INSTITUT FEMTO-ST

UMR CNRS 6174

Blinky Blocks Time Protocol (BBTP) : protocole de synchronisation des horloges internes de dispositifs embarqués

Version 1

André Naz — Benoît Piranda — Julien Bourgeois — Seth Copen Goldstein

Rapport de Recherche n° RR–FEMTO-ST–XXXX

DÉPARTEMENT DISC – January 13, 2015

Blinky Blocks Time Protocol (BBTP) : protocole de synchronisation des horloges internes de dispositifs embarqués

Version 1

André Naz , Benoît Piranda , Julien Bourgeois , Seth Copen Goldstein

Département DISC

OMNI

Rapport de Recherche no RR –FEMTO-ST–XXXX January 13, 2015 (23 pages)

Résumé : Ce rapport décrit les travaux menés sur la synchronisation des horloges internes de dispositifs embarqués dans le cadre du projet COordination and COmputation in Distributed Intelligent MEMS (CO_2Dim) [2] au sein de l'équipe Optimization, Mobility, NetworkIng (OMNI) au département d'Informatique des Systèmes Complexes (DISC) à Femto-ST. Le projet CO_2Dim est un projet international en collaboration avec des équipes de recherche de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) à Rennes, de l'Université Polytechnique de Hong-Kong (PolyU), ainsi que de l'Université de Carnegie Mellon (CMU) aux États-Unis. L'objectif ultime du projet CO_2Dim est de proposer un environnement de programmation pour les systèmes composés de microsystèmes électromécaniques intelligents distribués (DiMEMS): des micro-objets dotés d'une capacité de calcul, de capteurs et des actionneurs dont certains leur permettent de communiquer. Cet environnement comprend des DiMEMS physiques, un langage de programmation appelé Meld [1] développé à CMU et un simulateur nommé VisibleSim [3] réalisé à Femto-st. Un des buts phares du projet CO_2Dim est d'étendre Meld en un langage temps réel. C'est-à-dire un langage offrant aux programmeurs la possibilité de spécifier des contraintes temporelles et de coordonner des actions dans un système de DiMEMS. Dans les systèmes distribués, chaque acteur possède sa propre horloge interne. Si chaque DiMEMS a une heure différente, il est impossible de coordonner une action à une date précise. A travers ce rapport, nous contribuons au projet CO_2Dim en proposant BBTP : Blinky Blocks Time Protocol, un protocole de synchronisation des horloges internes de dispositifs embarqués appelés Blinky Blocks [7]. Ces robots modulaires conçus à CMU servent de support pour des grilles de capteurs et d'actionneurs formés de DiMEMS. BBTP a été codé en langage C et a été intégré au firmware des Blinky Blocks. Nous avons mené des expériences sur plusieurs configurations de Blinky Blocks matériels afin d'étudier la robustesse de BBTP: ce protocole peut synchroniser correctement jusqu'à 27 775 Blinky Blocks avec une précision de 40 millisecondes pour 99% du temps.

Mots-clés : systèmes distribués, robotqiue modulaire, protocole de synchronisation d'horloge

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tél : (33 3) 81 66 64 00 – Fax : (33 3) 81 66 64 23 – e-mail : brigitte.bataillard@univ-fcomte.fr

Blinky Blocks Time Protocol (BBTP): internal clock synchronization protocol for embedded systems

Version 1

Abstract:

The following report describes our work on clock synchronization protocol for embedded systems in the project COordination and COmputation in Distributed Intelligent MEMS (CO_2Dim) [2] led by the Optimization, Mobility, NetworkIng (OMNI) team at Femto-ST. CO_2Dim is a partnership project with the Institute for Research in IT and Random Systems (IRISA) at Rennes, the Hong-Kong Polytechnic University (PolyU) and Carnegie Mellon University (CMU) at Pittsburgh.

The aim of the project CO_2Dim is to propose a programming environment for systems composed of distributed intelligent microelectromechanical systems (DiMEMS): micro-objects with computational and communication capabilities, sensors and actuators. This environment is composed of hardware DiMEMS, Meld [1] a programming language developed at CMU, and VisibleSim [3] a simulator developed at Femto-st.

A key target of the project CO_2Dim is to extend Meld into a real-time programming language. That is to say, to give to the programmer the possibility to specify time constraints, and to express coordination of actions in Meld programs. However, in distributed systems, all the entities have their own internal clock. It is impossible to coordinate actions among a group of DiMEMS, if all of them have a different clock value.

In this report, we propose BBTP: Blinky Blocks Time Protocol, a distributed clock synchronization protocol for specific embedded systems called Blinky Blocks [7]. These modular robots are used as support for grids of sensors and actuators.

BBTP has been coded in C language and integrated inside the firmware of the Blinky Blocks. We evaluated the robustness of BBTP on various configurations of Blinky Blocks. We observe that our protocol can correctly synchronize up to 27,775 Blinky Blocks with a precision of 40 milliseconds 99% of the time.

Key-words: distributed systems, modular robotic, time synchronization protocol

FEMTO-ST Institute, DISC research department
UFR Sciences - Route de Gray - F-25030 BESANÇON cedex FRANCE
Tél : (33 3) 81 66 64 00 – Fax : (33 3) 81 66 64 23 – e-mail : brigitte.bataillard@univ-fcomte.fr

Blinky Blocks Time Protocol (BBTP) : protocole de synchronisation des horloges internes de dispositifs embarqués

André Naz , Benoît Piranda , Julien Bourgeois , Seth Copen Goldstein

January 13, 2015

1 Introduction

Ce rapport décrit les travaux menés sur la synchronisation des horloges internes de dispositifs embarqués dans le cadre du projet COordination and COmputation in Distributed Intelligent MEMS (CO_2Dim) [2] au sein de l'équipe Optimization, Mobility, NetworkIng (OMNI) dans le département d'Informatique des Systèmes Complexes (DISC) à Femto-ST. Le projet CO_2Dim est un projet international en collaboration avec des équipes de recherche de l'Institut de Recherche en Informatique et Systèmes Aléatoires (IRISA) à Rennes, de l'Université Polytechnique de Hong-Kong (PolyU), ainsi que de l'Université de Carnegie Mellon (CMU) aux États-Unis.

Le projet CO_2Dim s'intéresse à la coordination d'actions entre micro-systèmes électromécaniques intelligents distribués (DiMEMS), des micro-objets dotés d'une capacité de calcul, de capteurs, d'actionneurs. L'absence d'horloge de référence pose un problème majeure à la coordination temporelle d'actions dans les systèmes distribués. Il est évident qu'il est impossible de planifier une action à une date précise si chaque DiMEMS a une heure différente.

A travers ce rapport, nous contribuons au projet CO_2Dim en proposant BBTP : Blinky Blocks Time Protocol, un protocole de synchronisation des horloges internes de dispositifs embarqués, les Blinky Blocks [7], servant de supports à des capteurs et des actionneurs constitués de DiMEMS.

La première partie de ce rapport est consacrée à une mise en contexte. Nous abordons ensuite les objectifs et les problématiques liées à la synchronisation des systèmes distribués. Puis, nous détaillons BBTP et nous proposons une évaluation de celui-ci sur différentes combinaisons de blocs réels.

2 Contexte

Cette partie présente le projet CO_2Dim. Dans un premier temps nous présentons les équipes travaillant sur ce projet ainsi que le lien entre ce rapport et les objectifs de CO_2Dim. Dans un deuxième temps, nous exposons l'environnement de programmation pour Blinky Blocks.

2.1 Présentation du projet CO_2Dim

Le projet CO_2Dim est mené par l'équipe Optimization, Mobility, NetworkIng (OMNI) au sein du département d'Informatique des Systèmes Complexes (DISC) à l'institut Franche-Comté Électronique, Mécanique, Thermique et Optique - Sciences et Technologies (Femto-st). Ce projet est développé en partenariat avec des équipes de recherche de l'Institut de Recherche

en Informatique et Systèmes Aléatoires (IRISA) à Rennes, de l'Université Polytechnique de Hong-Kong (PolyU) ainsi que de l'Université de Carnegie Mellon (CMU) aux États-Unis.

Le projet CO_2Dim a pour but d'offrir un environnement de programmation pour DiMEMS. Cet environnement comprend des DiMEMS physiques, des supports pour ces DiMEMS tels que les Blinky Blocks [7] conçus par l'équipe de Seth Goldstein à CMU, un langage de programmation appelé Meld [1] développé par l'équipe de Seth Goldstein à CMU et un simulateur nommé VisibleSim [3] réalisé par l'équipe de Julien Bourgeois à Femto-st.

Un des objectifs phares du projet CO_2Dim est d'étendre Meld en un langage temps réel. C'est-à-dire un langage offrant aux programmeurs la possibilité de spécifier des contraintes temporelles, dont le respect est aussi important que le résultat logique des programmes [16]. La version temps réel de Meld facilitera la coordination d'actions entre DiMEMS. L'absence d'horloge de référence pose un problème majeure à la coordination d'actions dans les systèmes distribués. Il est évident qu'il est impossible de coordonner une action à une date précise si chaque DiMEMS a une heure différente. Ce problème se rencontre dans la vie de tous les jours avec les personnes humaines: comment se donner rendez-vous à la même heure si nos montres sont désynchronisées ? Nous avons contribué au projet CO_2Dim en développant BBTP, un protocole de synchronisation des horloges internes de Blinky Blocks. L'équipe de Jiannong Cao à PolyU travaille sur l'ajout d'élément de langage dans Meld afin de permettre au programmeur d'exploiter les outils de synchronisation temporelle décrits dans ce rapport.

Le projet Smart Blocks [13] mené par la même équipe de recherche propose une application industrielle intéressante de coordination d'actions entre DiMEMS. Ce projet consiste à mettre en place une surface de convoyage modulaire afin de transporter des objets légers d'un point A à un point B sans contact humain. Cette surface de convoyage est destinée à être utilisée en zones non-accessibles, par exemple parce qu'on veut préserver la stérilité de celles-ci (salles blanches), ou parce qu'elles sont dangereuses pour l'Homme (centrale nucléaire, Lune). Cette surface repose sur une grille de Smart Blocks auxquels sont connectés des DiMEMS rendant possible le convoyage d'objets: soit des jets d'airs, soit un système de cils. La synchronisation temporelles des Smart Blocks et des DiMEMS formant cette surface est nécessaire. En effet, deux blocs voisins doivent pouvoir effectuer des actions conjointes de telle façon à ce que les actions des jets d'airs ou des cils soient correctement combinées afin d'obtenir le déplacement souhaité de la pièce. De plus, cette surface modulaire est reconfigurable. C'est-à-dire qu'on peut en modifier la géométrie en demandant aux blocs de se déplacer les uns par rapport aux autres. La synchronisation temporelle est ici aussi importante. En effet, les moteurs des Smart Blocks doivent synchroniser leurs actions de façon à permettre le déplacement cohérent des blocs pour la reconfiguration.

2.2 Environnement Blinky Blocks

Nous allons à présent détailler l'environnement de programmation pour Blinky Blocks proposé dans CO_2Dim. Dans un premier temps nous présentons les Blinky Blocks, puis le langage Meld et finalement VisibleSim. La figure 1 présente de façon schématique l'environnement Blinky Blocks. La contribution apportée par ce rapport y est en rouge.

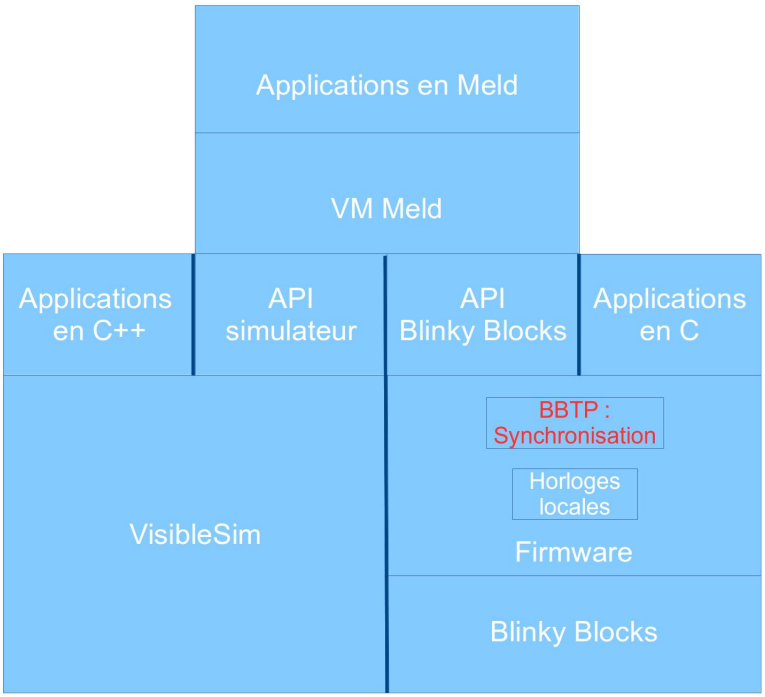


Figure 1: Environnement Blinky Blocks.

2.2.1 Blinky Blocks

Les Blinky Blocks [7] sont utilisés dans CO_2Dim comme des supports pour DiMEMS. Ces blocs ont été développés à l'Université de Carnegie Mellon. Ils ont été conçus afin d'étudier les problèmes de tolérance aux fautes, de passage à l'échelle, et d'auto-reconfiguration dans les systèmes massivement distribués sur du matériel concret, grâce à leur faible coût de production, et leur taille raisonnable.

Un Blinky Block peut être vu comme une brique LEGO intelligente. C'est un cube de 40 millimètres de côté, doté d'un micro-contrôleur Atmel ATxMega256a3, de six ports séries, d'une led multicolore, et de différents capteurs. Les Blinky Blocks sont capables de traiter de l'information, de communiquer entre eux via leurs ports séries, de changer de couleur, de détecter des mouvements, et de jouer des sons (cf. figure 2). Les Blinky Blocks ont une capacité mémoire très restreinte avec seulement 16Ko de mémoire vive. Tous les blocs contiennent les même composants matériels.

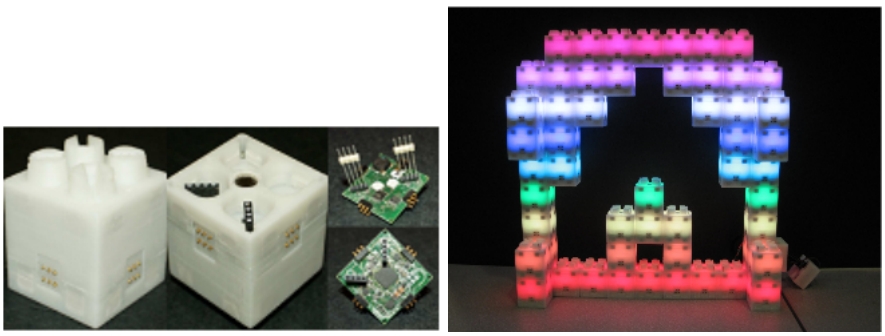


Figure 2: Blinky Blocks [7].

Ces blocs sont programmables en C et en Meld. Un firmware offrant des fonctionnalités de base, facilitant ainsi l'écriture d'applications au niveau a été développé (cf. figure 1). Tous les blocs d'un même système exécute la même application. Cette application est programmée sur un ensemble de blocs à l'aide d'un câble série et d'un logiciel fourni par CMU.

2.2.2 Meld

Motivations

Meld [1] est un langage déclaratif logique distribué développé à l'Université de Carnegie Mellon. Ce langage a été conçu afin de réduire les efforts des programmeurs d'applications massivement distribuées. En effet, la programmation pour les systèmes distribués est une tâche complexe où le programmeur doit veiller à gérer les communications entre les entités et la répartition des tâches entre chaque entité tout en garantissant le bon fonctionnement de son programme. Meld gère implicitement les communications et la répartition des tâches et permet au programmeur de ne se préoccuper que de l'aspect fonctionnel de son application. Les programmes Meld sont d'ailleurs entre 10 à 40 fois plus courts que leur équivalent en C++ [15]. De ce fait, les programmes Meld sont aussi plus faciles à lire et à comprendre.

De plus, le fait que Meld soit un langage de programmation logique et concis rend plus facile l'élaboration de raisonnements et de preuves formelles sur les programmes. En programmation logique, un programme est composé de faits initiaux et d'un ensemble de règles utilisant certains faits pour en dériver d'autres. Grâce à cet aspect, les programmes Meld sont aussi naturellement tolérants aux fautes. En effet, les faits dérivés de règles qui ne sont plus vérifiées sont automatiquement supprimés. Ainsi, lorsqu'un bloc s'aperçoit de la perte d'un voisin, les faits associés à ce voisin sont supprimés de la base de données.

Syntaxe et exemple

Nous allons étudier quelques éléments de syntaxe à travers un exemple simple afin de mieux comprendre Meld [1]. Le programme Meld de la figure 3 propage la couleur rouge dans un ensemble de blocs à partir du bloc avec l'identifiant unique 1. Un programme est composé d'une base de données comprenant des faits, et d'un ensemble de règles utilisant des faits pour en dériver d'autres. Les règles sont composées d'un *corps* et d'une *tête*: *corps -o tete*. Une règle est appliquée lorsque les faits présents dans le *corps* existent dans la base de données du programme. Les faits dans la *tête* sont alors générés dans la base de données.

Par exemple, *init(A) -o setcolor(A, R, G, B)* se comprend: si la base de données du bloc *A* contient le fait *init(A)*, alors A consomme ce fait et insère le nouveau fait *setcolor(A,R,G,B)* dans sa base de données. *setcolor(A,R,G,B)* est un prédicat système qui est immédiatement converti en action: *A* prend la couleur *(R,G,B)*.

L'annexe 1 montre un code C équivalent à ce programme. Le code Meld est 5 fois plus court et parait plus simple à comprendre.

[1] Manuel: https://github.com/claytronics/meld/blob/master/docs/manual.pdf

```
/* Déclaration des prédicats du programme */
type linear propagate-color(node,int,int,int).
type linear propagate-color-if-waiting(node,int,int,int).
type linear waiting(node).

/* Tous les blocs attendent de recevoir un ordre de changement
* de couleur
*/
waiting (A).

/* Le bloc d'ID 1 commencera la propagation de la couleur rouge */
propagate-color-if-waiting(@1,255,0,0).

/* Un bloc A change de couleur (R,G,B) et la propage , s'il en reçoit
* l'ordre et s'il n'avait jusqu'à présent reçu aucun ordre
*/
propagate-color-if-waiting(A,R,G,B), waiting (A) -o
setcolor(A,R,G,B), propagate-color(A,R,G,B).

/* A propage la couleur (R,G,B) à tous ses voisins */
propagate-color(A,R,G,B) -o
/* Propage la couleur (R,G,B) à tout voisin C de A connecté
* par la face X
*/
{C,X| ! neighbor (A,C,X) |
propagate-color-if-waiting(C,R,G,B)}.
```

Figure 3: Exemple de programme Meld: propagation de la couleur.

Fonctionnement

Un programme écrit en Meld doit être compilé en byte-code [2] avant d'être exécutée par une machine virtuelle (VM) [3] codée en C++ (cf. figure 1). Cette machine virtuelle peut être exécutée sur des Blinky Blocks matériels ou par le simulateur VisibleSim.

2.2.3 VisibleSim

VisibleSim [3] est un simulateur de DiMEMS développé par des chercheurs à Femto-st (cf. figure 4). Il permet de simuler entre autres l'environnement Blinky Blocks. Les applications simulées peuvent être codées en C++ ou en Meld. Ce simulateur est particulièrement utile pour évaluer des algorithmes sur un grand ensemble de blocs puisque seul un nombre limité de blocs matériels sont à notre disposition. Par contre, VisibleSim ne simule pour l'instant que l'aspect fonctionnel des programmes. Les horloges internes ne sont par exemple pas simulé. VisibleSim ne peut donc pas être utilisé pour tester notre mécanisme de synchronisation.

[2] Documentation du byte-code: https://github.com/claytronics/meld/blob/master/docs/vm_format.pdf
[3] Documentation de la VM: https://github.com/claytronics/meld/blob/master/docs/vm.pdf

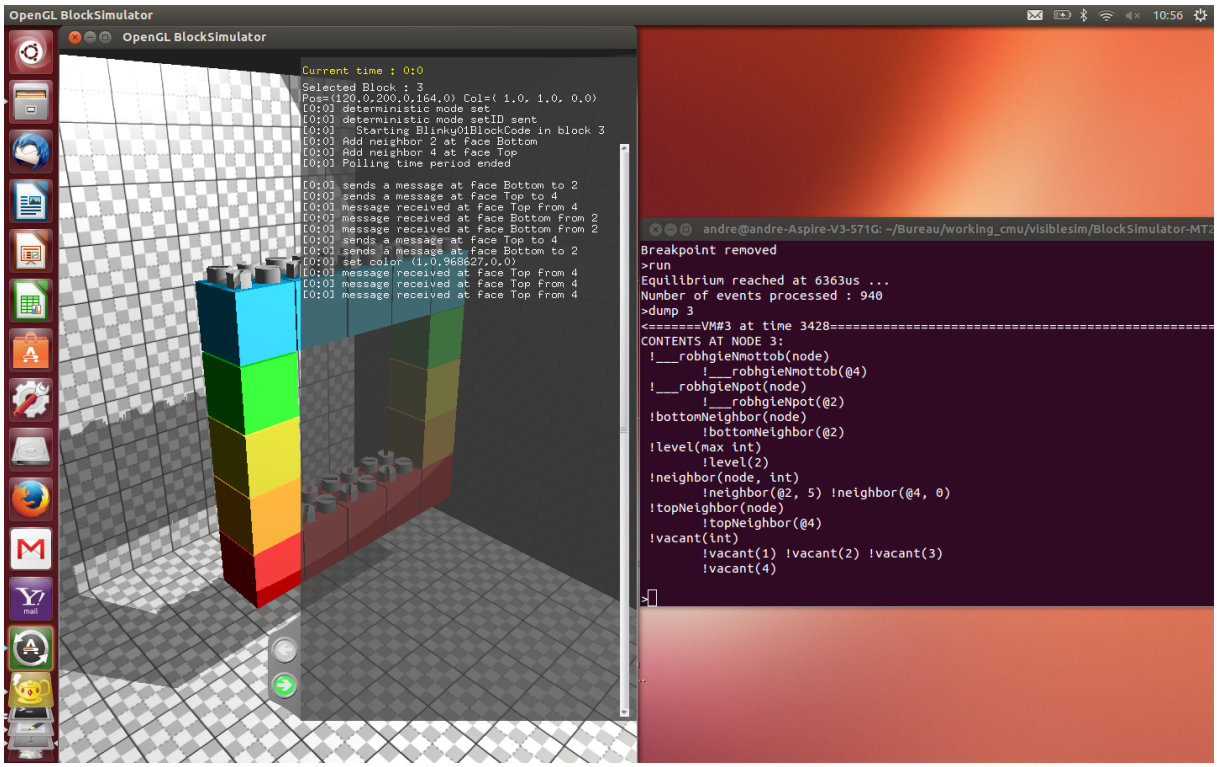


Figure 4: VisibleSim.

3 Objectifs et problématiques

Notre but est de proposer un protocole de synchronisation des horloges internes des Blinky Blocks. Nous avons décidé d'articuler ce développement autour d'une application simple: **coordonner le changement de couleur simultané et périodique de Blinky Blocks à une date précise avec une précision suffisante pour obtenir une perception visuelle d'un changement de couleur simultané des blocs**. Pour rappel, la persistance rétinienne de l'oeil humain est égale à $1/24Hz = 42ms$.
Cet exemple soulève différents problèmes, qu'il a fallu résoudre:

- Pour un bloc, le temps 0 correspond au démarrage de celui-ci. Il commence ensuite à compter le temps écoulé au fur et à mesure de l'exécution du programme grâce à son horloge interne. Ainsi chaque bloc a sa propre notion du temps en tant qu'entité à part entière et aucune notion de temps global n'est partagée par l'ensemble des blocs du système. Lorsqu'un bloc est au temps local 150, il se peut qu'un autre bloc ne soit qu'au temps local 100. Si aucun bloc n'a la même heure, il est évident qu'il est impossible de coordonner une action dans un grand ensemble de blocs à une date précise. Ce problème se rencontre dans la vie de tous les jours entre humains: comment se donner rendez-vous à une heure précise si nos montres sont désynchronisé ?

- On pourrait supposer qu'il suffit de démarrer les blocs au même instant pour qu'ils aient plus ou moins la même heure et par conséquent qu'ils soient synchronisés. Mais ce n'est pas suffisant, car les horloges internes de tels systèmes dépourvus de quartz ne sont pas précises. L'horloge locale d'un bloc peut avancer plus ou moins vite que celle d'un autre bloc. On dit que les horloges dérivent. Nous avons donc étudié cette dérive sur des systèmes réels.

- Les Blinky Blocks forment un système autonome. Ils doivent être capables de se synchroniser à travers des échanges inter-blocs, sans communication avec le monde extérieur. Un bloc particulier du système doit être désigné comme référence ou maître du temps et synchroniser régulièrement les autres blocs. De façon générale, l'élection d'un leader dans un système distribué est considérée comme impossible dans le cas général à cause des pannes, erreurs et pertes de messages qui peuvent survenir. Nous avons décidé de poser l'hypothèse que nous sommes dans un monde parfait, où aucune panne ou erreur ne survient durant l'élection du maître du temps. De plus, chaque bloc possède un identifiant unique permettant de le différencier des autres dans le système. Il est aussi important de noter que les Blinky Blocks forment des systèmes reconfigurables. En effet, des blocs peuvent être manuellement enlevés ou ajoutés au système. Les reconfigurations manuelles pouvant entrainer la disparition du maître du temps ou l'apparition d'un second, il faut que notre protocole veille à ce qu'un et un seul bloc assure toujours ce rôle.

- Le maître du temps envoie périodiquement l'heure courante aux différents blocs du système. Avec la précision de synchronisation qu'on souhaite atteindre, les délais induits par les communications ne sont pas négligeables. En effet, l'heure contenue dans le message reçu par un bloc n'est plus l'heure courante lorsque ce bloc reçoit le message. Le bloc récepteur doit estimer le temps de transmission du message et corriger l'heure qu'il contient.

- Aucune qualité de service n'est garantie pour les communications inter-blocs, qui sont effectuées en mode meilleur effort. Autrement dit, un message envoyé n'est pas forcément reçu correctement. Il peut être perdu ou arriver erroné. La connexion peut aussi être momentanément ou définitivement perdue. Le taux d'erreurs étant quasi-nul nous avons décidé d'ignorer ce point pour l'instant. Notre mécanisme de synchronisation fonctionne donc en mode meilleur effort.

- Les Blinky Blocks sont des systèmes embarqués très restreints en mémoire et capacité de calcul. Un bloc possède seulement 16Ko de mémoire vive et un message contient seulement 17 octets de données. La solution mise en place doit en tenir compte.

- Les systèmes de DiMEMS tendent à être de taille très importante. BBTP doit pouvoir passer à l'échelle. Dans des applications pratique, on aimerait faire fonctionner BBTP sur des systèmes de plusieurs milliers de blocs.

- La topologie du système est susceptible d'évoluer au cours du temps. Des blocs peuvent être ajoutés ou retirés. Le nouveau système doit être capable de s'auto-organiser: décider d'un nouveau maître du temps et se resynchroniser correctement et de manière cohérente. Par exemple, l'ajout d'un ensemble A de nouveaux blocs à un ensemble B doit impliquer un choix judicieux entre l'horloge de référence de A et celle de B. On choisit l'horloge la plus avancée afin d'éviter des soucis d'incohérence dans le temps.

Nous nous sommes donc fixés les objectifs suivants:

1. Étudier les durées communications entre Blinky Blocks et modéliser la dérive de leur horloge.

2. Proposer une solution à la problématique de synchronisation des Blinky Blocks suivant le modèle ci-dessous et respectant les contraintes énoncées plus haut:

   (a) Élection d'un bloc comme maître du temps.

(b) Construction d'un arbre de diffusion.

(c) Synchronisation des horloges.

3. Implémenter cette solution dans le firmware des Blinky Blocks.

4. Réaliser un jeu d'expérimentations permettant de valider BBTP.

4 Etat de l'art

La synchronisation des horloges dans les systèmes distribués a été étudiée à plusieurs reprises et de nombreuses solutions ont été proposées: Network Time Protocol (NTP) [12], Precise Time Protocol (PTP) [6], Reference Broadcast Synchronization (RBS) [5], Flooding Time Synchronization Protocol (FTSP) [11] pour n'en citer que quelques-unes.

RBS et FTSP ont été proposés pour les réseaux sans-fils et utilisent la notion de domaine de diffusion (broadcast). Ces mécanismes ne sont pas utilisables dans notre cas puisque les Blinky Blocks ne peuvent communiquer à un instant donné qu'avec leurs six voisins directs potentiels (liaison point-à-point).

NTP et PTP, comme la plupart des protocoles de synchronisation utilisent un noeud particulier du système comme référence ou maître du temps. Ce maître du temps envoie périodiquement, sur demande ou non, l'heure courante aux autres noeuds du système. Le problème posé par cette approche est que selon la précision de synchronisation qu'on souhaite atteindre, les délais induits par les communications ne sont pas négligeables. C'est-à-dire que l'heure contenue dans le message reçu par un noeud n'est plus l'heure courante.

Ces deux protocoles tentent d'estimer les délais induits par les communications sur la base d'allers-retours de messages entre le maître du temps et l'élément synchronisé afin de tenir compte du trajet dans le calcul de l'heure courante. Mais ces délais sont souvent variables selon la topologie du réseau, sa charge, ainsi que la charge de travail des équipements impliqués dans la communication. NTP et PTP permettent d'atteindre des précisions de l'ordre de 100 μs et même en-deçà pour PTP.

Pour compenser la dérive des horloges, c'est-à-dire le fait que chaque horloge avance à une vitesse légèrement différente des autres, NTP et PTP les synchronisent de façon périodique. RBS et FTSP utilisent le fait que sur un intervalle de temps raisonnable, la vitesse de chaque horloge est constante et montrent qu'il est possible de compenser la dérive des horloges par régression linéaire.

De manière générale ces protocoles sont assez sophistiqués, il n'est pas nécessaire d'en garder tous les aspects dans notre cas. Ainsi, avant de proposer une solution adaptée aux Blinky Blocks, il est nécessaire de connaître leur comportement physique, notamment en ce qui concerne leur horloge interne et les communications.

5 Étude du comportement des Blinky Blocks

Afin de proposer une solution adaptée aux Blinky Blocks, nous avons étudié leur horloge interne ainsi que leurs communications.

5.1 Modélisation de la dérive des horloges internes

Les Blinky Blocks sont équipés de plusieurs horloges temps réel (HTR) qui permettent de décompter le temps écoulé depuis le démarrage des blocs. Sur un bloc donné, une seule HTR peut être activée à la fois. Les HTR disponibles utilisent des oscillateurs RC et sont très

imprécises comparées aux horloges utilisant des oscillateurs à cristal présentes dans la plupart des ordinateurs actuels. L'HTR la plus précise des Blinky Blocks a une précision d'1% et une résolution d'une milliseconde.

Avec cette dernière HTR, après 15 secondes l'horloge du bloc le plus rapide aura théoriquement au plus avancé de $15 + 1\% * 15 = 15,150$ secondes, et celle du plus lent de $15 - 1\% * 15 = 14,850$ secondes. Ainsi, en seulement 15 secondes, la dérive entre l'horloge la plus rapide et l'horloge la plus lente peut atteindre en théorie 300 millisecondes. Nous rappelons que notre contrainte est de 42 millisecondes.

Nous avons mis en place un banc expérimental afin d'évaluer la dérive des horloges sans ajouter de code dans le firmware des blocs qui pourrait perturber leur fonctionnement. Cette expérience consiste à enregistrer à l'aide d'une caméra des changements de couleurs T-périodiques. Avec T égale à 3 secondes, le changement de couleur parait instantané à la date T. En revanche après $5T$, les changements de couleur apparaissent décalés dans le temps. A l'aide d'une caméra, nous avons évalué à 90 millisecondes l'écart entre le bloc le plus rapide et le plus lent. Autrement dit, le bloc le plus rapide avance 1,006 fois plus vite que le bloc le plus lent.

Une des caractéristiques importantes des oscillateurs RC est leur stabilité: leur fréquence d'oscillation est relativement constante. Autrement dit, une HTR utilisant un oscillateur RC dérive de façon plus ou moins stable. Ainsi, la dérive de l'HTR d'un bloc par rapport à l'HTR d'un autre doit être quasiment linéaire. Nous l'avons vérifié expérimentalement en démarrant 3 blocs simultanément et en les faisant changer de couleur périodiquement. La figure 5 représente les écarts entre les instants de changement de couleur des blocs 2 et 3 par rapport aux instants de changement de couleur du bloc 1 de référence. Nous avons aussi vérifié que ce comportement ne dépend pas de la forme du système.

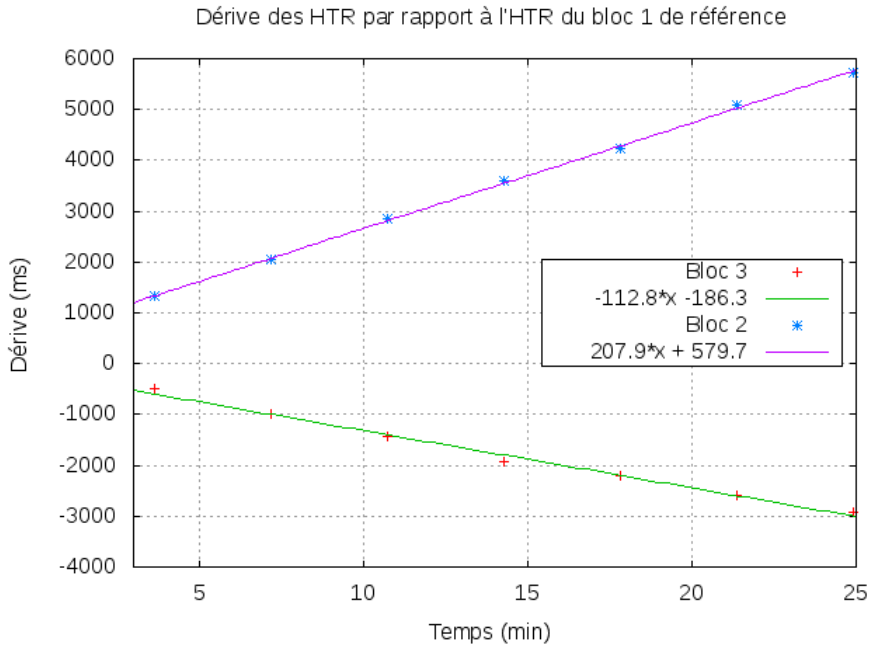


Figure 5: Dérive des HTR par rapport à l'HTR d'un bloc de référence.

Un coefficient peut donc être utilisé pour corriger la dérive des HTR. Nous avons aussi remarqué que d'une exécution à une autre cette dérive varie. Ainsi, ce coefficient doit être calculé à chaque exécution.

5.2 Étude des durées de communications

Dans cette partie nous nous intéressons à déterminer la durée moyenne de l'échange d'un message entre deux blocs. Nous sommes dans le cas particulier d'un réseau point-à-point, les Blinky Blocks ne communiquent qu'avec leurs voisins directs. Tous les messages, appelés Chunks, ont une taille fixe de 21 octets dont 17 octets de données. Le schéma en Annexe 2 présente le modèle des communications des Blinky Blocks.

Nous décomposons la durée du processus d'envoi d'un message en plusieurs étapes simples dont nous cherchons à déterminer la durée (modèle simplifié inspiré de l'article [8]) (cf. figure 6):

- Durée d'envoi: temps écoulé entre l'appel à la fonction d'envoi de message et l'envoi du premier bit sur le lien série. Cette durée représente principalement le temps d'attente du message dans la file d'émission. Ce délai est variable en fonction de la charge du lien considéré.
- Durée de transmission: temps nécessaire à la transmission de l'ensemble des bits du message sur le lien série.
- Durée de propagation: temps nécessaire au déplacement des bits du message pour aller de l'émetteur au récepteur sur le lien série.
- Durée de réception: temps qu'il faut au récepteur pour recevoir du lien série les bits et en construire un message.
- Durée d'arrivée: temps d'attente du message dans la file de réception avant son traitement. Ce délai est variable en fonction de la charge du réseau.

La figure 6 schématise cette décomposition.

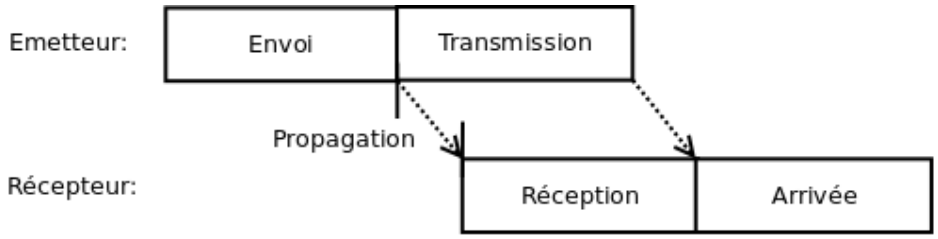


Figure 6: Décomposition de la durée d'envoi d'un message.

Les temps de transmission, de propagation et de réception dépendent en théorie presque uniquement de la connexion (composants, débit), et de la taille du message. Dans notre cas, les blocs contiennent les même composants, se connectent les uns aux autres de la même façon, le débit est fixe, et la taille des messages l'est aussi. Ces temps sont donc en théorie déterministes, nous allons le vérifier par l'expérience. Par convention, nous appellerons temps de transfert d'un message, la somme des temps de transmission, de propagation et de réception de celui-ci.

Nous avons estimé le temps moyen de transfert ($T_{transfert}$) d'un message dans un système dense où chaque bloc envoie continuellement un message à ses voisins qui lui répondent aussitôt. On parle d'allers-retours (cf. figure 7).

$$T_{transfert} = \frac{(T4 - T1) - (T3 - T2)}{2}$$

T1, T3 représentent respectivement les dates d'émission du premier bit pour respectivement l'aller et le retour. T2, T4 représentent respectivement les dates de réception complète pour respectivement l'aller et le retour (cf. Figure 7). Cette équation prend comme hypothèses que les liens sont symétriques et que la dérive des horloges locales est négligeable entre T2 et T3.

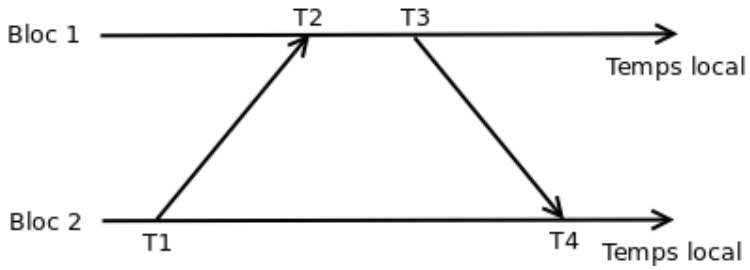


Figure 7: Echange de messages entre deux blocs.

Nous avons étudié cette durée sur 20000 allers-retours. Nous en déduisons que ce temps est relativement constant, toujours compris entre 5 et 7 millisecondes. La légère variation observée est due à la précision d'une milliseconde des horloges locales. Le temps de transfert moyen est d'environ 5,98 millisecondes. Le nombre de communications simultanées et la charge de calcul des blocs ne semble pas avoir d'impact sur le temps de transfert.

Nous avons décidé d'approcher la durée de transfert d'un message entre deux blocs à 6 millisecondes pour simplifier nos protocoles. Nous arrondissons le résultat précédent à cause de la précision d'une milliseconde des horloges locales.

Remarque: les temps de transfert ont été obtenus via un système de log distribué que j'ai développé. Ce système permet de récupérer sur un ordinateur connecté des informations textes du comportement du programme pendant son exécution. Afin de ne pas faire de digression par rapport à notre problématique les spécifications du système de log ne sont pas présentées dans ce rapport.

6 Election du maître du temps et construction de l'arbre de diffusion dans BBTP

L'élection du maître du temps ainsi que la construction de l'arbre de diffusion se font en même temps via un algorithme d'élection de leader dans un réseau asynchrone quelconque présenté dans le livre de M. Raynal [14].

Cet algorithme est basé sur les algorithmes de parcours de graphe. Chaque bloc initie un parcours de graphe avec message retour. Tous les messages liés aux parcours du réseau contiennent l'identifiant du bloc qui l'a lancé. Lorsqu'un bloc reçoit un message d'un nouveau parcours du réseau, il renonce au parcours en cours et prend part au nouveau parcours si l'identifiant qu'il contient est plus petit que celui du parcours courant. En revanche, si l'identifiant contenu dans ce nouveau parcours est plus grand, le nouveau parcours est ignoré. L'arbre de diffusion est construit en même temps grâce au message retour.

Notre protocole est tolérant aux fautes car le maître du temps est automatiquement ré-élu si la topologie du système est modifiée. En effet, lorsqu'un groupe de blocs est retiré ou ajouté au système, il se peut que le maître du temps ait disparu, ou qu'un deuxième ait apparu. Une nouvelle élection est donc déclenchée dès qu'un bloc détecte l'arrivée d'un nouveau voisin ou le départ d'un voisin. Lors d'une ré-élection, l'horloge de référence prend la valeur de l'horloge la plus en avance du système. Ainsi lors de la première synchronisation, tous les blocs avanceront

leur horloge locale à l'heure de référence, peu importe le retard qu'affiche leur horloge locale. On évite ainsi toute incohérence dans le temps.

7 Synchronisation des horloges dans BBTP

Cette partie présente le protocole de synchronisation des horloges à proprement parler, c'est-à-dire comment le maître du temps propage l'heure de référence dans le système.

7.1 Protocole de synchronisation

La synchronisation des blocs se fait de proche en proche via l'arbre de diffusion construit lors de l'élection du maître du temps. Puisque la durée de transfert d'un message entre deux blocs est quasi-constante et a été estimée à 6 millisecondes, il n'est pas nécessaire que nous utilisions les mécanismes similaires à ceux utilisés par NTP ou PTP, où l'estimation de ce délai est basée sur la mesure du temps d'aller-retour de messages.

Le protocole de synchronisation comporte un seul type de message: `SYNCHRONISATION <`H`>`. avec H l'heure courante du bloc émetteur insérée juste avant le début de l'émission du message (temps d'envoi, cf. section 5.2: décomposition des durées de communication). Le protocole fonctionne comme décrit ci-dessous (cf. figure 8).

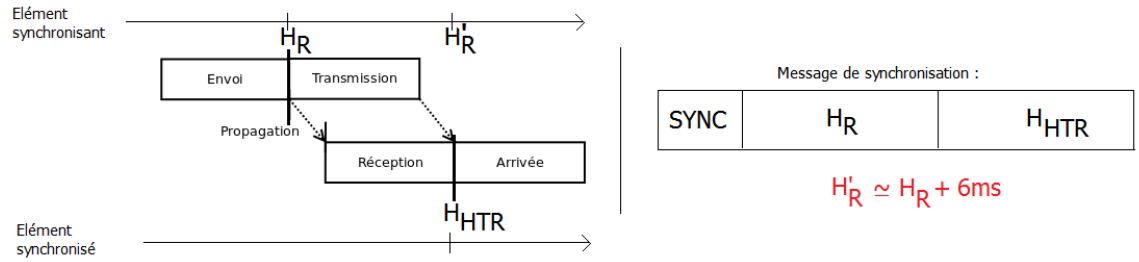


Figure 8: Schéma des messages de synchronisation.

Le maître du temps envoie `SYNCHRONISATION <`H_R`>` à tous ses voisins. Au moment où ceux-ci reçoivent le message `SYNCHRONISATION <`H`>`, H n'est plus égale à H_R. Les blocs doivent estimer l'heure de référence du système en prenant compte du temps de transfert du message : $H_R \approx H'_R = H + 6$ ms. Ils mettent ensuite à jour leur horloge locale H_L, et envoient leur estimation de l'horloge de référence à leurs voisins: `SYNCHRONISATION<`H'_R`>`. De façon générale un bloc qui reçoit un message `SYNCHRONISATION <`H`>` estime l'heure globale du système par $H_R \approx H'_R = H + 6$ ms, met à jour son horloge H_L, et synchronise ses voisins en leur envoyant un message `SYNCHRONISATION <`H'_R`>`.

L'heure de référence H_R est estimée pour le temps de réception, c'est-à-dire l'instant auquel le message est reçu (cf. section 5.2: décomposition des durées de communication). Puisque les messages dans la file d'attente sont traités relativement rapidement, nous considérons comme constant la décalage entre l'horloge de référence et l'horloge locale pendant la durée qui sépare la réception d'un message et son traitement (durée d'arrivée).

Une fois l'avance ou le retard calculé, le bloc récepteur peut mettre à jour son horloge. Un bloc avec une horloge locale H_L en retard, avance son horloge. En revanche, un bloc avec une horloge locale en avance, ne peut pas reculer son horloge. Il la bloque pendant $H_L - H'_R$ afin d'être à nouveau réglé sur l'heure de référence. Puisque l'avance d'un bloc est faible, la différence de vitesse entre l'horloge du bloc en question et l'horloge du maître du référence est

négligeable sur $H_L - H'_R$ millisecondes. Attendre $H_L - H'_R$ est équivalent à attendre que H_R rattrape H_L quelque soit l'horloge utilisée pour mesurer cette durée.

Nous avons vu dans la section 5.1 que la dérive des HTR est quasi-linéaire par rapport à l'horloge de référence: $H_R \approx a{*}H_{HTR}{+}b$. Les blocs sauvegardent les points $< H_{HTR}, H'_R >$ des 5 dernières synchronisations, et effectuent des calculs de régression linéaire [4] afin de déterminer le modèle (ie. a et b) suivi par leur HTR par rapport à l'horloge de référence sur cette fenêtre temporelle. Le cinquième couple $< H_{HTR5}, H'_{R5} >$ est considéré comme le plus récent.

$$\overline{H_{HTR}} = \frac{1}{5} * \sum_{i=1}^{5} H_{HTRi}$$

$$\overline{H'_R} = \frac{1}{5} * \sum_{i=1}^{5} H'_{Ri}$$

$$a = \frac{\sum_{i=1}^{5}(H_{HTRi} - \overline{H_{HTR}})(H'_{Ri} - \overline{H'_R})}{\sum_{i=1}^{5}(H_{HTRi} - \overline{H_{HTR}})^2}$$

$$b = H'_{R5} - a * H_{HTR5}$$

$$H_L = a * H_{HTR5} + b$$

Corriger la dérive rend plus précise l'approximation et permet donc d'augmenter la période entre les synchronisations.

Le protocole de synchronisation comporte 2 phases: une phase de calibration et une phase nominale. La seule différence entre ces deux phases est la période de synchronisation. La phase de calibration dure 30 secondes, et permet aux blocs d'apprendre le modèle de leur HTR. Le maître du temps y synchronise les blocs toutes les 2 secondes, contre 5 secondes lors de la phase nominale. Les périodes choisies sont discutées dans la section 7.2.2.

7.2 Évaluation des limites de BBTP

7.2.1 Tests des rayons synchronisables

La synchronisation des horloges se fait de proche en proche. A la réception d'un message de synchronisation, l'approximation des délais de communication entraîne une approximation de l'heure de référence. Cette approximation s'accumule à chaque bloc traversé. Le but de cette section est de déterminer la taille de système que notre protocole est capable de synchroniser.

Pour ce faire, nous avons mener l'expérience suivante:

- Le système est composé du maximum de blocs disponibles (28) répartis en forme de ligne (cf. figure 9).
- Le maître du temps est placé sur une extrémité.
- Le maître du temps synchronise les blocs toutes les 500 millisecondes. La période est faible afin que la dérive des horloges ne perturbe pas l'expérience.
- Les blocs changent de couleur toutes les 3 secondes.

[4] Régression Linéaire: http://tice.inpl-nancy.fr/modules/unit-stat/chapitre7/

Nous avons enregistré l'expérience à l'aide d'une caméra d'une résolution temporelle de 40 millisecondes. Afin de vérifier que les blocs soient synchronisés avec une précision de 42 millisecondes, il faut identifier sur les images de la vidéo une image à laquelle commence un changement de couleur (cf. figure 9, photo de gauche). A l'image suivante, c'est-à-dire 40 millisecondes plus tard, la distance du bloc le plus éloigné ayant changé de couleur correspond au rayon maximal que le maître du temps est capable de synchroniser. Dans notre cas (cf. figure 9: photo de droite), tous les blocs ont changé de couleur. Ainsi, le maître du temps arrive à correctement synchroniser un rayon de 27 blocs, et nous n'avons pas assez des blocs pour déterminer le rayon maximal.

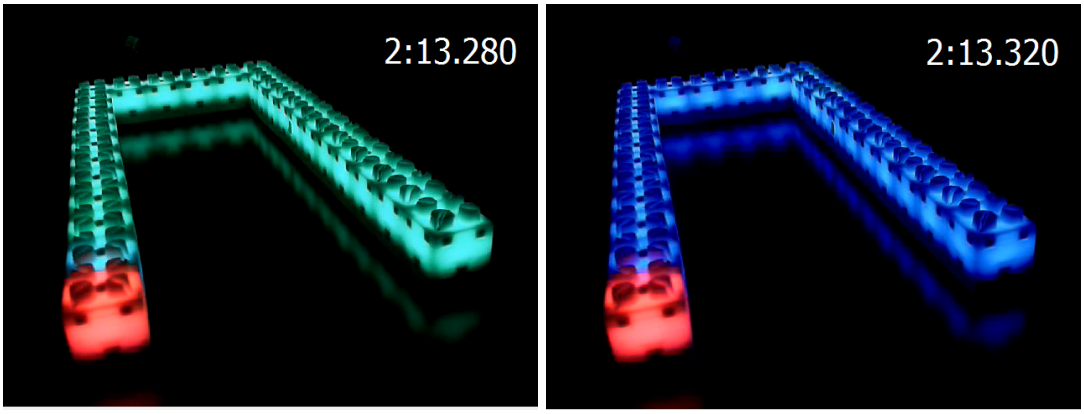


Figure 9: Deux images successives dans une vidéo de changements de couleur périodiques d'une ligne de 28 Blinky Blocks.

Si le maître du temps est placé au centre, notre protocole est donc au minimum capable de synchroniser correctement tous les systèmes ayant un rayon jusqu'à 27 blocs. Pour avoir un ordre de grandeur, un octaèdre régulier formant un réseau de 27 blocs de rayon regroupe 27 775 blocs!

7.2.2 Périodes de synchronisation

Cette section discute le choix du schéma du protocole de synchronisation présenté plus haut. Nous avons déterminé expérimentalement les valeurs des périodes de synchronisation des phases de calibration et nominale ainsi que la durée de la phase de calibration.

A l'aide du système de log, nous avons réalisé l'expérience suivante sur une ligne de 28 blocs (cf. figure 9) avec différentes valeurs des paramètres que nous cherchons à estimer: lorsqu'un bloc reçoit un message de synchronisation, il reporte avec probabilité $\frac{1}{4}$ la dérive de son horloge par rapport à l'horloge de référence. La dérive est définie comme la différence entre l'heure de référence estimée grâce au message de synchronisation et l'heure de son horloge locale. Il est important de noter que cette méthode prend pour hypothèse que l'heure de référence est correctement estimée à partir d'un message de synchronisation. Les blocs ne reportent pas systématiquement leur dérive, afin de ne pas saturer le réseau.

La figure 10 montre la dérive des horloges locales de deux blocs par rapport à l'horloge de référence avec et sans synchronisation. On y constate que la dérive des HTR par rapport à l'horloge de référence est quasi-linéaire (courbes: "Sans synchronisation"). Bien qu'un modèle linéaire a été mise en place pour corriger la dérive des HTR, les horloges locales dérivent légèrement. Ces dérives sont principalement dues à l'irrégularité de la fréquence d'oscillation des HTR. On constate que les valeurs de ces dérives dépendent du matériel (cf. figure 10). La figure 10 montre qu'avec une période nominale de 10 secondes, la dérive du bloc 1 est restée

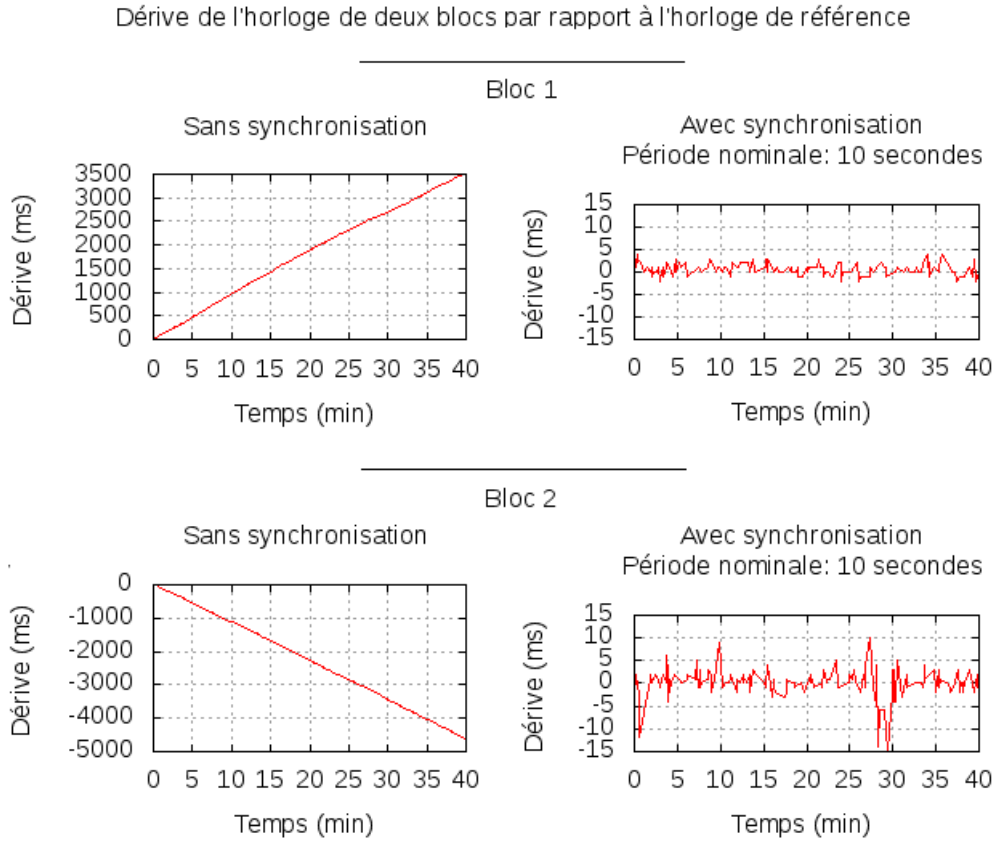


Figure 10: Dérive de l'horloge de deux blocs par rapport à l'horloge de référence.

bornée entre [-5;5] millisecondes sur 40 minutes, alors que la dérive du bloc 2 est seulement restée bornée entre [-15;10] millisecondes.

Il nous faut donc étudier le système complet. La figure 11 regroupe l'ensemble des dérives mesurées sous forme de nuage de points pour des périodes nominales de 5 secondes à droite et de 10 secondes à gauche. La phase de calibration de 30 secondes avec une période de 2 secondes est suffisante pour avoir une précision de 42 millisecondes. Avec une période nominale de 10 secondes, 99.6% des valeurs de dérive mesurées sont inférieures à 20 millisecondes contre 99.9% pour une période nominale de 5 secondes.

On peut remarquer la présence de quelques points extrêmes sur les courbes de la figure 11: à de rares occasions la dérive a atteint 30 à 60 millisecondes. Ces points ne sont pas dus à des pertes de messages, ni à des délais dans l'acheminement des messages de synchronisation. Nous pensons sans avoir pu le vérifier qu'ils sont dus à des comportements ponctuellement irréguliers de l'HTR de certains blocs.

Nous avons opté pour une période nominale de 5 secondes car la précision globale est meilleure sur les 40 minutes d'expérience, ce qui nous offre une légère marge dans notre objectif initial, et parce que la courbe avec cette période nominale affiche moins de points extrêmes.

En occupant 6 millisecondes par lien toutes les 5 secondes, ce protocole utilise à peine 0.12% de la bande passante sur cette période.

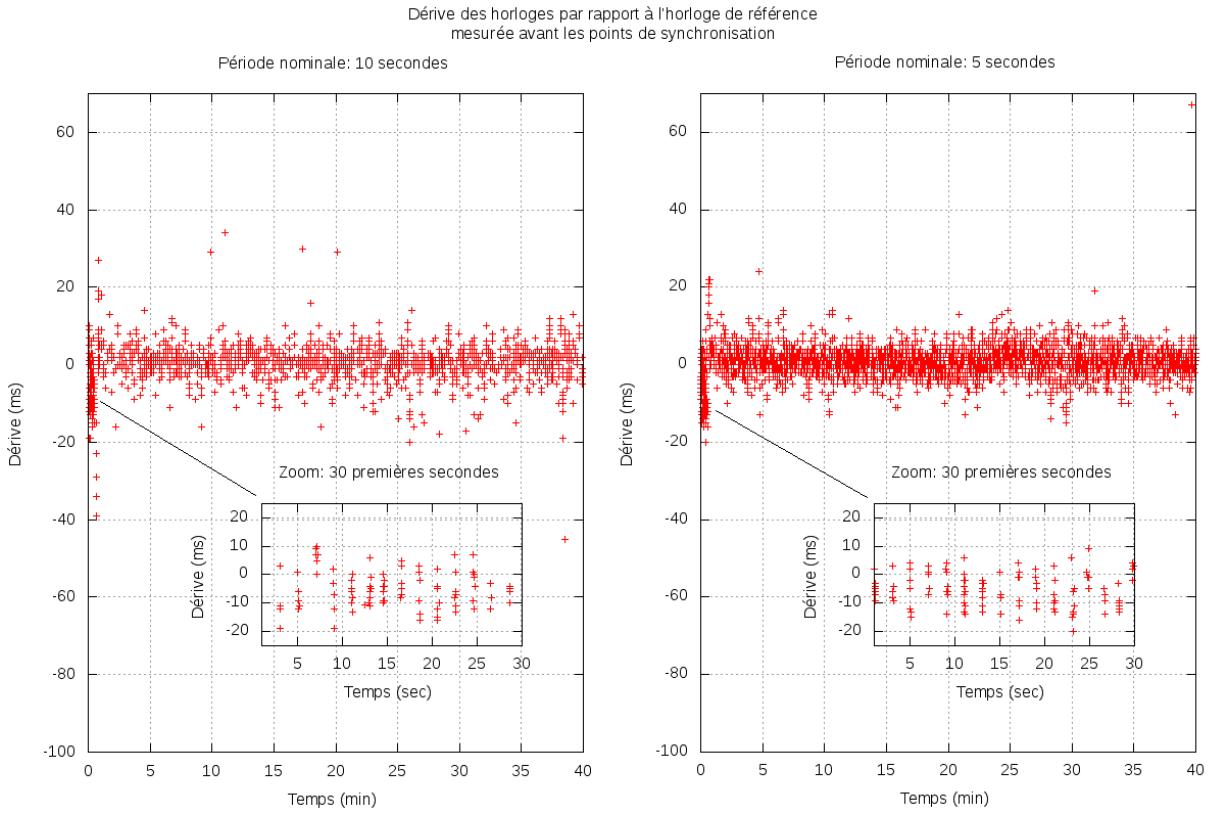


Figure 11: Dérive des horloges par rapport à l'horloge de référence.

Les résultats obtenus avec d'autres configurations sont moins bons, mais restent satisfaisants (cf. figure 12). Respectivement, 99.6% et 98.7% des dérives mesurées sur 10 minutes sont inférieures à 20 millisecondes.

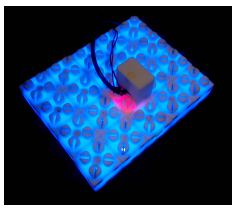
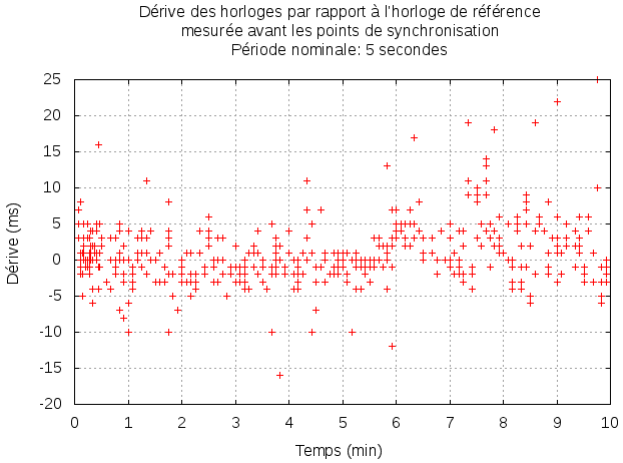
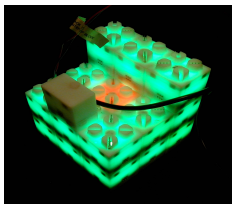
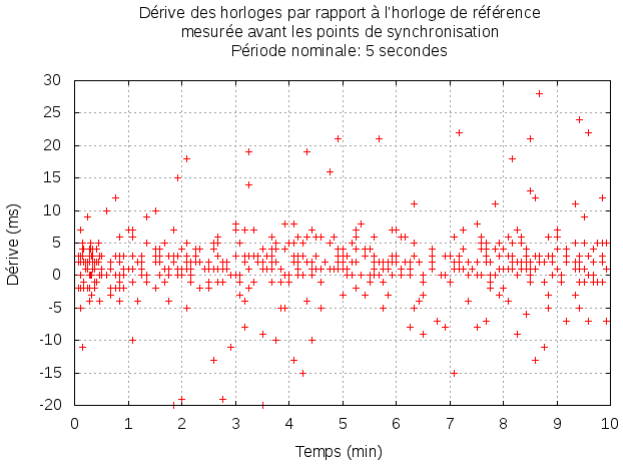
Photos	Nom	Dérive
	Dense 2D: 4*5	
	Dense 3D: 3*3*2	

Figure 12: Évaluation du protocole de synchronisation avec différentes formes.

8 Vérification expérimentale de la nécessité de BBTP

Cette partie illustre la nécessité de BBTP à travers l'exemple de changements de couleur périodiques (3 secondes), en parallèle dans un système synchronisé et dans un système non-synchronisé (cf. figure 13).

Dans le système non-synchronisé, les changements de couleur paraissent décalés dans le temps dès le deuxième changement de couleur. Au bout de 4 minutes d'expérience, un des blocs a plus de 3 secondes de retard par rapport au reste du système. Après une heure d'expérience, le système parait totalement désynchronisé. Ainsi, sans synchronisation, les horloges locales dérivent très vite les unes par rapport aux autres. Le système avec BBTP reste quand à lui totalement synchronisé pendant plus d'une heure d'expérimentation. Le maître du temps y reste en rouge.

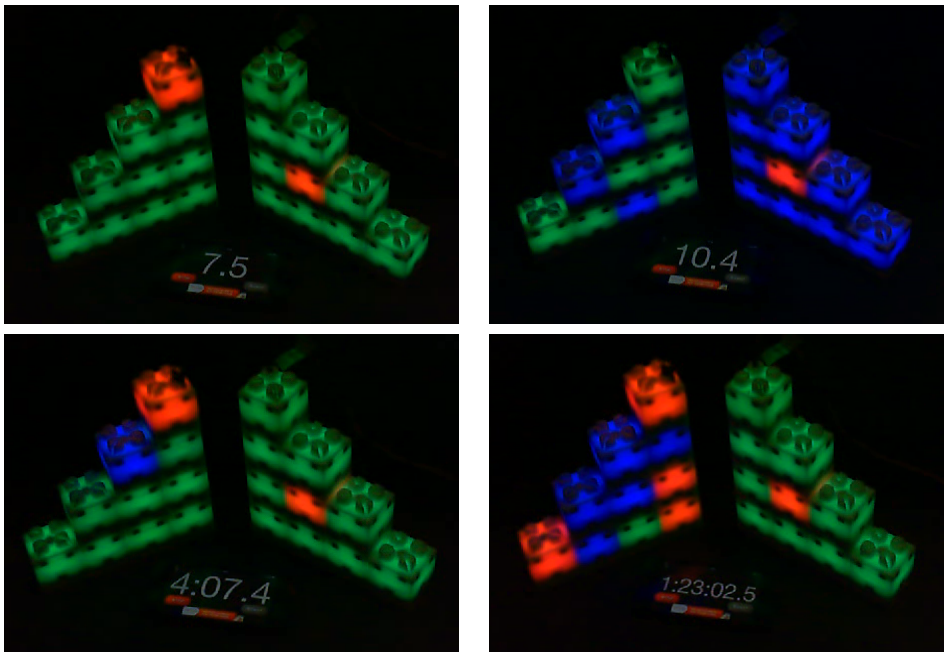


Figure 13: Comparaison entre un système non-synchronisé (gauche) et un système synchronisé (droite).

9 Conclusion

A travers de rapport, nous avons proposé BBTP : Blinky Blocks Time Protocol, un protocole de synchronisation des horloges internes des Blinky Blocks.

BBTP a été codé en langage C et a été intégré au firmware des Blinky Blocks. Dans BBTP, un Blinky Block est élu par le système pour synchroniser périodiquement les autres Blinky Blocks. Nous avons mené des expériences sur plusieurs configurations de Blinky Blocks matériels afin d'étudier la robustesse de BBTP: ce protocole peut synchroniser correctement jusqu'à 27 775 Blinky Blocks avec une précision de 40 millisecondes pour 99% du temps.

10 Travaux futurs

Les Blinky Blocks forment un réseau point-à-point où chaque bloc ne peut communiquer qu'avec ses voisins directs. Il est évident que plus un bloc est loin du maître du temps plus l'estimation de l'heure reçue dans les messages de synchronisation est imprécise. Ainsi, il paraît plus judicieux de placer le maître du temps au centre du système. De plus, placer le maître du temps au centre du système permet aussi de minimiser le temps nécessaire à ce que tout le système soit synchronisé, et de minimiser la charge du réseau liée à la synchronisation. Les algorithmes existants d'élection d'un noeud central dans les systèmes distribués [9] [10] [4] sont trop coûteux soit en temps, soit en mémoire, pour être utilisés par des dispositifs embarqués restreints en capacité de calcul et de stockage. Ainsi, nous travaillons actuellement sur l'élaboration d'un algorithme d'élection de leader au centre du système, afin de permettre l'utilisation de BBTP sur des systèmes de plus grande taille.

De plus, pour permettre un développement plus rapide d'applications temps réel pour Blinky Blocks, nous sommes aussi actuellement en train d'intégrer BBTP dans notre simulateur VisibleSim.

L'étape suivante dans CO_2Dim est la modification de Meld pour permettre au programmeur de spécifier des contraintes de temps. Cette étape va être réalisée à l'Université Polytechnique de Hong-Kong par le professeur Jiannong Cao et son équipe. Deux modifications majeures sont nécessaires. Dans un premier temps, il est nécessaire de modifier le langage Meld, c'est-à-dire d'y ajouter des éléments de langage afin de spécifier ces contraintes de temps. Dans un deuxième temps, il faut intégrer dans Meld un ordonnanceur temps réel qui garantira le respect des contraintes de temps fixées. Ces modifications de Meld exploiteront pleinement les outils de synchronisation temporelle décrits dans ce rapport.

References

[1] Michael P. Ashley-Rollman, Seth Copen Goldstein, Peter Lee, Todd C. Mowry, and Padmanabhan Pillai. Meld: A declarative approach to programming ensembles. In *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS '07)*, October 2007.

[2] Julien Bourgeois, Jiannong Cao, Michel Raynal, Dominique Dhoutaut, B Piranda, Eugen Dedu, Ahmed Mostefaoui, and Hakim Mabed. Coordination and computation in distributed intelligent mems. In *Advanced Information Networking and Applications (AINA), 2013 IEEE 27th International Conference on*, pages 129–136. IEEE, 2013.

[3] Dominique Dhoutaut, Benoît Piranda, and Julien Bourgeois. Efficient simulation of distributed sensing and control environments. In *iThings 2013, IEEE Int. Conf. on Internet of Things*, pages 452–459, Beijing, China, August 2013.

[4] Antoine Dutot, Damien Olivier, and Guilhelm Savin. Centroids : a decentralized approach. In *ECCS - European Conference on Complex System*, Vienna, Autriche, September 2011.

[5] Jeremy Elson, Lewis Girod, and Deborah Estrin. Fine-grained network time synchronization using reference broadcasts. *ACM SIGOPS Operating Systems Review*, 36(SI):147–163, 2002.

[6] IEEE. Ieee 1588-2008: Standard for a precision clock synchronization protocol for networked measurement and control systems. Technical report, IEEE, 2008.

[7] Brian T. Kirby, Michael Ashley-Rollman, and Seth Copen Goldstein. Blinky blocks: a physical ensemble programming platform. In *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '11, pages 1111–1116, New York, NY, USA, 2011. ACM.

[8] H. Kopetz and W. Ochsenreiter. Clock synchronization in distributed real-time systems. *IEEE Trans. Comput.*, 36(8):933–940, August 1987.

[9] E Korach, D Rotem, and N Santoro. Distributed algorithms for finding centers and medians in networks. *ACM Trans. Program. Lang. Syst.*, 6(3):380–401, July 1984.

[10] Marco Mamei, Matteo Vasirani, and Franco Zambonelli. Self-organizing spatial shapes in mobile particles: The tota approach. In *Engineering Self-Organising Systems*, pages 138–153. Springer, 2005.

[11] Miklós Maróti, Branislav Kusy, Gyula Simon, and Ákos Lédeczi. The flooding time synchronization protocol. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 39–49. ACM, 2004.

[12] David L Mills. Internet time synchronization: the network time protocol. *Communications, IEEE Transactions on*, 39(10):1482–1493, 1991.

[13] Benoît Piranda, Guillaume J Laurent, Julien Bourgeois, Cédric Clévy, Sebastian Möbes, and Nadine Le Fort-Piat. A new concept of planar self-reconfigurable modular robot for conveying microparts. *Mechatronics*, 23(7):906–915, 2013.

[14] Michel Raynal. *Distributed algorithms for message-passing systems*. Springer, 2013.

[15] Todd C. Mowry Seth Copen Goldstein. A logicial approach to programming concurrent systems.

[16] John A. Stankovic. Misconceptions about real-time computing: A serious problem for next-generation systems. *Computer*, 21(10):10–19, October 1988.

Annexe 1

```
#include "handler.bbh"
#include "block.bbh"
#include "led.bbh"
#include "ensemble.bbh"
#include "block_config.bbh"
#include "memory.bbh"

#define MYCHUNKS 12

Chunk myChunks[MYCHUNKS];
threadvar byte waiting;

void freeAllMyChuncks(void);
Chunk* getNextChunk(void);
byte propagateColor(Color c);

void myMain(void)
{
        freeAllMyChuncks();
        waiting = 1;
        if(getGUID() == 1) {
                waiting = 0;
                delayMS(1*1000);
                setColor(RED);
                propagateColor(RED);
        }
        while(1);
}

byte messageHandler(void)
{
        if(thisChunk == NULL) {
                return 0;
        }
        if (!waiting) {
                return 1;
        }
        waiting = 0;
        setColor(thisChunk->data[0]);
        propagateColor(thisChunk->data[0]);
        return 1;
}

byte propagateColor(Color c)
{
        byte msg[17];
        Chunk* cChunk;
        byte x;

        msg[0] = c;

        for( x = 0; x < NUM_PORTS; x++) {
                if (thisNeighborhood.n[x] == VACANT) {
```

```
                        continue;
                }

                cChunk = getNextChunk();
                if(cChunk != NULL) {
                        if(sendMessageToPort(cChunk, x, msg,
                    1*sizeof(byte), messageHandler, NULL) == 0) {
                                freeChunk(cChunk);
                        }
                }
        }
        return 1;
}

void userRegistration(void)
{
        registerHandler(SYSTEM_MAIN, (GenericHandler)&myMain);
}

/* Chunk management */

Chunk* getNextChunk(void)
{
        static byte i = 0;

        Chunk *c = &(myChunks[i%MYCHUNKS]);
        i++;
        return c;
}

void freeAllMyChuncks(void)
{
        byte x;

        for(x = 0; x < MYCHUNKS; x++) {
                myChunks[x].status = CHUNK_FREE;
                myChunks[x].next = NULL;
        }
}
```

Annexe 2

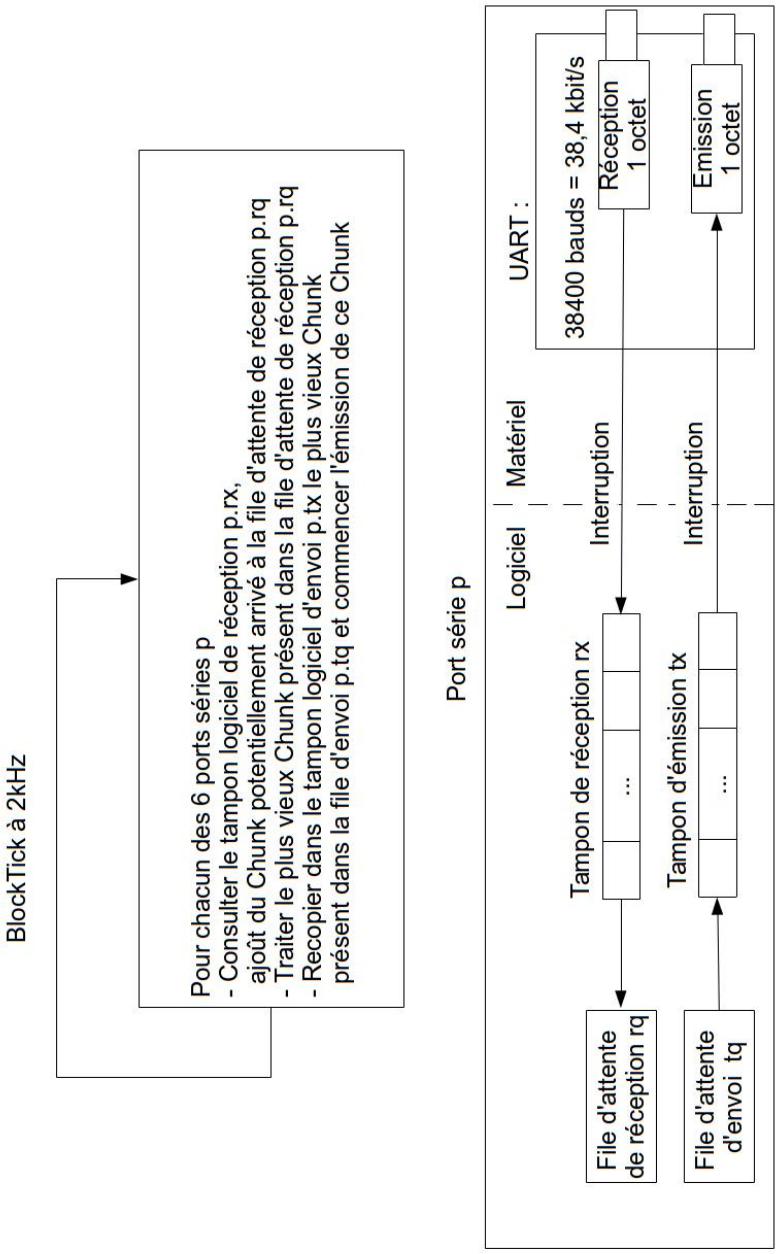


Figure 14: Système de communications des Blinky Blocks.

FEMTO-ST INSTITUTE, headquarters
32 avenue de l'Observatoire - F-25044 Besançon Cedex FRANCE
Tél : (33 3) 81 85 39 99 – Fax : (33 3) 81 85 39 68 – e-mail: contact@femto-st.fr

FEMTO-ST - AS2M : TEMIS, 24 rue Alain Savary, F-25000 Besançon
FEMTO-ST - DISC : UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST - ENERGIE : Parc Technologique, 2 Av. Jean Moulin, Rue des entrepreneurs, F-90000 Belfort France
FEMTO-ST - MEC'APPLI : 24, chemin de l'épitaphe - F-25000 Besançon France
FEMTO-ST - MN2S : 32, rue de l'Observatoire - F-25044 Besançon cedex France
FEMTO-ST - OPTIQUE : UFR Sciences - Route de Gray - F-25030 Besançon cedex France
FEMTO-ST - TEMPS-FREQUENCE : 26, Chemin de l'Epitaphe - F-25030 Besançon cedex France

http://femto-st.fr