Cours Système Embarqué avec µClinux

Présentation de la carte STM32F429I-DISC0/1

La carte, fabriquée par STMicroelectronics, repose sur le microcontrôleur STM32F429ZIT6 comprenant un ARM Cortex-M4 à 180 Mhz, 2 Mo de flash et 256 Ko de RAM. Le kit est complété de 64 Mb de SDRAM (8 Mo), d'un écran LCD TFT QVGA (240×320) de 2"4, d'un motion sensor 3 axes, d'un connecteur USB OTG et de la classique interface de développement ST-LINK/V2 avec alimentation (via USB). La carte DISC1 est équipée du ST-LINK/V2b avec Virtual COM port.

Le point très intéressant, en dehors de la présence de l'écran est, bien entendu, la SDRAM directement interfacée avec le STM32F4x. Ceci, couplé aux 2 Mo de flash laisse envisager l'utilisation de systèmes très complets nécessitant un volume non négligeable de mémoire.
Le microcontrôleur présent, lui-même, est très bien équipé avec ces 4 USART, 4 UART, 6 SPI, 3 I2C, 2 CAN, son accélérateur graphique dédié à l'interface LCD, 17 timers, un module crypto hardware, FPU, DSP, RNG, un gyroscope 3 axes MEMS L3GD20 avec capteur de température, etc.

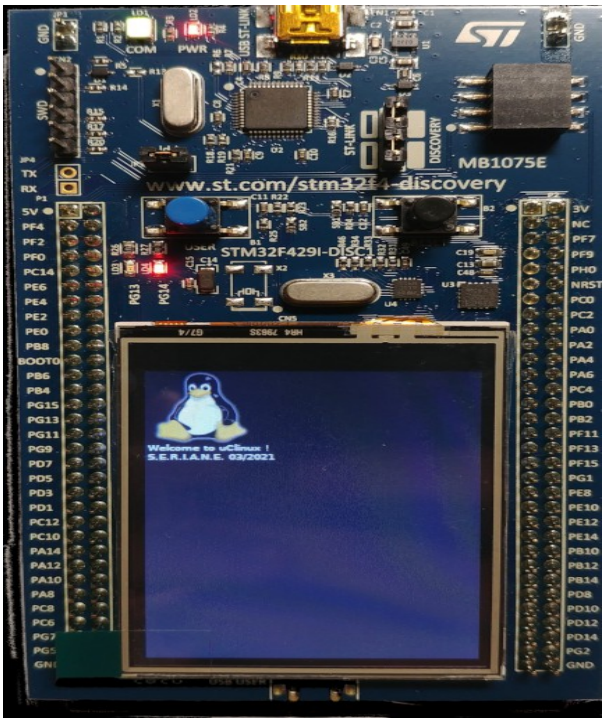
L'ensemble est disponible à la vente, sous blister, pour **moins de 35 euros**.

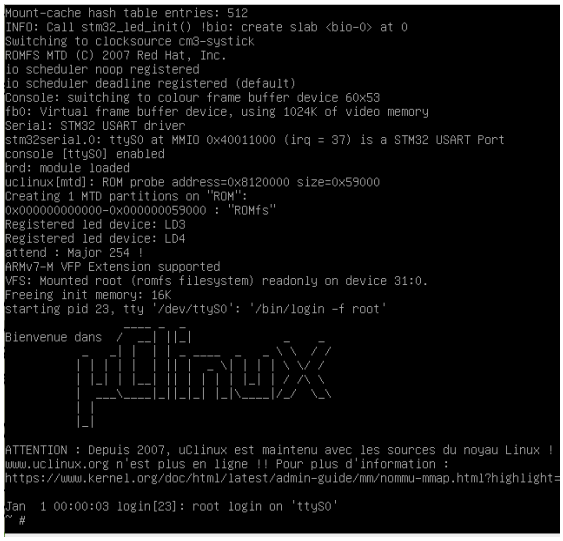
Avec l'arrivée des microcontrôleurs 16 et 32 bits, les ressources et les fonctionnalités offertes ont été décuplées. Naturellement, le développement « bare metal » où un code est exécuté seul sur la plateforme laisse doucement la place, dans certains domaines, à des architectures plus évoluées : des systèmes d'exploitation.
Certains des plus connus dans le domaine de l'opensource sont TinyOS, Contiki, Lepton, BeRTOS, Nut/OS, ScmRTOS, CooCox, Chibios/Rt, Ecos et surtout FreeRTOS. Mais les systèmes offrant des facilités et des primitives particulières pour le développement d'applications temps réel ne sont pas les seuls. L'écosystème des OS pour microcontrôleur est plein de vie actuellement. Parmi les nombreuses solutions, un nom revient souvent : µClinux.

Pourquoi µClinux ?

Mais pourquoi vouloir absolument disposer d'un noyau Linux/µClinux sur une aussi petite plateforme comme peut l'être un Cortex M3/M4 ? La réponse est simple et tient en deux mots : **stabilité et fonctionnalité**.
L'évolution vers les microcontrôleurs 32 bits et le Cortex M en particulier découle directement du besoin d'intégration de nouvelles fonctionnalités dans les systèmes embarqués : USB OTG, GUI, carte SD avec FAT, wireless, NFC... Autant de choses qu'il n'est certes pas impossible d'accomplir sans un OS mais qui, d'une plateforme à l'autre, change énormément en termes d'implémentation (malgré la généricité de certaines solutions). L'intégration de ces fonctionnalités nécessite le développement d'un support et souvent l'acquisition de ces connaissances de bout en bout.
S'adapter à un système d'horloge d'un microcontrôleur AVR à un MSP430 demande déjà du travail.
Imaginez le temps et l'énergie nécessaires s'il faut faire de même avec l'USB au plus bas niveau...
Il existe cependant une alternative : Linux.

Avec un code excessivement audité et robuste, le noyau Linux intègre de manière fiable et documentée une masse très importante de fonctionnalités qui ne nécessitent alors que quelques ajustements pour être adaptées à une plateforme. Il n'est plus nécessaire de développer l'ensemble du système mais uniquement de le porter sur la nouvelle plateforme.
Un problème est cependant à prendre en considération : le noyau Linux est « taillé » pour un cahier des charges minimal qui n'est pas celui auquel peut répondre un microcontrôleur 16 ou 32 bits comme un Cortex M. Un certain nombre d'éléments doivent, en effet, être présents comme une certaine quantité de mémoire ou encore certaines fonctionnalités du processeur comme la présence d'un MMU (Memory Management Unit). C'est là le point de différence majeur entre Linux et µClinux. Ce dernier est destiné aux architectures dites « MMU-less ».





On se « logue » sur la carte à l'ancienne, via une liaison série, qui nécessite un petit câblage sur la version DISC0. L'écran tactile LCD TFT est disponible pour des applications graphiques.

µClinux a vu le jour en 1998 avec pour plateforme principale de démonstration, le processeur Motorola 68000 du PalmPilot III. Des portages Coldfire et ARM ont rapidement suivi, ainsi que le passage des développements d'une base Linux 2.0 à 2.4 puis 2.6. On retrouve ainsi µClinux dans bon nombre de produits, du routeur à la caméra IP en passant par toutes les « appliances » multimédias.

µClinux intègre tout ce qui fait la force de Linux comme par exemple la pile IP, l'ensemble des systèmes de fichiers supportés de base, le sous-système USB, la facilité de portage des applications, une masse importante de bibliothèques, outils et applications, etc. Tout cela, sans support MMU, objet même de l'existence de µClinux.

Autre élément important : depuis 2007 il est maintenu avec les sources du noyau Linux, ce qui signifie une reconnaissance, de la part de l'équipe de Linus Torvalds, de la pertinence et du sérieux du projet. Et ce qui apporte une meilleure fusion des outils communs aux deux familles de noyau.

Attention cependant, car cela n'est pas une mince affaire sans MMU :

- Pas de mémoire virtuelle et donc une nécessité pour le noyau d'utiliser d'autres mécanismes pour régler le problème de fragmentation mémoire.
- Pas de protection mémoire, tout est partagé. Une application peut utiliser et corrompre la mémoire du noyau ou celle des autres applications. Cette gestion repose ainsi plus que jamais sur le développeur et la qualité de son code. Dans le cas d'un système embarqué, ce genre de chose est

normalement de mise car une application qui crée des « segmentation fault » signifie de toutes façons des problèmes pour les utilisateurs.
- Pas de swapping. Là encore ceci n'est pas vraiment un problème étant donné que ce genre de fonctionnalité n'a pas lieu d'être dans l'embarqué (enfin... normalement).
- Pas de changement de contexte car pas de VM et d'espaces d'adressage séparés entre noyau et application. Ceci signifie plus un avantage qu'un handicape car synonyme de gain en rapidité.
- Pas d'appel système *fork* ou du moins pas dans le sens strict puisque *vfork* est disponible et, dans les grandes lignes, partage un fonctionnement très similaire au *fork* de Linux.

µClinux est donc parfaitement adapté si l'on cherche à disposer de toutes les fonctionnalités de Linux en contrepartie des limitations imposées qui, rappelons-le, sont également présentes avec d'autres OS pour une même plateforme modestement équipée.

Les possibilités offertes

Ce projet ouvre la voie pour faire découvrir, à des élèves ayant déjà une bonne pratique du langage C, un ensemble de choses indispensables pour ceux qui veulent faire carrière dans les systèmes embarqués, avec entre autres :

- ✔ la compilation croisée,
- ✔ les systèmes de gestion de fichiers dédiés pour l'embarqué,
- ✔ l'étude de boot-loader (ici u-boot),
- ✔ le paramétrage et la compilation d'un noyau Linux (ici 2.6.33),
- ✔ la configuration et la mise en place d'une busy-box,
- ✔ les outils open source pour flasher de la mémoire à travers une interface JTAG,
- ✔ la gestion des arborescences de code avec *make*,
- ✔ l'intégration du code d'une application dans le système complet,
- ✔ la programmation en multi-threading,
- ✔ les accès simples au hardware (leds, boutons) en *user space*,
- ✔ le développement de drivers pour les nombreuses interfaces (GPIO, I2C, …),

  etc.

Notons que la carte DISC0 nécessite de lui ajouter un adaptateur convertisseur USB vers TTL de 3,3 V pour s'y connecter en mode console. Cela peut faire l'objet d'un petit TP. La version DISC1 permet de s'y connecter via la liaison USB de l'alimentation, mais ce qui n'empêche pas d'y mettre également un adaptateur pour une seconde liaison.

Pour toute information complémentaire : **secretariat@seriane.org**