Capture de Mouvement par Réseau de Capteurs Sans Fil

Nicolas ESTEVES

Juillet 2007

Résumé

A faire : description générale du stage

Table des matières

Première partie

Introduction

Chapitre 1

Contexte du stage

1.1 Présentation de l'INRIA

L'INRIA (Institut National de Recherche en Informatique et Automatique) est un établissement public à caractère scientifique et technologique qui mène des recherches avancées dans le domaine des sciences et technologies de l'information et de la communication. Ce domaine inclut l'informatique et l'automatique, mais aussi les télécommunications et le multimédia, la robotique, le traitement du signal et le calcul scientifique. Les recherches de l'INRIA sont réparties sur cinq unités (plusieurs projets sans lieu géographique sont regroupés dans l'unité INRIA Futurs) qui sont :

- INRIA Lorraine
- INRIA Rhône-Alpes
- INRIA Rennes
- INRIA Rocquencourt
- INRIA Sophia Antipolis

1.2 Unité de recherche INRIA Rhône-Alpes

L'unité de recherche INRIA Rhône-Alpes est composée d'environ 500 personnes réparties dans 26 équipes de recherches :

- 159 chercheurs et enseignants-chercheurs
- 60 ingénieurs, techniciens, administratifs
- 154 doctorants et post-doctorants dont 33
- 35 ingénieurs en développment
- 130 stagiaires accueillis régulièrement

L'unité de recherche est aussi impliquée dans des projets internationaux avec 8 équipes associées avec des laboratoires étrangers (USA, Canada, Brésil, Pays Bas, Suisse), 26 contrats de recherche européens en cours et la participation à 10 réseaux de recherche européens. Côté économique, l'INRIA Rhône-Alpes avait un budget de 6,2 Meuros en 2005 dont 3,2 Meuro en

ressources contractuelles, 271 contrats actifs dont 53 contrats de recherche directs avec l'industrie et 14 start-up ont été créées depuis 1998, dont Polyspace et Kelkoo, ayant généré plus de 560 emplois.

1.3 Le service SED

A faire : mettre a jour le paragraphe avec les infos du site sed

Le SED, service de support des expérimentations et de developpement logiciel, a une importance capitale dans le bon déroulement des recherches de l'INRIA. En effet, c'est cette équipe qui crée et développe tous les outils utiles aux expérimentations. Elle gère également la bonne utilisation de la halle robotique. Ce service apporte le support à trois plates-formes expérimentales :

- Les plates-formes Robotique et Vision : Composées d'une Halle robotique regroupant les véhicules Cycab, les robots bipèdes et les ateliers mécanique/électronique.
- La plate-forme Réalité-Virtuelle : Plate-forme de 160m basée sur un couple de supercalculateurs graphiques SGI et composée d'une salle d'immersion plein-pied (écan cylindrique), d'un plateau de prise de vue avec fond bleu et d'une salle de manipulation.
- Les plates-formes Grappes : Composées d'une grappe d'Itanium2 (i-cluster2) et d'une grappe de PC connectée sur un mur d'images interactif (GrImage).

Ce service est en charge de mon accueil à l'INRIA, plus précisément sur la plate-forme Robotique et Vision, et met à ma disposition les capteurs de mouvement et les noeuds sans fils.

Chapitre 2

La capture de mouvement

A faire : decrire capture, mettre infos Rodolphe

Chapitre 3

Déroulement du stage

3.1 Objectifs du stage

De nombreuses solutions existent déjà pour la capture de mouvement, avec des caméras ou par des capteurs filaires. Les méthodes utilisant du traitement d'images demandent un équipement important (plusieurs caméras, grande capacité de calcul, environnement spécial) et ne sont pas adaptées à l'étude du mouvement dans un but médical par exemple ou en extérieur. D'autres méthodes utilisant des capteurs reliés à un ordinateur existent déjà, mais ne sont pas utilisables simplement et ne permettent pas une grande liberté de mouvement à cause des fils reliant la personne à l'ordinateur traitant les données. C'est pourquoi l'utilisation de noeuds sans fils pour transmettre les données permet de garder la grande précision des réseaux de capteurs en évitant le problème de l'encombrement. Le but final est la mise au point d'outils permettant le déploiement d'un réseau de noeuds sans fils dédié à la capture de mouvements de manière synchronisée et fiable.

3.2 Problèmes posés

Deux problèmes majeurs se posent lors la mise en place de noeuds sans fil.

Tout d'abord les noeuds utilisés sont très limités en puissance et en mémoire. Le stockage des données sur les noeuds n'est pas possible, le transfert par radio est lent et la fréquence d'acquisition nécessaire aux études de mouvements est élevée (100-1000Hz). Il faut donc trouver une solution pour permettre une acquisition rapide et régulière.

L'autre problème qui se pose est celui de la synchronisation des données entre les noeuds. Il faut que les données soient parfaitement synchronisées pour être cohérentes, et l'utilisation d'une communication sans fil pose de nombreux problèmes. Le temps d'accès au réseau est plus long, le débit moins important et les vitesses de propagation ne sont pas constantes. Ainsi

un signal émis par un noeud n'arrivera pas forcément au même instant T à deux autres noeuds. De plus chaque noeud à sa propre horloge interne qui peut dévier par rapport aux horloges des autres noeuds et entraîner un décalage dans les mesures.

3.3 Organisation du stage

La durée du stage se divise en quatre étapes
- Découverte du matériel, du système TinyOS et des méthodes de programmation des noeuds
- Évaluation des performances du matériel
- Mise au point des solutions aux problèmes d'acquisitions et de synchronisation
- Validation des outils

Après avoir découvert les méthodes de programmation des noeuds j'ai dû étudier le fonctionnement du système TinyOS pour comprendre comment il gérait les différents composants. Ce travail a pris du temps car le système n'est documenté que pour l'utilisateur final, avec des documentations générales et quelques exemples d'utilisation des composants les plus courants. Le coeur du système n'est pas documenté, le code pratiquement pas commenté et pas toujours bien organisé. J'ai aussi étudié les caractéristiques et le fonctionnement du Micro-contrôleur Msp430 pour voir ses possibilités. Cela m'a permis de comprendre comment modifier les différents composants de TinyOS pour les adapter pour les besoins de rapidité de la capture de mouvements. La difficulté à ce moment a été la lecture de la documentation du Micro-contrôleur très complète mais surtout très technique pour moi. J'ai dû améliorer mes connaissances sur l'architecture des processeurs et apprendre à lire les diagrammes de fonctionement des différents composants. Le résultat de ces recherches est décrit dans la partie 2.

J'ai ensuite commencé les tests de performance du matériel afin de trouver les limites des composants utilisés. Puis, une fois les problèmes identifiés j'ai commencé à chercher des solutions à partir de la documentation. Les test effectués sont décrits dans la partie 3 ainsi que les modifications que j'ai apportées aux composants.

Je traiterais la résolution des problèmes de synchronisation et la mise en place du réseau de capteur durant la deuxième moitié de mon stage (juillet à septembre). Une étude bibliographique m'a permis de voir quelles solutions existent déjà et lesquelles seraient adaptées à nos besoins, je les présenterais brièvement dans la partie 4.

Chapitre 4

Présentation du matériel

Pour effectuer la capture, nous utilisons des capteurs qui communiquent leurs mesures à l'ordinateur chargé du traitement des données grâce à un réseau de noeuds de communication sans fil.

4.1 Les capteurs

Plusieurs types de capteurs peuvent être utilisés pour la reconstitution du mouvement. Le principe de ces capteurs est de mesurer un phénomène physique (accélération, impulsion électrique dans les muscles...) sous forme d'un signal analogique.

Les capteurs actuellement utilisés au SED sont des capteurs du CEA-LETI (2ème génération) comportant un accéléromètre et un magnétomètre tri-axes aui permettent d'obtenir l'orientation du capteur en trois dimensions après calibrage du magnétomètre.

La troisième génération de capteurs du CEA (système Starwatch) sont sans fils, ils incluent un module radio et une batterie ce qui les rend autonomes. L'avantage d'utiliser la solution avec un capteur relié à un noeud

FIG. 4.1 – Capteur CEA-LETI

sans fil permet de coupler plusieurs capteurs sur un même noeud.

4.2 Les noeuds

Les noeuds sont des modules conçus spécialement pour l'utilisation dans des réseaux de capteurs. Leur principal avantage est leur taille réduite, leur très faible consommation électrique et surtout leur capacité à communiquer sans fil, ce qui permet une grande liberté de mouvement par rapport aux noeuds filaires. Plusieurs produits existent sur le marché, de capacités plus ou moins équivalentes. Pour mon stage nous avons utilisés des capteurs Telos B fabriqués par Crossbow et développés par l'université de Berkeley.

Principales caractéristiques

- Micro-contrôleur MSP430 de Texas Instrument cadencé à 8MHz, avec 10Kb de RAM et 48Kb de flash
- Radio Chipcon Wireless Transceiver CC2420 2.4GHz à 250kbps respectant la norme IEEE 802.15.4
- Convertisseurs analogique-digital et contrôleur DMA intégrés
- Liaison USB
- Très faible consommation électrique
- Temps de réveil inférieur à 6us
- Un support d'expansion 16 ports
- Encombrement réduit
- Supporté par TinyOS

FIG. 4.2 – Noeud Telos B

Les noeuds utilisent le système d'exploitation TinyOS pour permettre la capture des données et leur envoi à l'ordinateur.

4.3 Présentation de TinyOS

TinyOS est un système d'exploitation open-source conçu pour des réseaux de capteurs sans-fil développé et maintenu par l'université de Berkeley et de nombreux contributeurs. Il permet de respecter les contraintes de mémoire et de puissance des noeuds en utilisant une programmation par composants, qui permet de réduire le code nécessaire à son fonctionnement. L'avantage de ce système est de permettre une programmation simple mais puissante des noeuds tout en gardant la portabilité du code pour les nombreuses plateformes supportées.

A faire : transition

Deuxième partie

Étude du matériel et de TinyOS

Chapitre 5

Le telosb

Nous allons détailler ici les différents composants matériels du telosb utilisés.

5.1 Le Micro-contrôleur Msp430

Le Msp430 contient un processeur RISC 16 bits et un système d'horloges flexible conçut spécialement pour une utilisation avec une batterie. Le système d'horloges met à disposition de l'utilisateur trois sources différentes en fonction des besoins en consommation et en fréquence :

- ACLK (Auxillary clock) pour les applications qui doivent consommer très peu et qui n'ont pas besoin de très hautes fréquences (32768Hz max)
- MCLK (Master clock) qui est utilisée par le CPU, utilisée pour les applications qui ont besoin de hautes fréquences (jusqu'a 8Mhz)
- SMCLK (Sub main clock) utilisable pour les modules périphériques

Il contient aussi deux timers internes configurables indépendemment, un module ADC 12 bits et un contrôleur DMA que j'ai dû étudier et adapter à nos besoins.

5.1.1 Timer

Le Msp430 possède deux Timer internes : Timer A et Timer B qui ont presque les mêmes caractéristiques, le B ayant quelques possibilités en plus. Nous allons voir le principe très simplifié de fonctionnement du Timer A pour comprendre comment le temps est géré par le noeud.

Le Timer A contient deux registres TAR (compteur) et TACCR0 (capture/comparaison) et est relié à une source d'horloge configurable. Plusieurs modes de comptage sont au choix : mode continu (CONTINUOUS) dans lequel le TAR est incrémenté jusqu'a sa valeur maximale 0xffff avant de générer une interruption, le mode montant (UP) dans lequel le TAR est incrémenté

jusqu'a la valeur contenue dans TACCR0 avant de générer une interruption et le mode alterné (UP/DOWN) dans lequel TAR est incrémenté jusqu'a la valeur de TACCR0 puis décrémenté jusqu'a TACCR0 avant de générer l'interruption. L'incrémentation ou la décrémentation est déclenchée par des fronts montants sur l'entrée du registre de comparaison en fonction des cycles d'horloge. Le timer possède d'autres registres TACCR1, TACCR2 (le timer B en contient 6 au total) permettant de générer des interruptions à des intervalles différents. Plusieurs flags sont modifiables pour controler le timer par exemple remettre le TAR à zéro ou désactiver les interruptions.

5.1.2 ADC12

L'ADC 12 (Analog to Digital Converter 12 bits) permet la conversion de signal analogique entrant sur un ou plusieurs ports en valeur numérique de 12 bits à un débit allant jusqu'a 200 000 conversions par seconde. Plusieurs modes de conversion sont possible : Single Channel (un seul port est lu) ou Multiple Channel (plusieurs ports sont lus à la suite), on peut configurer le nombre de conversions à effectuer avant d'émettre l'interruption de fin de conversion et le nombre de cycles d'horloges avant la prochaine série de conversions. L'horloge utilisée pour le déclenchement des conversions est comme pour les Timer au choix. L'ADC contient un tampon de 16 entrées de 12 bits permettant de convertir les données et de stocker les résultats sans avoir à passer par le CPU.

5.1.3 DMA

Le Msp430 contient un contrôleur DMA (Direct Memory Access) qui permet de transférer des données d'une addresse mémoire à une autre sans intervention du CPU. L'utilisation de la DMA à deux avantages : les transferts sont plus rapides (ils ne nécessitent que deux cycles d'horloge) et permet de ne pas occuper le CPU et donc de pouvoir le laisser dans un mode de basse consommation électrique et surtout de le laisser libre pour d'autres tâches. Le contrôleur DMA peut utiliser trois canaux de transferts indépendemment auxquels on peut assigner des priorités et qu'on peut configurer en fonction du type de transfert à effectuer. On choisit pour chaque canal la taille des données à transférer, octet (byte = 8 bits) ou mot (word = 16 bits) et le mode de transfert. Les transferts sont activés par un *trigger* au choix (fin de conversion ADC, Timer A ou B, fin de transmission USB...). Un flag permet d'activer la DMA en mode répétition qui permet de ne pas éteindre le contrôleur après chaque transfert qui évite des temps de réveils qui ralentissent les transferts.

Trois modes de transfert sont configurables

- Single : le transfert est fait un octet (ou mot) à la fois
- Block : le transfert se fait pour un bloc d'octets

– Burst-block : le transfert se fait comme pour le mode Block mais avec deux cycles d'horloges supplémentaires tous les 4 octets exécutés par le CPU.

5.2 Le module radio CC2420

5.3 La mémoire flash

Chapitre 6

Concepts du système TinyOS

Le système TinyOS est un exo noyau qui utilise une approche composants et non objet comme beaucoup d'autres systèmes. Nous verrons dans ce chapitre la différence entre un exo noyau et les noyaux plus classiques et les différences entre la programmation par composants et la programmation orientée objet. Nous verrons ensuite comment TinyOS est conçu.

6.1 Différents types de noyaux

Plusieurs types de noyaux existent correspondant à des époques et des besoins différents.

6.1.1 Les noyaux monolithiques

Dans un noyau monolithique (modulaire ou non) les pilotes matériels et les services sont intégrés dans un même bloc. L'avantage est une grande rapidité d'exécution car les services étant directement intégrés au noyau ne générent pas de coûteux appels système. Un des inconvénients de cette approche est que lorsqu'une erreur survient dans un des modules ou services du noyau elle menace la stabilité du système entier. Dans ce type de noyau, les programmes de l'espace utilisateur font appel au noyau pour accéder au matériel. C'est le premier type de noyau développé et qui est encore utilisé par les systèmes Linux, BSD et Solaris.

6.1.2 Les micro noyaux

Pour limiter le risque d'instabilité du système lors d'erreurs dans les services du noyaux, les micro noyaux séparent les services du noyau pour les placer dans l'espace utilisateur. Si une erreur survient dans un de ces services elle est limitée à ce service et ne compromet pas tout le système. Le noyau fournit uniquement les abstractions les plus basiques nécessaires. L'avantage de cette approche est d'améliorer en théorie la fiabilité du système et de permettre au programmeur de pouvoir mieux contrôler le matériel en n étant pas obligé d'utiliser des abstractions de haut niveau comme dans un noyau monolithique. En contrepartie, la séparation des services engendrent de nombreux appels systèmes et les mécanismes de communication entre les services deviennent complexes et lourds en temps de traitement, il en résulte un système aux performances diminuées. C'est le type de noyau utilisé par Mac OS X et Windows XP.

6.1.3 Les exo noyaux

Le principe des exo noyaux et de laisser la liberté au programmeur de se passer d'abstractions matérielles. Le système fournit uniquement des interfaces donnant au programmeur un accès direct au matériel sans rajouter de fonctionnalité. Des niveaux d'abstractions plus hauts sont disponibles mais le programmeur n'est pas forcé de les utiliser et peut très bien les remplacer par les siens. Ces abstractions étant fournies dans des librairies extérieures le programmeur est libre de les utiliser ou non et seules les librairies vraiment nécessaires sont utilisées, contrairement aux noyaux monolithiques.

6.1.4 Le choix des exo noyaux

Un noyau monolithique engendrerait un programme trop lourd puisqu'il intégrerait toutes les fonctionnalités du système. Un micro noyau serait plus léger en taille mais la gestion des services trop complexe et les nombreux appels système ne sont pas adaptés à des noeuds dont la puissance est limité et la vitesse d'exécution essentielle. La possibilité de n'utiliser que les librairies nécessaires offerte par les exo noyaux permet aux programmes d'être plus légers et permettent une grande modularité. L'accès direct au matériel est important puisqu'il permet au programmeur d'adapter le système à ses besoins en évitant de passer par des couches d'abstraction limitant les performances du programme. C'est pourquoi les exo noyaux sont les plus adaptés pour les noeuds sans fils.

6.2 Tâches, événements et applications

TinyOS est basé sur la gestion de tâches et d'évenements. Une tâche est un bloc d'instruction, un évenement est l'équivalent logiciel d'une interrup-

tion matérielle et a priorité sur les tâches. Chaque tâche est activée ou interrompue en fonction de l'apparition d'un évenement et TinyOS n'étant pas préemptif les tâches ne peuvent pas s'interrompre entre elles, mais peuvent l'être par un évenement.

A faire : rajouter un paragraphe sur exo kernel et un plus détaillé sur la programmation par composants

Gestion des tâches

Chaque tâche activée est mise en attente dans une file d'attente de type FIFO (First In First Out : première arrivée première sortie), lorsque la file des tâches est vide le système se met en veille en attendant le prochain évenement. Ce mécanisme de tâches a pour avantage d'empêcher une tâche d'en interrompre une autre, pouvant bloquer le système, mais il a aussi pour inconvénient de ne pas permettre une gestion en temps réel.

Pour les tâches de longue durée TinyOS possède un mécanisme permettant de *fragmenter* l'exécution d'une tâche nommé *split-phase* qui permet de ne pas bloquer le système. Ce mécanisme est utilisé dans l'initialisation de composants qui demandent du temps au démarrage, comme la radio par exemple.

Les Évenements

Lorsqu'une interruption matérielle a lieu, l'évenement correspondant reçoit un signal et prend la main de manière asynchrone, c'est à dire qu'il n'attend pas la fin de la tâche courante pour s'exécuter. Des évenements peuvent être signalés ne correspondant pas directement à une interruption matérielle. Il existe également des évenements synchronisés : ils sont mis en attente dans la liste des tâches, avec une priorité supérieure aux tâches en attente mais n'interrompent pas la tâche courante (cas de certains Timers).

Les applications

Les applications basées sur TinyOS sont formées de composants réutilisables et portables, (comme les Timers, les convertisseurs de signal ou la radio) qui sont *reliés* entre eux. Ces composants peuvent être directement liés au matériel (composant gérant les DEL par exemple) ou un regroupement de plusieurs composants de bas niveau (composants gérant les envois de données par la radio). Les composants sont implémentés en utilisant les tâches, les évenements et des commandes qui permettent de faire appel aux fonctionnalités d'autres composants auxquels ils sont liés.

Exemple d'application

Une application devant mesurer la température d'une pièce régulièrement et transmettre les données à un ordinateur utilisera plusieurs composants :

- un composant de mesure de température
- un composant qui se chargera de l'envoi des données par le port USB
- un composant de mesure du temps
- un composant de gestion des DEL pour afficher la fréquence des mesures

...

Chapitre 7

L'abstraction matérielle

TinyOS utilise une architecture d'abstraction du matériel (HAA : Hardware Abstraction Architecture) qui permet de satisfaire les besoin de portabilité et de réutilisabilité des applications et des composants tout en préservant l'efficacité et les performances de chaque plateforme. L'utilisation de ce type d'abstraction dans le développement de systèmes d'exploitation a permis de simplifier grandement le développement des applications en *cachant* le matériel aux développeurs, mais elle comporte aussi l'inconvénient de réduire les performances du système. C'est pourquoi il faut que le choix de cette architecture soit fait de manière optimale.

L'architecture d'abstraction du matériel de TinyOS est organisée en trois couches distinctes de composants Chaque couche a un but bien précis et est dépendante des interfaces fournies par les couches inférieures. Plus on se trouve haut dans les couches, moins le code utilisé dépend du matériel pour finalement arriver à des applications complétement indépendantes de la plateforme sur laquelle ils tournent, permettant leur portabilité.

7.1 Couche HPL (Hardware Presentation Layer)

Cette couche est la couche de niveau le plus bas : elle se situe juste à l'interface entre le matériel et le logiciel. Son but est de "présenter" les possibilités du matériel en mettant à disposition de la couche supérieure un ensemble de méthode qui permettent de contrôler le matériel. Chaque composant de cette couche doit fournir au minimum les méthodes suivantes :

- Des commandes d'initialisation, de démarrage et d'arrêt du composant matériel pour assurer une gestion optimale de l'énergie
- Des méthode d'accès aux registres du matériel (lecture, écriture)
- Des méthodes permettant de modifier simplement les *flags* les plus utilisés (sans avoir à modifier tout le registre qui les contient)
- Des commandes permettant d'activer ou de désactiver les différentes interruptions générées par le matériel

– La gestion des interruptions matérielles pour les transmettre à la couche supérieure

7.2 Couche HAL (Hardware Adaptation Layer)

C'est la couche centrale de l'architecture et a le rôle le plus important. Les composants de cette couche utilisent les interfaces fournies par la HPL pour construire des méthodes permettant d'utiliser au mieux les capacités de la plateforme pour lesquels ils sont créés. Par exemple un composant devant faire de la conversion de valeurs analogiques en valeurs digitales (ADC) ne sera pas le même en fonction de la plateforme sur laquelle on l'utilise. Certaines plateformes permettent la copie des valeurs converties par un contrôleur DMA qui accélère la copie. La couche HAL du convertisseur pour ces plateformes doit en tenir compte et utiliser ce contrôleur DMA afin de fournir un module ADC le plus performant possible.

7.3 Couche HIL (Hardware Interface Layer)

La couche HIL met a disposition du développeur les composants logiciels qui doivent être utilisables quelle que soit la plateforme sur laquelle s'exécute l'application. En fonction des performances des plateformes pour lesquelles ces composants sont développés il faut faire un choix parmi les possibilités offertes par chacune et "brider" certaines possibilités au niveau de la HIL pour garantir l'indépendance du composant par rapport au matériel. Dans le cas d'applications destinées uniquement à une plateforme, qui demandent un grand contrôle sur le matériel ou qui doivent être optimisé au maximum, on peut se passer de la HIL.

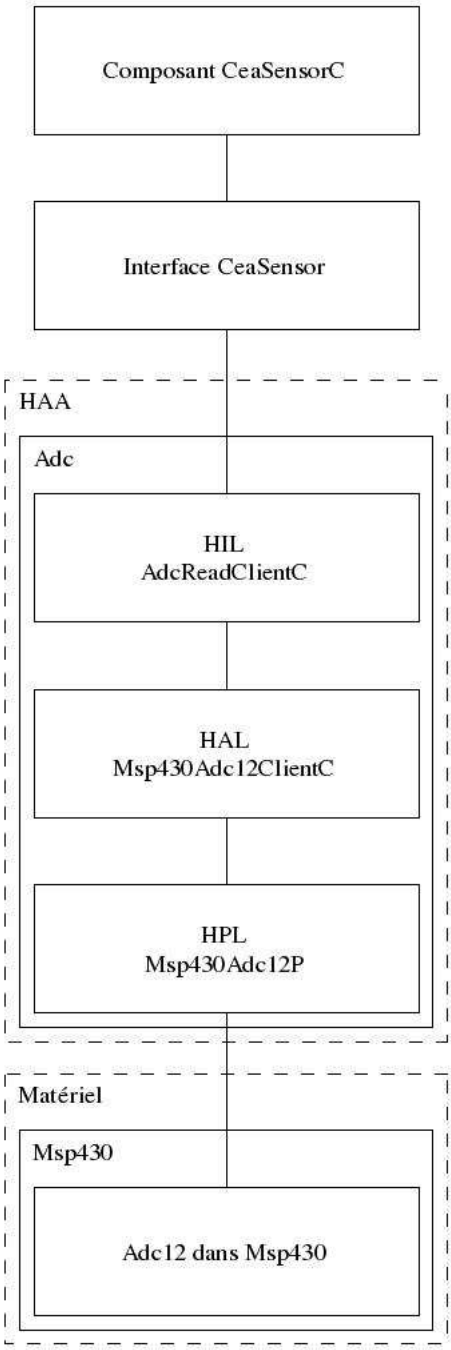

FIG. 7.1 – Exemple des différentes couches pour un composant utilisant l'ADC

Chapitre 8

Les composants utilisés

Une étude des différents composants utilisés est nécessaire avant de commencer à les utiliser ensemnoble, pour comprendre comment chacun fonctionne seul, comment l'utiliser correctement, quelles sont ses limites et comment les dépasser éventuellement.

8.1 Gestion du temps

La gestion du temps est essentielle pour permettre le bon fonctionnement de l'application. Il existe plusieurs composants et plusieurs manières de gérer le temps avec TinyOS.

8.1.1 Les composants HIL

TinyOS met à disposition différents types de composants pour gérer le temps notamment :

- Les Counter : un compteur qui calcule le temps écoulé
- Les Alarm : une extension du Counter qui signale un événement quand la valeur du Counter atteint une valeur donnée
- Les Timer : pour effectuer une tâche périodique on utilise un Timer qui signale un événement à chaque répétition

Grâce à un système de virtualisation TinyOS permet de créer 256 instances de Timer et fournit des méthodes de conversion entre Alarm, Counter et Timer. Les Timer se distinguent des autres par le fait qu'ils soient synchronisés, c'est à dire que les événements signalés par un Timer n'interrompent pas la tâche courante mais attend qu'elle se termine pour prendre la main.

De plus chaque instance de composant est configuré avec un paramètre de taille précisant la taille maximale du délai (16 ou 32 bits) et avec un précision correspondant au nombre d'intervalles de temps contenus dans une seconde (puissances de 2 donc pas multiples de 10) :

- TMicro : 1 seconde est divisée en 1048576 unités de temps

– T32Khz : 1 seconde est divisée en 32768 unités de temps
– TMilli : 1 seconde est divisée en 1024 unités de temps

8.1.2 Les abstractions

La couche HPL est implémentée dans le composant Msp430TimerP qui permet de contrôler un timer interne (A ou B). Au niveau de la HAL le composant Msp430Timer32KhzMapC fournit six interfaces Msp430Timer correspondant aux six registres de contrôle du timer B. Un autre composant Msp430Timer32KhzC permet d'allouer dynamiquement à la compilation ces six interfaces en fonction des besoins de l'application.

8.1.3 Les limites

Les Counter sont les plus adaptés pour les hautes fréquences puisqu'ils sont prévus pour les trois types d'intervalles mais ne permettent pas de définir une périodicité puisqu'ils signalent un événement lorsque le registre du processeur à un débordement mais ne permet pas de définir la taille du registre.

Les Alarm permettent de définir un temps avant d'émettre le signal mais ne permettent pas la répétition de l'événement. L'avantage des Counter et des Alarm est qu'ils ne sont pas synchronisés et permettent de recevoir l'événement directement.

Le composant Timer est le plus simple à utiliser et permet d'exécuter les tâches d'échantillonage à intervalles réguliers. Il est cependant limité et ne permet pas de supporter les hautes fréquences nécessaires à un bon échantillonage (1000Hz). En théorie il est prévu pour fonctionner pour les trois types d'intervalles (TMicro, T32Khz et TMilli) mais en pratique seul le TMilli est supporté car du fait de son fonctionnement synchronisé, même si le compteur interne du processeur émet une interruption elle sera mise dans la file d'attente des tâches et ne sera pas traitée en temps réel. Ce mécanisme ne marche bien que pour des délais de l'ordre de quelques millisecondes ou de la milliseconde si aucune tâche n'est en attente.

8.1.4 Les améliorations

Pour obtenir les performances voulues j'ai dû compléter la couche HPL de gestion des timer matériels (Msp430TimerP) qui ne permettait pas de contrôler tous les registres directement. Une fois la couche HPL complétée j'ai pu créer dans la couche HAL un nouveau composant qui permet de signaler un événement de manière asynchronisée périodiquement basée sur le timer A du Msp430 configuré en mode UP et en mettant le délai choisi dans le registre TACCR0. Comme ce composant utilise une configuration des registres qui est spécifique au Msp430, et donc qui ne peut pas être

garantie sur chaque plateforme supportée par TinyOS, l'application l'utilise directement sans passer par la couche HIL.

8.2 Lecture du capteur

Le capteur transmet six mesures au noeud auquel il est relié : les mesures sur les trois axes pour l'accéléromètre et le magnétomètre. Les mesures sont des signaux analogiques qui arrivent séparement sur six ports du noeud et le programme du noeud les lit périodiquement. Pour lire un signal analogique sur un port et le traiter il faut le convertir en signal digital. Pour cela on configure le port de lecture pour que sa valeur soit convertie l'ADC. On peut aussi lire un signal numérique directement sans passer par l'ADC en utilisant le même port configuré différemment.

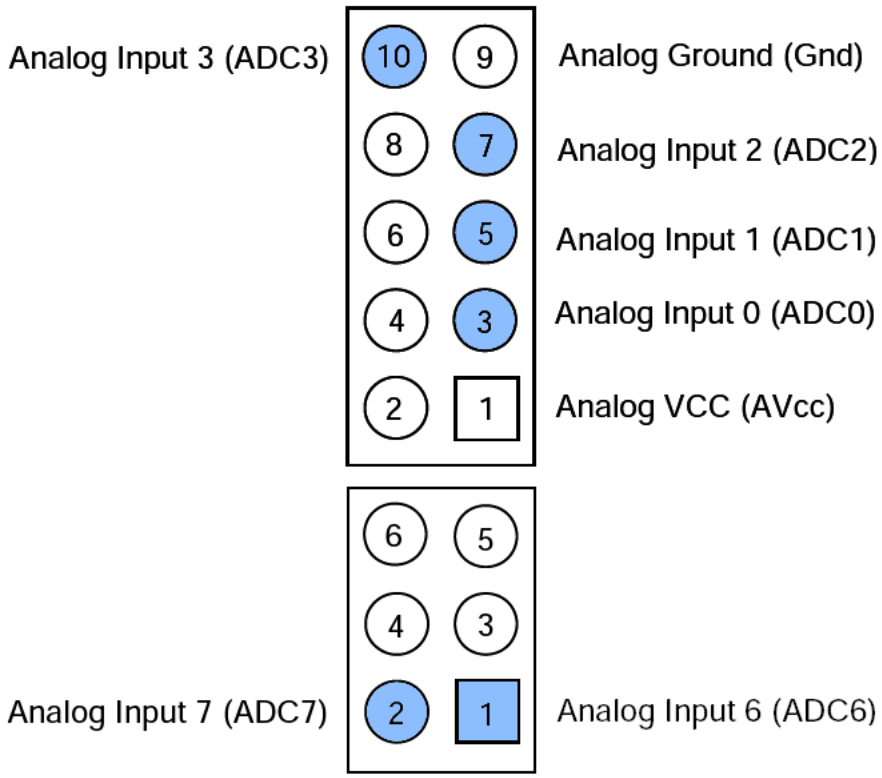

FIG. 8.1 – Les six ports du connecteur du Telos B reliés à l'ADC

8.2.1 Les composants HIL

Trois composants sont disponibles pour la lecture des ports configurés en ADC :

- AdcReadClientC : pour lire des valeurs sur un seul port (synchronisé)
- AdcReadNowClientC : pour lire des valeurs sur un seul port (non synchronisé)
- AdcReadStreamClientC : pour lire des valeurs de plusieurs ports (synchronisé)

Ces composants ne profitent pas de tous les avantages de l'ADC du Msp430, puisqu'ils appartiennent à la HIL, c'est pourquoi il est intéressant de s'intéresser à la couche HAL qui permet d'utiliser certaines fonctionnalités du Msp430 qui améliorent grandement la vitesse d'acquisition et sacrifiant la portabilité de l'application.

8.2.2 Les composants HAL

Le contrôleur DMA du Msp430 qui permet la copie de données entre les registres de l'ADC et la RAM sans passer par le processeur, on peut donc copier directement avec le matériel les résultats dans la mémoire sans avoir à passer par le logiciel. Le transfert par le contrôleur DMA est déclenché par l'interruption de fin de conversion de l'ADC donc tout se fait sans intervention de l'application et évite de mettre des tâches en attente qui ralentiraient la capture. Le processeur contient aussi un générateur de voltage de référence qui peut être utilisé par l'ADC pour se calibrer et ainsi améliorer la précision de la conversion. Ces deux fonctionnalités sont implémentées dans les composants de la HAL.

- Msp430Adc12ClientC : Utilise l'ADC sans utiliser la DMA et le voltage de référence
- Msp430Adc12ClientAutoRVGC : Utilise l'ADC avec le voltage de référence
- Msp430Adc12ClientAutoDMAC : Utilise l'ADC avec la DMA
- Msp430Adc12ClientAutoDMA_RVGC : Utilise l'ADC avec la DMA et le voltage de référence

Dans la version actuelle de TinyOS on ne peut pas configurer l'ADC pour la lecture de plusieurs ports de manière répétée et transfert par la DMA. J'ai donc dû compléter les composant de la HAL pour supporter plusieurs ports avec la DMA, sans répétition pour l'instant mais qui est en développement dans la version non stable de TinyOS.

8.3 Stockage dans La mémoire flash

A faire : transition

Troisième partie

Tests de performance

A faire : changer disposition pour éviter page presque vide

Cette partie présente les résultats des différents tests effectués sur les composants TinyOS. Le but de ces tests est d'évaluer le temps mis par les composants pour effectuer certaines tâches et pour voir lesquelles posaient un problème pour les acquisitions à hautes fréquence. Les tests sont regroupés en deux catégories qui correspondent à deux méthodes d'évaluation différentes :

- Les tests effectués avec le MilliTimer TinyOS
- Les tests effectués avec un oscilloscope

Chapitre 9

Tests avec le composant MilliTimer

Les test effectués avec le MilliTimer TinyOS sont les premiers que j'ai effectués, ils correspondent à une première manière simple mais limitée d'évaluer le temps mis par une certaine tâche. L'application de test utilise un MilliTimer qui fait deux mesures de temps, une avant le début de la tâche et une après, la différence donnant le temps mis par la tâche. La première mesure est prise juste avant l'appel de la fonction qui lance la tâche et la deuxième juste après si la tâche est courte ou alors lors de l'événement de fin de tâche dans le cas d'une tâche longue en split-phase. Les mesures sont effectuées un grand nombre de fois (plusieurs milliers) pour effectuer une analyse statistique des données. Chaque mesure est envoyé par le port USB à l'ordinateur qui les collecte.

9.1 Temps d'émission d'un paquet par la radio

L'émission d'un paquet par le module radio est une opération lente car l'accès à la couche MAC est long. Le processus d'envoi est en split-phase, une fois le paquet envoyé la radio émet une interruption signalée à l'application par un événement qui effectue la deuxième mesure de temps. Pendant le temps de l'envoi le processeur est libéré mais la radio ne peut émettre qu'un paquet à la fois. L'acquisition à haute fréquence pose alors un problème si le temps d'envoi est supérieur à la période souhaitée.

Description du test

J'ai effectué deux tests avec deux tailles de paquets différentes, le premier avec des paquets de 4 octets et le deuxième avec des paquets de 28 octets qui est la taille maximale d'un paquet supporté par la radio.

Résultats

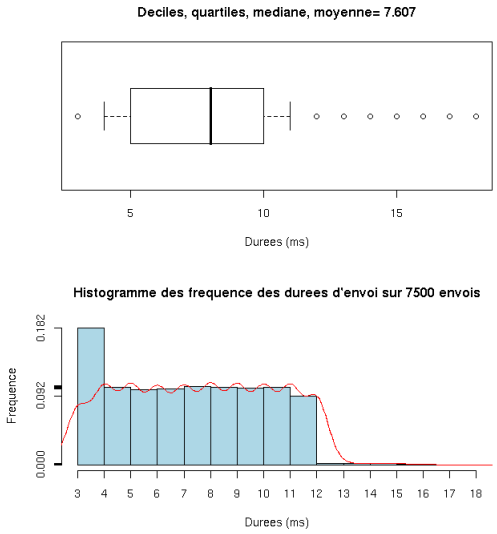

FIG. 9.1 – Test avec des paquets de 4 octets

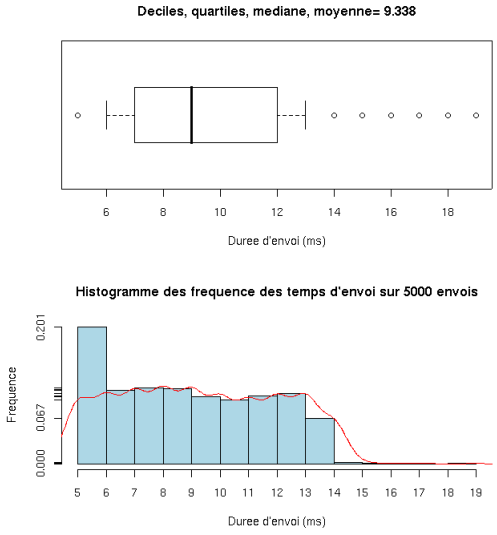

FIG. 9.2 – Test avec des paquets de 28 octets

Analyse et conclusion

On voit que les temps moyens d'émission sont trop élevés : entre 7 et 9.5 millisecondes, qui est largement supérieur à un temps nécessaire d'au plus une milliseconde. Ces temps correspondent à l'émission d'un paquet sans aucune acquisition ni autre traitement. Le transfert individuel de chaque mesure n'est pas envisageable, d'autant que les réception de tous les noeuds de mesure se font par un même noeud relié à l'ordinateur de collecte et dont le temps de réception d'un paquet est du même ordre que le temps d'émission. Une solution envisageable est de stocker plusieurs mesures dans un tampon et de les envoyer périodiquement ou de les stocker dans la mémoire flash du noeud et de les récupérer à la fin de la capture. On constate aussi que les temps d'émission ne sont pas constants et dispersés, il faut donc trouver la cause de ces différences pour essayer de les réduire.

9.2 Temps de trajet d'un paquet entre deux noeuds

Le temps de trajet entre deux noeuds est important, puisqu'ils correspond au décalage entre le moment où le noeud de liaison envoie le signal de début de la capture et le moment où le noeud de capture commence réelement les mesures (en négligeant le temps d'envoi et de réception du signal). Il faut donc savoir si ce temps est constant dans le temps et s'il est le même pour deux noeuds à des distances différentes. Il est aussi intéressant de regarder le temps de trajet en fonction de la taille des paquets pour optimiser la quantité de données à envoyer dans le cas d'une sauvegarde des mesures dans un tampon.

Description du test

Pour ces tests j'ai utilisé deux noeuds : le premier, relié à l'ordinateur qui collecte les résultats, envoie un paquet au deuxième qui le reçoit et le renvoie au premier. Je calcule le temps mis entre le début de l'envoi du paquet et la réception de la copie sur le premier noeud et l'envoie à l'ordinateur qui stocke les mesures. J'ai fait ce test pour différentes tailles de paquets (4, 8, 12, 16, 20, 24 et 28 octets) et 7500 paquets émis.

J'ai également cherché à savoir si la distance entre les noeuds avait une influence sur le temps de trajet, mais la vitesse de propagation du signal est telle que la différence de temps n'est pas mesurable avec ce dispositif.

Résultats

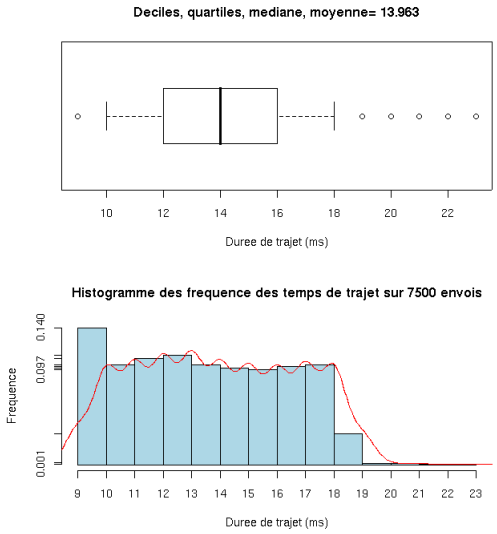

FIG. 9.3 – Test avec des paquets de 28 octets

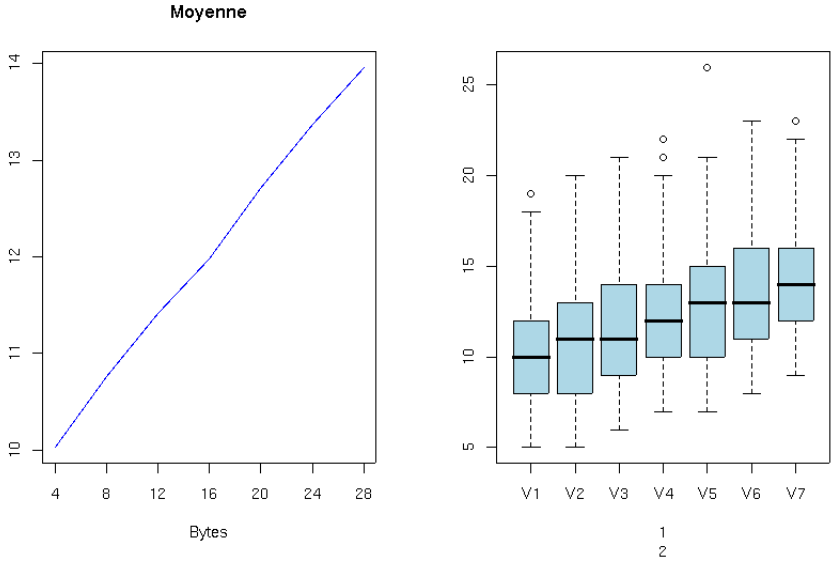

FIG. 9.4 – Comparaison des temps de trajet en fonction de la taille des paquets

Analyse et conclusion

On remarque que le temps de trajet moyen est linéaire en fonction de la taille des paquets mais qu'il n'est pas constant avec des grandes variations (entre 9 et 18ms pour 28 octets). Une première recherche bibliographique m'a permis de trouver des algorithmes conçuts spécialement pour compenser ces retards. J'étudierai leur efficacité lors de leur mise en place.

9.3 Temps de lecture d'un Adc

Le temps de lecture d'un Adc est normalement très court, de l'ordre de quelques microsecondes. Ce test a pour but de vérifier que les différentes abstractions de TinyOS ne ralentissent pas trop la lecture de l'Adc.

Description du test

Pour ce test j'ai effectué des mesures d'un des ports du noeud par l'Adc un grand nombre de fois en utilisant le composant AdcReadClientC.

Résultats et analyse

Les tests ont donné un temps moyen de lecture de 7ms, ce qui est loin des performances attendues. Cette différence est dûe à l'utilisation du composant AdcReadClientC qui est synchronisé, donc qui attend la fin de la tâche courante pour effectuer la conversion. De plus ce composant appartenant à la HIL ne profite pas du contrôleur DMA, la copie des données est donc faite par le CPU. Ce test met en évidence le problème de l'utilisation de ces composants de la HIL.

Il faudra utiliser un composant permettant la conversion de plusieurs canaux en répétition avec transfert des données par la DMA. Un tel composant n'existant pas dans la HAL de l'Adc j'ai dû en concevoir un qui permet la conversion de plusieurs canaux avec transfert par la DMA mais sans utiliser le mode répétition de l'ADC. Après en avoir discuté avec les développeur par la mailing list TinyOS, le composant a été rajouté dans la liste des choses à faire et au bout de deux semaines le développeur responsable de cette branche de TinyOS l'a ajouté à la version CVS de TinyOS.

9.4 Limite de ces tests

Les résultats de ces tests ne sont pas suffisemment précis et fiables pour qu'ils soient considérés comme valables et j'ai dû mettre en place une autre méthode de mesure plus fiable et précise pour les vérifier. Ils ne sont pas fiables car ils reposent sur l'utilisation du MilliTimer qui est synchronisé

donc l'écart entre les mesures peut être dû au retard occasionné par la mise en attente des requêtes du Timer dans la file des tâches.

Chapitre 10

Tests avec l'oscilloscope

La deuxième méthode est basé sur la mesure des temps par un oscilloscope et plus par un composant logiciel. Le principe est d'émettre sur une des sorties numérique du Telos B un signal en créneau dont la mesure de la période nous donnera le temps mis par la tâche à mesurer. On envoie sur la sortie un signal de 3V avant l'appel de la fonction à évaluer et un signal de 0V quand la tâche est terminée. On visualise sur l'écran de l'oscilloscope la période du signal et on en déduit la durée de la tâche.

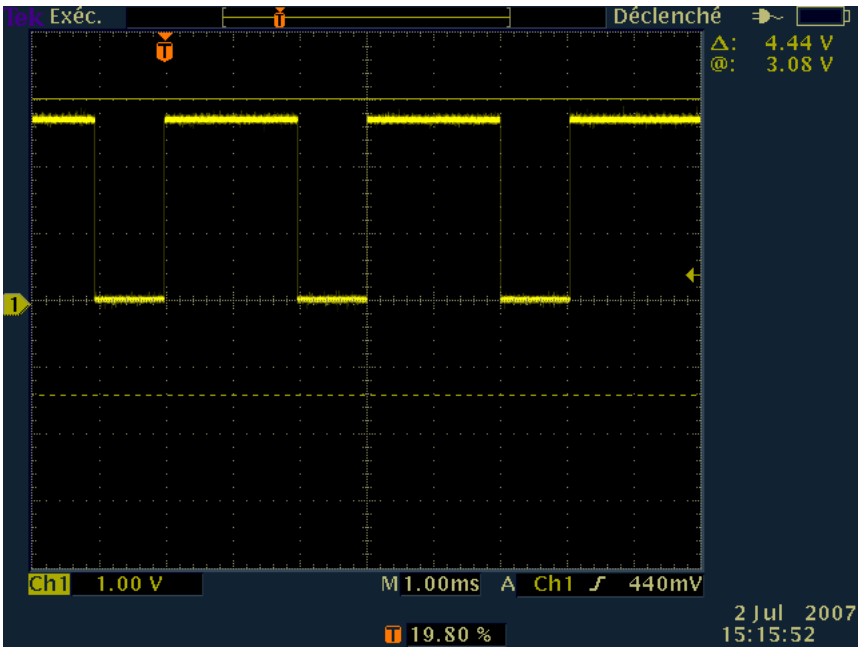

FIG. 10.1 – Mesure de temps avec l'oscilloscope, ici la tâche dure 2ms et deux tâches sont espacées de 1ms

Une autre manière de mesurer les temps avec un oscilloscope est d'utiliser

un générateur de fréquences relié au noeud, le signal du générateur est lu avant le début de la tâche et il est écrit sur un autre port du noeud. En visualisant la différence de temps entre le signal émis par le générateur et celui émis par le noeud on déduit le temps écoulé. Le temps de lecture et d'écriture sur les ports du noeuds sont négligeables.

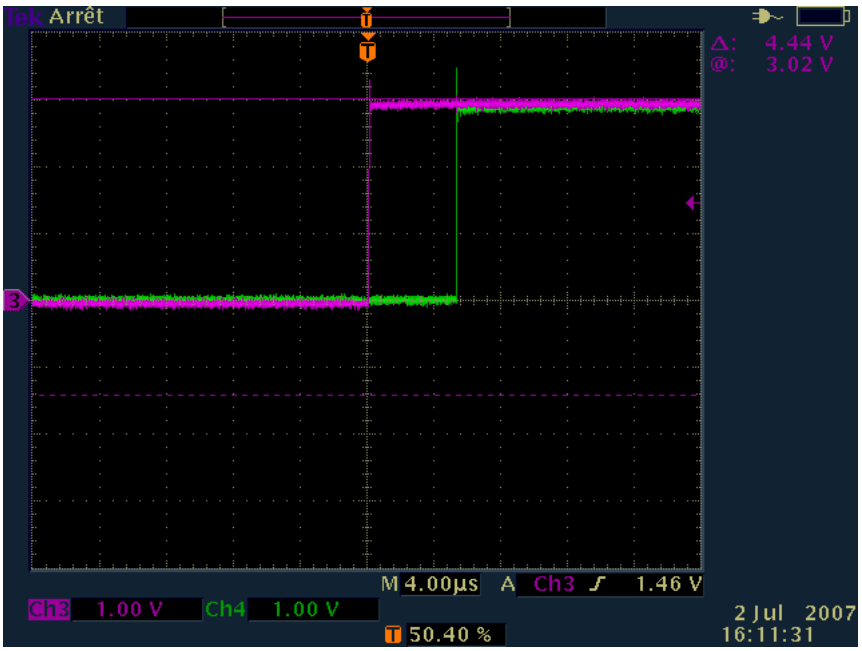

FIG. 10.2 – Mesure de temps avec l'oscilloscope et le générateur de fréquences, la tâche dure environ 5.6us

10.1 Temps d'envoi d'un paquet

Description du Test

J'ai utilisé la première méthode de mesure à l'oscilloscope pour ce test. Un timer déclenche un envoi de paquet toutes les 50ms, un signal de 3V est envoyé sur le port relié à l'oscilloscope avant l'envoi du paquet. Quand le paquet est envoyé un signal de 0V est envoyé sur le port jusqu'a la prochaine mesure.

Résultats

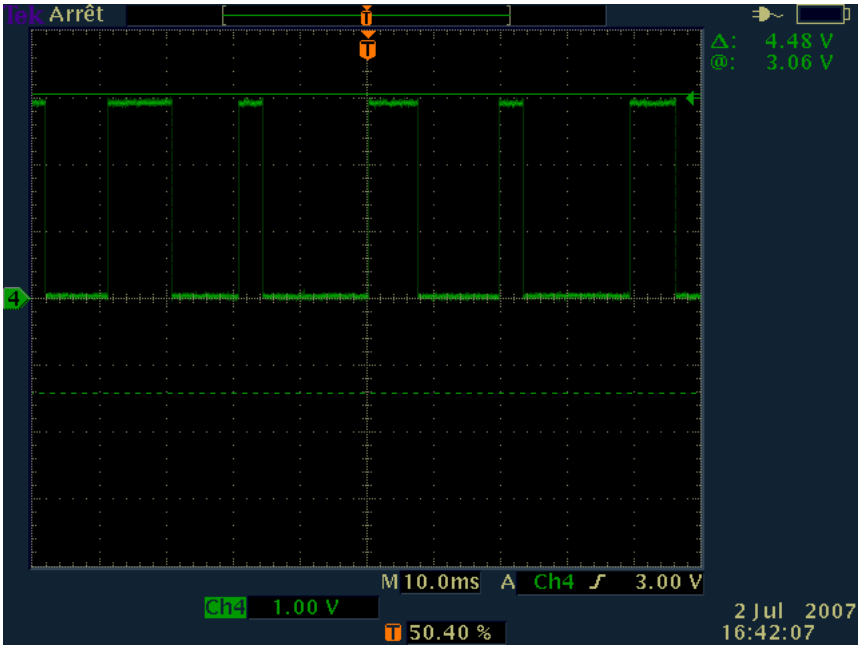

FIG. 10.3 – Envoi d'un paquet de 4 octets

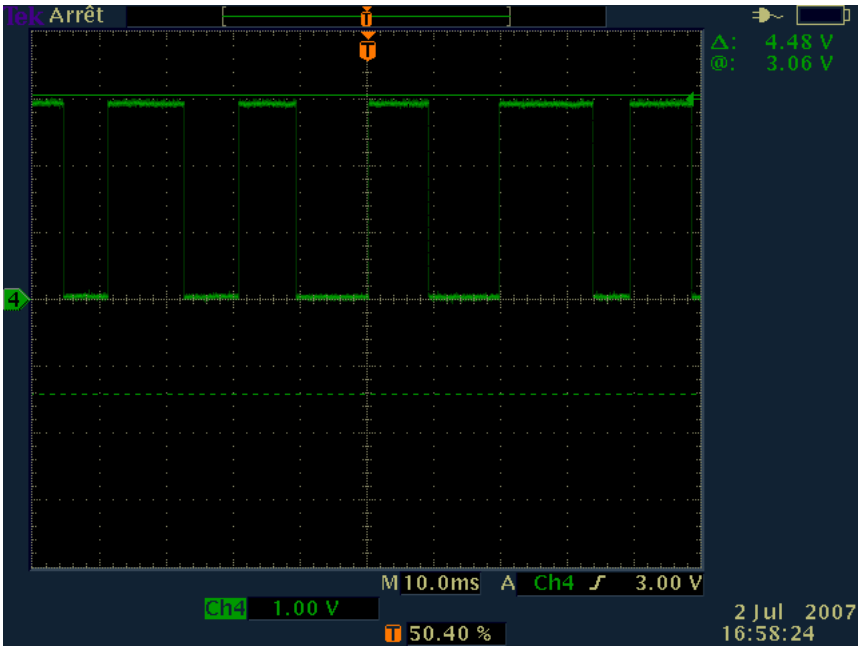

FIG. 10.4 – Envoi d'un paquet de 28 octets

Analyse des résultats et conclusion

On retrouve les résultats obtenus avec le MilliTimer : les temps d'envois ne sont pas constants et oscillent entre 6 et 13 ms. La taille des paquets n'influence pas le temps d'envoi.

10.2 Temps de mesure d'un ADC

A faire : finir avec les autres composants

Description du Test

J'utilise la première méthode pour calculer le temps mis par les différents composants de lecture de l'ADC pour lire un port plusieurs fois de suite.

Résultats

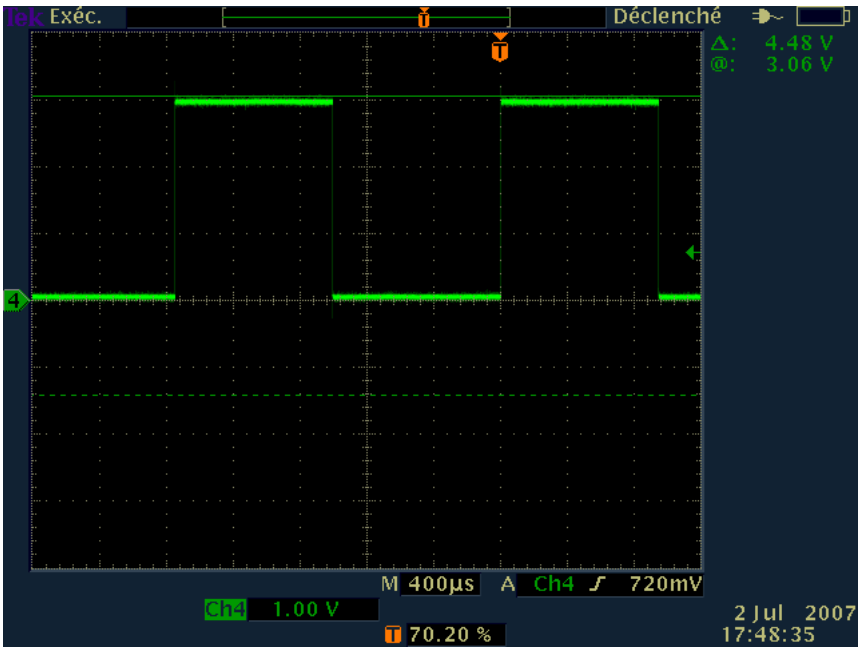

FIG. 10.5 – Lecture avec le composant AdcReadClientC

Analyse et conclusion

Le composant AdcReadClientC permet la lecture d'un simple canal sans répétition en environ 1ms, ce qui est très insuffisant. Il faudrait lire 6 canaux dans la moitié de ce temps pour permettre aux autres composants de tourner et garder une période de 1ms.

Quatrième partie

Mise en place du réseau de capteurs