APPROCHE DE CONCEPTION DE SYSTEME D'EXPLOITATION TEMPS REEL POUR UNE ARCHITECTURE MULTIPROCESSEUR RECONFIGURABLE

Kaïs LOUKIL, Hajer KRICHENE, Nader BEN AMOR, Mohamed ABID
Kais_loukil@ieee.org RESUME
Ecole Nationale d'Ingénieur de Sfax,

Actuellement on assiste à une large diffusion des produits électroniques embarqués chez une large gamme d'utilisateurs. Ces systèmes permettent d'exécuter divers types d'applications de plus en plus complexes. Par conséquent, l'architecture multiprocesseur est l'une des solutions pour répondre aux exigences des nouvelles applications. Nous proposons dans ce papier une approche de conception de système d'exploitation temps réel pour les architectures multiprocesseur. Il s'agit d'une couche générique de communications inter-processeur qui permet d'adapter les systèmes d'exploitation monoprocesseur aux architectures multiprocesseur. Nous présentons également les étapes qui ont conduit à la mise en place d'une plateforme de prototypage multiprocesseur reconfigurable. Cette couche a été validée à travers l'application de synthèse d'images 3D.

Mots clefs

MPSoC, RTOS, couche de communication inter-processeur.

1 INTRODUCTION

Avec le progrès de la capacité d'intégration de centaines de millions de transistors sur une seule puce et l'avancement au niveau de la conception des cœurs de processeurs embarqués. Les nouvelles technologies s'orientent vers l'intégration sur une même puce de plusieurs processeurs, DSP, IP matériels et logiciels, mémoires, bus partagés, etc. Nous parlons ainsi de systèmes multiprocesseurs mono puce (MPSoC) [1, 11, 12]. En fait, les systèmes multiprocesseurs sont l'une des solutions pour répondre à la complexité croissante des systèmes intégrés utilisés pour des applications telles que les applications multimédia.

En plus, et du fait de la complexité croissante de ces systèmes, de la présence de fortes contraintes temps réel, de la limitation des ressources disponibles, tant en mémoire qu'en énergie disponible et donc en puissance de calcul, mais également de la pression exercée par le marché sur ces produits, l'usage de systèmes d'exploitation temps réel (RTOS) est devenu indispensable dans les systèmes embarqués [2, 13, 14]. Par ailleurs actuellement, la plupart des RTOS existants ne supportent pas les architectures multiprocesseur d'où la nécessité de trouver des méthodes pour répondre aux exigences de tel systèmes. Deux solutions peuvent être distinguées la première consiste à développer de nouveaux systèmes d'exploitation qui supportent des architectures multiprocesseurs. La deuxième solution consiste à étendre les systèmes existants par d'autres fonctionnalités pour qu'ils puissent supporter ces architectures. Dans ce cadre se situe notre travail qui consiste à étendre un RTOS existant par une couche de communication inter processeur pour gérer la communication entre les processeurs de notre système.

Ce papier est organisé de la façon suivante. La première section, est consacrée pour la présentation des différents modèles de communication inter-processeur. La deuxième section présente les caractéristiques de l'environnement utilisé ainsi que la topologie de l'architecture

multiprocesseur adopté. La troisième section illustre la couche de communication interprocesseur implémentée ainsi que sa validation. On terminera par quelques conclusions.

2 Couche générique de communication inter-processeur

Dans le cadre d'un système multiprocesseur, il est nécessaire que les processeurs se communiquent. Cette communication est accomplie par envoi de messages (Message Passing) [9].

2.1 Modèle de communication inter-processeur

Généralement, les communications peuvent être synchrones ou asynchrones, bloquantes ou non bloquantes. Les communications synchrones sont plus lentes et généralement bloquantes, c'est à dire que les deux processeurs engagés dans la communication doivent attendre la fin de la communication pour continuer.

Par contre les communications asynchrones sont la plupart du temps non bloquantes. En fait, quand un processeur veut envoyer un message à un autre, il envoie le message, et il peut immédiatement reprendre son exécution sans se soucier de l'autre processeur quand il recevra le message. C'est le principal avantage des communications asynchrones.

Dans le modèle de la couche de communications inter processeur implémentée, nous avons considéré une mémoire partagée dont l'accès est protégé par le mutex hardware et une file de messages (Message Queuing) à travers laquelle les processus des nœuds envoient et reçoivent des messages. Ainsi, la couche de communications développée est un ensemble de routines permettant d'envoyer et de recevoir des messages de différentes façons en combinant les paramètres suivants :

- Envoi/Réception bloquant et non bloquant ;
- Envoi/Réception de messages uniques ou composés ; • Envoi de messages synchrone ou asynchrone.

2.2 Différents modèles de communication

Il existe deux modèles principaux de communications pour l'échange d'information, appelés aussi modèles de synchronisation, le modèle synchrone et le modèle asynchrone [10].

2.2.1. Modèle synchrone : Dans le cadre d'une communication synchrone (Synchronous Message Passing), l'expéditeur est sûr que le destinataire a reçu le message. D'autre part, l'expéditeur d'une demande doit attendre une réponse du récepteur prévu avant de pouvoir effectuer le traitement restant. Le délai d'attente de l'émetteur dépend entièrement du temps nécessaire au récepteur pour traiter la demande et envoyer une réponse. Typiquement, le temps d'attente de la réponse à la demande est assez long.

2.2.2. Modèle asynchrone : Dans le cadre d'une communication asynchrone (Asynchronous Message Passing), l'expéditeur envoie son message dès qu'il soit prêt, et le récepteur peut éventuellement attendre si le message n'est pas encore arrivé.

Avec ce modèle, l'émetteur n'attend jamais, mais aussi avec moins de contrôle sur la réception. La communication asynchrone peut être soit bloquante, soit non bloquante, ou bien par interruptions.

2 .2.2.1. Mode bloquant : Pour l'envoi de message, l'appel à la primitive d'envoi se termine lorsque le message a quitté l'expéditeur. Toutefois, cela ne signifie pas que le destinataire l'a bien reçu. Pour la réception, l'appel à la primitive de réception se termine quand le message est arrivé et qu'il est copié dans le buffer de réception.

2.2.2.2. Mode non bloquant : Le but de ce mode est de diminuer le temps d'attente. En fait, La couche logicielle expédie ou reçoit dès que c'est possible. L'exécution d'une telle communication asynchrone non bloquante s'effectue aussi sans attendre qu'elle soit finie. Il s'agit donc de masquer le temps des communications.

2.2.2.3. Mode par interruptions : Ce mode est défini pour les communications asynchrones, et il est rarement disponible. Dans le cadre de ce mode, l'arrivée d'un message génère une interruption au niveau de l'application.

2.3 Choix conceptuelle:

Partant des deux modèles de communication présentés, nous avons été amenés à adopter le modèle asynchrone comme modèle de communication (Asynchronous Message Passing). En fait, la communication à travers des files d'attente est, par nature, asynchrone dans la mesure où les messages sont envoyés à une file d'attente et reçus d'une file d'attentes dans des processus différents.

2.3.1. Gestionnaire de messages : La gestion des messages de communication est tributaire d'une part à la gestion de la file de communication et ses buffers de messages, et d'autre part à la gestion du support de communication (mémoire partagée).

2. 3.1.1. Gestion de la file de communication : La file d'attente est un objet permettant la communication asynchrone de messages entre des tâches. Les principales propriétés mises en jeu pour la gestion de cette file de communications sont les suivantes :

- *Capacité maximale de la file* : En fait, la file d'attente (pending-message-queue) définit un nombre maximal et bien déterminé de messages en attente. Cette propriété est indispensable pour éviter l'échec de l'envoi de messages.
- *Propriété d'expiration* : Cette notion de « timeout », liée au temps, dispose d'un meilleur contrôle à nos messages. Elle détermine la durée maximale d'existence d'un message dans le système avant son élimination. Cette propriété est utilisée en cas d'une communication asynchrone bloquante.
- *Propriété d'optimisation* : Il s'agit, en fait, d'employer la notion d' « importance » (ou d' « urgence ») qui affecte l'ordre dans lequel les messages sont envoyés vers la file de communications. Le paramètre de l'importance d'un message détermine l'emplacement du message dans la file d'attente. Les messages envoyés avec un degré d'urgence élevé sont placés plus haut dans la file d'attente, tandis que ceux affectés d'une importance faible sont placés plus bas dans la file. Cette propriété est utilisée dans les deux cas de communications étudiés : bloquante et non bloquante.

2.3.1.2. Gestion des tampons de messages : Un message est caractérisé par une longueur, un type (unique ou composé), un degré d'urgence et une identification. La longueur de message est mesurée par octets. Cependant, le nombre maximal d'octets définissant la longueur est fixé lors de la création de la file de messages. Ainsi, si la taille du message à envoyer est supérieure à celle déjà définie lors de l'initialisation de la communication, le message correspondant sera rejeté.

2.3.1.3. Adressage mémoire : La mémoire partagée est repartie en un nombre bien défini de blocks ayant tous la même taille en octets. En fait, la taille d'un block est égale à la taille maximale d'un buffer de message. Ainsi, le nombre de buffers de la file d'attente ne doit pas dépasser le nombre total de blocks de la mémoire partagée, sinon la création de la file sera échouée et donc la communication ne sera pas établie.

Pour ce faire, et afin de pouvoir gérer les buffers de messages transmis vers la mémoire partagée, nous avons tenu en considération :

- Attribution d'un block mémoire à un seul buffer de messages.
- Utilisation des identificateurs pour designer les tâches.
- Utilisation des identificateurs pour designer les processeurs.

3 Etude de cas : Conception d'architecture multiprocesseur à travers l'environnement d'Altera

Afin de mettre en place un système multiprocesseur, plusieurs topologies d'architectures multiprocesseurs sont envisagées [3] :

- *Les systèmes à mémoire partagée* : ce type de mémoire présente l'avantage de permettre un partage immédiat des données, facilitant la programmation. Mais le nombre de processeurs ajoutés sur une même mémoire est limité [4].
- *Les systèmes à mémoire distribuée* : dans ce cas, chaque processeur possède sa propre mémoire. La modification par l'un des processeurs de sa propre mémoire n'a pas d'influence sur celle des autres processeurs. Cela suppose donc de mettre en place une communication explicite entre les processeurs. C'est au programmeur de gérer la plupart des détails de la communication entre les processeurs. Cette topologie rend également difficiles les échanges complets de structures de données, elle pose des problèmes d'accès non uniformes dans le temps, et elle rend la cohérence de données plus dure à maintenir.
- *Les systèmes à mémoire distribuée, partagée* : Ce type de mémoire est un mélange des deux premiers. Dans cette architecture, il y a plusieurs groupes de processeurs partageant de la mémoire. Cela permet, dans une certaine mesure, de tirer les avantages des deux précédentes architectures et d'en réduire les inconvénients [5].

Dans le but de choisir une topologie d'architecture multiprocesseur il a été convenu d'étudier les caractéristiques de notre environnement de conception (le kit excalibur d'ALTERA). Cet environnement utilise le bus avalon qui peut être vu comme un ensemble de signaux prédéfinis, permettant de connecter un ou plusieurs blocks IP. En plus, il est généré automatiquement par le NIOS-II Builder. Le bus Avalon a comme caractéristiques principales c'est qu'il est simultané multi maîtres ; les maîtres peuvent accéder simultanément à leurs esclaves et en cas de besoin les maîtres peuvent échanger des données à travers une mémoire partagée. Généralement l'accès à cette mémoire est géré par le système d'exploitation utilisé [7]. Comme solution on va utiliser un module d'un RTOS implémenté en hardware « mutex » pour assurer la gestion des accès concurrents aux mémoires partagées entre les différents processeurs [8]. Le mutex fournit une opération « test-and-set » à base de matériel, permettant au logiciel dans un environnement multiprocesseur de déterminer le processeur qui possède l'accès à une ressource partagée. Le mutex est utilisé dans la conjonction avec la mémoire partagée pour mettre en œuvre des dispositifs de coordination d'inter processeurs complémentaires.

Après l'étude faite sur l'environnement d'altera nous avons adopté la topologie d'architecture multiprocesseur à mémoire distribuée, partagée. Cette plateforme se compose d'un ensemble de sous systèmes qui peuvent communiquer ensemble à travers des mémoires partagées dont l'accès est protégé par des mutex Hardware fournis avec l'environnement d'Altera (fig1). Chaque sous système possède son propre RTOS et en cas de besoin ils peuvent échanger des données à travers la mémoire partagée.

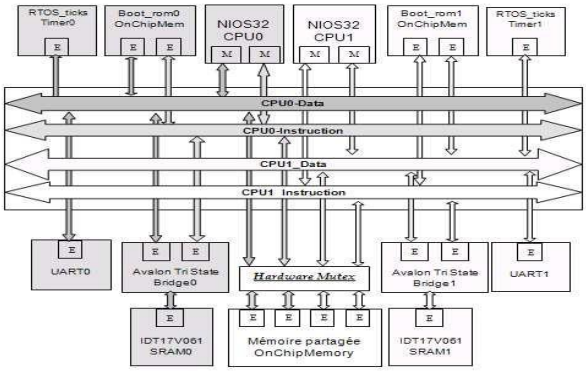

Figure 1. topologie adoptée pour une architecture multiprocesseur

Ainsi notre environnement de prototypage multiprocesseur est mis en place. Dans la suite de ce papier nous proposons une couche générique de communications inter-processeur, qui permet d'adapter un système d'exploitation monoprocesseur aux architectures multiprocesseur.

4 Expérimentation : Implémentation des routines de communication:

4.1 Communication inter-processeur:

L'établissement d'une communication entre plusieurs processeurs nécessite différentes phases telle que l'initialisation de la communication, l'envoi et la réception des messages

4.2.1. Initialisation de la communication :

Une étape d'initialisation de la communication est nécessaire avant que les processeurs ne commencent l'envoi et la réception des messages. Cette phase contient les étapes suivantes :

- Définir les propriétés de la queue de message et ce à travers la fonction « definirPropriétésMsgQueue() » ○ S'il y a un espace suffisant dans la mémoire partagée la queue de message sera crée sinon un message d'erreur apparaît.

4.2.2 Réception d'un message :

La phase de réception contient les étapes suivantes :

- La vérification de l'existence d'une queue de message ○ Recevoir le message avec la fonction « getMessageBuffer() »
- Si le message reçu est NULL donc il y a une erreur d'émission sinon on décrément le nombre de messages dans la queue de message et on dé alloue le bloc mémoire réservé au message reçu

4.2.3 Envoi d'un message :

La phase d'envoi d'un message comporte les étapes suivantes : ○

Vérifier l'existence d'une queue de message

- S'assurer que la taille du message à envoyer est inférieure à celle définie lors de l'étape d'initialisation.
- Lorsque la queue de message est pleine et il existe un message qui a un degré d'urgence inférieur à celui du message à envoyer on dé alloue le bloc mémoire associé au message de plus petit degré d'urgence.
- Si notre queue de message peut contenir d'autres messages on crée un buffer de messages et on l'insère dans la queue de message.
- Incrémenté le nombre de messages dans la queue de messages

4.2 Validation :

Après avoir implémenté les différentes routines de communication inter-processeur, dans un premier temps, nous avons opté à des tests pour chaque routine à part. Dans nos tests on a utilisé le système d'exploitation temps réel monoprocesseur MicroC_OS-II fournit avec le kit excalibur d'altera. Il est à signaler que le code source est ouvert et il écrit en langage C, ce qui

a facilité l'intégration de la couche de communications inter-processeur implémentée dans ce système d'exploitation. Le portage de notre nouveau système d'exploitation « multiprocesseur » a été fait en utilisant l'environnement IDE fourni avec notre kit de conception.

Dans un second temps, et après avoir validé toutes les fonctions implémentées nous avons optés à la validation de cette couche à travers l'application de synthèse d'images 3D sur la plateforme multiprocesseur mise en place. Pour ce faire en premier lieu on a extrait le graphe de tâche de cette application qui représente les différentes tâches de l'application ainsi que les données échangées entre eux. En suite, et d'une façon manuelle on est passés a l'étape de partitionnement de l'application sur les différents processeurs.

5 Conclusion

Le travail entrepris a permis d'étudier de près les contraintes et les problèmes engendrés par le prototypage des systèmes multiprocesseurs temps réel sur des architectures reconfigurables. En premier lieu, des études bibliographiques sur les architectures des systèmes multiprocesseur, a été faite pour explorer le domaine et avoir une idée sur leurs caractéristiques. En second lieu on a visé la mise en place d'une plateforme multiprocesseur ainsi que la proposition d'une couche générique de communication inter-processeur qui permet d'adapter les systèmes d'exploitation monoprocesseur pour des architectures multiprocesseur. On a terminé par la validation de la couche implémentée sur le système d'exploitation temps réel MicroC_OS-II. Notre objectif à court terme consiste à valider la couche de communication proposé sur d'autres systèmes d'exploitation.

Les résultats obtenus durant ce travail ont permis d'ouvrir divers axes de recherche. Le premier axe vise l'exploration de l'espace des solutions architecturales. Le concepteur se trouve devant divers types d'architectures d'où la nécessité de mettre en place des outils qui aident le concepteur à choisir l'architecture adéquate à son systèmes et surtout dans le domaine des systèmes sur puce puisqu'ils sont mobiles avec des ressources limitées et évoluant dans des environnements variables. Le deuxième axe concerne le partitionnement automatique de l'application sur une architecture multiprocesseur.

6 References

[1] Bambha, N., Kianzad, V., Khandelia, M., and Bhattacharrya, S.S., Intermediate Representations for Design Automation of Multiprocessor DSP Systems. In Design Automation for Embedded Systems, vol. 7, 307-323, Kluwer Academic Publishers, 2002.

[2] Le Moigne, R.; Pasquier, O.; Calvez, J.-P.; "A generic RTOS model for real-time systems simulation with systemC", Design, Automation and Test in Europe Conference and Exhibition, Feb. 2004.

[3] Conception d'un système à haute performance, le calcul parallèle , CETMEF 2004.

[4] Amer BAGHDADI: Exploration et conception systématique d'architectures multiprocesseurs monopuces dédiées à des applications spécifiques thèse PhD, TIMA France.

[5] CM: The CM-5 Connection Machine : A Scalable Supercomputer, W. Daniel Hillia and Lewis W.Tucker. Communication of the ACM, November 1993, Vol. 36, No. 11.

[6] J.J. Labrosse, « Micro C/OS-II, the Real-Time Kernel", Second Edition.

[7] http:// www.altera.com

[8] K. loukil, Y. aoudni, G. Gogniat, M.abid, J.L. philippe, Estimation du temps d'exécution des systèmes sur puce temps réel. GEI 2007

[9] « Présentation rapide de MPI : Message Passing Interface », Géraud Krawezik, LRI – Université de Paris Sud, EADS CCR – Blagnac, 28 Octobre 2003.

[10] Ph. Marquet, « Bibliothèque de communications », Maîtrise en Informatique, Université des sciences et technologies, Lille.

[11] L.Wang and N. Manjikian. A performance study of chip multiprocessors with integrated dram. In Proc. 2003 Symp. on Perf. Eval. of Computer and Telecommunications Systems, Montreal, Quebec, July 2003.
[12] N. Manjikian. Multiprocessor enhancements of the SimpleScalar tool set. ACM Computer architecture News, 29(1):8–15, March 2001.
[13] Dongkun Shin and Jihong Kim. Power-Aware Scheduling of Conditional Task Graphs in Real-Time Multiprocessor Systems. In Proc. International Symposium on Low Power Electronics and Design (ISLPED), August 2003.
[14] Victor B. Lortz, Kang G. Shin, Fellow, IEEE, and Jinho Kim [14]MDARTS: A Multiprocessor Database Architecture for Hard Real-Time Systems IEEE transactions on knowledge and data engineering, VOL. 12, NO. 4, JULY/AUGUST 2000
[15] M. Ben Said, K. Loukil, N. Ben Amor, M. Abid, Jean Philippe Diguet «A timing constraints control technique for embedded real time systems» Design and technology of integrated Systems (DTIS 2010) March 2010
[16] Hanen Abbes, Kais Loukil, Hafedh Abid, Mohamed Abid, Ahmad Toumi « Implementation of Photovoltaic Maximum Power Point Tracking Fuzzy Logic Controller on FPGA » Journal of Information Assurance and Security, pp. 097 – 106, Vol. 11, Issue 2, 2016