombre d'états du compteur de 16 bits soit 65536 (2^16). Finalement, on divise par la fréquence du PIT afin d'obtenir une période en seconde. Par

exemple, des valeurs de PIT de 20 et de 10 donneront un temps de calcul du (20-10) / 78125 = 128 µs. La figure 20 montre l'algorithme final du séquenceur.

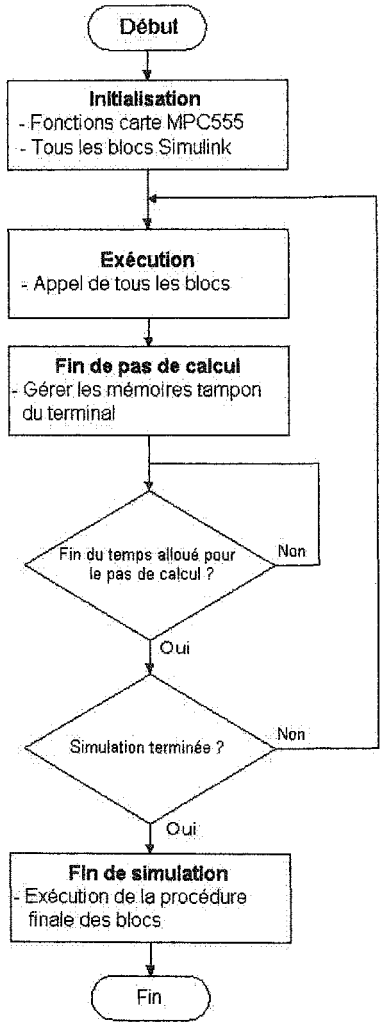

Figure 20 Organigramme du séquenceur final

6.6 Conclusion

L'ajout des fonctions de synchronisation et d'analyse du temps de calcul complète le développement du séquenceur du projet. Tous les éléments sont maintenant en place pour l'exécution de tout modèle Simulink sur le MPC555. Dans le chapitre suivant, on

développe des blocs permettant l'interaction avec le modèle à partir d'un terminal et l'utilisation de fonctions d'entrée-sortie du MPC555.

CHAPITRE 7

BLOCS D'ENTRÉE-SORTIE

7.1 Introduction

Cette étape vise à développer une bibliothèquee de blocs Simulink afin de permettre l'interaction avec le modèle à partir d'un terminal ainsi que l'utilisation des fonctions d'entrées-sorties requises pour le contrôle d'un moteur. Les deux fonctions d'entrées-sorties sont la sortie numérique à largeur d'impulsion variable et le décodage de signaux en quadrature.

Dans un premier temps, on présente les différents concepts liés à la création d'un bloc Simulink. Ensuite, on présente chaque bloc dans une sous-section différente. On y retrouve la définition des entrées, des sorties et des paramètres de configuration ainsi que la decription du processus de développement et des tests effectués.

7.2 S-function

Tout comme le reste du projet, les blocs Simulink sont programmés en langage C dans l'environnement CodeWarrior. Afin d'être compatible avec Simulink et RTW, on doit respecter le format C Mex S-function. Ce format définit plusieurs fonctions qui doivent être incluses dans chaque bloc Simulink. De plus, une autre bibliothèque de fonctions permet d'effectuer plusieurs tâches spécifiques à Simulink tels la lecture de paramètres, la lecture des entrées ou sorties des blocs, l'ajustement du pas de calcul du bloc et plusieurs autres. Les sous-sections 7.2.1 et 7.2.2 donnent un aperçu des fonctions les plus utiles.

7.2.1 Fonctions principales requises dans un bloc Simulink

- `mdlInitializeSizes(SimStruct *S)` : Initialise le nombre d'entrées et de sorties du bloc ainsi que le nombre de paramètres;
- `mdlInitializeSampleTimes(SimStruct *S)` : Ajuste le pas de calcul du bloc;
- `mdlInitializeConditions(SimStruct *S)` : Fonction d'initialisation;

- `mdlOutputs(SimStruct *S, int_T tid)` : Fonction de calcul appelée à tous les pas lors de l'exécution;
- `mdlTerminate(SimStruct *S)` : Fonction de fin.

7.2.2 Fonctions utilitaires les plus utilisées

- `ssSetNumOutputPorts(S,nOutputPorts)` : Ajuste le nombre de sorties du bloc;
- `ssSetOutputPortWidth(S,port,val)` : Ajuste la largeur d'une sortie;
- `ssSetNumInputPorts(S,nInputPorts)` : Ajuste le nombre d'entrées du bloc;
- `ssSetInputPortWidth(S,port,val)` : Ajuste la largeur d'une entrée;
- `ssSetSampleTime(S,sti,t)` : Ajuste le pas de calcul du bloc;
- `ssGetInputPortRealSignalPtrs(S,i)` : Obtient un pointeur sur une des entrées;
- `ssGetOutputPortRealSignal(S,i)` : Obtient un pointeur sur une des sorties;
- `ssGetSFcnParam(S,index)` : Obtient un pointeur sur un des paramètres du bloc.

7.2.3 Exécution d'une S-function

Chaque S-function doit pouvoir être exécutée à la fois dans l'environnement Simulink et sur le MPC555. Pour l'exécution dans Simulink, il est requis de compiler le code C de la S-function et de créer une bibliothèque dynamique (DLL, Dynamic Link Library). Pour l'exécution sur le MPC555, le code C est simplement compilé et assemblé dans CW avec la méthode habituelle.

Puisque les fonctions matérielles du MPC555 ne sont pas disponibles sur le PC, les sections de code C accédant à celles-ci ne sont considérées que si certaines options de compilation sont sélectionnées. On se retrouve donc généralement avec un bloc fonctionnel, mais inutile lors de l'exécution dans Simulink et un bloc pleinement fonctionnel lors de l'exécution sur le MPC555.

7.2.4 Création d'un masque de bloc Simulink

Par défaut, une s-function est accédée en utilisant le bloc `S-Function` de la section Real-Time Workshop de la bibliothèque de blocs Simulink. Afin de simplifier l'entrée des paramètres de chaque bloc et leur utilisation, Simulink permet la définition d'un masque. Celui-ci permet d'insérer une documentation sommaire de bloc et de présenter chaque paramètre sous différentes formes graphiques. Pour crér un masque pour le bloc `S-Function`, on n'a qu'à sélectionner la rubrique `Mask s-function` accessible en cliquant avec le bouton droit de la souris au dessus du bloc. La section `Icon` de la fenêtre qui apparaît permet de définir l'apparence du bloc en ajoutant du texte ou une image qui apparaîtra au dessus du bloc. Afin d'imprimer la valeur de certains paramètres, il est possible d'utiliser la fonction `sprintf`. La section `Initialization` permet la définition de la liste de paramètres. Pour chaque paramètre, il est possible d'en choisir la présentation sous forme d'une case simple (`Edit`), d'une case à cocher (`Checkbox`) ou d'un menu déroulant (`Popup`). La section `Documentation` permet d'ajouter une description sommaire du bloc ou un lien sur un document HTML.

7.3 Blocs d'interaction avec le terminal

Le but des blocs Op MPC555 Term Read et Op MPC555 Term Print est de fournir à l'usager une interface permettant de lire ou de modifier certains signaux lors de l'exécution d'une simulation. En fonction des liens de communications disponibles sur le MPC555 et sur un PC standard, l'utilisation du port série RS-232 a été préférée aux ports CAN ou JTAG qui nécessiteraient tous deux des connecteurs spéciaux ainsi que le développement d'un logiciel d'interface sur le PC. Avec le port série RS-232, il est possible d'utiliser tout logiciel de communication série disponible sur le marché dont HyperTerminal qui est fourni avec toutes les versions de Microsoft Windows.

7.3.1 Op MPC555Term Print

7.3.1.1 Description

Ce bloc permet d'afficher des valeurs de signaux lors de l'exécution du modèle. Puisque le système n'utilise qu'une tâche, cette fonction doit être optimisée afin de ne pas affecter la performance temps réel. Ainsi, on permet l'affichage de la valeur du signal lors d'un changement ou à intervalle fixe généralement plus lente que le pas de calcul du modèle Simulink.

7.3.1.2 Bloc Simulink

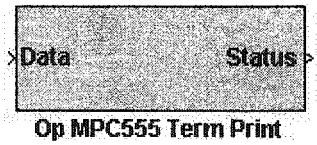

Figure 21 Op MPC555 Term Print

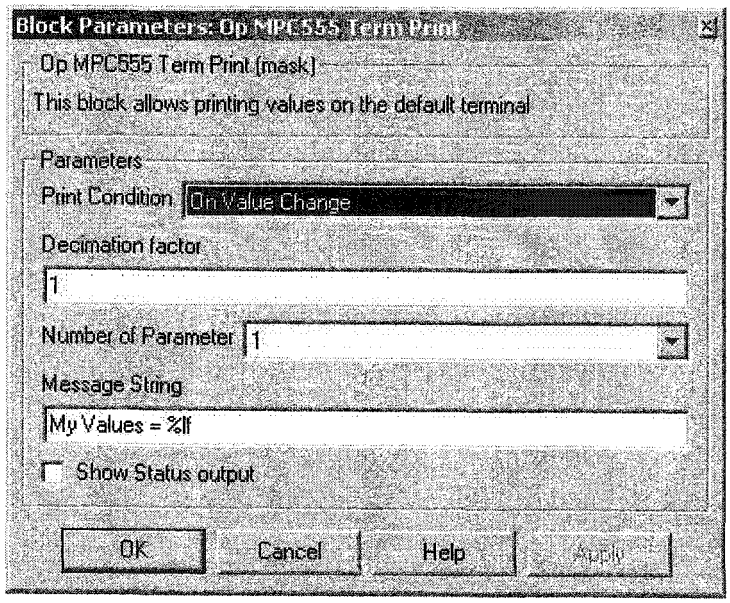

Figure 22 Masque du Op MPC555 Term Print

7.3.1.3 Paramètres

Print Condition : Condition pour laquelle on doit imprimer un message; Toujours ou seulement quand une des valeurs d'entrée change.

Decimation factor : Nombre de pas de calcul minimum entre deux impressions successives.

Number of parameter : Nombre d'entrées du bloc.

Message String : Message à afficher respectant la nomenclature de la fonction `printf` en C. Lors de l'affichage, les %lf seront remplacés par les valeurs à l'entrée du bloc.

Show Status output : Ajoute une sortie d'état au bloc.

7.3.1.4 Entrées

Data : Valeurs qui seront utilisées pour le message.

7.3.1.5 Sorties

Status : Sortie optionnelle, indique si un message a été affiché ou non (0 ou 1).

7.3.1.6 Implémentation

L'idée initiale pour l'implémentation de ce bloc a été d'utiliser la fonction `printf` qui est disponible sur le MPC555. À chaque fois que la condition d'impression et la contrainte de décimation sont respectées, on apelle la commande `printf` qui est automatiquement redirigée sur le port série de la carte MPC555.

7.3.1.7 Tests et corrections

Des tests initiaux ont permis de constater que le temps de calcul était largement influencé par ce nouveau bloc. Le tableau VIII présente le temps de calcul moyen en fonction de la largeur des messages envoyés avec le bloc.

Tableau VIII

Temps de calcul moyen en fonction des envois sur le port série

Largeur du message (octets)	Temps de calcul moyen (ms)
1	0.089
10	8.4
20	18.8

Ces chiffres sont facilement expliqués par la configuration du port série à 9600 bauds avec un bit d'arrêt où l'on peut transmettre 9600 bits par secondes soit environ un octet (8 bits) par ms. Les temps obtenus semblent directement liés à cette vitesse. La limite derrière ces chiffres est que la fonction `printf` utilisée pour envoyer les messages ne contient aucune gestion de mémoire tampon et bloque jusqu'à ce que tout le message soit envoyé sur le port série. Ce type de fonctionnement est évidemment inacceptable pour une application temps réel. De plus, l'architecture du module de communication SCI permet l'utilisation de mémoire tampon pour les envois. Lors de l'utilisation de cette mémoire, on copie simplement les données sans avoir à attendre la fin de transmission qui est gérée de façon transparente et automatique par le SCI.

Le problème a été réglé avec l'implémentation des deux fonctions suivantes :

`int RTPRint( char* str )` : Fonction qui reçoit le message à afficher et qui copie le tout dans une mémoire tampon circulaire sans pour autant effectuer l'envoi sur le port série. La fonction retourne 0 si la copie a eu lieu ou -1 si la mémoire tampon est pleine. Cette fonction est appelée par tous les blocs Op MPC555 Term Print.

`void emptyPrintBuffer()` : fonction qui vérifie l'état de la mémoire tampon du SCI et qui copie un ou plusieurs octets de la mémoire tampon de messages vers le SCI. Cette fonction est appelée dans la boucle d'attente de la fin du pas de calcul du séquenceur d'exécution. L'impact sur le temps de calcul est donc minimal.

Suite à ces modifications, le temps d'envoi de messages par le bloc Op MPC555 Term Print a été réduit à moins de 100 μs, et ce, peu importe la taille des messages envoyés.

7.3.2 Op MPC555Term Read

7.3.2.1 Description

Ce bloc permet de recevoir des commandes lors de l'exécution du modèle. Tout comme le bloc Op MPC555 TermWrite, cette fonction doit être optimisée afin de ne pas affecter la performance temps réel. Les commandes sont reçues de manière asynchrone à partir d'un terminal.

7.3.2.2 Bloc Simulink

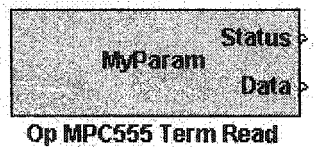

Figure 23 Op MPC555 Term Read

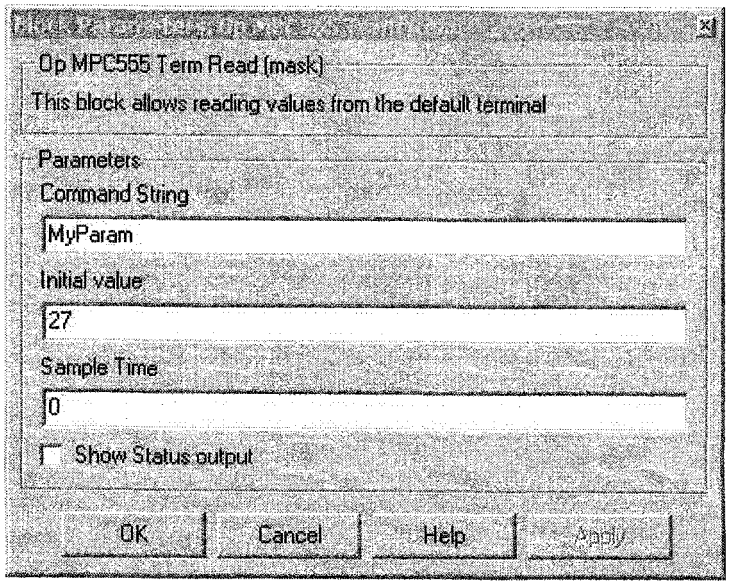

Figure 24 Masque du Op MPC555 Term Read

7.3.2.3 Paramètres

Command String : Nom de la commande. Pour modifier la valeur à partir du terminal, l'usager doit envoyer la nouvelle valeur dans le format suivant : nom_commande= XX 'Entrée'.

Initial value : Valeur initiale de la commande.

Sample Time : Pas d'intégration du bloc.

Show Status output : Ajoute une sortie d'état au bloc.

7.3.2.4 Entrées

Aucune

7.3.2.5 Sorties

Status : Sortie optionnelle, indique si un message a été affiché ou non (0 ou 1).

Data : Valeur actuelle de la commande.

7.3.2.6 Implém/entation

Avant de débuter l'implémentation, il était prévu d'utiliser des fonctions C non bloquantes telles `keypressed()` et `getc()` pour obtenir un à un les caractères reçus sur le port série. Après vérification, il a été établi que seule la fonction bloquante `getch()` est disponible dans la bibliothèque C pour le MPC555. Utiliser cette fonction bloquante n'est pas acceptable puisque nous ne savons pas quand les caractères seront reçus sur le port.

La solution retenue est d'implémenter les fonctions suivantes :

- `registerCommand` : Fonction d'enregistrement qui est appelée à l'initialisation par le bloc OpMpc555TermRead afin de déclarer une commande;
- `emptyInputBuffer` : Fonction non bloquante qui se charge de lire les caractères reçus sur le port série et d'analyser la correspondance avec les commandes enregistrées.

Dans la fonction d'enregistrement de commande, on crée une liste chaînée double dont la séquence est établie selon l'ordre alphabétique des noms de commande. Chaque élément de la liste chaînée contient les sous-éléments suivants :

- `nom` : Nom de la commande;
- `état` : Permet au bloc TermRead de savoir si une nouvelle valeur a été reçue;
- `valeur` : Valeur actuelle de la commande;
- `prev` : pointeur sur l'élément suivant;
- `next` : pointeur sur l'élément précédent.

La commande `emptyInputBuffer` est beaucoup plus complexe, elle consiste à gérer la lecture des données sur le port série et à analyser les messages reçus. Au niveau SCI, il doit être configuré en mode 'Queue'. Dans ce mode, la mémoire tampon du SCI est décomposée en deux banques de 8 registres de données qui sont utilisés comme une mémoire circulaire. Les différents registres du SCI permettent de connaître la position où le prochain caractère sera écrit et de confirmer si une des deux banques de données est pleine. La fonction doit indiquer au SCI que les données ont été lues en remettant les indicateurs de banque pleine à zéro. La séquence suivante présente les différentes étapes de la boucle de réception :

1. vérifier si un nouveau caractère est disponible (Registre QSCI1SR bits QRPNT, QBHF, QTHF);
2. lecture d'un caractère dans la mémoire tampon (Registre SCRQ[X]);
3. si une des banques est pleine (QSCI1SR bits QBHF et QTHF) et les données sont lues, effacer le bit;
4. vérifier et effacer tous les bits d'erreurs d'overrun (SC1SR bit OV), Framing (SC1SR bit FE), Parité (SC1SR bit PF), Bruit (SC1SR bit NF) et Queue Overrun (QSC1SR bit QOR).

Pour ne pas influencer les performances temps réel, la fonction `emptyInputBuffer` est appelée par le séquenceur d'exécution lors de l'attente de la fin du pas de calcul.

Aussi, l'analyse des données se fait graduellement et non à la suite de la réception de la touche 'Entrée' ce qui répartit l'effort d'analyse sur plusieurs pas de calcul plutôt que sur un seul. Lorsque la fonction `emptyInputBuffer` identifie une commande valide, elle met à jour les items `valeur` et `état` correspondants dans la liste chaînée. La logique de la fonction `mdl_output` de chaque bloc OpMpc555TermRead se résume à vérifier l'item `état` de la commande dans la liste chainée et de rafraîchir les données à la sortie du bloc.

7.3.2.7 Tests et corrections

Plusieurs tests ont été nécessaires afin de tester les différentes séquences d'entrée de commande sur le terminal. Les principales variables sont le nombre et la taille des commandes ainsi que les similarités entre celles-ci. Les erreurs initiales étaient dues à la fonction d'analyse qui n'arrivait pas à bien décoder les commandes semblables telles ABC et ABCD. Ceci fut vite réglé en traçant le code dans l'environnement CodeWarrior.

Ces tests ont permis de constater qu'il serait vraiment utile de supporter les cas spéciaux où l'usager utilise la touche de retour arrière (Backspace) pour faire une correction ou lorsque celui-ci ajoute des espaces entre le caractère '=' et la valeur de la commande. Ces modifications ont toutes deux été effectuées.

Finalement, des tests de limite de la mémoire tampon ont été effectués en bloquant la fonction de réception dans le débogueur et en envoyant 18 caractères soit un de plus que la taille de la mémoire tampon (16) et du registre de donnée interne du SCI (1). Après une telle séquence, la réception ne fonctionnait plus et aucun caractère n'était reçu. Après quelques vérifications de registres, il a été établi que les bits QOR (Queue overrun, registre QSCI1SR) et OR (Reception overrun, registre SCI1SR) étaient tous deux actifs, mais que le bit OR n'était jamais remis à zéro malgré le code prévu à cet effet. Suite à plusieurs lectures du manuel d'usager pour le MPC555 et de nombreux essais, il a été établi que les bits d'erreur ne sont pas remis à zéro avec la même méthode. Ainsi, les bits QOR (Queue Overrun) et NF (Noise Flag) sont remis à zéro suite à la

lecture de la mémoire tampon circulaire (SCRQ[X]) alors que les bits OR (Reception Overrun), PF (Parity Flag) et FE (Framing Error) sont remis à zéro suite à la lecture du registre de réception interne du SCI (SC1DR).

7.4 Blocs d'entrées-sorties

Le but des blocs Op MPC555 Pwm Out et Op MPC555 Quad In est d'utiliser les fonctions d'entrées-sorties du MPC555. Ces blocs seront utilisés pour l'application finale.

7.4.1 Op MPC555Term Pwm Out

7.4.1.1 Description

Ce bloc permet de générer un signal numérique à fréquence et largeur d'impulsion variable (PWM) à l'aide du module MIOS du MPC555.

7.4.1.2 Bloc Simulink

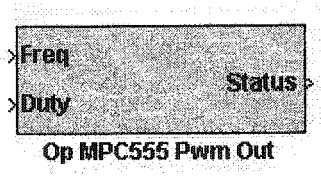

Figure 25 Op MPC555 Pwm Out

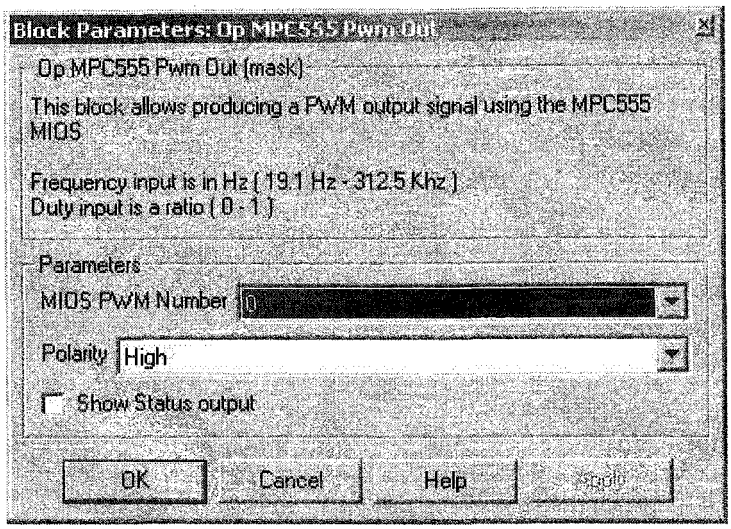

Figure 26 Masque du Op MPC555 Pwm Out

7.4.1.3 Paramètres

MIOS PWM Number : Canal du MIOS utilisé (0 à 3).

Polarity : Permet de définir la polarité du signal (High ou Low).

Show Status output : Ajoute une sortie d'état au bloc.

7.4.1.4 Entrées

Frequency : Fréquence du signal (19.1 Hz à 312.5 Khz).

Duty : Fraction correspondant au temps de polarité active du signal par rapport au temps inactif (0 à 1).

7.4.1.5 Sorties

Status : Sortie optionnelle, indique si un message a été affiché ou non (0 ou 1) .

7.4.1.6 Implémentation

Le module MIOS (Modular Input/Output Subsystem) contient plusieurs fonctionnalités d'entrée-sortie telle la mesure de fréquence et de période, la génération de signaux à modulation de largeur d'impulsion (PWM), des compteurs multi-usages et des entrées-sorties numériques simples.

Pour le bloc PWM out, on utilise la fonction de PWM qui est disponible sur les canaux 0 à 3 et 16 à 19. Au besoin, il serait aussi possible d'utiliser les canaux du TPU A et B afin d'obtenir la même fonctionnalité.

Les tableaux IX et X présentent le pseudo code de l'initialisation et de l'exécution du bloc.

Tableau IX

Pseudo code de l'initialisation du bloc Op MPC555 Pwm Out

Séq.	Fonction	Pseudo code
1	ajuster le diviseur du MIO à 16	MCPSMSCR bit PSL = 0
2	permettre l'exécution du MIO	MIOS1MCR bit STOP = 0
3	ajuster la polarité	MPWMSM0SCR bit POL = paramètre polarité
4	ajuster le diviseur à compteur 1	MPWMSM0SCR bit CP = 0xFF
5	remettre le compteur à zéro	MPWMSM0CNTR = 0x0000
6	ajuster la période initiale	MPWMSM0PERR = 0x0
7	ajuster la largeur d'impulsion initiale	MPWMSM0PULR = 0x0
8	permettre l'exécution du canal	MPWMSM0SCR bit EN = 1

Tableau X

Pseudo code de l'exécution du bloc Op MPC555 Pwm Out

Séq.	Fonction	Pseudo code
1	ajuster la période	MPWMSM0PERR = 40000000 / 16 / entrée fréquence (horloge interne / diviseur / fréquence)
2	ajuster la largeur d'impulsion	MPWMSM0PULR = entrée duty * période actuelle

7.4.1.7 Tests et corrections

Différents essais ont été effectués avec des fréquences allant de 19.1 Hz à plus de 312.5 Khz et des fractions de largeur d'impulsion allant de 0 à 100% (1). Le tout fonctionne sans problème.

7.4.2 Op MPC555Term Quad In

7.4.2.1 Description

Ce bloc permet d'obtenir une position en fonction des signaux d'un encodeur en quadrature. Ce type d'encodeur produit des signaux A et B qui sont déphasés de plus ou moins 90 degrés selon la direction ainsi qu'un signal optionnel Z qui permet de détecter le point initial (zéro degré). La figure 27 illustre les signaux d'un encodeur en quadrature.

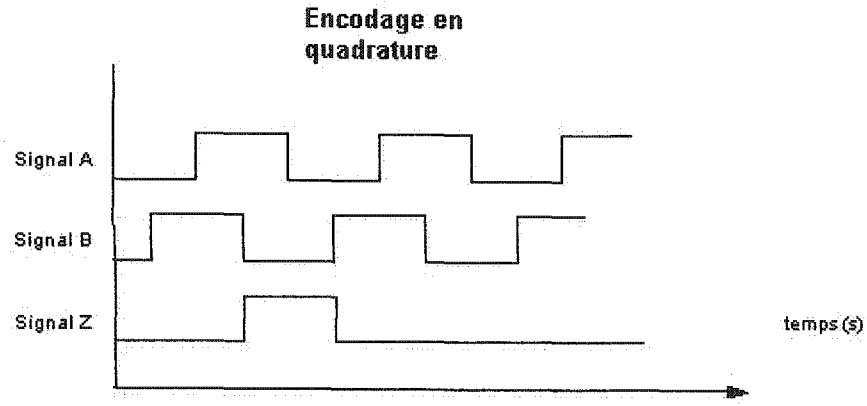

Figure 27 Signaux d'un encodeur en quadrature

La fonction Fast Quadrature Decode du TPU (FQD) permet le décodage de tels signaux. Cependant, comme les signaux A et B doivent être analysés, cette fonction utilise deux canaux successifs du TPU.

7.4.2.2 Bloc Simulink

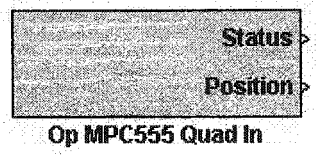

Figure 28 Op MPC555 Quad In

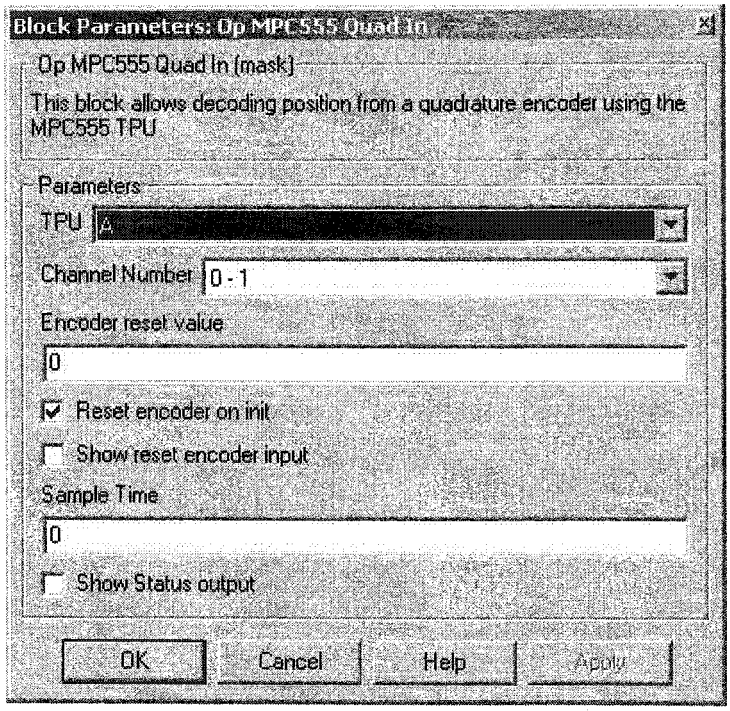

Figure 29 Masque du Op MPC555 Quad In

7.4.2.3 Paramètres

TPU : Choix du TPU utilisé (A ou B).

Channel number : deux canaux utilisés (De 0-1 à 14-15).

Encoder reset value : Valeur utilisée lors d'une reconfiguration de position.

Reset encoder on init : Force une reconfiguration de position lors de l'initialisation du bloc.

Show reset encoder input : Ajoute une entrée pour forcer une reconfiguration de position.

Sample time : Pas d'intégration du bloc.

Show Status output : Ajoute une sortie d'état au bloc.

7.4.2.4 Entrées

Encoder reset : Entrée optionnelle, permet de forcer une reconfiguration de position lorsque la valeur d'entrée est supérieure ou égale à 1.

7.4.2.5 Sorties

Status : Sortie optionnelle, permet de confirmer si une nouvelle position a été lue (1) ou non (0).

Position : Position lue sur 16 bits (0-65535).

7.4.2.6 Implémentation

Les deux canaux utilisés se nomment respectivement canal primaire (CHP) et secondaire (CHS). Le canal primaire est celui où se trouve l'information sur la position actuelle alors que le canal secondaire est géré à l'interne par le TPU. Chaque canal du TPU utilise une section mémoire contenant de un à six paramètres qui varient selon la fonction utilisée. Dans le cas de la fonction Fast Quadrature Decoding, certains paramètres sont utilisés comme pointeurs afin que chaque canal puisse référer aux informations de l'autre.

Les tableaux XI et XII présentent le pseudo code de l'initialisation et de l'exécution du bloc.

Tableau XI

Pseudo code de l'initialisation du bloc Op MPC555 Quad In

Séq.	Fonction	Pseudo code
1	désactive les deux canaux	CPR[0:1] bits CHP et CHS = 0
2	ajuste la fonction TPU à Fast Quadrature Decoder (FQD) pour chaque canal	CFSR[0:3] bits CHP et CHS = TPU_FUNCTION_FQD (0x06)
3	défini la position initiale	CHP paramètre 2 = XX
4	ajuste les pointeurs de chaque canal sur son canal complémentaire	paramètres 5 (CORR_PIN) et 6 (CORR_EDGE) de chaque canal = pointeur autre canal
5	définit le canal primaire (CHP)	HSQ[0:1] bits CHP = TPU_FQD_PRIM_NORMAL(0x0)
6	définit le canal secondaire (CHS)	HSQ[0:1] bits CHS = TPU_FQD_SEC_NORMAL(0x1)
7	initialise les deux canaux	HSR[0:1] bits CHP et CHS = TPU_FQD_INIT (0x03)
8	ajuste la priorité de chaque canal	CPR[0:1] bits CHP et CHS = TPU_PRIORITY_HIGH (0x03)

Tableau XII

Pseudo code de l'exécution du bloc Op MPC555 Quad In

Séq.	Fonction	Pseudo code
1	lit la dernière position mesurée	position = CHP paramètre 2

7.4.2.7 Tests et corrections

Les tests du bloc ont été effectués à l'aide d'un moteur DC avec un encodeur de position intégré. Les fils A et B ont été reliés aux canaux 0 et 1 du TPU A. L'alimentation 5 volts requise pour alimenter l'encodeur a été obtenue du connecteur X3 de la carte de développement.

En bougeant manuellement l'axe du moteur, il a été établi qu'une rotation complète (360 degrés) fait varier la position de l'encodeur de 2000. Cette variation est constante et linéaire peu importe le sens de rotation. Puisque le décodage de signaux en quadrature requiert 4 comptes par dent, on en déduit que l'encodeur du moteur possède 500 dents (2000 / 4).

7.5 Conclusion

Le développement de tous les blocs d'entrée-sortie complète l'environnent de développement requis pour ce projet. Afin de confirmer le bon fonctionnement de tous les éléments dans le cadre d'un besoin réel, la prochaine étape consiste à développer un contrôleur en position utilisant ces blocs.

CHAPITRE 8

APPLICATION DES OUTILS DÉVELOPPÉS

8.1 Introduction

Le but de cette partie est d'utiliser les blocs Simulink et autres outils développés précédemment afin de concevoir une application où un moteur est contrôlé en position.

On y décrit tout d'abord les équipements et branchements nécessaires. Dans une deuxième étape, on présente le modèle Simulink et l'interface usager qui ont été développés.

8.2 Matériel utilisé

La composante principale de la démonstration est un moteur DC Pittman GM14900 avec un encodeur de position intégré. Un pont en H soudé sur circuit imprimé est utilisé afin de fournir la puissance au moteur. Le circuit a été développé par les étudiants de l'ÉTS. Il utilise un pont LMD18200 de National Semiconductor. Au niveau du fonctionnement, le pont fournit la tension provenant d'une source externe en fonction d'une commande numérique à largeur d'impulsion variable. Deux entrées numériques additionnelles permettent de contrôler la direction du moteur et l'arrêt d'urgence (frein) qui désactive le circuit, peu importe la commande reçue.

La figure 30 montre les branchements entre la carte de développement PhyCORE MPC555, le pont en H, la source d'alimentation externe et le moteur. Le pont en H est alimenté par une source externe continue de 12 volts pouvant fournir jusqu'à 3 ampères ce qui est suffisant pour le moteur utilisé. Les signaux de commande, direction et frein proviennent des canaux 0,1 et 2 du module MIOS. Ces canaux utilisent le bloc de sortie numérique à largeur d'impulsion variable développé lors du chapitre précédent. Étant donné que les signaux numériques de direction et frein sont statiques, on utilise des

largeurs d'impulsion de 0% pour obtenir un 0 ou 100% pour obtenir un 1. Les signaux A et B de l'encodeur de position sont reliés aux canaux 0 et 1 du TPU A.

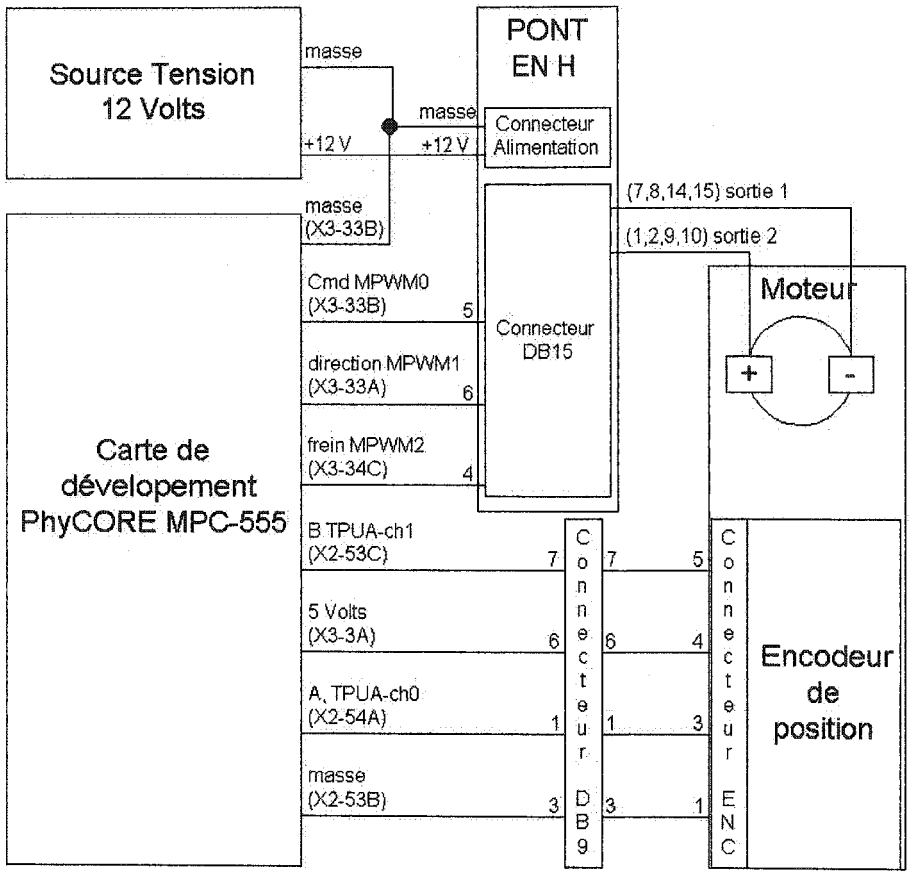

Figure 30 Branchements requis pour l'application de contrôle

8.3 Modèle Simulink

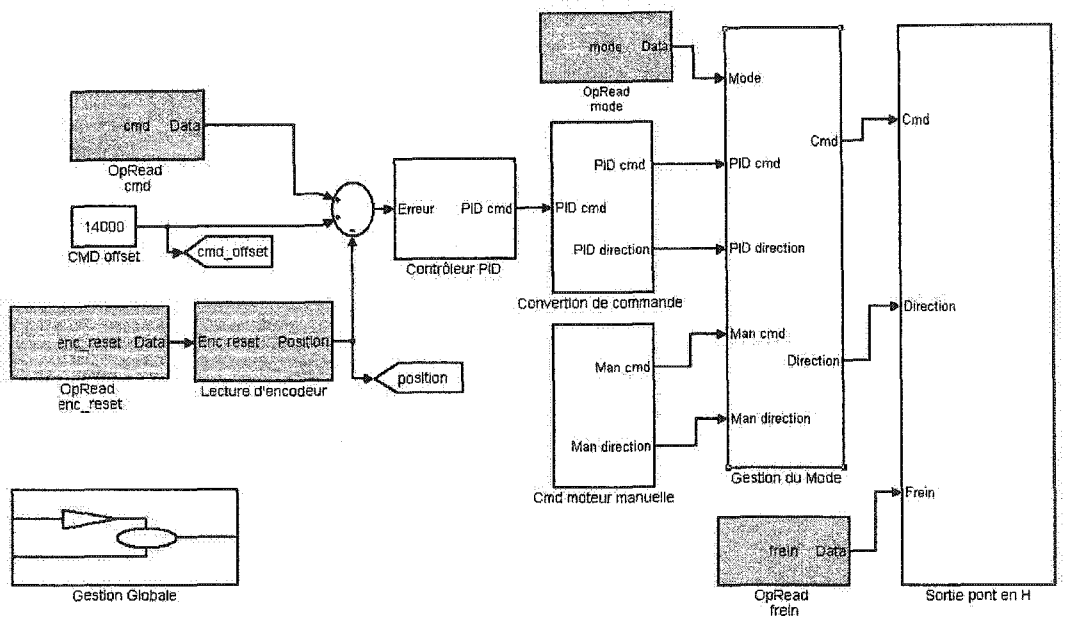

Figure 31 Modèle Simulink du contrôleur de position

La figure 31 présente le modèle développé pour le contrôle du moteur. Celui-ci utilise une commande provenant du terminal dont la valeur est convertie de l'intervalle [0, 36000] à [14000, 50000] afin de ne pas rencontrer le cas où la position passe de 65535 à 0 ce qui pourrait rendre le contrôleur instable. Parcourant le modèle de gauche à droite, l'erreur de position est d'abord calculée et transmise au sous-système `Contrôleur PID` qui produit une commande. La commande est convertie en valeur absolue et direction tel que requis par le pont en H. Le sous-système `Gestion du Mode` permet de sélectionner la commande du PID ou une commande manuelle en fonction de la variable mode provenant du terminal. Finalement, la commande et la direction sélectionnées ainsi que la variable frein sont envoyées au sous-système `Sortie pont en H` où se trouvent trois blocs de sortie numérique à largeur d'impulsion variable. Le sous-système `Gestion globale` contient un bloc d'impression sur le terminal pour la position actuelle ainsi qu'un bloc de lecture de variable (commande quit) dont la

sortie est reliée au bloc `Stop Simulation` de Simulink. Ceci permet de forcer l'arrêt de la simulation.

À l'intérieur du sous-système `Contrôleur PID` se trouvent les gains KI, KP et KD ainsi qu'un gain global (att). Ce dernier sert à convertir en radian l'erreur qui est initialement en comptes d'encodeur. Puisque que l'encodeur retourne 2000 comptes par rotation et que 2π radians correspond à 360 degrés, ce gain a été ajusté à $2.\pi$ / 2000 ou 0.003.

8.4 Interface usager

Tout comme pour les tests précédents, il est possible d'interagir avec le modèle à l'aide du logiciel HyperTerminal ou de tout autre logiciel de terminal. Le désavantage de cette méthode est que l'on doit entrer toutes les variables à modifier lors de chaque exécution ce qui est peu efficace pour l'ajustement de gains ou lorsque le modèle contient beaucoup de paramètres. Dans le cadre de cette application, un panneau d'interface Labview a été développé afin de palier à ces limitations.

La figure 32 présente l'interface développée. Du côté gauche, on y retrouve un graphe de la commande et de la position actuelle ainsi que les contrôles du mode manuel et quelques boutons de configuration. Il est possible de remplacer le graphe par un terminal de réception en désélectionnant le bouton `Show Scope`. Du côté droit, on retrouve un indicateur de la position actuelle ainsi que la commande et tous les gains du PID. L'interrupteur mode permet de griser ou d'activer la section de commande manuelle ou la section de commande et gains du PID.

Aussitôt qu'un des contrôles du panneau Labview est modifié, la nouvelle valeur est transmise au modèle. Au niveau du graphe, il est important de mentionner que celui-ci est synchronisé à une fréquence de 10 Hz (100ms) par les réceptions des données du modèle. Advenant une perte de données sur le port série, il y aura une erreur de 100ms

dans le graphe. Aucun mécanisme de resynchronisation n'est disponible dans la version actuelle de l'interface.

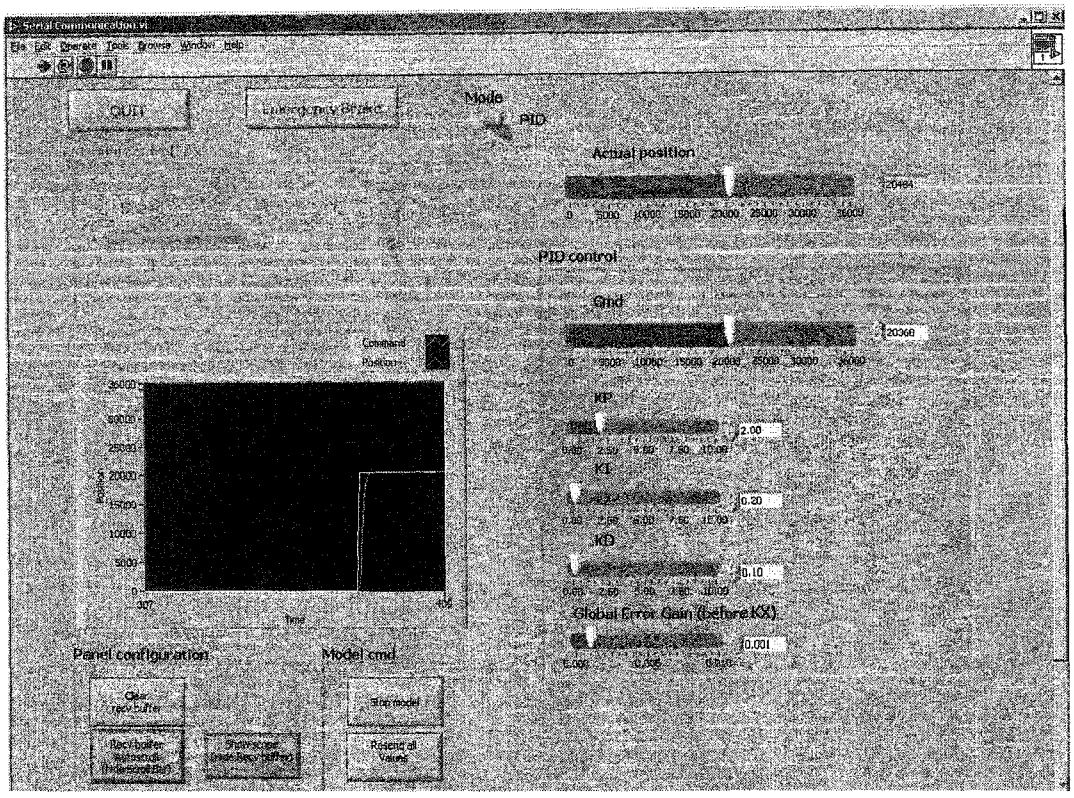

Figure 32 Panneau d'interface Labview

Le programme du panneau Labview est défini à l'aide d'un schéma bloc semblable à ceux développés dans l'environnement Simulink. L'interface avec le lien RS-232 se fait à l'aide des blocs suivants:

- `Serial Port Init.vi` : Ouvre le port sélectionné;
- `Bytes At Serial Port.vi` : Retourne le nombre de caractères disponibles dans la mémoire tampon de réception;
- `Serial Port Read.vi` : Lit le nombre spécifié de caractères dans la mémoire tampon de réception. Si non disponible, bloque jusqu'à la réception de ceux-ci;

- `Serial Port Write.vi` : Transmet le nombre spécifié de caractères sur le port série;
- `Close Serial Driver.vi` : Ferme le port sélectionné.

Le programme contient une boucle à trois phases qui s'exécute continuellement jusqu'à ce que le bouton `Quit` soit appuyé. La première phase vérifie si de nouveau caractères sont disponibles et les lit. Chaque caractère reçu est envoyé sur le terminal de réception. On repère la position actuelle du moteur en cherchant une ligne respectant le format de la position soit POSITION=>XXXXX<= où XXXXX est la valeur de la position. Puisque le nombre de caractères pour la valeur est toujours de 5, on utilise la fonction de conversion `Decimal String To Number` qui extrait les caractères et les convertit en nombre décimal. La deuxième phase vérifie tous les contrôles et envoie les nouvelles valeurs si nécessaire. Finalement, la dernière phase gère les boutons globaux tel le bouton `Mode` qui requiert de griser ou non certaines sections du panneau.

8.5 Identification du moteur

Afin d'ajuster les gains du contrôleur PID, l'identification du moteur a d'abord été effectuée. L'équation 8.1 représente la fonction de transfert du moteur.

$$\frac{\Omega}{U} = \frac{k}{\tau s + 1} \tag{8.1}$$

On cherche à trouver les valeurs de τ et de k. Un bloc dérivé a été utilisé afin d'obtenir la vitesse à partir du signal de position de l'encodeur. La figure 33 montre la réponse du moteur à une commande constante de 2.5 volt. Puisque la vitesse en régime permanent (s=0) est de 150 rad/s, on obtient une valeur de k de 60 (équation 8.2).

$$\frac{\Omega}{2.5} = \frac{k}{1}, k = \frac{150}{2.5} = 60 \qquad (8.2)$$

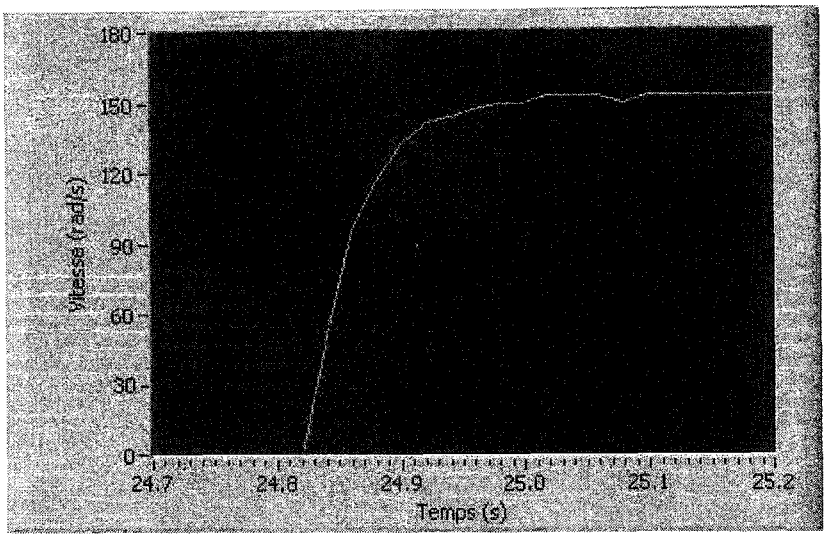

Figure 33 Identification du moteur

τ est égal au temps requis pour obtenir 63% de la valeur en régime permanent soit 94.5 rad/s. Selon la figure 33, on obtient environ 40 ms. Cette valeur est cependant approximative puisque la fréquence d'échantillage du système est limitée à 20 ms.

En fonction des paramètres du moteur et de l'équation discrète de celui-ci, on a établi les gains afin d'obtenir un dépassement de moins de 5 % :

Kp = 5.73

Ki = 0

Kd = 0.23

La figure 34 montre le résultat d'une simulation effectuée sous MATLAB avec les valeurs de gains calculés. La figure 35 montre les résultats obtenus avec ces mêmes gains dans l'environnement réel (avec le moteur). On constate que les résultats sont semblables. L'identification du moteur est donc juste. Il est à noter que le temps de

l'échelon à la figure 35 n'est pas de 0s puisque celui-ci est provoqué pas une commande manuelle provenant de l'interface usager.

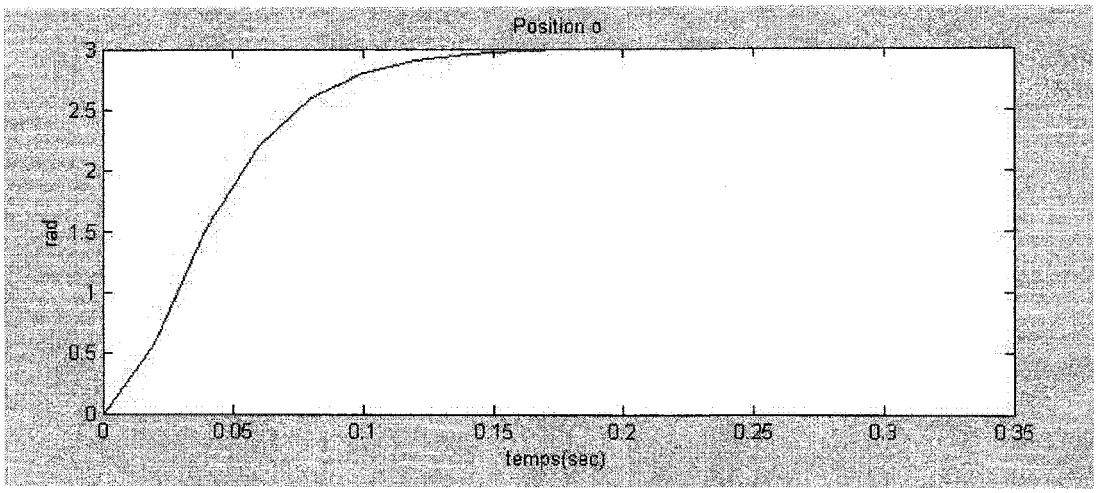

Figure 34 Position simulée en fonction du temps

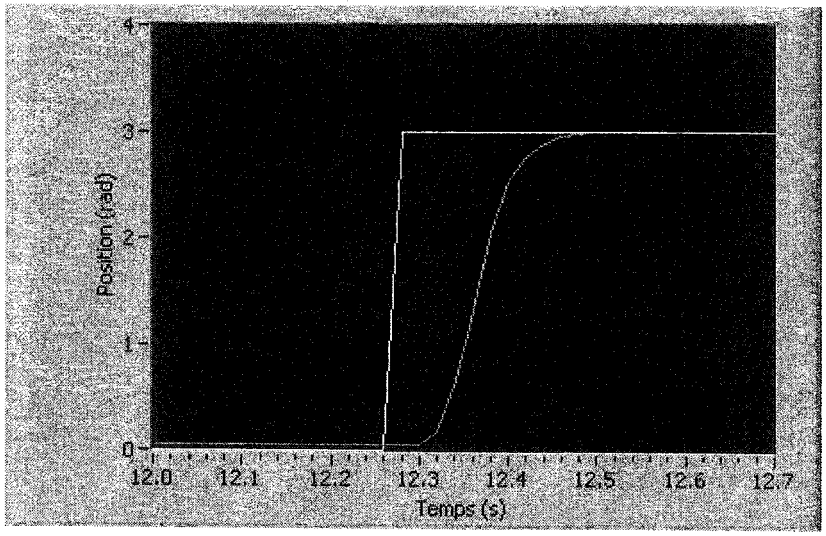

Figure 35 Position réelle en fonction du temps

8.6 Tests

De nombreuses compilations ont dû être effectuées afin de faire fonctionner le modèle final. Lors de ce processus, plusieurs blocs OpMpc555TermPrint ont été utilisés afin de vérifier l'exactitude de certaines valeurs intermédiaires.

Aucun problème majeur n'a été noté si ce n'est que le temps de calcul semble significativement influencé lorsque plusieurs blocs OpMpc555TermPrint sont utilisés. Le lecteur est prié se de référer au chapitre 9 pour plus de détails sur les temps de calcul obtenus.

8.7 Conclusion

Cette étape nous a permis de constater qu'il est possible de rapidement développer des programmes à l'aide de l'environnement Simulink et des modules développés pour ce projet. Lors du prochain chapitre, nous évaluerons les performances du modèle lorsque la complexité et le nombre de blocs augmentent.

La vitesse d'acquisition de données constitue une des limites significatives du système qui réduit la précision du processus d'identification.

CHAPITRE 9

ANALYSE DE PERFORMANCE

9.1 Introduction

L'objectif de ce chapitre est de déterminer si la solution développée peut être utilisée dans des applications réelles où la charge de calcul et le nombre de blocs d'entrées-sorties sont très élevés. Pour ce faire, on notera les temps de calcul pour plusieurs combinaisons de modèles. Si les temps de calcul sont relativement bas et évoluent de façon linéaire, on pourra déclarer que le système est évolutif et pourra être utilisé dans des projets de plus grande envergure. On pourrait être tenté de déclarer que la linéarité est une caractérisque certaine, mais ce n'est pas toujours le cas. En effet, la gestion interne du 'pipeline' d'instruction et de l'antémémoire ainsi que les programmes et le système d'exploitation utilisés sont des facteurs qui peuvent influencer significativement la linéarité.

Le deuxième test consiste à vérifier la taille de l'exécutable et l'utilisation de mémoire en fonction de la complexité du modèle.

Finalement, on évalue les coûts de la solution développée par rapport au prix d'un générateur de code de production et on analyse les résultats obtenus pour l'ensemble du projet.

9.2 Tests de performance

Le tableau XIII présente les valeurs minimum, maximum et moyennes du temps de calcul pour différents modèles Simulink. Pour chaque bloc Simulink développé, on mesure les performances avec un, quatre et huit blocs. De plus, deux tests ont été effectués avec le modèle de contrôleur en position du chapitre précédent ainsi qu'une nouvelle version permettant le contrôle de deux moteurs. Lors de ce dernier test, le deuxième moteur n'était pas relié mais, toutes les fonctions d'entrées-sorties étaient utilisées.

Tableau XIII

Sommaire des performances en fonction du nombre de blocs

Nom du modèle	Description	Temps min (us)	Temps max (us)	Temps moy (us)
vide.mdl	Modèle vide	12.8	25.6	16.8
TRead_1.mdl	1 bloc OpMpc555TermRead	12.8	25.6	16.8
TRead_4.mdl	4 blocs OpMpc555TermRead	25.6	38.4	27.75
TRead_8.mdl	8 blocs OpMpc555TermRead	38.4	51.2	38.4
TPrint_1s10.mdl	1 bloc OpMpc555TermPrint avec une variable et un message de dix caractères qui est imprimé à tous les 1000 pas (1 sec).	12.8	230.4	20.8
TPrint_4s10.mdl	4 blocs OpMpc555TermPrint avec une variable et un message de dix caractères qui est imprimé à tous les 1000 pas (1 sec).	38.4	857.6	39.2
TPrint_8s10.mdl	8 blocs OpMpc555TermPrint avec une variable et un message de dix caractères qui est imprimé à tous les 1000 pas (1 sec).	64.0	1702.4	65.6
PwmOut_1.mdl	1 bloc OpMpc555PwmOut	25.6	38.4	25.6
PwmOut_4.mdl	4 blocs OpMpc555PwmOut	38.4	294.4	51.2
PwmOut_8.mdl	8 blocs OpMpc555PwmOut	76.8	320.0	84.5
QuadIn_1.mdl	1 bloc OpMpc555QuadIn	12.8	25.6	21.4
QuadIn_4.mdl	4 blocs OpMpc555QuadIn	25.6 / 25.6	51.2 / 217.6	35.3 / 37.36
QuadIn_8.mdl	8 blocs OpMpc555QuadIn	51.2 / 51.2	64.0 / 230.4	51.2 / 55.8

Nom du modèle	Description	Temps min (us)	Temps max (us)	Temps moy (us)
pid_final.mdl	Modèle de démonstration contenant : - 11 OpMpc555TermRead - 1 OpMpc555TermPrint - 3 OpMpc555PwmOut - 1 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	115.2	358.4	130.8
pid_final_2.mdl	Modèle de démonstration pour deux moteurs contenant : - 22 OpMpc555TermRead - 2 OpMpc555TermPrint - 6 OpMpc555PwmOut - 2 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	230.4	563.2	245.8

Les variations de temps de calcul obtenues pour les blocs OpMpc555TermRead, OpMpc555PwmOut et OpMpc555QuadIn se situent à moins de 10 µs par bloc ce qui est raisonnable. On constate cependant que certains tests affichent des maximums d'environ 200 µs au dessus des valeurs moyennes ce qui n'est à permière vue pas normal. En affichant plusieurs maxima pour chaque test, il a été établi que ces valeurs maximales apparaissent entre 0 et 3 fois sur une période d'environ 30 secondes (30 000 pas de calculs). Comme le cas se produit dans des modèles avec des blocs différents, la source du problème se trouve possiblement à l'intérieur du séquenceur. À ce jour, rien n'a été trouvé à ce sujet.

Les résultats du bloc OpMpc555TermPrint affichent des temps maximums très élevés qui correspondent toujours au pas de calcul où l'affichage est effectué. Afin de mieux comprendre ces maxima, des mesures de temps ont été ajoutées dans le code du bloc. Il a été établi que la fonction `sprintf` utilisée pour création des messages est exigeante en temps de calcul comme le démontre le tableau XIV. On constate d'ailleurs que la

création d'un message sans aucune variable est très rapide alors la conversion d'une valeur de variable prend environ 120 µs par variable. Ceci devra être sérieusement considéré lors du développement de modèle.

Tableau XIV

Temps moyen d'exécution de la fonction `sprintf`

Nombre de caractères	Nombre de variables	Temps moyen du `sprintf` (us)
1	0	25.6
30	0	38
60	0	64
10	1	128
10	2	243
10	3	358
10	4	473
10	5	588
10	10	1152
10	10 (Avec format 3.0)	1164

Finalement, les tests avec le modèle de démonstration *pid_final_2.mdl* démontrent que le contrôle de deux moteurs pourrait se faire à 1 ms puisque le temps de calcul maximum est de 563 µs. Le temps de calcul de ce dernier modèle est d'ailleurs environ deux fois plus élevé que le temps correspondant du contrôleur pour un moteur *(pid_final.mdl)*.

9.3 Taille du Code

Cette partie vise à confirmer que la taille de l'exécutable obtenu et la quantité de mémoire RAM sont appropriées pour un MPC555. Considérant que la taille de la mémoire flash interne pour un microcontrôleur MPC555 est de 448 Ko et que la carte PhyCoreMPC555 permet d'ajouter au plus 4 Mo supplémentaire, on désire que la taille des exécutables soit à l'intérieur de ces limites. La carte MPC555 possède 26 Ko de RAM interne et jusqu'à 4 Mo supplémentaires sur la carte de développement phyCORE MPC555.

Le tableau XV présente la taille de l'exécutable et la quantité de mémoire utilisée pour différents modèles Simulink. On y constate qu'avec 16 contrôleurs PID, on utilise 383 Ko de Flash et 284 Ko de mémoire RAM. La quantité de RAM et de flash augmente d'ailleurs linéairement avec la complexité du modèle. Les données du tableau ont été obtenues directement de l'environnement CodeWarrior qui fournit les items *Code* (Taille de l'exécutable) et *Data* (Mémoire RAM utilisée) pour le programme complet ainsi que pour chaque fichier compilé.

Tableau XV

Taille de l'exécutable et de la mémoire en fonction du modèle

Nom du modèle	Description	Taille de l'exécutable (Ko)	Mémoire RAM utilisée (Ko)
simple.mdl	Modèle vide	136	20
pid_final.mdl	Modèle de démonstration contenant : - 11 OpMpc555TermRead - 1 OpMpc555TermPrint - 3 OpMpc555PwmOut - 1 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	150	36
pid_final_2.mdl	Modèle de démonstration pour deux moteurs contenant : - 22 OpMpc555TermRead - 2 OpMpc555TermPrint - 6 OpMpc555PwmOut - 2 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	165	53

Nom du modèle	Description	Taille de l'exécutable (Ko)	Mémoire RAM utilisée (Ko)
pid_final_4.mdl	Modèle de démonstration pour deux moteurs contenant : - 44 OpMpc555TermRead - 4 OpMpc555TermPrint - 12 OpMpc555PwmOut - 4 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	196	86
pid_final_8.mdl	Modèle de démonstration pour deux moteurs contenant : - 88 OpMpc555TermRead - 8 OpMpc555TermPrint - 24 OpMpc555PwmOut - 8 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	258	152
pid_final_16.mdl	Modèle de démonstration pour deux moteurs contenant : - 176 OpMpc555TermRead - 16 OpMpc555TermPrint - 48 OpMpc555PwmOut - 16 OpMpc555QuadIn Les impressions du bloc OpMpc555TermPrint sont effectuées à tous les 100 pas (0.1 sec).	383	284

9.4 Comparaison de coût

Afin de pouvoir utiliser la solution développée dans le cadre de ce projet, l'utilisateur doit se procurer, en plus de Matlab et Simulink, le générateur de code Real-Time Workshop qui se détaille à CAN$ 12,000 pour les entreprises.

Dans le cas où l'on veut utiliser la solution pour code de production de MathWorks, on devra débourser CAN$ 8,000 supplémentaires pour le générateur Real-Time Workshop

Embedded Coder et CAN$ 6,400 pour le module spécifique au MPC555 (Embedded Target for MPC555) soit CAN$ 14,400 de plus.

Considérant ce coût et le temps de développement requis, on peut considérer que le choix de l'une ou l'autre des solutions sera influencé par la nécessité d'obtenir un code de taille réduite et le nombre de postes de développement requis.

9.5 Discussion et interprétation des résultats

Les principaux résultats dont nous pouvons faire l'analyse sont ceux de fonctionnement global et ceux de performance.

Pour les tests de fonctionnement global, on devait vérifier chaque programme ou modèle de tests. Une des méthodes de test a été d'insérer des fonctions d'impressions qui affichent des valeurs intermédiaires lors de l'exécution. Avec cette technique, il a été possible d'isoler chaque section de code. De plus, l'utilisation des DEL de la carte de développement a permis de vérifier le temps d'exécution approximatif de la fonction de synchronisation. Finalement, le débogage du code C dans l'environnement CodeWarrior permet de tracer chaque ligne de programme. On a pu ainsi comprendre certaines erreurs, mais aussi rapidement en provoquer en forçant la valeur de certaines variables. Face à toutes ces techniques et à la répétition des tests à au moins trois reprises, on considère que ceux-ci sont valables et pourront être reproduits en tout moment.

Pour les tests de performance, les mesures sont effectuées à l'aide du compteur PIT qui est cadencé à une fréquence de 78.125 Khz. La résolution correspondante est de 12.8 µs soit 1 / 78125. Par rapport à un pas de calcul typique d'un modèle de contrôle qui est de l'ordre de la ms, une telle résolution est suffisante.

Au niveau des résultats obtenus, on constate que chaque bloc OpMPC555TermRead terminal, OpMPC555PwmOut ou OpMPC555QuadIn utilise moins de 10 µs de temps de calcul. Le bloc OpMPC555TermPrint fait cependant figure d'exception puisque son temps d'exécution varie grandement selon le nombre de variables à afficher. Cette

performance est justifiée par l'utilisation de la fonction `sprintf` qui doit convertir chaque variable en chaîne de caractères pour ensuite concaténer toutes les chaînes en une seule. Il est donc normal que tous ces mouvements de mémoire aient un impact sur le temps d'exécution.

9.6 Conclusion

Les tests effectués ont permis de constater que le temps de calcul augmente linéairement avec la complexité du modèle. La performance des blocs spécifiques au MPC555 est raisonnable mis à part le bloc d'impression OpMpc555TermPrint dont la conversion de valeur en chaîne de caractères nécessite environ 120 μs par variables.

En fonction des résultats obtenus, on peut établir que le générateur de code Real-Time Workshop est suffisament optimisé et configurable pour permettre le ciblage du code sur un processeur embarqué.

CONCLUSION

La production d'une solution de génération de code pour un processeur embarqué tel le MPC555 requiert un travail modulaire réparti entre les fonctionnalités de l'environnment Simulink et celle du processeur embarqué.

Du côté du MPC555, on a dû apprendre l'environnement CodeWarrior pour ensuite implémenter les fonctions de synchronisation à l'aide du PIT , les fonctions de communications à l'aide du SCI ainsi que les fonctions d'entrées-sorties à l'aide du TPU et du module MIOS. Les fonctions du SCI permettent la communication à travers une interface RS-232 mais nécessitent une gestion efficace de la mémoire tampon d'entrée et de sortie, ce qui n'est pas fourni avec les fonctions de base du système. Quant aux fonctions d'entrées-sorties, le TPU a été utilisé pour la fonction d'entrée d'encodeur en quadrature et le MIOS pour la sortie numérique à largeur d'impulsion variable.

Au niveau de l'environnement Simulink, on note tout d'abord qu'un modèle de génération avec les fichiers de configurations TMF et TLC doit être fourni à Real-Time Workshop lors de la génération de code. Un séquenceur écrit en langage C constitue la coeur de chaque programme. Il se charge de la synchronisation et de l'appel de fonctions d'exécution générées par RTW. Les S-fonctions de Simulink ont été utilisées pour la création des blocs Simulink créés pour ce projet. Ces S-function sont des fichiers en C contenant des fonctions standardisées qui seront appelées par le séquenceur lors de l'exécution.

Comme suite aux analyses de performance, on constate que Real-Time Workshop permet la génération d'un code suffisamment compact pour être utilisé sur le MPC555. De plus, le contrôle d'un moteur DC a permis de confirmer la pertinence de cette solution dans le cadre de projet de prototypage rapide.

Face à la rapidité du développement d'une application complète et au nombre élevé de fonctions d'entrées-sorties intégrées dans un MPC555, la solution actuelle pourrait

s'avérer très utile pour plusieurs applications industrielles tel le contrôle de robot et autres.

RECOMMANDATIONS

Dans la solution actuelle, l'acquisition de données se fait à l'aide du bloc OpMPC555TermPrint dont la fonction est d'afficher les messages d'état du système et certaines informations ponctuelles. Les envois sont faits en format texte ce qui est approprié pour ce type de fonctionnalité, mais non idéal pour faire de l'acquisition rapide. Il serait donc utile d'ajouter un mécanisme d'acquisition de donnée performant au système. Ceci pourrait être fait à l'aide d'un des ports de communication CAN fonctionnant à 1Mbits/s. Ceci nécessiterait cependant une carte CAN sur le PC qui reçoit les données. Une autre option serait d'utiliser le deuxième port série du SCI afin d'envoyer de manière périodique les données provenant du modèle. Afin de pouvoir acquisitionner à très haute vitesse, un concept de trame avec condition d'acquisition pourrait être mis en place. Ceci permettrait de sauvegarder des données à haute vitesse pendant une certaine période et de compléter le transfert au PC lorsque l'acquisition est complétée. Avec une telle fonction, il sera possible de faire de l'identification de système.

L'ajout de fonction d'entrée-sortie est un autre point qui augmentera la flexibilité du système. Les nouvelles fonctions pourraient être le convertisseur analogique numérique, les sorties numériques statiques ainsi que la possibilité d'utiliser le TPU pour générer des signaux numériques à largeur d'impulsion variable.

Un des désavantages de la solution actuelle est que l'usager doit d'abord aller dans Simulink afin de générer le code pour ensuite utiliser CodeWarrior afin de compiler et exécuter le programme sur la carte MPC555. Il serait intéressant de créer une application maîtresse qui se chargerait de tout en appuyant sur un bouton. Il a été établi que Matlab, Simulink et Real-Time Workshop peuvent être contrôlés par une application externe. Quant à CodeWarrior ce n'est pas certain, il faudra vérifier le tout.

Finalement, bien que Real-Time Workshop génère un code suffisamment compact pour le MPC555, il serait possible d'utiliser la version embarquée de RTW (Real-Time

Workshop Embedded Coder) qui contient des modèles de génération afin de réduire la taille des structures internes. L'intérêt de cette modification est dans une application industrielle de haut volume ou chaque kilo-octet de mémoire et de Flash est important. Si tel n'est pas le cas, les coûts additionnels de RTWEC ne sont sûrement pas justifiés.

ANNEXE 1

Procédure complète de génération et d'exécution d'un modèle

Procédure complète de génération et d'exécution d'un modèle

1. démarrer MATLAB et Simulink
2. créer le modèle Simulink désiré. La bibliothèque de blocs spécifiques au MPC555 est située dans la section `MPC555` de la fenêtre des bibliothèques Simulink.
3. dans la section `Simulation/Simulation Parameters` du menu Simulink, sélectionner un algorithme `Fixed Step Size` sous l'onglet `Solver` et choisir un pas de calcul dans l'item `Fixed Step Size`.
4. dans le cas où le temps de simulation (`Stop Simulation Time`) est ajusté à `inf`, le modèle ne s'arrêtera que lorsque le bloc Stop de Simulink sera appelé avec une entrée unitaire. Une méthode simple est de relier un bloc MPC555 Term Read au bloc Stop ce qui permettra l'arrêt du modèle à l'aide d'une commande provenant du terminal.
5. sous l'onglet `RealTime Workshop`, sélectionner le fichier `mpc555.tlc` pour l'item system target file.
6. démarrer la génération de code en appuyant sur `Build`. Cette étape génére le code C correspondant au modèle Simulink dans un un sous-répertoire. Le nom du sous répertoire a la forme suivante : `<Nom du modèle>_mpc555_rtw`. Il est créé par rapport au répertoire courant de MATLAB au moment de la commande `Build`. Il est donc recommendé d'être dans le répertoire du modèle à générer.
7. copier le modèle de projet Code Warrior dans le répertoire désiré.
8. ouvrir le projet dans l'environnement CodeWarrior
9. dans la section `Files`, sous la rubrique `INSERT_MODEL_SRC_HERE`, insérer tous les fichiers .c contenus dans le répertoire du code généré par RTW à l'étape du Build. Il n'est pas nécessaire d'insérer les fichiers .h puisque que ceux-ci sont implicitement ajoutés.
10. compiler et exécuter le modèle en appuyant sur le bouton `RUN` de CW. CodeWarrior compilera alors tout le code et transférera automatiquement le tout sur le MPC555.
11. à partir d'un logiciel de terminal tel HyperTerminal et d'un cable null modem reliant un port du PC et le port1 du MPC555, il est maintenant possible d'interagir avec le modèle en fonction des commandes qui ont été préalablement définies dans le modèle Simulink.

BIBLIOGRAPHIE

Altia, (2002). *Altia Design.* Site de Altia, [En ligne]. http://www.altia.com/products/design/altiadesign.html (Page consultée le 12 mai 2002)

Applied Dynamic International (2002). *ADI / Beacon for Simulink and Stateflow,* Site de ADI. [En ligne] http://www.adi.com/products_be_bss.htm (Page consultée le 18 avril 2002)

Automotive Test Report, (2002). *Group Defines Computer-Aided Automotive Test Standards.* Site de Automotive Test Report, [En ligne]. http://www.autotestreport.com/2001_Articles/01_group.htm (Page consultée le 20 avril 2002)

Axiom (2002) CME-0555 Single Board Computer , site de Axiom [En ligne]. http://www.axman.com/ Section Products/CME-0555 (Page consultée le 19 mai 2002)

Bostic, D., Chutinan, A., Cook, J., Wang, Y. (2001) *Embedded Software Analysis and Generation* [En ligne]. http://vehicle.me.berkeley.edu/mobies/papers/embedded_challenge.pdf (Page consultée le 13 avril 2003)

Chen C., Novick G., Shimano K. (2002) *risc vs. cisc*, site de Université de Stanford [En ligne]. http://cse.stanford.edu/class/sophomore-college/projects-00/risc/risccisc/index.html (Page consultée le 12 mai 2002)

Département de la justice américaine (DOJ, 2002) *Justice department reaches settlement with The MathWorks* Inc, site du département de la justice [En ligne]. http://www.usdoj.gov/atr/public/press_releases/2002/200164.htm (Page consultée le 3 novembre 2002)

dSPACE GmbH. (2002). *Target Link 1.3 Datasheet.* Site de dSpace, [En ligne]. http://www.dspaceinc.com/Download/PDFFiles/Products/TargetLink13.pdf (Page consultée le 4 mars 2002)

Etas (2002) MPC555-based Evaluation Board, site de Etas [En ligne]. http://www.etasinc.com/products/embedded_control/ES200text.html (Page consultée le 19 mai 2002)

MathWorks, (2002). *Real-Time Workshop Overview*, Site de MathWorks. [En ligne]. http://www.mathworks.com/products/rtw/description/overview.shtml (Page consultée le 20 avril 2002)

MathWorks, (2002). *Real-Time Workshop User Manual*, Site de MathWorks. [En ligne]. http://www.mathworks.com/access/helpdesk/help/toolbox/rtw/rtw_ug/intro_t3.shtml#17297 (Page consultée le 9 juillet 2002)

MathWorks, (2002). *Real-Time Workshop Embedded Coder Overview.* Site de MathWorks, [En ligne]. http://www.mathworks.com/products/rtwembedded/description/overview.shtml (Page consultée le 20 avril 2002)

MathWorks, (2001). *The MathWorks et WindRiver Systems renforcent leur partenariat.* Site de MathWorks, [En ligne]. http://www.mathworks.fr/company/pressroom/press_windriver.shtml (Page consultée le 19 avril 2002)

MathWorks, (2002). *xPC Target 1.3*. Site de MathWorks, [En ligne]. http://www.mathworks.com/products/xpctarget/description/accessing.shtml (Page consultée le 12 mai 2002)

McElroy, J. (2002). *Code generation speeds debug cycle* [En ligne]. http://www.eetimes.com/in_focus/embedded_systems/OEG20021115S0038 (Page consultée le 13 avril 2003)

Motorola, (2002) *Magneti Marelli Powertrain Selects Motorola 32-Bit Microcontrollers for Advanced Engine Management Systems*, site de Motorola [En ligne]. http://www.motorola.com/mediacenter/news/detail/0,1958,767_523_23,00.html (Page consultée le 12 mai 2002)

Motorola, (2002) *Motorola History*, site de Motorola [En ligne]. http://www.motorola.com/General/Timeline/ (Page consultée le 12 mai 2002)

National Instruments, (2003). *National Instruments Acquires MATRIXx Simulation and Control Software* [En ligne]. http://digital.ni.com/worldwide/bwcontent.nsf/web/all/EEE30C6BF638943986256CD200783703 (Page consultée le 27 mars 2003)

Noguera, J., M.Badia, R. (2002). *Dynamic Run-Time HW/SW Scheduling Techniques for reconfigurables architectures* [En ligne]. http://www.sigda.org/Archives/Proceed-

ingArchives/Codes/Codes2002/papers/2002/codes02/pdffiles/7_4.pdf (Page consultée le 13 avril 2003)

Phytec (2002) phyCORE-MPC555, site de Phytec [En ligne]. http:// www.phytec.com/mmppc555.html (Page consultée le 19 mai 2002)

ScreamingMedia, (2001). *Motorola's Line of Powerpc(TM) Microcontrollers Becoming The Standard for Automotive Applications*, site de Sun-Java [En ligne]. http:// industry.java.sun.com/javanews/stories/story2/0,1072,35336,00.html (Page consultée le 12 mai 2002)

Vaughan-Nichols, S. (1999). *Special Millennium Report, Day Two.* Site de ZDNET, [En ligne]. http://techupdate.zdnet.com/techupdate/stories/main/0,14179,2412259,00.html (Page consultée le 18 avril 2002)

White, S. (2001). *A Brief History of Computing*, site de Stephen White [En ligne]. http://ox.compsoc.net/~swhite/history/timeline-PRODUCT.html#118 (Page consultée le 12 mai 2002)

Wilson, R. (2002). *Éditorial : When the embedded system is a chip.* Site de EEDesign, [En llgne]. http://www.eedesign.com/isd/columns/OEG20020227S0049 (Page consultée le 18 avril 2002)

Wind River Systems, (2002). Autocode *for MATRIXx*. Site de MathWorks, [En ligne]. http://www.mathworks.com/products/matrixx/autocode.pdf (Page consultée le 19 avril 2002)