Cible p-Soc pour l'apprentissage de Linux en DUT GEII

P. Aygalinc[a], S. Calvez[b]

[a] Dép. GEII, IUT d'Annecy et LISTIC, Université de Savoie-Mont Blanc, France
[b] LISTIC, Polytech Annecy Chambéry, Université de Savoie-Mont Blanc, France
Contact email : pascal.aygalinc@univ-smb.fr

Les systèmes embarqués modernes utilisent souvent comme système d'exploitation ***Linux***. Pour le ***DUT GEii***, les architectures ***p-Soc*** (***p****rogrammable-****S****ystem* ***o****n* ***c****hip*) trouvent grandement leur intérêt car elles permettent, en plus de l'apprentissage de ce système sur ses deux niveaux (***user***/***kernel***), d'entretenir et d'enrichir les connaissances acquises sur la description matérielle et l'informatique industrielle bas niveau. L'expérience menée ici décrit la plateforme développée dans ce cadre afin d'aborder d'une part les objectifs d'un système d'exploitation et de ses propriétés pour le développement d'applications de contrôle/commande en mode *user*, et d'autre part la conception de pilotes de périphériques du mode *kernel* sur des composants *custom* décrit en ***VHDL***. Cet article traite aussi des prérequis nécessaires ainsi que des moyens indispensables à mettre en œuvre par l'enseignant pour la mise en place de cet enseignement (*prototypage rapide aussi bien au niveau matériel que logiciel*).

I. Introduction

Compte tenu de ses propriétés de portabilité, ***Linux*** (1) est devenu incontournable pour la réalisation de systèmes embarqués qu'ils soient communicants ou non. Ceux basés sur des architectures ***p-Soc*** (***p****rogrammable* ***S****ystem* ***o****n* ***c****hip*) apparaissent comme d'excellentes cibles pour l'apprentissage de ce système d'exploitation car ils permettent de découvrir naturellement *Linux* aussi bien au niveau ***user*** qu'au niveau ***kernel***. De plus, comme il est possible d'ajouter sur ce type de cible des ***composants custom*** plus ou moins complexes, ceci autorise un enseignement progressif de leur utilisation dans le monde *user* et il en est de même dans le monde *kernel* lors de la conception de leur pilote associé.

L'objet de cet article est de montrer que dans le cadre des formations de type *DUT* en ***G****énie* ***É****lectrique et* ***I****nformatique* ***I****ndustrielle* (*GEII*) les cibles ***p-Soc*** par rapport à des systèmes ***Soc*** comme la *Raspberry PI (2), l'Aria (3), ...* sont un plus pour découvrir le fonctionnement d'un système d'exploitation (*mode user/kernel, notion de processus, d'attente passive/active, virtualisation, multiprogrammation, ...*). En effet, tout en découvrant les apports d'un système d'exploitation pour le développement d'applications en informatique industrielle, les cibles ***p-Soc*** permettent aussi d'entretenir les connaissances acquises lors des deux premiers semestres de cette filière (*langage de description matérielle*, *langage informatique structuré, programmation bas niveau, outils de modélisation*). Ces prérequis seront décrits au paragraphe 2, et sont indispensables pour la mise en œuvre des concepts abordés lors de l'apprentissage d'un système d'exploitation de type ***Linux*** basé sur ce type de cible.

Article publié par EDP Sciences et disponible sur le site https://www.j3ea.org ou https://doi.org/10.1051/j3ea/20191008

La maquette ***p-Soc*** réalisée a été conçue en fonction des objectifs pédagogiques à atteindre (figure 1). Le paragraphe 3 décrira la partie matérielle de cette maquette, ainsi que les développements qu'elle a nécessités. Le paragraphe 4 donnera le contenu de l'enseignement sur le système *Linux* qu'elle permet d'aborder. Il constitue un module complémentaire d'environ une trentaine d'heures, et s'inscrit dans le parcours orienté informatique qu'offre le département *GEII d'Annecy* au semestre 3.

Fig.1. Plateforme de type p-Soc pour l'apprentissage du système d'exploitation Linux

Au paragraphe V, on traitera des moyens matériels et logiciels à mettre en œuvre par l'enseignant pour arriver à rendre opérationnelle une salle informatique équipée de plateformes ***p-Soc***.

Pour conclure, un bilan sera fait concernant les possibilités de créativité qu'offre ce type de cible et ce qu'il est possible de faire durant un TP au niveau *user*, puis au niveau *kernel* lors de l'ajout d'un composant *custom*.

II. Les prérequis

Les formations en *IUT* sont régies par des ***P****rogrammes* ***P****édagogiques* ***N****atio*naux (*PPN*) (4). Ils précisent les objectifs de la formation, les volumes horaires, les modalités pédagogiques et de contrôles de connaissances et des aptitudes. Pour la partie pratique, le département est libre de ses choix technologiques.

En *GEII* à *l'IUT d'Annecy*, au semestre 1, les étudiants découvrent l'électronique numérique, un langage de description matérielle (*le VHDL*) et la modélisation des systèmes séquentiels à l'aide de graphe d'états *(module M1102)*. Pour la partie TP, les cartes sont des *DE2-Board* de l'intégrateur *Terasic*® (5) équipées de *FPGA* du constructeur *Intel FPGA*® (6) (*anciennement Altera*®). L'environnement de développement est *Quartus* et *ModelSim* pour la partie simulation. Parallèlement, la programmation classique est enseignée à l'aide du *langage C (module M1103)*.

Au semestre 2, l'initiation à la programmation bas-niveau est effectuée dans un module d'informatique industrielle utilisant des microcontrôleurs *(module M2103)*. La configuration des coupleurs, les mécanismes d'accès aux données de ces derniers *(mappage en mémoire, scrutation cyclique ou sur interruption*) sont alors traités sur des périphériques de base (*port d'entrée/sortie, timer, liaison série, bus i2c,…*). Dans ce semestre débute également un module d'automatisation (*module M 2102)* abordant la modélisation du parallélisme et de la synchronisation à l'aide de l'outil de spécification *Grafcet*, et le fonctionnement d'un automate (*exécution périodique du cycle en mode Run, repli des sorties en mode Stop*).

Au début du semestre 3, la programmation « objet » est enseignée (*module M3105 C*) et fournit d'autres outils de modélisation comme les diagrammes de séquences qui, appliqués à des processus concourants, permettent de décrire leur dialogue via des messages.

Pour renforcer ces apprentissages, des projets tuteurés ainsi que des travaux de réalisation sur ces thèmes sont aussi effectués. Ainsi, en fin de semestre 2, les étudiants disposent de compétences suffisantes pour aborder une programmation de type « *baremetal* » sur une plateforme matérielle utilisant des périphériques d'E/S classiques de l'informatique industrielle. Il convient alors de leur faire découvrir au semestre 3 l'apport d'un système d'exploitation pour établir des applications de contrôle/commande à l'aide d'une plateforme ***p-Soc*** tout en consolidant ce qui a été vu lors des deux premiers semestres.

III. La plateforme p-Soc développée

Partie matérielle

Afin de conserver les acquis sur l'environnement *Quartus*, la cible ***p-Soc*** est basée sur une carte de développement de type *DE1-SoC* (5) intégrant un *Cyclone V* du constructeur *Intel FPGA*® (6).

Parmi les interfaces que proposent cette carte, ont été retenues pour la découverte de *Linux* celles que l'on rencontre classiquement en informatique industrielle telles que les afficheurs sept segments, les boutons poussoirs et le convertisseur analogique digital (*deux de ses entrées sont reliées respectivement à un potentiomètre et l'autre à un capteur de luminosité passif de type LDR*). Les dix leds disponibles sont utilisées comme un « *flux de debug* » pour la partie matérielle ; elles permettent ainsi de visualiser facilement l'état interne des composants *custom* (*état d'un registre, état d'un graphe, ...*) sans faire appel à la simulation conjointe (*software/hardware*) hors de portée du public visé.

Pour compléter ces interfaces, une carte fille a été développée. On dispose alors des ressources suivantes :

- un écran ***oled*** (7) accessible via une liaison série qui autorise l'emploi d'une ***IP*** (***I****ntellectual* ***P****roperty*). Son intégration permet la découverte de l'outil « ***Platform Designer*** » (*anciennement Qsys*) intégré à *Quartus* et un début d'apprentissage du bus ***Avalon*** (*configuration de l'instance de l'IP, ses connections au bus et avec le niveau supérieur, décodage mémoire, ...*). Par ailleurs, il est à noter que la page d'accueil qu'il propose lors de son démarrage permet d'afficher des informations mettant en garde les étudiants sur les ressources que réclame le démarrage de la plateforme comme on peut le voir sur la vidéo de la figure 2 du paragraphe 5,
- trois paires d'afficheurs sept segments qui permettent d'instancier plusieurs fois le même composant *custom* mais sur des régions mémoire différentes,
- un écran ***lcd*** (8) dont le composant *custom* a pour objectif ici une adaptation temporelle (*temps de setup, temps de maintien*) entre les signaux du bus *Avalon* et ceux de l'écran. C'est ici l'aspect dynamique des différents bus qui peut être vu et approfondi,
- une matrice 3*3 composée de neuf leds ***RVB*** dimmables. Afin de limiter le nombre de fils de commande, la structure matricielle adoptée permet de découvrir les aspects ***co-design***. Un composant assure le décodage périodique de la colonne à afficher et la synchronisation des différentes ***PWM*** dont les valeurs temporelles sont fixées par un autre composant accessible, lui, à l'aide du bus *Avalon*,

Il est clair que des composants *custom* nécessaires pour certains périphériques atteignent de tels degrés de difficulté qu'il n'est pas possible de les aborder sur une séance de tp au niveau DUT. En voici quelques exemples :

- pour le convertisseur analogique digital, il nécessite une sérialisation d'un registre pour transmettre l'ordre de conversion en respectant un certain protocole, et une dé-sérialisation pour obtenir la valeur convertie précédente,

- pour les boutons poussoirs, ils sont munis d'un filtre anti-rebond programmable et afin d'autoriser des attentes passives au niveau du noyau, ils peuvent en fonction de leur configuration provoquer une interruption sur un des cœurs du ***HPS*** (***H****ard* ***P****rocessor* ***S****ystem*) sur chaque changement d'états. Ils disposent alors d'un registre d'état dont la lecture permet de connaitre s'ils sont responsables de l'interruption levée et de l'acquitter si c'est le cas.

Partie logicielle

Pour tous les périphériques retenus à l'exception de l'écran *oled*, des pilotes de type « *caractère* » ont été développés (***misc driver***). Dans la mesure du possible, ils doivent permettre une utilisation des périphériques à l'aide des deux systèmes de fichiers virtuels que propose *Linux* (*devfs et sysfs*), ainsi qu'une politique d'accès unitaire (*i.e. dès qu'un processus réserve via le service* ***open****( ) un périphérique*, *il est le seul à pouvoir l'utiliser jusqu'à sa libération obtenue en effectuant l'appel system* ***close*** *( ))*.

- Pour chaque paire d'afficheurs sept segments, le pilote permet d'obtenir sous *devfs*, deux fichiers de périphérique dont l'un assure le décodage *Ascii/7Segments* alors que l'autre effectue le décodage *Hexadécimal/7Segments*. L'objectif est de montrer aux étudiants que plusieurs virtualisations sont possibles pour un même périphérique à l'aide d'un même pilote ainsi que la notion de réentrance (*capacité à effectuer une même mission dans des contextes différents*). Sous *sysfs*, chaque paire d'afficheurs offre deux attributs : le premier permet de valider ou non de façon sélective l'afficheur actif (*0 : aucun, 1 : celui de droite, 2 : celui de gauche, 3 : les deux*). Le second permet de définir la valeur à afficher.
- Pour chaque bouton-poussoir, le pilote autorise des attentes passives au niveau du noyau lorsque l'on utilise son fichier de périphérique (***poll***) et offre deux attributs : l'un pour lire son état, l'autre pour modifier/connaitre le temps de maintien du filtre.
- Pour chacune des 8 voies multiplexées du convertisseur analogique digital, la valeur convertie est donnée soit à l'aide d'un fichier de périphérique, soit par un attribut sous *sysfs*.
- Pour l'écran *lcd*, aucun attribut n'est disponible et tout doit s'effectuer via le fichier de périphérique sous *devfs*. Ce dernier assure l'interprétation des codes de contrôle usuels (*LineFeed, Carriage Return, …*), et permet grâce au service ***ioctl****( )* de configurer certains de ses paramètres (*type de curseur, affichage clignotant, permanent ou pas, …*).
- Pour chacune des neuf leds *RVB* de la matrice 3*3, le pilote permet de lire/écrire l'intensité de chacune des composantes via *devfs*, et il en est de même pour *sysfs*.

La création du système de fichiers ***rootfs*** a été obtenue à l'aide du système de construction d'une distribution ***Yocto*** (9) dans lequel on a ajouté une couche (*meta-geii*) permettant d'inclure différents scripts et exécutables. Pour le noyau, il est obtenu à partir des sources disponibles sous *https://github.com/altera-opensource/linux-socfpga.git*.

IV. Objectifs pédagogiques

Le module complémentaire associé permet de découvrir un système d'exploitation et ses propriétés avec pour l'illustrer le système *Linux* (1). On restreint son apprentissage à la conception d'applications de contrôle/commande nécessitant un comportement périodique autorisant une certaine latence. C'est le *langage C* qui est utilisé pour la programmation.

Le premier TP permet à l'aide des fonctions d'horodatage la prise en main de la plateforme et des outils de développement en mode *user*. Pour familiariser les étudiants à ce que l'on peut attendre de la virtualisation des périphériques, les TPs portent tout d'abord

sur les accès synchrones avant d'aborder au travers des techniques de multiplexage la communication asynchrone (10). Voici quelques sujets de tps qui sont proposés à l'aide de la plateforme :

- lecture périodique de la valeur digitale associée au capteur de luminosité avec horodatage. La périodicité est obtenue par une mise en sommeil du processus pour maintenir la fluidité du système et un *gestionnaire de signal* est mis en place pour terminer proprement l'application Les appels système visés sont ***open( )***, ***read( )***, ***close( )*** et ***sleep( )***,
- écriture de cette valeur sur l'écran *lcd* nécessitant une configuration particulière de son curseur. On rajoute à la liste précédente des appels système les services ***write( )***, et ***ioctl( )***. Par ailleurs, pour effectuer en parallèle le décompte du temps, plusieurs solutions sont disponibles : les fonctions ***pause( )*** et ***alarm( )*** pour une période d'au moins une seconde, la multiprogrammation (***fork( )***) pour les périodicités moindres,
- au lieu d'utiliser le *lcd,* on peut réaliser le même cahier des charges avec les afficheurs sept segments et leurs attributs de *sysfs,* ou bien encore avec l'écran *oled* accessible au travers de sa liaison série disponible sous *devfs*. Ceci permet de découvrir la structure spécifique ***termios*** pour la configuration.

Viennent ensuite les TPs portant sur l'intégration de composants *custom* et la conception de leur pilote en mode *kernel* (11, 12). Comme ces composants ne sont pas découvrables par le système *Linux* (*au sens classique d'un bus PCI par exemple*), la notion de ***Device Tree*** est abordée. Elle consiste à ajouter au noyau une description de la configuration matérielle sous la forme d'un fichier binaire « ***d****evice* ***t****ree* ***b****lob* (*.dtb*) » produit à partir de différents fichiers « ***d****evice* ***t****ree* ***s****ource* (*.dts*) » écrits dans un langage spécifique. C'est le mot clé *« compatible »* que l'on trouve dans le *Device Tree* et le pilote qui permet au noyau de sélectionner le bon pilote. Dans la phase de test, les pilotes sont des ***modules*** qu'on installe avec la commande ***insmod*** évitant une compilation du noyau. Pour chaque TP portant sur l'intégration d'un nouveau composant, un squelette de pilote est donné. Il contient les structures nécessaires à mettre en œuvre quand on utilise le couple « *platform driver, platform device* » et les services à écrire pour l'exploration et le retrait. Au fur et à mesure des TPs, le squelette s'enrichit pour permettre la création d'attributs pour *sysfs*, puis l'obtention d'un fichier de périphérique sous */dev*. Notons que le système *sysfs* se révèle très efficace car plutôt que de développer le service ***ioctl( )*** et un programme en mode *user* qui l'utilise, quelques commandes de type ***echo***, ***cat*** munies d'une indirection suffisent pour valider qualitativement le fonctionnement.

V. Poste de développement

L'ensemble des machines de type PC du département fonctionnent sous *Windows 10* ®. Afin que les étudiants découvrent aussi l'environnement graphique de *Linux* dans ce module, le poste de développement est une machine virtuelle (*Oracle VM Virtual Box*® (13)) utilisant une distribution ***Ubuntu*** sur laquelle on dispose de l'environnement *Quartus* et de la suite *Intelfpga's SoC* ***E****mbedded* ***D****esign* ***S****uite* (*EDS*). La machine hôte a deux interfaces réseau et un port *USB* pour la liaison console avec la plateforme. Aussi, cette solution permet d'installer différents services autorisant un boot réseau (14) :

- le service *DHCP* dont l'objectif est de fournir l'adresse *IP* de la carte, qui, associé au service *TFTP*, autorise les téléchargements nécessaires au boot de la carte (*le noyau linux*, *le fichier binaire représentant la programmation de la partie FPGA*, *le fichier binaire associé au Device Tree décrivant l'ensemble des périphériques*),
- le service *NTP* est nécessaire pour l'horodatage (*la cible* ***p-Soc*** *n'a pas de RTC sauvegardée par pile*),

- le service *NFS* qui a pour objectif de rendre distant le système de fichiers *rootfs* qui contient, entre autres, les *modules* à tester et les *applications*.

La configuration de ces services est effectuée par l'enseignant ainsi que celle de « *u-boot* » (15) et permet de procéder à des tests sur tous les niveaux (*matériel, logiciel mode User et Kernel*).

Fig.2. Vidéo associée au boot de la plateforme

VI. Conclusion

Il est clair que la plateforme ***p-Soc*** créée offre une multitude de possibilités pour la création de tps sur le système *Linux* portant aussi bien sur le mode *user* que *kernel*. La créativité est pour notre part sans limite avec ce type de cible. Pour le mode *user*, on ne rencontre pas de problème particulier. Par contre, faire l'intégration d'un composant *custom* et son pilote dans une même séance de TP est assez difficile sauf si l'on restreint assez fortement le nombre de services que doivent renseigner les étudiants dans chaque séance. Comme les services fonctionnent généralement par couple (*pour la détection (.probe, .remove), pour un attribut sous sysfs (.show, .store), pour un fichier de périphérique sous devfs ((.open, .release), (.read, .write), ...),* l'expérience nous a montré qu'au plus un de ces couples devait faire l'objet d'un TP.

Remerciements

Les auteurs souhaitent remercier vivement *Thierry Gil*, Ingénieur de recherche au CNRS, laboratoire LIRMM, Université de Montpellier pour sa disponibilité, et ses encouragements sous toutes ses formes, *Gilles Sicardi*, Technicien au département GEII d'Annecy pour le routage de la carte fille et la réalisation de la plateforme p-Soc, ainsi que le GIP-CNFM (16) et le programme FINMINA (17) pour leur soutien.

Références

1. Linux Documentation: *website:* https://www.kernel.org
2. Raspberry PI : *website:* https://www.raspberrypi.org/
3. Aria : *website:* https://www.acmesystems.it/aria
4. Programmes pédagogiques des DUTs : *website:* http://www.enseignementsup-recherche.gouv.fr/cid53575/programmes-pedagogiques-nationaux-d.u.t.html
5. Terasic : *website:* http://www.terasic.com
6. Intel FPGAs : *website:* https://www.intel.fr/content/www/fr/fr/products/programmable/fpga.html
7. uOLED-128-G2 : *website:* https://www.4dsystems.com.au/product/uOLED_128_G2/
8. LCD162COG : *website:* https://www.displaytech-us.com/16x2-character-lcd-displays
9. Yocto project : *website:* https://www.yoctoproject.org
10. C. Blaess, *Développement système sous Linux, 4ème édition,* 964, Eyrolles (2016).
11. J. Corbet, A. Rubini, G. Kroah-Hartman, *Linux Device Drivers,* 640, *O'Reilly Media* (2005).
12. *Linux kernel and Driver Development Training : Online course*: https://bootlin.com/doc/training/linux-kernel/linux-kernel-slides.pdf
13. Oracle VM Virtual Box® machines virtuelles : https://www.virtualbox.org
14. P. Aygalinc, S. Calvez, : *Introduction des systèmes Soc en DUT GEII*, J3eA, Vol 16, (2017).
15. U-boot : *Website:* https://www.denx.de/wiki/U-Boot
16. GIP-CNFM: Groupement d'Intérêt Public - Coordination Nationale pour la formation en Microélectronique et en nanotechnologies. *Website:* http://www.cnfm.fr

17. IDEFI-FINMINA : Initiative d'Excellence - Formation Innovante en MIcroélectronique et Nanotechnologies, ANR-11-IDFI-0017. *Website : http://www.cnfm.fr/VersionFrancaise/actualites/FINMINA.htm*