Idées de projet de PRT

# Carte uCLinux:

* Créer une carte à base de Cortex M3/4 pouvant faire tourner uCLinux
* Nécessite de la RAM externe (64Mo)
* Nécessite de la mémoire flash externe (qq Mo)
* Nécessite un ARM STM32F4: c’est les seuls chips pour lesquels j’ai trouvé des semblants d’instructions pour faire tourner uCLinux
* Pas d’assurance d’y arriver sans trop de problème: il n’y a pas énormément de ressources sur uCLinux et sur les portages.

# Cluster de Cortex M3/4:

* Création d’une carte mère basique, avec juste un Cortex M3/4 dessus et des headers. On peut prendre une Tiva comme exemple.
* Raccordement de plusieurs cartes entres elles
* Etablissement d’un modèle de programmation permettant de tirer partie du tout.
* Les différents usages:
  + Plusieurs cartes pour plusieurs fonctions (carte moteur, carte asserv, etc)
  + Plusieurs cartes pour plus de perf: nécessite un découpage de l’application. Il est sûrement préférable de découper selon les tâches, et de mettre un micro-contrôleur plus puissant (passer d’un M3 à un M4 par exemple) quand c’est nécessaire à une tâche, plutot que d’essayer de découper une même tâche sur plusieurs mcu lents.
* L’objectif global: Obtenir un cluster de cartes se programmant relativement facilement (par rapport à une carte toute seule), et extensible à souhait.
* Les différents sous projets possibles:
  + Utilisation d’une carte maître pour flasher les différents esclaves au démarrage du cluster.
    - Avantages: Cela peut permettre d’éviter de flasher les esclaves à la main (moins de branchements JTAG, cartes inaccessibles, oublie de reflasher entrainant l’usage de version incompatibles de soft sur certaines cartes).
    - Inconvénients: Nécessite d’avoir de la mémoire en rab sur la carte maître, car elle devra stocker le programme de toutes les cartes filles.
  + Mise au point d’un système permettant de programmer ce genre de cluster en facilitant la communication et la mise au point d’une application distribuée sur le cluster. J’avais déjà fait un travail assez conséquent dans ce sens là l’été dernier, le résultat était fonctionnel.
* Le matériel:
  + Usage d’ARM Cortex M de chez TI car:
    - Fonctionnalités attendues (bus, mémoire, etc.)
    - Documentation au top
    - Suite de développement au top:
      * IDE Linux/Windows/Mac basé sur Eclipse (Code Composer)
      * Chaîne de compilation + débuggeur + JTAG entièrement supportés sous Linux (cf l’expérience de Nico). C’est du gcc et gdb mainstream, et le JTAG c’est openocd comme serveur gdb.

# Soft pour cluster

Plusieurs solutions:

* Uniquement simplifier la communication, en gérant la partie décapsulation/encapsulation et attente de messages (fifo)
* Faire du RPC de manière implicite: pouvoir prendre n’importe quelle fonction, et faire tourner son code sur une board distante plutôt que localement. Exemple:

La fonction move\_arm permet de bouger un bras du robot. Problème: le hard du controle du bras a été changé de board, car il ne marchait pas sur la carte de départ, ou on a voulu réutiliser l’UART qu’il utilisait par exemple. Seulement le soft de la board de départ appelait move\_arm directement, et le modifier pour envoyer des messages à la nouvelle board fait perdre du temps et n’est pas forcément trivial. Solution: move\_arm sur la board de départ est remplacé par un stub, qui s’occupe de contacter la nouvelle board et d’executer la fonction en lui envoyant ses paramètres et en récupérant le résultat.

L’inconvénient, c’est que les les variables globales ne seront pas synchronisées entre le deux board. De la même manière, toutes les fonctions appelés par move\_arm doivent être implémentés sur la nouvelle carte. Ceci dit, si aucune variable globale n’est utilisée dans tout le code appelé par move\_arm, ça peut marcher sans trop de soucis. Il y a aussi moyen de vérifier qui c’est bien le cas (analyse du code source avec Clang, ce genre de truc se fait sans trop de problèmes).

C’est un peu faire du monkey patching, pour le coup, ça présuppose pas un truc qui marche bien au départ, ça réduit juste le coût soft des modifs de l’archi hard ^^

Nico => Pour le club il y a quelque chose qui nous semble miraculeux c’est la modularité !

Mais je comprends pas pourquoi ! On a toujours tendance à partir en se disant on va faire un système qui va fonctionner nikel et comme ça on le touchera plus et on pourra juste changer ou remplacer un carte chaque année.

Je pense pas que ça soit vraiment une bonne idée parce que le savoir se perds et les gens ne savent plus pourquoi on fait comme ça.

Je sais pas trop quelle est la solution mais je pense que tous les ans il faut refaire les cartes, les améliorer et toucher à tous le système. Après les gros morceaux qui prennent du temps ne sont pas toucher chaque année.

Cette année par exemple en refaisant le système j’ai compris que certains choix sur l’ancien système était finalement mieux mais il m’a fallut m’y frotter pour le comprendre.

Ensuite je ne comprends pas pourquoi on peut pas souder des composants comme des M4 ou des M3 … D’autres club avec moins de moyens le font …

Dans les procs il y a un soucis c’est que dès qu’on veut sortir d’un proc par tâche et faire une guirlande il faut sortir des petits uC …

L’enjeu d’un cluster de M3/M4 c’est de pouvoir justement avoir moins de cartes mais plus puissantes axées autour d’un gros proc.

La BBB centralise ce cluster et le commande plus facilement. Et elle permet de flasher les procs.

Douglas => @Nico

Pour ce qui est de la modularité et pourquoi ça semble miraculeux, regarde du côté de ROS ;). Ça permet globalement de faire des choses très intéressantes, très vite, notamment en phase de prototypage. Si trop de choses sont à réaliser de manière ad-hoc, tester plusieurs solutions prend un temps tellement grand que au final, on n’en teste qu’une. Après c’est peut être pas adapté au club en effet.

Du coup au final, tu penses quoi du genre de solutions que je propose ? Ça te paraît trop entrer dans la modularité problématique ou au contraire que ça permet de faire un truc suffisamment intégré pour pas poser de contraintes trop fortes à la création ?

L’avantage du cluster de M3/4, c’est en effet d’avoir des noeuds corrects niveau perf/fonctionnalités/outillage de dev. L’approche du reflashage en live peut simplifier le cycle de vie d’une appli.

Après si je comprends bien, tu es plutôt pour utiliser des M3 en multitâche, plutôt que de faire un M3<=>une fonctionnalité. L’avantage est en effet de pouvoir faire des cartes comme on veut, avec autant de trucs qu’on veut, sans tomber dans le sur-découpage. L’inconvénient, c’est que quand on procède comme ça, on se retrouve souvent à faire des trucs un peu bancals en terme de communication entre les blocs, parce que ce n’est plus indispensable à chaque bloc. Et au final, ça peut poser des problèmes d’intégration ou tout simplement limiter les choses possibles dans les tâches de supervision de haut niveau (si les boîtes sont trop noires et avec trop de fonctionnalités disparates, ça peut poser soucis à une IA par exemple).

Selon moi, c’est le principal problème à résoudre. Si c’est fait de manière satisfaisante, on peut se permettre de faire du from scratch et aggréger les fonctionnalités sur une même board sans que ça pose de problèmes. D’où l’idée de proposer un système de com niveau soft qui soit pas trop invasif, mais qui augmente la maintenabilité et permette de faire un truc bien foutu sans y passer des heures. En bref, que ça soit plus intéressant du point de vue temps passé à la mise en place d’utiliser ça plutôt que de céder à un flux de float sur une UART pour donner les positions par exemple ^^. (dans cet exemple, le protocole est tellement simple que c’est ultra rapide à mettre en oeuvre. Cependant, une extension du protocole est compliquée et sera probablement faite de manière bancale pour pas recommencer les applis ces 2 côtés.)

Un autre point qui est amélioré par cette méthode, c’est que si les différentes API sont séparées, même si elles tournent sur la même board à la base, ça permet de les réimplémenter sur une board séparée en cas de pb. Par exemple, si il s’avère que une fonctionnalité hard déconne sur une board, il faut pouvoir la réimplémenter séparément, sans que ça soit trop compliqué et trop long. Hors, si on a des grosses board qui font trop de trucs, ça sera très couteux car il faudra modifier toutes les parties du soft impactée par la séparation en plusieurs cartes.

Si on arrive à résoudre ce problème de manière satisfaisante (pas forcément parfaite), ça permettrait quand même de pouvoir se resservir de ce qu’on peut, sans pour autant toucher à tout même quand on s’en passerait.