

Implémentation Matérielle des Services d'un RTOS sur Circuit Reconfigurable

Pierre Olivier*, Jalil Boukhobza*, Jean-Philippe Babau⁺, Damien Picard⁺ et Stéphane Rubini⁺

**Lab-STICC UMR 3192*

UEB-UBO, Brest, France

pierre.olivier@etudiant.univ-brest.fr

jalil.boukhobza@univ-brest.fr

⁺*LISyC EA 3883*

UEB-UBO, Brest, France

nom.prénom@univ-brest.fr

Résumé — Ce papier s'inscrit dans un projet d'implémentation sur plateforme reconfigurable d'un système d'exploitation temps-réel à base de composants. Ces composants peuvent être à l'état logiciel ou matériel. Ils peuvent être configurés en phases de conception, et reconfigurés de manière dynamique en phase d'exécution. Sous certains scénarios, un composant peut être accéléré en passant à l'état matériel.

Au sein du projet, ce papier traite des principes de déport matériel de services d'OS dans un contexte temps-réel. On prend l'exemple appliqué de l'implémentation du gestionnaire de tics système.

L'objectif de cette approche est l'adaptation et l'extensibilité des systèmes à long termes, ainsi que l'accroissement des performances.

I. INTRODUCTION

Aujourd'hui, le système d'exploitation temps-réel (*Real Time Operating System*, RTOS) est une couche de plus en plus intégrée au sein de nombreux systèmes embarqués. D'une part, le RTOS effectue une abstraction de la couche matérielle sous-jacente, et d'autre part, il apporte un *exécutif* contenant un ensemble de services. Ainsi, le RTOS amène des facilités de programmation dans un contexte où la complexité et la diversité des systèmes embarqués vont en augmentant, tout comme la pression des délais de mises sur le marché. De plus, l'utilisation d'un RTOS permet l'application de standards et la conformité à leurs spécifications.

Au niveau exécutif, les principaux services fournis par le système d'exploitation temps-réel sont les suivants :

- *L'ordonnancement des tâches* : l'ordonnanceur est l'unité du RTOS qui choisit la tâche à exécuter. La méthode la plus utilisée dans les systèmes d'exploitation temps réel actuels est basée sur une priorité assignée à chaque tâche, la tâche de plus haute priorité se voyant choisie pour être exécutée sur le processeur ;
- *La gestion du temps* : l'horloge (système) permet au RTOS d'offrir une notion du temps. La résolution du *timer* du système détermine à quelle fréquence l'interruption de tic, responsable entre autres du réveil de tâches mises en attente pour un temps donné, est exécutée ;
- *La communication et la synchronisation entre les tâches* : Les services de communication entre les tâches (*Inter-Process Communications*, IPC) sont gérés par le RTOS. Ce

sont des mécanismes tels que les sémaphores (d'exclusion mutuelle et de comptage), les files de messages, etc.

Les contraintes de prédictibilité et de déterminisme temporel sont des contraintes fortes des systèmes temps-réel. Elles doivent être impérativement respectées dans les systèmes temps-réel critiques. La mise en œuvre d'un tel RTOS génère des surcoûts en termes de temps d'exécution, qui peuvent avoir un impact négatif sur ces contraintes. Dans la suite de cet article on désigne ces surcoûts par le terme *latence*.

La latence est due aux services principaux fournis par le système : le temps d'exécution de l'ordonnanceur est fonction de sa complexité et du nombre de tâches s'exécutant dans le système, et la latence due aux IPC dépend du taux d'utilisation de ces entités. Pour ce qui est de la gestion du temps, il faut effectuer un compromis entre un système réactif avec une haute résolution de timer, ou un système avec des temps de réponse plus importants mais des temps de latence réduits.

Pour minimiser cette latence, une solution envisagée dans la littérature est le déport au niveau matériel des services du RTOS : ce déport permet d'accélérer ces opérations en profitant du parallélisme spatial (deux unités de calcul travaillant en parallèle) et temporel (*pipelining*) offert par une telle implémentation. Cette accélération des services permet de réduire la latence, et d'améliorer les performances au niveau des concepts clés que sont le déterminisme et la prédictibilité temporelle. Cela génère néanmoins des coûts matériels supplémentaires pour chaque service déporté dans le matériel.

La flexibilité offertes par les circuits logiques reconfigurables (de type *Field-Programmable Gate Array* ou *FPGA* dans notre cas) dans des environnements de codesign matériel / logiciel, et leurs capacités de reconfiguration dynamique et partielle en font de bons candidats pour l'implémentation de services du RTOS en matériel. En effet, ce type de circuit permet de gérer dynamiquement le déport de différents services en matériel en fonction des besoins/contraintes, et ce sur un même circuit.

Dans cet article, on présente un projet d'implémentation du système temps réel $\mu C/OS-II$ sous forme de composants logiciels et matériels reconfigurables dynamiquement en phases de conception et d'exécution. On étudie plus précisément les bénéfices apportés par une implémentation au niveau matériel d'un service du RTOS, en prenant comme

exemple pratique le gestionnaire de tics systèmes.

Cet article est organisé comme suit : la partie 2 présente un ensemble de travaux actuels réalisés dans le domaine de l'accélération matérielle pour les systèmes d'exploitation temps-réel. Dans la partie 3, on traite des bénéfices apportés par le déport de services du RTOS au niveau matériel, et des nouvelles contraintes soulevées par ce genre d'implémentation. En partie 4 on présente un projet de mise en œuvre d'un système d'exploitation temps réel embarqué sous forme de composants sur un circuit logique reconfigurable de type FPGA. Enfin, une conclusion est donnée dans la partie 5.

II. BIBLIOGRAPHIE

Cette partie introduit différents travaux réalisés dans le domaine de l'accélération matérielle pour les systèmes d'exploitation temps réel.

Un RTOS doit garantir des réactions temporelles déterministes, en réponse à divers stimuli provenant de son environnement interne et externe. Les performances sont parfois perturbées par les latences dues à l'OS, induites par la complexité logicielle du système d'exploitation temps réel.

Ainsi, de nombreux travaux proposent des accélérateurs et des supports matériels pour réduire la latence des services du RTOS les plus consommateurs en termes de temps.

A. Ordonnancement des tâches

L'ordonnanceur d'un système d'exploitation temps réel doit, par défaut, s'exécuter le plus rapidement possible, son temps d'exécution étant considéré comme du temps de latence.

Dans [1], les auteurs présentent un ordonnanceur matériel flexible et configurable, supportant les politiques *priority-based*, *rate monotonic*, et *earliest deadline first*. L'implémentation matérielle de cet ordonnanceur permet d'alléger le système de la latence générée par l'ordonnancement et la gestion du tic système.

Spring Scheduling Coprocessor (SSCoP) [2] est une architecture qui résulte du déport de l'ordonnanceur en matériel. Ce système est capable de s'exécuter en un temps déterministe, et présente un facteur d'accélération jusqu'à 6,5 comparé à un ordonnanceur logiciel.

De leur côté, les auteurs de [3] présentent Real-Time Task Manager (RTM), une unité matérielle qui réduit les temps de latence associés à un système d'exploitation temps réel. Cela est fait en supportant au niveau matériel des opérations telles que la gestion du temps, l'ordonnancement, et la gestion des événements. En plus de la latence, le temps de réponse aux stimuli extérieurs des systèmes implémentant RTM se trouve considérablement réduit.

Le déport de l'ordonnanceur ne fait pas l'objet de ce papier, mais une attention particulière y sera accordée dans le cadre du projet global.

B. Gestion temporelle

Les coûts peuvent être négativement impactés par le temps d'exécution de l'interruption gérant le tic système, notamment

lorsque la résolution du timer est élevée, ou lorsque le nombre de tâches dans le système est important.

Fastchart [4] est un noyau temps réel implémenté entièrement en matériel. Pour minimiser les temps de latence relatifs aux timers, chaque tâche devant être mise en attente se voit attribuer un compteur matériel. Le même mécanisme est utilisé dans [3].

III. IMPLEMENTATION MATERIELLE DE SERVICES D'UN SYSTEME D'EXPLOITATION TEMPS REEL SUR CIRCUIT RECONFIGURABLE

Dans cette partie on effectue une réflexion sur les bénéfices apportés par un déport matériel de services d'un système d'exploitation temps réel exécuté sur un circuit reconfigurable de type *FPGA*. L'avantage de l'utilisation d'un tel circuit est la possibilité de reconfigurer le comportement et les propriétés des services implémentés en matériel au cours du cycle de vie du système.

Pour ce faire on prend l'exemple pratique de l'implémentation matérielle du gestionnaire de tics du système d'exploitation temps réel $\mu C/OS-II$ sous la forme d'un composant matériel.

A. Contexte

$\mu C/OS-II$ [5] [6] est un système d'exploitation temps réel préemptif multitâche, destiné aux systèmes embarqués à base de microprocesseurs ou de microcontrôleurs. Il est utilisé dans de nombreux domaines : avionique, équipements médicaux, automobile, etc. $\mu C/OS-II$ est hautement paramétrable en phase de pré compilation, via une série de macros dans les sources du système, ce qui permet de réduire l'empreinte mémoire de l'exécutable final.

On utilise pour cette étude une plateforme logique reconfigurable FPGA. La capacité de reconfiguration de ce type de circuit et leur possibilité de synthèse des microprocesseurs *softcore*¹ les destinent au codesign matériel / logiciel, et au prototypage de nombreux systèmes électroniques numériques. La puce utilisée ici est un FPGA *Xilinx Virtex 5* [7], intégré au sein d'une carte d'évaluation possédant de nombreux périphériques d'entrées / sorties.

La plate-forme matérielle sur laquelle a été porté $\mu C/OS-II$ est un *MicroBlaze* [8], un processeur *softcore* RISC développé par Xilinx et synthétisé sur la matrice reconfigurable. Sur une partie de la matrice restante, on a synthétisé le service système implémenté en matériel.

L'échange de données entre le processeur *softcore* et le composant matériel représentant le service du RTOS peut se faire via un bus de type *Fast Simplex Link* (FSL) [9]. Ce type de bus fournit une interface point à point unidirectionnelle sous la forme d'une file FIFO de longueur et de largeur paramétrables.

Alors que le microprocesseur *MicroBlaze* fournit un certain nombre d'interfaces FSL prédéfinies, il est nécessaire d'encapsuler le composant (service matériel) pour permettre

¹ Une architecture de processeur synthétisée sur le circuit FPGA

l'interfaçage avec un bus FSL : un *wrapper* générique est ainsi développé pour permettre le branchement de bus FSL sur de nombreux composants représentant autant de services RTOS au niveau matériel.

Une illustration de l'architecture proposée ci-dessus est présentée sur la figure 1.

Il est possible d'utiliser d'autres moyens de communication que le bus FSL entre le processeur et le composant, par exemple un bus de type *Wishbone* [12].

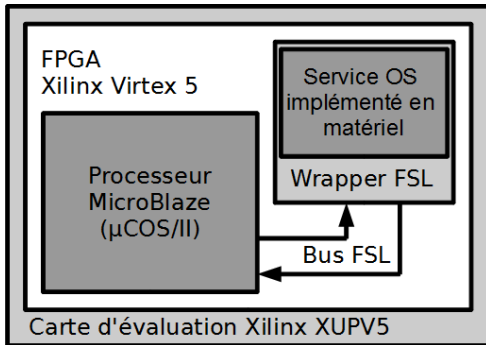


Figure 1 : Architecture d'un système synthétisé sur circuit FPGA présentant un RTOS exécuté sur un processeur MicroBlaze et un service implémenté en matériel communiquant avec le processeur. Le bus FSL est unidirectionnel, il en faut donc deux pour envoyer et recevoir des données.

B. Implémentation matérielle de services

Nous nous intéressons au déport matériel de services applicatifs et de services du RTOS.

L'implémentation d'un service de RTOS est nécessairement atomique : l'exécutif prend une décision par rapport à un contexte d'exécution qui doit rester constant le temps de la prise de décision. A cette fin, chaque service logiciel de $\mu\text{C}/\text{OS-II}$ suit le schéma suivant, où les interruptions sont inhibées le temps d'exécuter le service invoqué :

```
RTOS_Service (list_params)
{
    if (correctStateCondition)
    { OS_ENTER_CRITICAL() /*disable interrupts*/
      /* RTOS_Service Body */
      ...
      OS_EXIT_CRITICAL() /*enable interrupts*/
    }
    if (necessary) OSSched();
}
```

En suivant ces principes, l'appel au composant matériel doit être lui aussi exécuté en mode non interruptible sur le processeur.

Pour la communication de données entre le service logiciel et le composant matériel, nous avons implémenté un pilote effectuant des opérations d'écriture et de lecture sur les FIFO (matérielles) des bus FSL reliant le processeur au composant matériel. Les opérations de lecture/écriture peuvent se faire de manière bloquante ou non bloquante. Lors d'une opération bloquante, le pilote monopolise le processeur jusqu'à ce qu'il puisse lire / écrire les données nécessaires dans le bus. En mode non bloquant, le pilote tente une lecture / écriture sur le

bus et retourne directement, en renvoyant une erreur en cas d'écriture dans une FIFO pleine ou de lecture dans une FIFO vide. Du fait du contexte (atomicité de l'appel), on propose d'utiliser ici l'appel est en mode bloquant.

Pour le déport matériel de services applicatifs, le principe est différent et l'atomicité n'est pas toujours nécessaire. Dans ce cas, les interruptions ne sont pas inhibées et on propose d'utiliser le mode bloquant pour l'écriture et non bloquant pour la lecture : suite à une écriture, on boucle jusqu'à ce que la requête retourne un résultat valide. L'introduction d'un délai au sein de la boucle permet de libérer le processeur pour exécuter d'autres tâches.

Pour les tâches de haute priorité utilisant des ressources, le parallélisme spatial offert par une implémentation matérielle risque d'introduire un phénomène d'inversion de priorité. Dans ce cas, les appels aux services en matériel doivent se faire ou bien en mode bloquant, ou encore en mode non bloquant mais de manière atomique (en section critique) ce qui revient en grande partie à un appel bloquant.

C. Implémentation matérielle du gestionnaire de tics systèmes

Comme énoncé précédemment, la gestion du temps par le RTOS peut avoir un impact important sur le déterminisme temporel requis dans de nombreux systèmes critiques.

Dans $\mu\text{C}/\text{OS-II}$, l'utilisateur définit une précision temporelle (`OS_TICKS_PER_SEC`) qui permet de préciser le nombre d'interruptions tic générées à la seconde. Ensuite, une tâche qui s'exécute peut, à tout instant, se mettre en attente d'un certain nombre de tics via la primitive `void OSTimeDly(INT16U ticks)`. L'appel à cette primitive initialise un compteur local à la tâche appelé `OSTCBDly`. A chaque interruption correspondant à un tic, une routine incrémente un compteur (`OSTime`), et place dans la liste des tâches prêtes les tâches en attente de 1 tic (`--OSTCBDly == 0`) tout en décrémentant le compteur d'attente de toutes les tâches délayées.

Au vu de cette implémentation classique dans le monde logiciel, comme il est précisé dans la documentation de $\mu\text{C}/\text{OS-II}$, « *The actual frequency of the clock tick depends on the desired tic resolution of your application. However, the higher the frequency of the ticker, the higher the overhead, execution of Interrupt Service Routine each time a tick occurs, called OSTickISR.* » [5].

Le fait d'implémenter le gestionnaire de tics systèmes au niveau matériel, sous la forme d'un composant s'exécutant en parallèle du microprocesseur permet d'une part de soulager le processeur de ce service, et ainsi de réduire les latences du RTOS du système. D'autre part, le gestionnaire de tics accéléré, il est possible de choisir une résolution de timer plus importante, on bénéficie ainsi d'un système plus réactif sans augmentation des latences du RTOS.

Dans cette évolution, on peut aussi simplement définir de nouveaux services de mise en attente de tâches, par exemple avec réveil à une date absolue. La mise en place d'une tâche périodique, concept non natif avec $\mu\text{C}/\text{OS-II}$, en est grandement facilitée.

Le déport au niveau matériel d'un ou plusieurs services RTOS permet de réduire l'indéterminisme temporel et la réactivité du système en réduisant les latences dues au RTOS. En accélérant ces services il est également possible de gagner en performance.

IV. PERSPECTIVES : IMPLEMENTATION MATERIELLE D'UN SYSTEME D'EXPLOITATION TEMPS REEL A BASE DE COMPOSANTS

Le projet THINK-FPGA vise à porter une version à composant purement logicielle du système $\mu\text{C}/\text{OS-II}$ vers une version à composants logiciels et matériels implémentée sur un circuit FPGA. Dans cette partie on rappelle, très brièvement, les concepts de la programmation orientée composant, et les bénéfices à tirer d'une telle mise en œuvre.

A. Le modèle Fractal/THINK et $\mu\text{C}/\text{OS-II}$ à composants

Dans le modèle *Fractal* [10] les composants possèdent des *interfaces serveurs*, décrivant les services / fonctionnalités qu'ils offrent, et des *interfaces clients*, décrivant les fonctionnalités (venant d'autres composants) qu'ils requièrent. Les composants communiquent entre eux via des liaisons entre interfaces client et serveurs. Les propriétés d'un composant sont exprimées par les valeurs de ses *attributs*. Enfin, les composants fournissent des *interfaces de contrôle*, pour l'observation et la modification des propriétés du composant. Un composant peut être constitué d'un ou plusieurs sous composants. Au plus bas niveau de cette hiérarchie, les composants de bases peuvent être implémentés dans un langage de programmation quelconque. *THINK* [11] est une implémentation du modèle Fractal en langage C.

Dans [6], les auteurs présentent une implémentation du système d'exploitation temps-réel $\mu\text{C}/\text{OS-II}$ sous la forme de composant logiciels THINK. Les composant requis par le système sont exprimé en utilisant THINK, et à la compilation un exécutable adapté est généré pour une architecture cible donnée. En phase d'exécution, les composants peuvent être reconfigurés logiquement. Le coût en termes d'augmentation de l'empreinte mémoire varie entre 0,7% pour un système complètement statique, et 31,2% pour un système hautement reconfigurable [6].

B. Implémentation matérielle de la version à composant de $\mu\text{C}/\text{OS-II}$ sur circuit FPGA

Les composants décrits avec THINK se voient utilisés en phase de pré-compilation pour générer des représentations. Un exécutable est généré lors de la compilation, et en phase d'exécution chaque composant peut être activé, désactivé, et reconfiguré. D'un point de vue matériel, il faudrait être en mesure d'implémenter des concepts symétriques à ceux des composants. Les représentations intermédiaires seront exprimées en langage HDL (*Hardware Description Language*). Les exécutables logiciels auront comme symétrique les *bitstreams* de configuration du matériel permettant de configurer une zone du circuit pour une fonctionnalité donnée.

En phase d'exécution, la reconfiguration d'un composant logiciel se fait en branchant des pointeurs de fonction contenus dans une structure dédiée vers les fonctions décrivant le comportement correspondant, et ce directement dans le code binaire à l'exécution. Pour ce qui est des composants matériel, on profite des capacités de reconfiguration dynamique et partielle des circuits FPGA pour (ré) implémenter un module sous forme matérielle.

Un autre point important à prendre en considération est l'implémentation en matériel des interfaces composants. En effet, au même titre que pour le logiciel, l'interfaçage du service de l'OS sur le matériel et entre le matériel et le logiciel doit suivre un formalisme rigoureux améliorant ainsi la flexibilité du système.

V. CONCLUSION

Le déport de services RTOS au niveau matériel permet d'améliorer les performances du système en profitant du parallélisme spatial et temporel offert par ce type d'implémentation. Cependant, ce déport pose certaines contraintes (mises en attente de tâche prioritaires menant à des inversions de priorités) dont il faut tenir compte.

Pour aller plus loin, cet article présente un projet d'implémentation matérielle d'un RTOS à composant logiciels et matériels reconfigurables en phases de design et d'exécution permettant d'une part l'amélioration des performances, et d'autre part le maintient, l'extensibilité et l'adaptabilité du système dans son ensemble sur le long terme.

BIBLIOGRAPHIE

- [1] P. Kuacharoen, M. A. Shalan, V. Mooney, "A Configurable Hardware Scheduler for Real-Time Systems", *Proceedings of the IEEE International Conference of Computer Design : VLSI in Computers and Processors*, 2003.
- [2] W. Burleson, J. Ko, D. Niehaus, K. Ramamritham, J.A. Stankovic, G. Wallace, C. Weems, "The Spring Scheduling Coprocessor: a Scheduling Accelerator", *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol.7, no.1, Mar 1999.
- [3] P. Kohout, B. Ganesh, B. Jacob, "Hardware Support for Real-Time Operating Systems", *Proceedings of the 1st IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis (CODES+ISSS '03)*. ACM, New York, NY, USA, 2003.
- [4] L. Lindh, "Fastchart - a Fast Time Deterministic CPU and Hardware Based Real-Time-Kernel", *Real Time Systems, 1991. Proceedings., Euromicro '91 Workshop on*, vol., no., pp.36-40, 12-14 Jun 1991.
- [5] J. J. Labrosse, "Microc/os-ii : the Real-Time Kernel", 1999.
- [6] F. Loiret, J. Navas, O. Lobry, J.-P. Babau, "Component-based Real-Time Operating System for Embedded Applications", *Proceedings of the International Symposium on Component Based Software Engineering*, 2009.
- [7] Xilinx Inc., "Virtex 5 Family Overview", 2009.
- [8] Xilinx Inc., "MicroBlaze Processor Reference Guide", 2008.
- [9] Xilinx Inc., "LogiCORE IP Fast Simplex Link (FSL) V20 Bus (v2.11c), 2010.
- [10] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, J.-B. Stefani, "The FRACTAL component model and its support in Java: Experiences with Auto-adaptive and Reconfigurable Systems", *Softw. Pract. Exper.* 36, 11-12, Sept 2006.
- [11] M. Anne et al., "Think: View-Based Support of Non-functional Properties in Embedded Systems". *Proceedings of the 2009 International Conference on Embedded Software and Systems (ICCESS '09)*. IEEE Computer Society, Washington, DC, USA, 2009.
- [12] OpenCores, "WISHBONE B4 System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores", 2010.