Université Paris-Sud
IUT de Cachan
Département GEII-1
Parcours Traitement du Signal
Année 2009

Rapport de stage

Stagiaire chez New Vision Technologies

Mise à jour de ColiMe et réalisation d'un déplacement linéaire motorisé

Robin Cordier, 2009

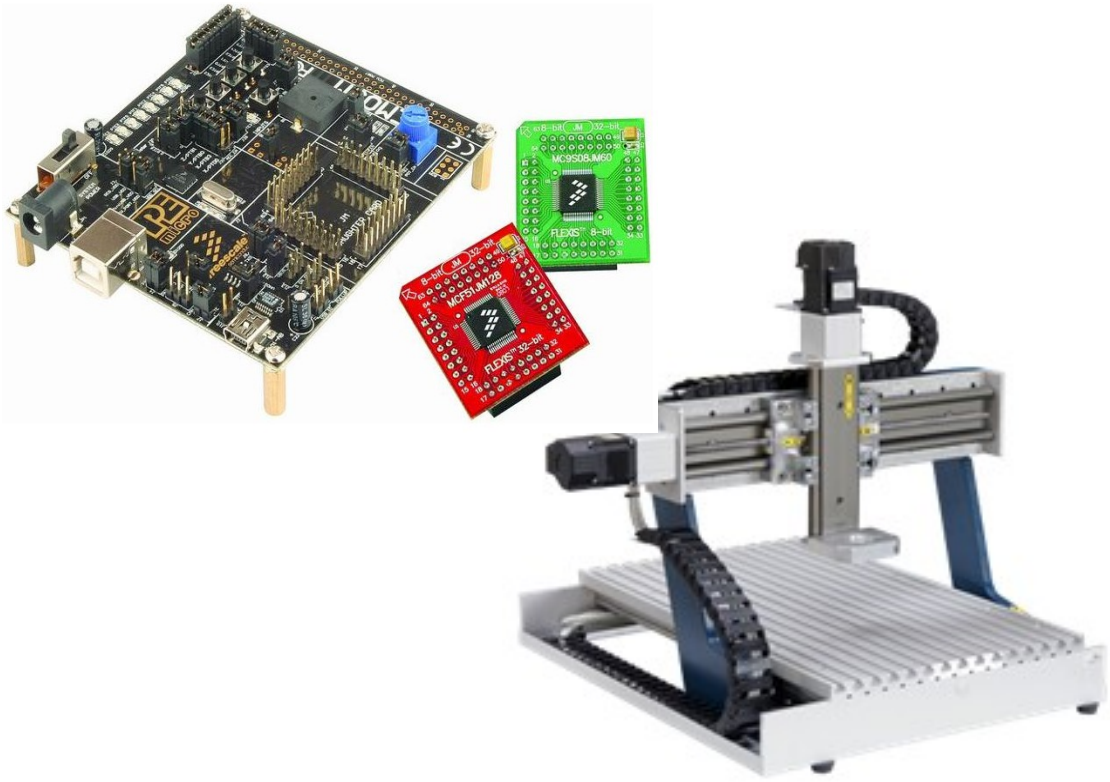

Table des matières

Index des illustrations

1. Remerciements

Je tiens à remercier toute l'équipe de New Vision Technologies, pour l'expérience qu'elle a su m'apporter durant ce stage. L'équipe m'a donné les moyens de prendre des responsabilités sur plusieurs plans et m'a accordé une totale confiance dans mon travail. Aussi, j'ai appris énormément de choses tant sur le plan technique que sur le plan professionnel, ce dont j'ai essayé de tirer le plus profit.

2. Résumé

Dans le cadre de mon DUT Génie Électrique et Informatique Industrielle, j'ai dû effectuer un stage de 2 mois en entreprise. Celle qui m'a chaleureusement accueillie est New Vision Technologies, basée à Champs sur Marne (77). Ce rapport a pour but de synthétiser travail, notes et expériences que j'ai pu accumuler durant ce stage. On y trouvera principalement une partie dédiée à la mise à jour du logiciel ColiMe sous LabVIEW, avec une explication détaillée de ce qui a été mis en œuvre. Dans un second temps, j'aborderai la réalisation d'un déplacement linéaire motorisé, en expliquant les différentes problématiques du projet ainsi que les solutions adoptées.

3. Abstract

To succeed in my Electrical Engineering and Industrial Computing two years university diploma, I had to do a placement of a few months in a company. The one who warmly welcomed me is New Vision Technologies, based in Champs sur Marne (77). The goal of this report is to synthesize work, notes and experiments that I have accumulated during this placement. In a first part, we will discuss about the update of the ColiMe software under LabVIEW with detailed explanations of what was done. In a second part, I will deal with the realization of a motorized linear translation, by explaining the various problems of the project as well as the adopted solutions.

4. Mots clés

Stage, DUT, GEII, Cachan, New Vision Technologies, LabVIEW, microcontrôleur, développement, programmation, ColiMe.

5. Introduction

Dans le cadre de mon DUT Génie Électrique et Informatique Industrielle, l'obtention du diplôme n'est effective qu'au terme de la validation de deux ans de formation continue et de 10 semaines de stage en entreprise. C'est dans ce contexte que je synthétise dans ce rapport mes notes, expériences et travaux effectués durant mon stage. L'entreprise qui m'a encadré s'appelle New Vision Technologies. Son secteur d'activité est la vision industrielle. C'est donc dans ce domaine que j'ai été amené à mettre à jour ColiMe, un de leurs logiciels, ainsi qu'à imaginer une solution de déplacement linéaire motorisé, pour répondre aux besoins internes de l'entreprise.

La réalisation de ces types de projets dans un milieu d'entreprise est une première pour moi, et au terme de ce stage, je suis amené à faire le point sur ce qu'il m'a apporté, tant d'un point de vue technique que d'un point de vue personnel. C'est dans l'optique de répondre à cette problématique que je détaillerai quelques points techniques intéressants de ce que j'ai du réaliser, ainsi que le retour de mes expériences professionnelles. Nous serons amenés à citer certains logiciels de programmation (LabVIEW, CodeWarrior), des notions de langage C et un peu d'électronique. Nous ferons volontairement un impasse sur les parties très techniques de mon stage.

Nous diviserons ce rapport en deux grandes parties, la première traitant de la mise à jour du logiciel ColiMe, tandis que la deuxième partie s'attachera à la réalisation d'un projet de déplacement linéaire motorisé. Mais avant de nous lancer dans le vif du sujet, une petite présentation de l'entreprise, dans laquelle j'ai travaillé 10 semaines, s'impose.

6. Présentation de New Vision Technologies

Du 14 avril au 19 juin, j'ai été stagiaire chez New Vision Technologies. Ce stage rentre à la fois dans le cadre de la validation de mon DUT, ainsi que de ma première expérience dans le monde du travail. L'équipe de l'entreprise se compose principalement de 5 personnes permanentes:

- Véronique Newland est la gérante de l'entreprise. Elle s'occupe principalement de toutes les relations commerciales, et prend les grandes décisions dans l'entreprise. Elle possède une formation d'ingénieur

optronique.

- Pascal Lucia est ingénieur informatique, directeur du développement technique, assure le support après-vente et s'occupe des études de faisabilité.
- Catherine Odiot est assistante administrative, elle s'occupe de l'accueil téléphonique, de tenir les comptes de l'entreprise, de passer les commandes et de la réception des clients.
- Éric Lunammachak est technicien supérieur dans le domaine de la vision et ingénieur technico-commercial. Il s'occupe des études de faisabilité et travaille en étroite collaboration avec Pascal Lucia.
- Jamel Bettahar est apprenti ingénieur dans le domaine de la vision. Il s'occupe de projets courts dans le temps, dû au fait qu'il alterne école et entreprise une semaine sur deux.
- Déborah Bourquin est une stagiaire qui s'initie au monde du travail, dans le cadre de son BTS optique.

Les locaux se situent à Champs sur Marne, en Seine et Marne (77). En illustration, un plan de situation suivi d'une photo des locaux de l'entreprise[1]:

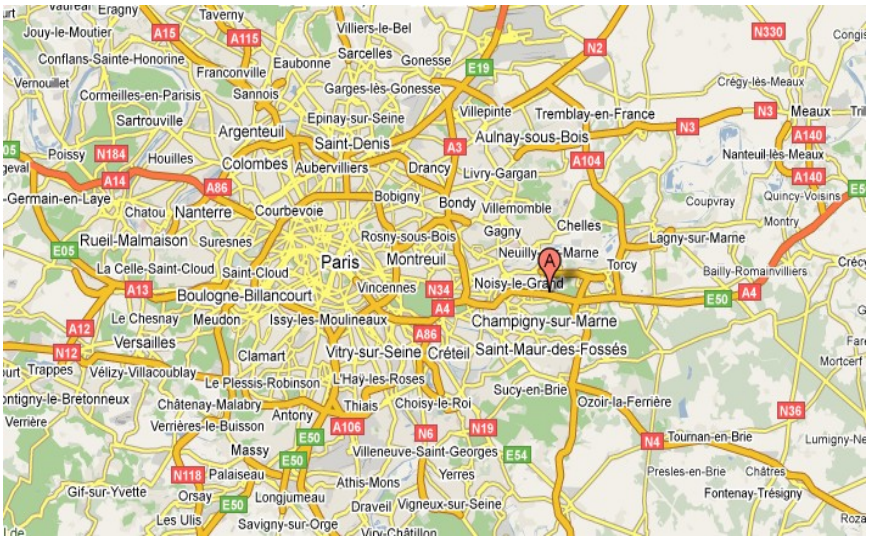

Illustration 1: Situation des locaux New Vision Technologies (Google Map)

1 Sources: http://maps.google.fr/maps?q=18+rue+albert+einstein,+champs+sur+marne

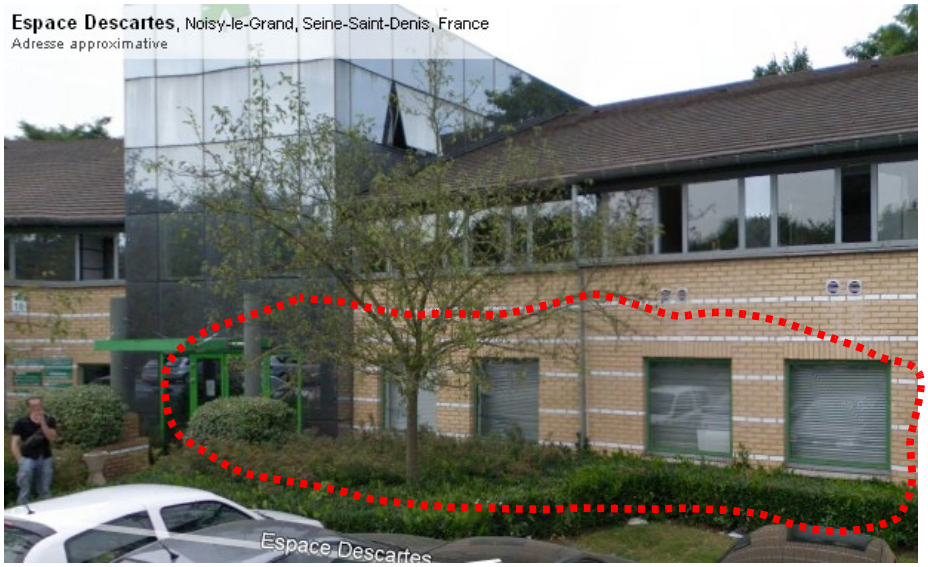

Illustration 2: Photo des locaux (Google Street View)

L'entreprise a été créée en janvier 2002, par Véronique Newland et Pascal Lucia. L'entreprise est une société à responsabilité limitée (SARL), au capital de 15000 euros. Elle se place dans le secteur de la vision industrielle. Que comprendre derrière ce terme? La vision industrielle est l'application du traitement de l'image assistée par ordinateur aux domaines industriels de production et de recherche[2]. Elle a une clientèle de 10 à 15 entreprises par an, et réalise un chiffre d'affaires inférieur à 2 millions d'euros par année.

Le domaine d'application de leur réalisation est pour le moins assez vaste: agroalimentaire, ferroviaire, métallurgie, vérification d'objets usinés… L'entreprise est d'ailleurs toujours en recherche de nouveaux clients, et chaque année elle participe à quelques salons spécialisés pour se donner une certaine visibilité: salon international de l'industrie ferroviaire (SIFER), National Instrument Weeks, salon de l'industrie… Il n'existe qu'une petite dizaine d'entreprises qui opèrent dans le même secteur d'activité, en France.

7. Mise à jour du logiciel ColiMe

ColiMe est un outil de mesure de surfaces colorées développé par New Vision Technologies. Cet outil permet de définir des plages et des nuances de couleurs sur une image (acquise via une caméra vidéo) que l'on souhaiterait mesurer. Une application simple de cette solution serait de mesurer la couleur d'une baguette après cuisson, et de décider sa mise en commercialisation ou sa destruction. En fonction de sa cuisson, la baguette présenterait des niveaux de couleurs différents. Si la surface d'une couleur représentant des zones brulées est plus importante que la surface des zones à point, alors, ColiMe vous alertera, et vous serez apte à prendre les mesures nécessaires quant à l'avenir de la baguette. Cette introduction à ColiMe, j'espère, vous aura permis de cerner globalement son utilité.

Dans le but de répondre aux nouveaux besoins de ses clients, New Vision Technologies a décidé de me confier la tâche de mettre à jour ColiMe. Ce dernier qui a été développé en 2003, sur LabVIEW 6.1, doit être mis à jour vers une version 8.6, qui supporte de nouvelles possibilités et fonctionnalités. Le but recherché est d'avoir une version plus performante, plus facile à faire évoluer et qui répond à de nouveaux besoins.

C'est dans ce sens que je vais aborder les différents points clés de la mise à jour de ColiMe. Dans un premier temps, nous serons amenés à remplacer les variables globales par des variables globales fonctionnelles, puis nous

2 Source: http://fr.wikipedia.org/wiki/Vision_industrielle

verrons comment créer une interface graphique souple et économe en ressources. La dernière partie concernera la répartition des tâches du programme dans un processeur.

Avant de rentrer dans le vif du sujet, je vous propose une petite introduction à LabVIEW, qui vous permettra d'être à l'aise avec les notions que l'on abordera plus tard.

7.1. Introduction à LabVIEW

LabVIEW est un produit de National Instrument (NI), qui est basé sur un langage de programmation graphique. Actuellement en version 8.6, ce logiciel existe depuis plus de 20 ans. La notion de langage graphique (également appelé langage G) n'étant pas forcement facile à comprendre du premier coup pour le néophyte, je vous propose une petite présentation.

Le principe se base sur des symboles graphiques (des fonctions) que l'on relie entre eux pour décrire une série d'actions séquentielle. À cela, on peut les encapsuler dans des structures qui vont nous permettre de répéter ce code un certain nombre de fois, ou jusqu'à ce qu'une condition soit satisfaite. L'analogie avec les langages séquentiels peut continuer, avec la notion de variables. Une variable permet de stocker ou de lire une valeur dans la mémoire de l'ordinateur. Les variables sont typées: un nombre entier, un nombre réel, une chaîne de caractères, un tableau, un booléen (vrai ou faux), un cluster (ensemble de variables différentes) … Comprendre ces notions de base est essentiel à la compréhension de ce qui suit. L'illustration 1 présente les différents types que nous avons précédemment mentionnés. Là où tous les fils convergent est une fonction qui consiste à regrouper plusieurs variables différentes dans un cluster.

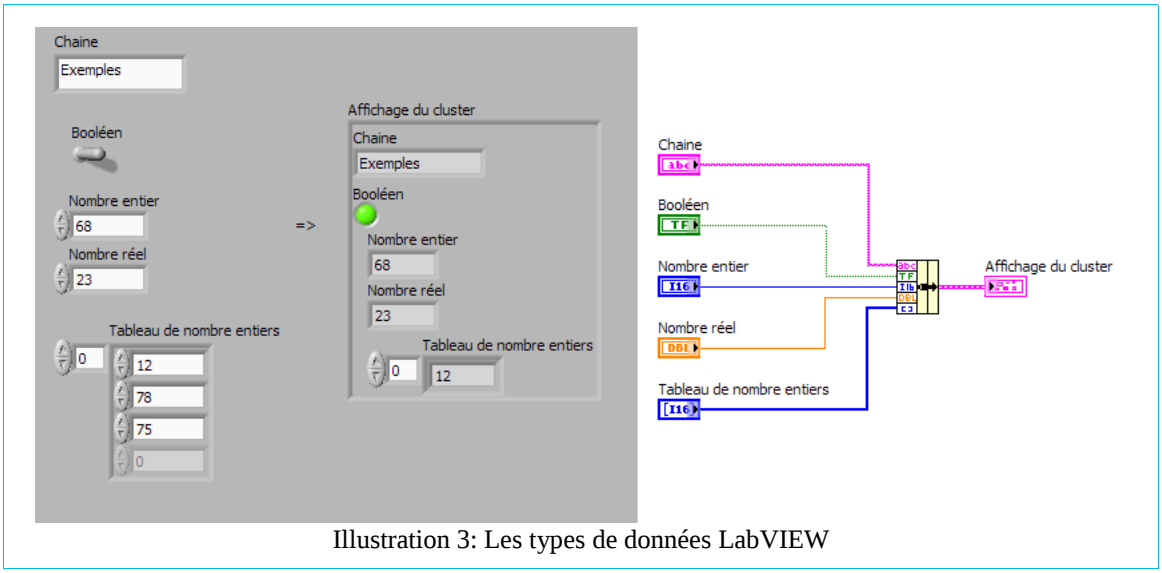

Illustration 3: Les types de données LabVIEW

Lors de la création d'un programme, LabVIEW présente deux visages: la face-avant, et le schéma. Les deux sont intimement liés. Le premier va être la partie que l'utilisateur final utilisera, sous forme d'application. Le second sera toute la partie algorithmique. Chaque bouton de la face-avant a une représentation dans le schéma sous la forme d'une variable que l'on peut lire ou écrire. Le schéma va donc utiliser les données qu'entrera l'utilisateur dans la face-avant, pour les traiter. Une fois le traitement fini, les résultats sont affichés dans la face-avant, sous une forme compréhensible par l'utilisateur. L'un étant lié à l'autre, on appelle cet ensemble fichier VI. L'illustration présente la face-avant (en haut) et le schéma (en bas).

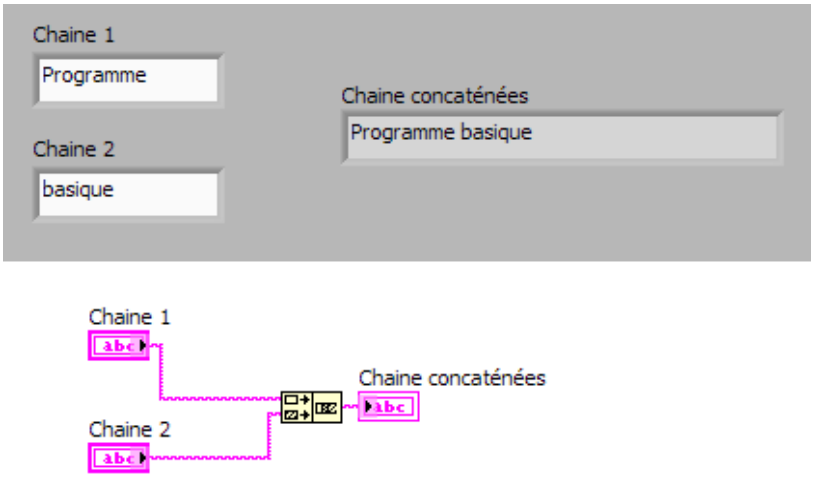

Illustration 4: Face-avant et schéma d'un VI

Maintenant que nous avons expliqué les bases de LabVIEW, nous pouvons nous intéresser aux modifications opérées sur ColiMe.

7.2. L'utilisation des Variables Globales Fonctionnelles optimise le programme

Dans quasiment tous les langages de programmation existe la notion de variable. Une variable permet d'accéder ou d'écrire une donnée dans la mémoire vive de l'ordinateur. On peut y stocker des données variées. Dans le cas de LabVIEW, on peut stoker un nombre, une chaîne, un cluster, un tableau… Il existe principalement deux variantes: les locales et les globales. Les premières ne sont accessibles qu'à un certain endroit du schéma, non dans tout le programme entier. Les secondes sont accessibles depuis n'importe quel endroit du programme, et c'est son principal intérêt. Mais, dans le cas de LabVIEW, elle a aussi de nombreux inconvénients, dont celui de rendre les schémas peu lisibles, et de ralentir l'exécution du programme. Nous allons voir en quoi les variables globales fonctionnelles (VGF) corrigent ce problème.

7.2.1. Les variables globales sont peu efficaces et mal optimisées sous LabVIEW

Les variables locales sont stockées dans un espace mémoire dédié au programme. Elles sont créées en même temps que l'ouverture du VI, et détruites à sa fermeture. Son accès en lecture et en écriture est donc très rapide. En revanche, les variables globales sont stockées en dehors de l'espace mémoire du programme. Ceci a plusieurs inconvénients: Lorsque l'on essaye de lire ou écrire dans une variable globale, LabVIEW va en faire une copie dans l'espace mémoire du programme. Une fois copié, le programme pourra effectuer une opération de lecture ou d'écriture. Dès que les opérations sont finies, LabVIEW, replace la variable globale en dehors de l'espace mémoire du programme. Donc, plus la variable est grosse, plus la quantité de données à déplacer est importante. La conséquence directe est que beaucoup d'espace mémoire est utilisé pour stocker les mêmes informations à deux endroits différents. On perd deux fois plus de mémoire, et cette perte est encore plus importante dans le cas de grosses variables. On peut noter également aussi que cette copie prend du temps.

Maintenant, imaginons que deux VIs fonctionnent en même temps, et qu'ils accèdent en même temps à cette variable globale. LabVIEW fera deux copies de la variable dans l'espace mémoire de chaque VI. Le problème est que si un VI modifie la valeur de la variable, l'autre programme n'en tiendra pas compte puisqu'il a dans sa propre mémoire une valeur qui n'est plus à jour. On peut se trouver à travailler sur des variables qui n'auront pas la valeur la plus actuelle, ou qui ne seront pas correctement enregistrées. C'est donc problématique.

On note aussi que l'équipe de développement de LabVIEW déconseille d'utiliser les variables globales,

surtout dans le cas de programmes complexes. D'après P. Lucia, cette fonctionnalité a été ajoutée « à la va-vite » dans une ancienne version du logiciel, pour répondre aux besoins des utilisateurs. Pourquoi l'équipe de développement de LabVIEW n'a-t-elle pas cherché à corriger ce problème? Tout simplement, parce que les variables globales restent pratiques dans le cas d'un tout petits programmes, et surtout parce qu'il existe une solution alternative bien plus intéressante.

7.2.2. L'utilisation des VGF est une alternative aux variables globales

Les variables globales fonctionnelles, que l'on appelle communément VGF, proposent une solution réellement intéressante aux variables globales classiques. Elles sont plus rapides à lire et à écrire, il n'y a pas de duplication dans la mémoire, et surtout elles introduisent beaucoup d'autres possibilités (ce que nous verrons plus loin). Nous allons expliquer le fonctionnement d'une VGF.

Le principe d'une VGF est qu'elle utilise ce que l'on appelle des registres à décalage non initialisés. Plutôt qu'une longue explication, prenez connaissance du schéma et de l'explication détaillée de ce qui se passe:

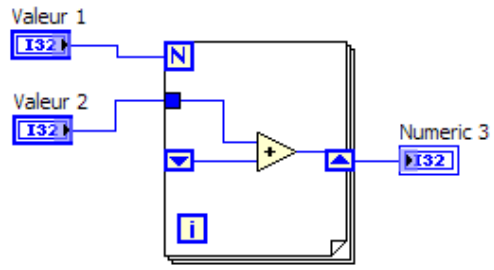

Illustration 5: Schéma d'une multiplication, avec registre à décalage non initialisé

Ce schéma effectue une opération de multiplication:

$$Valeur1 \cdot valeur2 = Numeric\ 3$$

Le registre à décalage permet de mémoriser une valeur, d'une itération à une autre. On ne les trouve donc que dans le cadre de structures *While* et *For*. Dans notre précédent exemple, la valeur est conservée, ce qui permet de faire <Valeur 1> fois la somme de <Valeur 2>. C'est donc une opération de multiplication.

On peut se demander ce qui se trouve dans un registre à décalage lors de la première itération. En réalité, il s'y trouve une valeur quelconque. Cependant, elle est initialisée automatiquement par LabVIEW lors du premier appel du code, après, elle est conservée tant que l'on n'écrit pas dedans. On utilise le même principe pour les VGF, sauf qu'il faut que l'on s'assure d'écrire quelque chose dans le registre à décalage lors du premier appel:

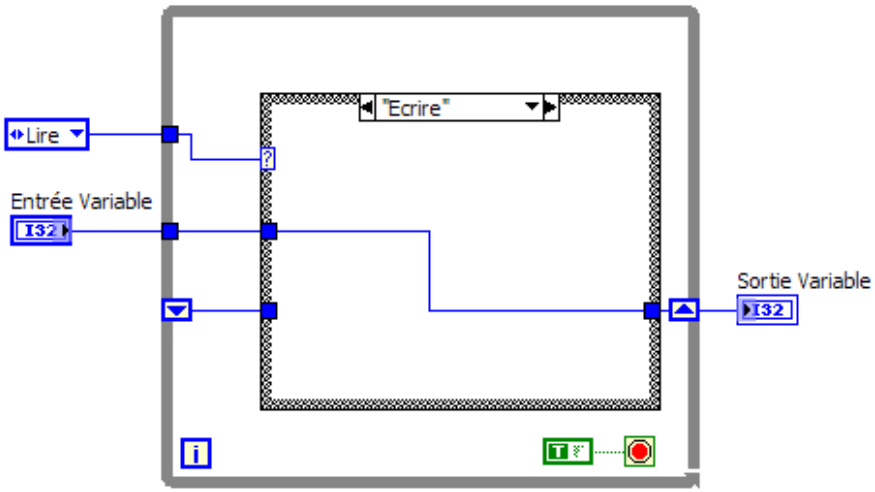

Illustration 6: Écriture dans une VGF

La valeur de 'Entrée Variable' est écrite dans le registre à décalage, si la condition est 'Ecrire'. L'ancienne valeur présente dans ce dernier est perdue. Pour lire la valeur, il suffit de placer la condition à 'Lire':

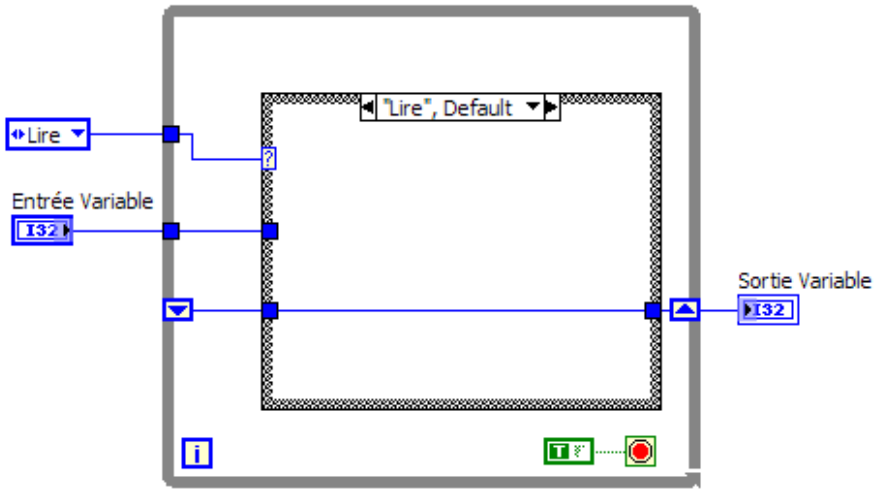

Illustration 7: Lecture d'une VGF

'Entrée Variable' n'est alors pas prise en compte, le contenu du registre à décalage est placé dans 'Sortie Variable'. On peut rappeler autant de fois la VGF, la valeur sera conservée dans le registre à décalage.

Tout à l'heure, nous avons mentionné la possibilité de faire d'autres actions, ce qui est tout à fait envisageable:

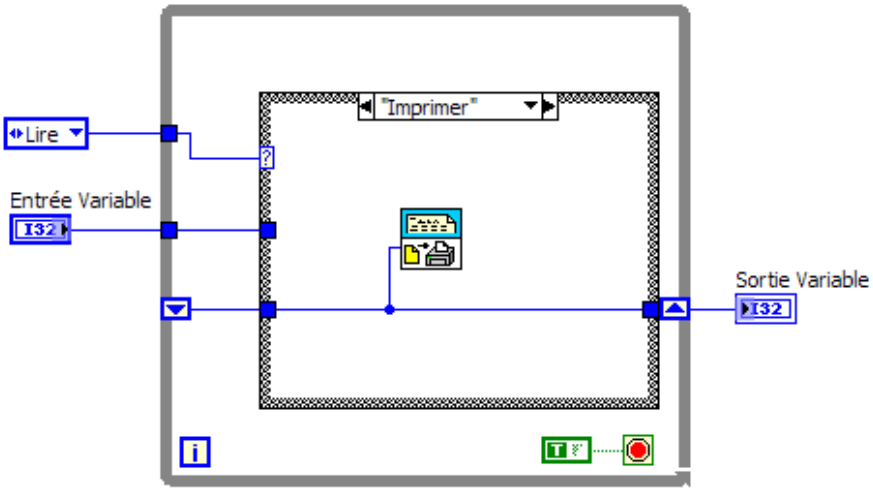

Illustration 8: Les autres possibilités des VGF

Comme nous pouvons le constater, si l'on place comme argument d'entrée 'Imprimer', le contenu du registre à décalage sera imprimé. Bien sûr, après, c'est au programmeur de choisir; il pourrait très bien faire une fonction de sauvegarde dans un fichier texte, de réinitialisation… il n'y a pas de limites.

Une fois comprise la théorie des VGF, il ne suffit plus qu'à les utiliser.

7.2.3. La mise en place des VGF apporte des améliorations sur plusieurs points

La mise en œuvre d'un VGF est montrée dans les deux exemples qui suivent. Le premier est simple, et montre comment est utilisée une VGF.

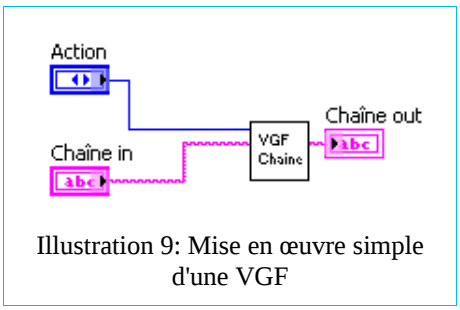

Illustration 9: Mise en œuvre simple d'une VGF

Les VGF sont certes un peu moins aisées à mettre en œuvre que les variables globales classiques. Elles nécessitent un temps d'apprentissage.

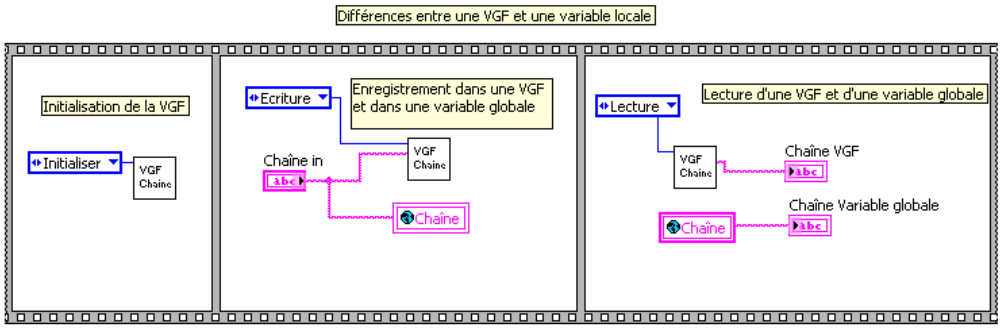

Illustration 10: Différences entre une VGF et une variable globale.

7.2.4. Les VGF optimisent le programme et amènent à plus de modularité

L'utilisation des VGF est donc très intéressante, car elles résolvent les problèmes que nous avions avec les variables globales classiques. Si nous synthétisons sous forme de points:

- La VGF s'utilise sous forme de fonction. Donc, tout ce qu'elle crée reste dans l'espace mémoire du programme. Il n'y a plus de duplication de données. C'est un gain de temps, surtout lorsque l'on utilise beaucoup de variables qui doivent être accessibles dans tout le programme.
- Le fait de pouvoir permettre d'autres actions que le simple fait de lire et d'écrire une valeur simplifie largement la tâche du programmeur.
- Il n'y a pas de possibilités de perte de données dans une VGF. On fixe le fait qu'un VI ne peut pas s'exécuter en parallèle si deux programmes différents l'appellent en même temps.
- Les VGF sont néanmoins moins faciles à mettre en œuvre, car elles nécessitent la création d'un VI, alors que ce n'est pas le cas pour les variables globales. Donc, dans le cas de petits projets, l'utilisation de variable globale peut se justifier.
- La fluidité globale du programme est sensiblement améliorée. Durant mon stage, j'ai travaillé sur un ordinateur ayant un processeur cadencé à 800 MHz, avec 512 Mo de RAM. En permanence tournaient LabVIEW, Firefox, un lecteur PDF, quelques fenêtres d'explorateur, sous Windows XP. L'impact du remplacement des variables globales par des VGF, dans ce contexte très précis, a été relativement important et appréciable. Si l'on parle en terme de temps processeur, c'est bien 5 ou 10 % d'économisé. Cependant, il est assez difficile de mesurer de manière absolue les économies réellement réalisées.
- Nous pouvons observer aussi une simplification de la partie débogage. En effet, nous pouvons savoir à tout moment quelle est la valeur de la VGF, même si le programme dans son ensemble est à l'arrêt. Concrètement, on teste un VI, puis on l'arrête. On peut accéder à la valeur directement en lançant la VGF juste après, et ainsi observer quelles modifications a opérées le VI sur la VGF.

Les VGF, par leurs nombreux avantages, optimisent donc le programme tant sur le plan du développement que sur leur fonctionnement.

7.3. Les évènements permettent de simplifier la gestion des faces-avants complexes

Comme nous l'avons mentionné dans l'introduction à LabVIEW, schéma et face-avant sont très intimes. L'importance d'avoir une interface graphique élaborée et simple d'utilisation peut amener à créer des faces avant très complexes à gérer, avec beaucoup de contrôles. En fonction de ce que fera l'utilisateur final, certains contrôles seront amenés à être désactivés, grisés, ou modifiés. Dans le cas de ColiMe, c'est un critère qu'il faut absolument prendre en compte. Nous avons tout à l'heure cité le très bon exemple du boulanger, qui veut savoir si son pain est commercialisable, ou non. Ce dernier n'est pas expert informatique, il faut qu'il puisse utiliser

simplement le logiciel. Tout est dans notre intérêt de bien travailler et de clarifier au maximum l'interface de ColiMe.

Cependant, la gestion complexe d'une interface graphique complexe peut nécessiter beaucoup de ressources au processeur, ce qui est synonyme de perte de performances. Nous allons voir comment trouver le juste compromis entre interface élaborée et faible demande de ressources, grâce aux structures évènements.

7.3.1. La surveillance par lecture périodique des contrôles est une opération couteuse en ressources

Quand ColiMe a été développé, la version de LabVIEW ne permettait pas d'autres alternatives que de lire la valeur d'un contrôle de manière constante. Le programme exécutait en permanence son code, en relisant à chaque fois les valeurs des contrôles de la face-avant, au cas où l'utilisateur aurait modifié une donnée. Il consommait donc autant de ressources quand il y a une modification que lorsqu'il n'y en a pas. Ce sont des ressources utilisées inutilement, sachant que cette boucle était exécutée toutes les 200 millisecondes. Si le code était lourd, alors, l'utilisateur avait une interface graphique peu fluide. Ce qui nuisait aux performances du programme, et à l'utilisateur. La technique consistait à placer son code dans une boucle while.

Le schéma ci-dessous nous présente la méthode ancienne, avec un exemple que nous avons déjà étudié.

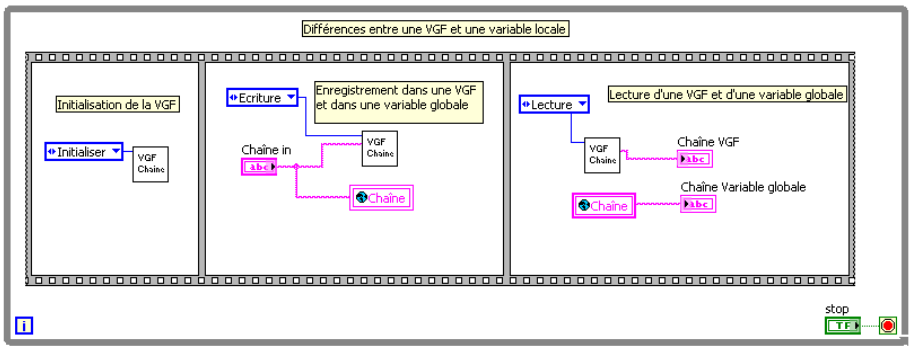

Illustration 11: Gestion d'une interface graphique sans structure évènement

Afin de prendre en compte le plus rapidement possible si la valeur de 'Chaîne in' a été modifiée, le programme va la lire de manière permanente, et traiter la valeur, même si elle n'a pas été modifiée.

À partir de LabVIEW 6, la notion d'évènement est apparue, pour résoudre ce problème.

7.3.2. La gestion des évènements permet d'éviter une relecture permanente des contrôles

Lorsque l'on crée un contrôle, LabVIEW lui associe des évènements. Ces derniers définissent un certain nombre d'actions que peut faire l'utilisateur sur un contrôle. Nous vous proposons une liste non exhaustive des plus utilisées:

- Changement de valeur
- Souris appuyée
- Souris relâchée
- Souris survolant l'objet
- Clic droit

Ces évènements peuvent être interceptés grâce à la structure événement. La structure événement a la

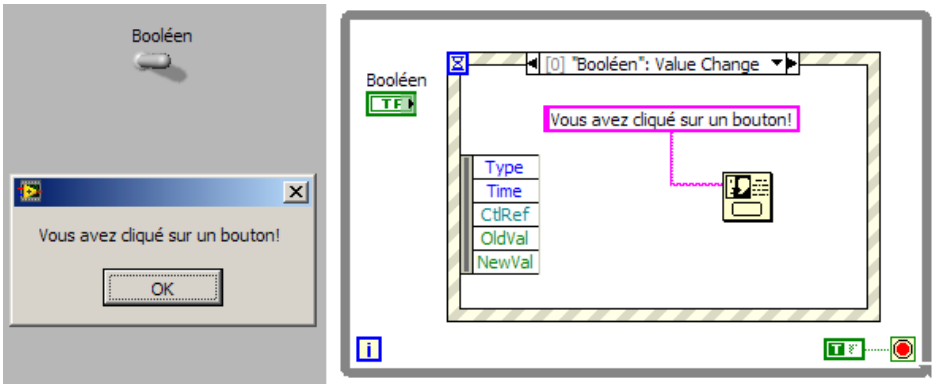

Illustration 12: Exemple d'une structure évènement

particularité d'être bloquante. C'est à dire que tant qu'elle n'a pas capté un évènement, elle se mettra en 'pause '. À ce moment, cela ne consomme plus que très peu de ressources. Dès qu'un événement est capté, alors, elle va exécuter la portion de code appropriée. Ci-dessous un exemple de structure évènement:

Cet exemple affichera le message « Vous avez cliqué sur un bouton » dés que l'utilisateur aura cliqué sur le bouton 'Booléen'. Cependant, on note tout de même la présence d'une boucle while. Sauf que la différence avec le cas précédent, c'est qu'elle ne va plus tourner à la vitesse de 200 millisecondes, mais qu'elle va rester bloquée tant qu'un événement n'a pas été capté.

7.3.3. Le passage vers des structures évènement nécessite de profonds changements dans les schémas

Dans le cas de ColiMe, il a fallu modifier tous les VIs pour qu'ils puissent gérer les évènements. En effet, la difficulté réside dans le fait que les schémas sont organisés de manière séquentielle. Si l'on peut diviser le VI par une succession d'algorithmes, nous pouvons nous apercevoir que chacun d'entre eux dépend du précédent. Il est alors dans certains cas difficile de faire le changement de manière simple, sans se plonger dans leur manière de fonctionner. Et parfois c'est impossible sans modifier une partie de ces algorithmes. C'est un gros investissement temporel, et non négligeable pour le développeur. Pour les VIs les plus complexes de ColiMe, il a fallu presque une journée de travail.

7.3.4. La notion d'évènement permet d'économiser des ressources et de simplifier la programmation

Si l'on synthétise ce que nous avons pu voir sur ce thème:

- L'utilisation de structure événement permet de rendre plus modulaire chaque partie du programme. En effet, en fonction de l'évènement généré, seule une partie du code total sera exécuté. Ce fait facilite beaucoup la maintenance et le débogage du programme, car on cible directement où est l'erreur, sans avoir besoin de tester étape par étape notre programme.
- Le programme n'exécute plus une boucle while de manière périodique, mais seulement lorsqu'un évènement est généré. C'est un gain de performances.
- Bien que mettre en place des structures événement est assez aisé, modifier un programme (qui gère son interface de manière séquentielle) vers une manière événementielle l'est beaucoup moins.
- Il est aussi possible de générer un événement sans que l'utilisateur fasse quoi que ce soit. On peut donc simuler un appui sur un bouton. On peut s'en servir pour créer des macros, comme dans Word ou Excel.

7.4. La programmation parallèle améliore les performances

Aujourd'hui, la plupart des processeurs sont multicœur. Le processeur ne possède plus une unité de calcul, mais plusieurs. Les ordinateurs grand public récents possèdent souvent deux cœurs, et dans les milieux professionnels, on trouve des processeurs à plus d'une centaine de cœurs. Concrètement, un programme peut répartir sur plusieurs cœurs différentes tâches. Malheureusement, la plupart des programmes actuels n'exploitent pas cette possibilité, et n'utilisent qu'une seule unité de calcul. Souvent, cela s'explique par la difficulté de développement avec des langages séquentiels, comme le langage C. Le système d'exploitation se contente juste de répartir les différents programmes sur différents cœurs, ou alors il répartissent de manière égale la charge des processeurs, comme le montre la capture d'écran de la charge des deux cœurs d'un processeur Intel Atom, sous Windows XP:

Contrairement aux langages séquentiels existants, LabVIEW permet très facilement de créer des programmes qui exploitent la répartition de différentes tâches sur des unités de calculs distinctes. C'est le concept de parallélisme, plusieurs tâches sont exécutées en même temps. Nous allons voir en quoi le parallélisme est particulièrement intéressant dans le cas de ColiMe, et de quelle façon nous pouvons le mettre en œuvre.

7.4.1. ColiMe n'exploite pas les capacités des processeurs multicœurs

La manière dont a été développé ColiMe est une répétition permanente d'une suite d'action. Cette manière de programmer est désormais aujourd'hui déconseillée. Comme nous l'avons expliqué dans le chapitre précédent traitant des structures évènement, la tendance actuelle est à la séparation d'un algorithme en plusieurs petites taches.

Dans le cas de ColiMe, nous retrouvons souvent une même présentation de l'interface graphique. La capture d'écran présente une ancienne version du logiciel:

Illustration 13: Interface en deux parties de l'ancienne version de ColiMe

On retrouve systématiquement une partie de commande, ainsi qu'une partie d'acquisition. Dans notre exemple, on fait l'acquisition d'une gamme de rouge à lèvres. Il se pose alors un problème: le principe de la vidéo sous LabVIEW est d'afficher de manière rapide une image de la caméra. Cela donne l'impression de vidéo[3] et de fluidité. Donc, en principe, il faut placer l'acquisition d'une image dans une boucle while. Mais, dans le cas de ColiMe, il faut aussi gérer l'interface graphique séquentielle, en plus du traitement sur chaque image de l'acquisition. Les ressources demandées sont considérables, et si elles ne suffisent pas, c'est le programme entier qui sera ralenti, et difficile à utiliser.

3 Le fait est de préciser qu'il s'agit de l'affichage successif d'images plutôt que d'un flux vidéo.

7.4.2. La programmation en parallèle est possible avec LabVIEW

Nous avons vu qu'il est possible de gérer l'interface graphique sans en grever la rapidité du programme, grâce aux structures événement. Nous avions alors dissocié interface utilisateur et traitement de l'algorithme. La solution a donc consisté à séparer le programme en différents sous-programmes. Dans notre cas, il faudrait séparer l'acquisition de l'image et son traitement.

LabVIEW permet de diviser un algorithme, et d'exécuter chacun de ses sous partis en même temps. Notez que cette technique n'a que très peu d'intérêt dans le cas d'ancien processeur (dit monocoeur) ne possédant qu'une seule unité de calcul. En revanche, LabVIEW répartira l'exécution des différentes sous-parties sur chaque cœur que dispose le processeur. Le programme pourra donc utiliser tout le potentiel de la puissance du processeur, et exécuter deux algorithmes en même temps.

7.4.3. Mise en application du parallélisme dans ColiMe

Pour mettre en place un traitement parallèle dans LabVIEW, il suffit de placer deux boucles while dans un schéma. On pourrait se poser la question de quelle boucle sera exécutée en premier? Eh bien la réponse est: aucune, car les deux seront exécutées en parallèle. Si le processeur a plusieurs cœurs, alors chaque boucle sera traitée par un cœur du processeur, et ce, de manière indépendante. Si le processeur n'a qu'un seul cœur, les boucles seront exécutées en même temps, mais cela prendra 2 fois plus de temps.

Pour tester, comprendre et vérifier le fonctionnement de la gestion du parallélisme dans LabVIEW, on crée un nouveau VI. Dans ce dernier, on affiche l'heure. On place ce code dans une boucle while. On fait une copie de cette boucle juste en dessous, comme le montre le schéma ci-dessous:

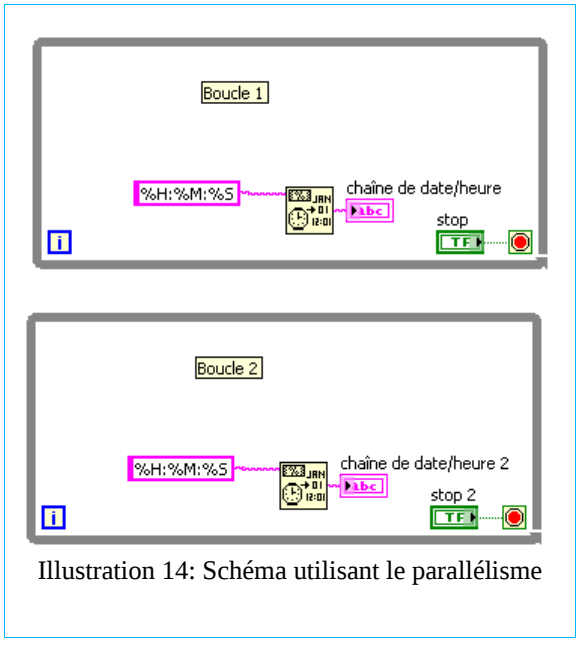

Illustration 14: Schéma utilisant le parallélisme

Si l'on observe l'utilisation des ressources durant l'exécution du programme, en arrêtant d'abord une boucle, puis la seconde grâce aux boutons stop et stop 2, on obtient les courbes suivantes:

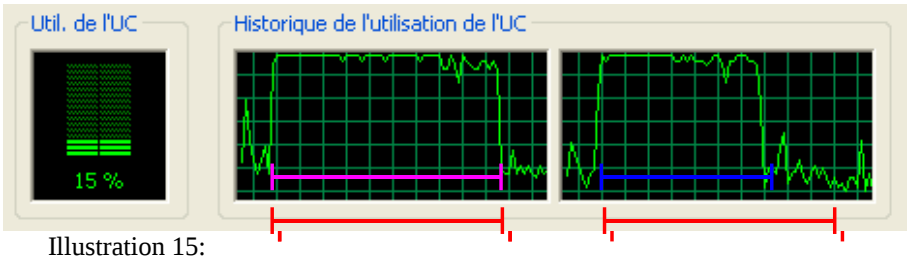

Illustration 15:
Utilisation des ressources sur chaque cœur du processeur

Le trait rouge représente la durée entre le moment où l'on a démarré le programme, et où on l'a arrêté. Le trait bleu correspond à la durée ou la boucle 1 a fonctionné. Le trait magenta représente la durée où la boucle 2 a fonctionné, son arrêt entrainant la fin du programme. Nous pouvons donc bien observer l'impact de l'exécution de ces deux boucles, sur les différents cœurs du processeur: chaque boucle est assignée à un cœur.

On appelle thread, le fait d'assigner un calcul sur un cœur. Notre programme génère donc deux threads.

7.4.4. L'exploitation de plusieurs cœurs permet une meilleure répartition des tâches

- Simplifie largement la mise à jour du programme. Si l'on veut rajouter un nouvel algorithme, le développeur a juste besoin de créer une nouvelle boucle while, et de placer son code dedans.
- S'il y a une erreur dans une boucle et qu'elle est amenée à s'arrêter, il n'y a pas d'impact sur les autres, puisqu'elles sont autonomes et dissociées.
- LabVIEW gère de manière totalement transparente pour le développeur la répartition des boucles sur chaque cœur. Un programme écrit pour un processeur à 32 cœurs pourra fonctionner sur un processeur mono-cœur. Il n'y a aucune configuration à faire. A cadence égale, un processeur multicoeur exécutera un programme deux fois plus vite gérant le multicoeur qu'un processeur monocoeur.
- La programmation parallèle à un inconvénient lorsqu'elles doivent s'échanger des données. Il faut alors mettre en place un système de pile, que l'on remplit et que l'on vide. Tandis que certaines boucles mettront des données dans une pile, d'autres boucles les liront et les traiteront. Mais la mise au point d'un tel système limite la vitesse du programme à la boucle la plus lente.
- Un autre problème que peut poser le fait d'avoir plusieurs tâches qui s'exécutent en même temps est l'impossibilité d'avoir une synchronisation entre leur exécution. Il faut alors mettre en place un système de sémaphore. On active un sémaphore signifiant qu'une opération est en cours, et on la libère dès que cette opération est terminée. Si une boucle veut faire la même opération en même temps, elle devra attendre que le sémaphore soit libéré pour exécuter l'opération. Dans le cas de ColiMe, le recours à ce procédé a été nécessaire lorsque plusieurs boucles essayaient d'accéder aux paramètres de la caméra via un port série.

8. Réalisation d'un déplacement linéaire motorisé

La seconde partie de mon stage a été consacrée à la réalisation d'un déplacement linéaire motorisé. Derrière cette appellation un petit peu barbare se cache un réel besoin de l'entreprise. En effet, la finalité de ce projet est de créer un chariot qui se déplace sur un axe. Sur ce dernier, l'équipe technique compte y poser une caméra, pour faire des acquisitions d'un objet long, ou volumineux, et de manière précise et régulière.

New Vision Technologies a décidé de me confier la conception et la réalisation de ce projet. Un cahier des

charges[4] m'a été imposé pour réaliser ce projet. C'est dans ce cadre que je vais vous présenter les différentes démarches que j'ai dû faire.

Partant de rien, j'ai dû imaginer comment réaliser ce projet de A à Z. J'ai été amené à traiter des technologies différentes que j'ai choisies, ainsi que la manière de faire communiquer un ordinateur et une carte microcontrôleur. Pour terminer, j'expliquerai certains points sur la carte DEMOJM.

8.1. Diviser le projet pour le conceptualiser

À partir d'un cahier des charges imposé, je me suis demandé comment j'allais réaliser ce projet, pour lequel je n'avais pas de cadre ou de limites, juste des contraintes à respecter. Dans ces contraintes, j'avais un coût maximal de 1000 euros, des contraintes de précisions et d'utilisation. Je vous propose un résumé du cahier des charges non exhaustif sous forme de liste:

- Budget maximal de 1000 euros
- Une vitesse de déplacement de quelques centimètres par seconde, sur une distance de 70 cm
- Une précision d'un centième de millimètre
- Un pilotage du système via LabVIEW
- Une charge de déplacement utile minimum de quelques kilogrammes
- Utiliser des éléments réutilisables, comme des profilés, des codeurs, des borniers

C'est avec à ces différentes contraintes que doit répondre la réalisation.

Il est important d'utiliser les technologies les plus adaptées aux différentes problématiques posées par le sujet. Une fois le cahier des charges défini par l'entreprise, il a fallu se pencher sur la manière de réaliser le projet. La principale difficulté est de trouver par où commencer. Imaginer comment réaliser ce projet, comment rassembler les connaissances acquises durant le DUT, les connaissances personnelles et le résultat de différentes recherches.

8.1.1. Comment conceptualiser un projet aussi vaste?

La réalisation d'un si grand projet est problématique sans expérience ou méthodes. En effet, c'est une démarche à laquelle je n'ai jamais été confronté à l'IUT. En général, quand nous devions créer un programme, ou réaliser une conception, nous avions une base: un logiciel de développement, un langage imposé, des composants imposés, une démarche proposée. L'exercice est tout de suite plus simple, sachant qu'un cadre est délimité. De plus, le projet fait intervenir un grand nombre de domaines de connaissances différents: informatique, électronique, programmation, mécanique.

J'avais déjà quelques idées pour certains points, en revanche, j'étais un peu bloqué dans d'autres. En évitant de demander trop d'aide à mon responsable de stage, j'ai commencé par une phase de recherche, pour savoir s'il existait des sites parlant de la réalisation d'un système similaire. C'est Éric qui m'a donné la première piste, en me parlant de CNC: Commande Numérique de Construction. Une recherche dans Google m'a vite indiqué quelques sites, dont un en particulier: http://www.cncloisirs.com

Sur ce site de passionnés sont abordées différentes méthodes pour commander des moteurs, créer des machines automatiques, etc. L'illustration ci-dessous vous permettra de mettre une image sur ce qu'est une machine de commande numérique de construction:

4 cf. annexes

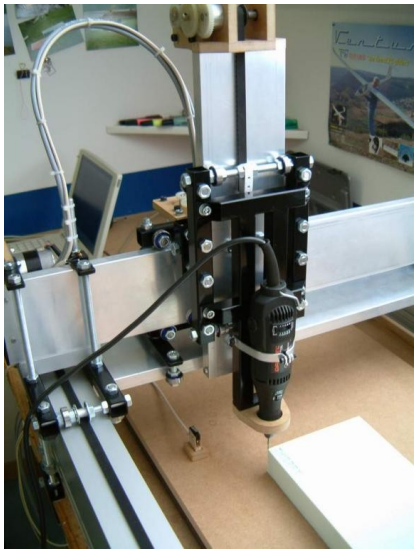

Illustration 16: Machine à Commande Numérique de Construction

CNC Loisir.net présente divers tutoriels, explications et forums afin d'aider à la création d'une machine de CNC. Cependant, il est vite apparu que ces machines étaient construites par des passionnés, car leur site n'ést pas forcement abordable au néophyte. Le site traite surtout de mécanique, d'électronique et de modélisme. Site intéressant, mais j'en ai conclu que ce n'est pas avec cette méthode que j'avancerai le plus vite.

8.1.2. Diviser, c'est régner

Après avoir bien analysé le cahier des charges, j'ai été amené à faire différentes recherches, pour clarifier certains points du sujet. Après un petit temps de réflexion, le réflexe de diviser le projet en sous projets m'est presque venu naturellement. J'ai alors rédigé un rapport complet sur ces différentes sous parties. Voici mon projet sous forme de liste:

- Mécanique: Quel moteur choisir et quel système moyen de déplacement mettre en œuvre.
- Électronique de puissance: Comment alimenter le moteur, pour qu'il soit assez puissant et rapide.
- Électronique de commande: Comment piloter le moteur.
- Connexion: Comment relier le PC à l'électronique de commande.
- Interface: Comment permettre à l'utilisateur final de pouvoir utiliser simplement notre système, via LabVIEW.

Le synoptique qui suit a pour but de synthétiser les interactions entre chaque partie:

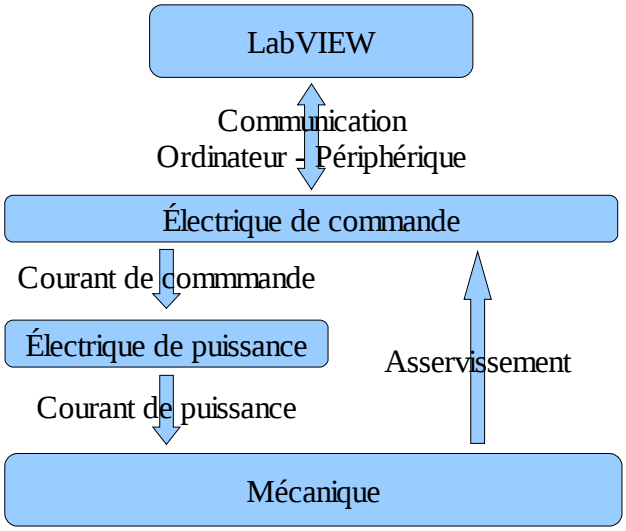

Illustration 17: Synoptique du projet

Une fois cette organisation établie, nous avons dû déterminer comment réaliser chaque sous-partie. L'association des connaissances acquises durant le DUT, du résultat de recherches sur Internet et de mon expérience personnelle m'a permis de proposer une solution pour chaque sous-partie. C'est ce que nous allons détailler, point par point:

- L'utilisateur pilote le système via LabVIEW.
- LabVIEW utilise le port série pour communiquer avec la partie commande du projet.
- La commande du moteur est une carte microcontrôleur de démonstration. Cette dernière peut à la fois générer les différents signaux logiques (sur 4, 6 ou 8 bits) pour commander le moteur, gérer une liaison série avec l'ordinateur, et assurer l'asservissement. Bien sûr, il s'occupe aussi de traiter ces signaux.
- La partie puissance convertira les signaux logiques en signaux de forte puissance, adaptés à l'alimentation du moteur.
- Le moteur est un moteur pas à pas, fixé sur un chariot à galets. Le moteur entraine une courroie fixe, ce qui permet de déplacer le chariot.
- Un codeur incrémental permet de vérifier le nombre de rotation du moteur, et envois des pulsations vers la carte microcontrôleur.

C'est à partir de l'élaboration de ce schéma que le dimensionnement pourra s'effectuer, puis aboutir sur l'achat du matériel.

8.1.3. Synthèse des choix techniques

Nous allons traiter points par point la manière dont sont réalisées les différentes parties, et déterminer quel type de produit nous conviendra le mieux. En effet, s'il faut acheter du matériel, il faut s'assurer qu'il nous conviendra. Un travail de documentation et renseignement est donc indispensable. Nous commencerons par traiter la partie mécanique (moteur, chariot et rail). C'est la base de tout le projet, si elle n'est pas ou mal dimensionnée, on ne peut pas créer la partie d'électronique de puissance, ni la partie de commande, et ainsi de suite.

1. Le moyen de déplacement est assuré grâce à un chariot à galets glissant sur un rail. L'avantage d'utiliser ce principe est, que si nous souhaitons agrandir la distance de déplacement, il suffit de racheter un autre rail plus long. Le choix du matériel s'est fait chez le constructeur/fournisseur ISEL. Un appel à leur service commercial a permis de définir quelle référence conviendrait le mieux à nos besoins. Le coût de l'ensemble rail et chariot s'élève à 230 €. La référence du matériel est 'Guidage linéaire MLF 2 '. Le schéma présente les différents éléments du chariot sur galet.

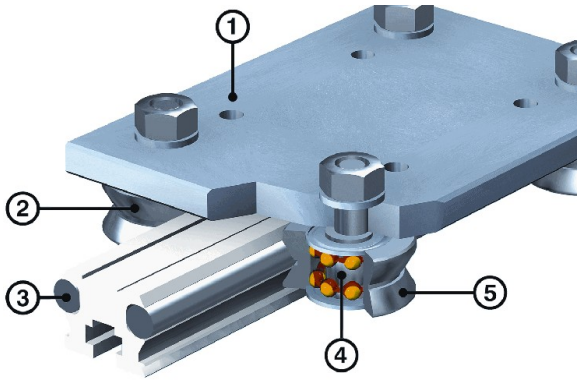

Illustration 18: Principe du chariot sur galet

1, Plateforme; 2, galet; 3,rail; 4,roulements à billes; 5,galet
Sources: www.isel.com

2. Pour le choix du moteur, toujours chez ISEL, il est de type pas à pas, et sa référence est 'MS 58 HT'.

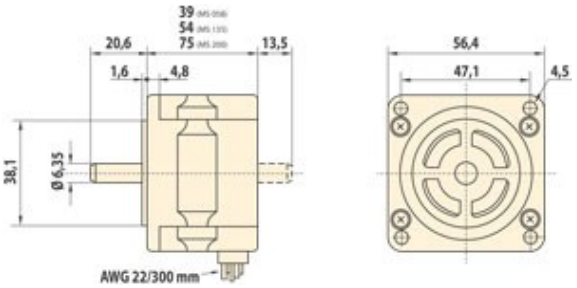

Illustration 19: Dimensions du moteur MS 58 H

Sources: www.isel.com

3. Le dimensionnement des engrenages et de la courroie a été la partie la plus difficile, au point de demander de l'aide à un camarade d'IUT en section GMP. Une fois mon problème posé, il n'a malheureusement pas pu me donner de chiffres précis pour le dimensionner. Avec l'accord de mon responsable de stage, nous avons décidé que le choix des engrenages serait empirique, à défaut d'avoir un stagiaire GMP. Les engrenages font le lien entre le rotor du moteur, la courroie et le rotor du codeur incrémental. La courroie est fixée aux extrémités du rail. Le moteur, attaché sur le chariot, va donc se servir d'elle comme d'une crémaillère. L'illustration ci-dessous montre comment s'organise tout cela (sauf le codeur).

Illustration 20: Placement du moteur sur une courroie crémaillère

Sources: cncloisir.net

4. La carte de puissance est développée « à la main ». Il existe différentes solutions, dont une appelée « pont en H » qui semble la plus intéressante.

5. La partie de commande sera assurée par une carte de démonstration microcontrôleur. A l'issue de la formation que j'ai reçue au cours de mon DUT sur de telles plateformes (F310 ou Star12), j'ai estimé que l'utilisation d'une carte microcontrôleur présentait le meilleur ratio simplicité/évolution/développement. Le choix de la carte s'est fait à l'aide de la communauté du forum officiel de Freescale, ainsi que par un mail envoyé au service commercial. La carte choisie est Flexis™ JM Demonstration Board, disponible sur le site du constructeur. Nous consacrerons plus loin un chapitre entier à l'étude de cette carte.

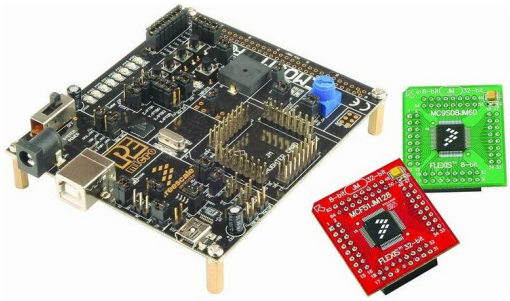

Illustration 21: Carte Flexis DEMOJM

Source: freescale.com

6. Les signaux de salves du codeur incrémental sont reliés aux entrées de la carte. C'est donc la carte que s'occupera de l'asservissement. C'est le codeur GI338 qui a était choisi, car nous en avions un en réserve.

Illustration 22: Codeur incrémental GI338

Source: ivo.fr

7. Un ordinateur standard avec LabVIEW, pour permettre une communication avec la carte microcontrôleur.

8.1.4. Retours sur les choix

À la fin de mon stage, je pouvais tirer quelques conclusions. Je n'ai pu m'occuper que de la partie carte de commande, liaison avec le PC, et interface avec le PC. Le reste n'a pas été par faute de temps, et de problèmes techniques (difficultés dans ce domaine, car en dehors de mes compétences)

La réalisation de ce projet m'a amené à me poser des questions sur certains choix technologiques à faire. Par exemple, faut-il privilégier la simplicité de développement d'une liaison via un port série entre le PC et la carte de commande, en dépit de la complexité d'utilisation pour l'utilisateur? Ou alors utiliser une liaison USB, difficile à développer, mais facile à utiliser pour l'utilisateur final. Pour la partie de commande, faut-il acheter une carte microcontrôleur, compliquée à développer, mais facilement évoluable, ou alors utiliser des interfaces de commande déjà usinées prêtes à l'emploi en dépit des limitations d'évolution du système et du prix important de ces appareils? C'est ce type de question que j'ai été amené à me poser durant la phase de conception de mon projet. Pour y répondre, il n'y a pas d'autres solutions que d'étudier les différentes technologies, et en fonction de leur caractéristique et de ce que veut le mandataire faire les bons choix.

Pour illustrer le paragraphe précédent, prenons l'exemple du moyen d'établir la communication entre le PC et la carte microcontrôleur:

- Communication directe par le protocole USB
- Utilisation d'un protocole série, du type RS232
- Utilisation d'un port série virtuel (VCP)
- Utilisation du protocole Ethernet
- Un protocole propriétaire pour commander une carte industrielle?
- Une liaison inventée à partir de rien

Ces choix détermineront quel type de matériel gérera la commande. En effet, le matériel qui s'occupera de la partie commande ne pourra pas proposer tous ces moyens de communication. D'où l'importance de bien choisir du premier coup.

8.2. Développement de programmes sur la carte Flexis DEMOJM

La Flexis™ JM Demonstration Board, aussi appelée carte DEMOJM, est une carte de démonstration. Elle permet à Freescale de présenter à couts minimes les capacités de deux de ses produits. En effet, pour 99 $, nous avons une carte avec différents accessoires dessus (potentiomètre, boutons, LED, entrées/sorties, port USB...),

deux cartes microcontrôleur amovibles, et des logiciels de développement. Les deux minicartes amovibles (appellées daughter card) proposent deux microcontrôleurs:

- MCF51JM128: ColdFire v1 32-bit, avec gestion de l'USB, sorti en 2008.
- MC09S08JM60: Microcontrôleur 8-bit avec gestion de l'USB, sorti en 2001.

Le constructeur met en avant la compatibilité broche à broche entre ces deux produits, issue de deux générations différentes. Ces deux produits proposent donc les mêmes fonctionnalités, mais des performances différentes. Dans le cadre de notre projet, pouvoir développer sur deux cibles différentes ne nous intéresse pas, nous serons amenés à toujours utiliser le ColdFire. Ce sont surtout ses fonctionnalités qui nous interressent:

- Cadence de fonctionnement à environ 40 MHz
- Gestion de l'USB, en mode hôte, périphérique ou « On the go ».
- Gestion d'un port série virtuel (VSP, pour Virtual Serial Port)
- Timers, PWM
- Rapid GPIO, qui sont des entrées/sorties capables de fonctionner à très haute fréquence
- Entrées/sorties classiques

Pour notre projet, nous pourrons utiliser à la fois les possibilités de la carte, et celle du microcontrôleur. A ce stade, nous nous demandons comment développer un programme pour cette carte de démonstration.

8.2.1. Comment exploiter cette carte de développement?

Après avoir reçu la carte microcontrôleur, nous pouvons d'après Freescale développer directement des programmes dessus. En effet, le constructeur propose plusieurs exemples: toutes les fonctions de gestion de la connexion USB sont fournies. Cependant, ces exemples ne sont pas du tout documentés, et pour les comprendre, il faut avoir une de solides notions du fonctionnement du protocole USB, les différentes méthodes de transfert, etc. Toutes ces notions sont présentes et disponibles dans la norme USB, sur le site www.usb.org. Cependant, l'apprentissage de cette norme exige de passer beaucoup de temps. Donc, je me suis vite retrouvé bloqué pour utiliser ces exemples. Nous allons voir comment j'ai fait pour contourner ce problème.

8.2.2. Plusieurs méthodes de communication via USB

Comme la création d'un programme pour la carte DEMOJM gérant le protocole USB semblait compromise, j'ai essayé de voir quelles solutions alternatives il existait pour créer une communication avec un ordinateur, mais toujours via le câble USB. J'ai trouvé deux méthodes, la première utilisant les fonctionnalités du microcontrôleur, la seconde en exploitant le travail d'autres personnes.

Le ColdFire dispose de plusieurs modules, qui ont chacun une fonction: timers, entrées/sorties, convertisseur analogique numérique (CAN), contrôleur réseau Freescale, USB, cryptographie, compteur, port série... L'étude du module SPI (Serial Peripheral Interface) semble résoudre notre problème. Ce module permet via le port USB de créer une connexion de type série. Ce type de connexion est relativement simple à mettre en œuvre, sachant que c'est un domaine qui a été largement bien étudié au cours de mon DUT. Néanmoins, la nouveauté est que cette connexion ne passe pas par un câble de type série, mais par la liaison USB. Le problème est donc un petit peu contourné. Cependant, lorsque l'on branche la carte via un câble USB sur l'ordinateur, il faut que ce dernier détecte un périphérique série, et non un périphérique USB. C'est le rôle du pilote.

Illustration23: Câble série (RS232) à gauche, câble USB à droite

Sources: http://www.stce.fr/catalog/images/CABLE-138.jpg, http://www.miwim.fr/blog/ressources/cable_usb_.jpg

Freescale ne fournissant pas de pilotes pour ce type de communication, il a fallu faire des recherches pour savoir comment résoudre ce problème. La création d'un pilote sous Windows se présente en deux parties:

- Fichier .inf: Ce fichier permet l'installation des pilotes, car il permet d'associer un périphérique grâce à un numéro d'identification unique, et la façon dont il faut l'utiliser. Créer un fichier .inf ne propose pas de problèmes particuliers, sachant qu'il existe des programmes qui le font de manière très simple (dont LabVIEW)
- Fichier .dll: Est la façon de l'utiliser. Pour communiquer, ce fichier propose des fonctions (au sens de la programmation) en langage C ou C++ pour pouvoir communiquer avec le périphérique. Il faut néanmoins créer ce fichier .dll, qui requiert des connaissances sur la manière de le créer, afin qu'il soit compatible avec les normes imposées par Windows. Pour l'USB, il existe différents pilotes ouverts (Open Source), qui permettent de modifier à sa guise le fonctionnement du périphérique. Pour utiliser un périphérique USB en périphérique série, il n'existe que deux pilotes disponibles. Un seul des deux a fonctionné dans mon cas (celui proposé par www.jungo.com)

Pour créer une communication série sur le microcontrôleur ColdFire, il faut paramétrer des registres. Pour le SPI, il y a 7 registres, certains sont accessibles juste en lecture, d'autres en lecture et écriture. Nous allons présenter ces registres, de manière succincte, pour comprendre ce mécanisme:

1. SPI0C1 (SPI Control Register 1): permet d'activer le module SPI, d'activer les interruptions logicielles, et quelques autres paramètres.
2. SPI0C2 (SPI Control Register 2): permet de paramétrer des paramètres optionnels du module SPI.
3. SPI0BR (SPI Baud Rate Register): permet de définir vitesse de transfert (baud rate) et le diviseur d'horloge (prescaler).
4. SPI0S (SPI Status Register): accessible seulement en lecture, ces bits permettent de savoir si des données sont en cours d'envois, si les données sont bien reçues, si la mémoire tampon est remplie, ou vide.
5. SPI0D (SPI Data Registers): c'est en réalité deux registres, dans lesquels on va placer les données à envoyer.
6. SPI0M (SPI Match Registers): deux registres seulement accessibles en lecture, qui permet de récupérer les informations envoyées par le PC.
7. SPI0C3 (SPI Control Register 3): active ou non les fonctions de FIFO (First Input, First Output).
8. SPI0CI (SPI Clear Interrupt Register): permet de lire l'état des drapeaux (flags) des interruptions.

Chacun de ces registres permet de configurer et/ou d'utiliser le module SPI du ColdFire. Ainsi, on peut mettre en œuvre une communication.

La seconde méthode trouvée est l'utilisation du travail que d'autres personnes ont déjà fait. En effet, il existe des programmes que certaines personnes mettent à disposition d'autrui. Le but est double, la gratuité du logiciel, et aussi profiter du fait que chacun va y ajouter une correction, ou une fonctionnalité. C'est le cas de FreeRTOS. C'est un mini OS temps réel pour microcontrôleur. Il a l'avantage de proposer toute une base pour le développement de programmes qui doivent s'exécuter très rapidement (d'où l'attribut temps réel). Il fournit plusieurs fonctions pour gérer des notions de priorité. Il faut également noter que la version pour ColdFire n'est pas disponible sur le site officiel, mais qu'elle est généreusement fournie par un membre du forum, qui s'est occupé de le faire fonctionner sur le ColdFire v1. Le principal problème de FreeRTOS est dû au fait qu'il soit multiplateforme, donc il ne propose pas d'utiliser les modules spécifiques de chaque cible. Il n'y a aucune fonction d'écrite gérant l'USB, ou quelconque autre moyen de communication.

StickOS est un autre OS pour microcontrôleur. Il n'est disponible que sur certains microcontrôleurs de Freescale, dont le nôtre. StickOS n'est pas complètement Open Source, seule une partie de son code est disponible. La forme binaire que l'on place directement sur le ColdFire permet de communiquer de manière directe avec la carte. Lorsque la carte est branchée en USB au PC, elle est détectée comme périphérique Série (grace au driver de Jungo), et on peut directement communiquer avec elle via un terminal (Hyperterminal, de Microsoft). Une fois la connexion établie, nous obtenons une interface de type ligne de commande. Il est possible de créer directement des programmes de cette manière, de les enregistrer sur la carte, et de les faire tourner en permanence. La capture d'écran ci-dessous présente l'interface avec StickOS, via Hyperterminal:

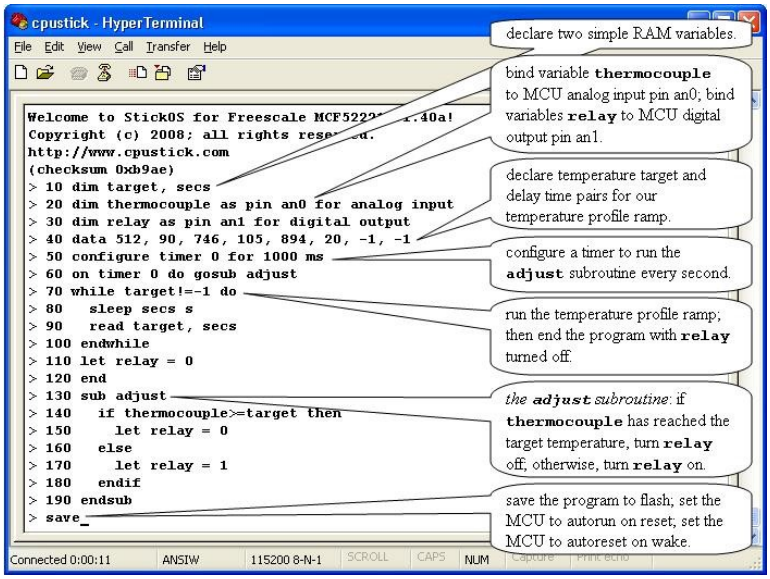

Illustration 24: « Dialogue » avec StickOS via Hyperterminal

Sources: http://www.cpustick.com/examples.htm

Le principal inconvénient, est que l'on n'a pas accès à tout le code source, ni à toutes les fonctionnalités qui nous intéressent dans le cadre de ce projet, n'étant pas dans la partie ouverte. Néanmoins, il semble tout à fait possible de créer notre programme (partie commande, réception des données du codeur et dialogue avec l'ordinateur) de cette manière. Une documentation complète est disponible sur le site de l'éditeur, expliquant la syntaxe et le fonctionnement du langage supporté.

8.2.3. Synthèse des programmes développés

Après avoir vu les différentes méthodes pour gérer une communication avec le PC, j'ai essayé de les mettre en application. J'ai commencé par la mise en place du protocole USB, sans trop de convictions. La mise en œuvre du port série virtuel a été la deuxième étape. Puis, j'ai essayé avec les deux OS que l'on a précédemment mentionnés, FreeRTOS et StickOS.

La programmation du protocole USB grâce aux exemples fournis par Freescale est très difficile pour un néophyte. Les exemples sont très compliqués à réutiliser, car il y a beaucoup de fonctions différentes, et de paramètres à effectuer, selon le type de transfert que l'on souhaite réaliser. Ajouté à cela le manque de documentations et de commentaires dans le code source des exemples, la tâche s'est vite révélée impossible.

La programmation avec la technique du VSP (Virtual Serial Port) semblait la plus prometteuse. J'ai réussi à mettre en place une communication, mais les données envoyées ne correspondaient pas aux données reçues. Surement quelques erreurs de paramétrage, mais impossible de le faire fonctionner correctement.

8.2.4. Analyse du travail effectué

La mise en place d'une communication via un protocole aussi complexe que l'USB n'est pas une chose facile pour le néophyte. J'ai essayé 3 méthodes différentes, sans obtenir de résultats réellement convenables ou exploitables. Au final, je n'ai eu qu'une communication ne fonctionnant pas bien avec via le VSP, ou alors, je devais utiliser StickOS, mais la solution ne me convenait que moyennement, sachant que je n'avais pas accès au code source.

La solution aurait surement été trouvée par l'utilisation d'une communication série classique, avec un câble RS232. En effet, c'est une technique que nous avons déjà abordée au cours du DUT.

A défaut d'avoir ce type de connexion, il peut être envisageable de tout gérer avec StickOS, et de développer dessus sur cet OS les programmes que nous souhaitions développer en langage C. Il faudra néanmoins s'assurer que cela puisse se faire.

9. Conclusion

Ce stage a donc été pour moi l'occasion de mettre en œuvre deux projets, la mise à jour de ColiMe, et la conception d'un déplacement linéaire motorisé. Nous avons abordé les points clés de chacun de ces projets. La mise en place des Variables Globales Fonctionnelles a permis de rendre ColiMe plus performant, et plus facile à développer, tandis que l'utilisation de structures événement à permit d'économiser des ressources. La programmation sur plusieurs cœurs d'un processeur a permis à ColiMe de gagner en performances. Dans le cas du second projet, déplacement linéaire motorisé, nous avons vu comment réaliser le projet dans son ensemble, ce qui a permis de séparer le projet en sous-projet, plus facile à réaliser. La découverte des différents moyens de créer une communication entre un ordinateur et un périphérique au travers d'un câble USB (sans utiliser forcement le protocole USB) a permis de poser certaines bases, propice à un éventuel collègue qui reprendrait le travail que j'ai commencé.

Néanmoins, ces deux projets étaient ambitieux, surtout en l'espace de 10 semaines. Le cahier des charges de ColiMe a été respecté, et le projet quasiment fini. En revanche, les conclusions du deuxième projet sont plutôt décevantes, car je n'ai réalisé que la partie de commande. Cela s'explique par un manque de temps certain, et des faiblesses personnelles dans certains domaines. C'est pour ce deuxième projet que j'ai réalisé un rapport d'une trentaine de pages expliquant toutes les méthodologies et résultats, afin qu'un éventuel stagiaire puisse prendre la suite de mon travail, le compléter, le rectifier, et surtout, le plus important, le réaliser!

10. Conclusion personnelle

Ce premier stage en entreprise aura été une très belle introduction au monde du travail. J'ai été amené à faire des choses que je n'aurai jamais imaginé faire! J'ai énormément appris sur LabVIEW, et certaines de ces

techniques (dont celles que j'ai abordées dans ce rapport), et ce, principalement grâce à Pascal, qui m'assistait quand je le demandais expressément. De l'autre côté du bureau, c'est Eric qui m'a conseillé pour certains choix techniques de la réalisation du déplacement linéaire. J'ai été amené aussi à présenter le fonctionnement de LabVIEW et de ColiMe à deux de mes collègues (Jamel et Déborah). Lors de la préparation au salon du SIFER, nous avons fait du modélisme: une maquette de train devait être remise en état pour présenter les solutions de l'entreprise. Cela m'a rappelé le temps ou je faisais beaucoup de modélisme. La présentation du travail que j'ai fait durant mon stage a remis en valeur certains enseignements de l'IUT, comme la préparation des PowerPoint.

La dimension sociale de l'entreprise a largement contribué au plaisir de travailler tous les jours. Les « pauses cafés », souvent proposées par Pascal, permettaient de sortir du contexte de l'entreprise pour parler de sujets divers et variés. Au début, de mon stage, j'ai eu la chance de pouvoir manger dans le restaurant d'entreprise commun, avec l'équipe de l'entreprise. C'est là que j'ai vraiment pu mieux connaître mes collègues de travail, en dehors des locaux. Je profiterai aussi de l'occasion pour me féliciter de la perte de certaines mauvaises habitudes que j'ai prises pendant mes deux années passées à Cachan, dont la fâcheuse habitude de ne pas me lever le matin (un point qui me tracassait dans bien des domaines).

A la fin du stage, les conclusions et avis ont été une bonne surprise: mon employeur m'a fait part de sa satisfaction quant à mon travail et mon comportement (mise à part une tenue vestimentaire pas toujours adaptée à l'entreprise). Pour me remercier et conclure mon stage, Véronique m'a invité un jeudi midi au restaurant. Nous avions parlé du fait que j'aurai une petite rémunération (peut-être pour m'acheter des vêtements que je pourrai mettre en entreprise) au terme de ce stage, ainsi qu'une possibilité de continuer à travailler chez eux. Déjà pour le mois de juillet, et puis si c'est encore possible, dans deux ans, en alternance après mon année à Glamorgan.

11. Annexes

- Cahier des charges 1.pdf: Premier cahier des charges fourni lors de mon arrivée dans l'entreprise, rédigé le 09/04.
- Cahier des charges 2.pdf: Mise à jour des objectifs, rédigé le 20/04.
- Étude de faisabilité de déplacement linéaire.pdf: C'est l'étude théorique que j'ai rédigée durant mon stage, pour le deuxième projet.
- Rapport de travail journalier.pdf: C'est une description du travail que j'ai faite au jour le jour, sous forme de notes.
- LabVIEW/: Dossier contenant un projet LabVIEW, contenant tous les exemples que j'ai mentionnés dans ce rapport.
- Images/: Toutes les images contenues dans ce rapport son disponible dans ce dossier.
- Datasheets/: Toutes les documentations techniques qui m'ont servi durant mon stage sont présente dans ce dossier.

12. Lexique

- Port série/ Port COM: Le port série (ou abusivement appelé port COM) est un protocole standardisé qui permet d'envoyer des données en série dans un câble.
- VI: Un VI est un fichier regroupant la face-avant et le schéma d'un programme. On peut inclure un ou plusieurs VI dans un VI. On parle alors de sous-VIs.
- Face-avant: C'est l'interface utilisateur d'un programme crée avec LabVIEW. Elle fait partie intégrante d'un VI.
- Schéma (diagramme), code: C'est la partie algorithmique d'un VI, effectuant tous les traitements.
- Mémoire: La mémoire permet d'enregistrer des données, puis de les récupérer à un moment donné.

- Boucle (while, for): C'est une portion de code qui va se répeter un certain nombre de fois, jusqu'à ce qu'une condition d'éxecution ne soit plus satisfaite. Par exemple, 'si le bouton vaut 1', alors, 'arrêter la boucle'.
- Codeur incrémental: Un codeur incrémental est un objet mécanique. Il envoi un signal logique tous les 'n' degrés de rotation de son rotor. Par exemple, un codeur incrémental enverra 1000 signaux logiques par rotation. On peut ainsi connaître le nombre de tour qu'a effectué un moteur, si l'on crée un asservissement avec un codeur incrémental.
- Octet, bit: Un octet, c'est 8 bits. Un bit est un état logique (0 ou 1) que l'on peut stocker dans une mémoire. On stocke les bits par octets.
- Registre: Un registre est une zone particulière de la mémoire, et c'est souvent un octet.
- Registre à décalage (Shift Register): C'est un registre qui va décaler ses bits de un pas. Dans le cas de LabVIEW, cela fait référence à une valeur qui va pouvoir être conservée lors de la prochaine itération d'une boucle.
- Mémoire tampon (buffer): C'est une zone mémoire temporaire, qui a besoin d'être remplie pour être vidée. Très utilisé dans le cadre d'une communication série.
- Interruptions logicielles: C'est une portion de code qui sera exécuté en priorité lors d'un évènement.
- Système d'exploitation (Operating System, OS): C'est un système qui permet l'exécution de programmes en son sein.
- Code source: Fait référence aux fichiers qui ont servi à la programmation d'un logiciel, avant que ces derniers ne soient transformés en fichier binaire (1 et 0).
- Open Source: Fait référence à une idéologie de distribuer des logiciels, en distribuant le code source, pour que quiconque puisse comprendre, étudier, améliorer et corriger des logiciels.

13. Sources et liens

- Page de garde:
  - L'image de la carte DEMOJM sur la page de garde: http://www.freescale.com/files/graphic/block_diagram/DEMOJMSKTBD.jpg
  - L'image d'une machine de CNC sur la page de garde: http://www.isel-cnc.fr/img_basic/CNC_Pure_Proto-1.png
- ColiMe:
  - Forum officiel de LabVIEW: http://forums.ni.com
  - Forum amateur d'aide sur LabVIEW: http://forums.lavag.org
  - Livres: LabVIEW Basics I et LabVIEW Basics II, LabVIEW Intermediate I et LabVIEW Intermediate II, LabVIEW Advanced
- Déplacement linéaire motorisé:
  - Site officiel de la carte DEMOJM: http://www.freescale.com/webapp/sps/site/prod_summary.jsp?code=DEMOJM
  - Forum officiel de Freescale: http://forums.freescale.com
  - FreeRTOS, pour ColdFire v1: http://www.freertos.org, http://forums.freescale.com/freescale/board/message?board.id=CFCOMM&message.id=5073
  - Site et forum amateur sur les microcontrôleurs: http://www.68hc08.net/

- StickOS: http://www.cpustick.com/
- Farnell: fournisseur de matériel divers, http://fr.farnell.com
- Images (par ordre d'apparition):
  - http://cncloisirs.com/CNCLoisirs/Accueil?action=dispimg&im=FraisTMonnot.p.jpg