GEI 2003
Mahdia, Tunisie, 18 au 20 Mars 2003

Portage d'un noyau multitâche temps réel sur un processeur embarqué

Fakhreddine GHAFFARI *& ; Michel AUGUIN * ; Maher BEN JEMAA ****
**Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis, les Algorithmes bat. Euclide, 2000, route des Lucioles BP 121, 06903 Sophia-Antipolis Cedex*
***Unité de recherche GMS. Ecole Nationale d'Ingénieurs de Sfax, BPW 3038 Sfax Tunisie.*
Ghaffari@i3s.unice.fr, auguin@i3s.unice.fr, Maher.Benjemaa@enis.rnu.tn

Résumé

Un noyau temps réel est un ensemble de programmes assurant la gestion des processus d'une application embarquée afin de garantir les contraintes temps réel. Dans ce papier, nous présentons le portage d'un système d'exploitation temps réel : *µC/OS-II* sur le processeur NIOS 32 d'ALTERA afin d'implanter une application de traitement d'images sur la carte EXCALIBUR.

1. Introduction

Un facteur important dans l'évolution des systèmes modernes est l'apparition de nouvelles architectures hétérogènes exploitant la synergie entre le matériel et le logiciel, basées sur la programmation des circuits matériels tels que les composants FPGAs (Field Programmable Gate Array). La conception efficace de ces systèmes hétérogènes nécessite une approche globale dans laquelle les parties matérielles et logicielles sont conçues en parallèle et de façon interactive [1]. L'approche de conception conjointe logiciel/matériel : Codesign consiste à chercher une adéquation application /architecture satisfaisant les nombreuses contraintes de conception telles que le coût, les performances, la surface, la consommation, le temps de conception [2].
Une de phase clés de cette approche est le partitionnement logiciel/matériel qui consiste à effectuer le choix pour une implémentation logicielle, matérielle ou mixte des différentes parties du système. Cette étape est généralement suivie par un ordonnancement des fonctionnalités de l'application sur les unités de l'architecture en fonction du temps. La gestion de tel ordonnancement n'est pas toujours évident surtout pour les architectures hétérogènes et elle nécessite une programmation assez optimisée pour respecter toujours les contraintes temps réel de l'application.
Le problème s'accentue lorsqu'il s'agit des processus à temps d'exécution variable en fonction du jeux des données : dans ce cas nous allons avoir différents résultats du partitionnements [3] et par suite plusieurs configurations de l'architecture.
Pour mieux gérer tous ces partitionnements et les ordonnancements correspondants il parait légitime d'introduire un système d'exploitation temps réel qui assure la gestion de tous les processus en impliquant à chaque fois l'ordonnancement adéquat de l'application.
Nous avons travaillé sur l'application détection de mouvements sur un fond d'image fixe [4] ; les résultats des partitionnements de cette application impliquent plusieurs ordonnancements possibles. Pour expérimenter nos travaux, nous avons considéré le système EXCALIBUR d'ALTERA qui intègre sur le même puce un processeur RISC et une unité réconfigurable [5].
Ce papier est organisée comme suit, dans le deuxième paragraphe nous présentons le rôle de système d'exploitation *µC/OS-II* suivi par ses fonctions dans un troisième paragraphe. Nous expliquons le mode de configuration du processeur Nios dans le paragraphe quatre. La mise en place de *µC/OS-II* est détaillée dans le paragraphe cinq. Avant de conclure nous terminons par un exemple d'application.

2. Le rôle du µC/OS-II

Le multitâche / monoprocesseur permet d'effectuer à l'échelle humaine plusieurs tâches simultanément. En effet l'utilisateur peut alors diviser son projet en plusieurs tâches indépendantes. Au niveau du processeur, une seule tâche est effectuée à la fois mais le multitâche permet d'éliminer les temps machines inutilisés (boucle d'attente...).

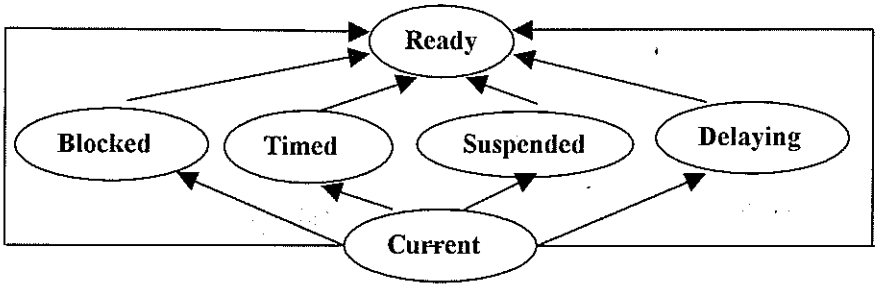


Figure 1– les différents états des tâches

Le fonctionnement du noyau temps réel sur le processeur INTEL est le suivant : lors de l'initialisation du programme µC/OS-II, les différents programmes de l'utilisateur sont considérés comme des tâches qui sont toutes créées pendant cette période d'initialisation. Le programmeur doit alors spécifier le point d'entrée de la tâche, l'emplacement des données pour cette tâche, l'adresse haute de la pile de la tâche et le degré de priorité de la tâche. Ainsi la tâche de plus haut degré de priorité et prête est exécutée par le microprocesseur (on dit que le noyau est **préemptif**). La figure 1 montre les différents états des tâches lors du fonctionnement du noyau multitâche. Dans le noyau µC/OS, l'utilisateur dispose de 63 tâches où chaque degré de priorité correspond à une tâche, c'est-à-dire que 2 tâches ne peuvent pas avoir le même degré de priorité (pas de "round-robin" ou "méthode du tourniquet").

Les caractéristiques essentielles du noyau temps réel µC/OS sont les suivantes :

- création et gestion de 63 tâches maximum.
- création et gestion de sémaphores binaires et comptés.
- fonction d'attente de tâche.
- changement de priorité des tâches.
- effacement de tâches.
- suspension et continuation de tâches.
- envoi de messages depuis une routine d'interruption (ISR) ou d'une tâche vers une autre tâche.

Les tâches peuvent ainsi communiquer avec d'autres tâches (grâce aux sémaphores, boîtes aux lettres, files d'attentes...) ou bien avec des périphériques grâce aux ISR (**Interrupt Service Routine)** dont la liste complète est disponible sur l'adresse Web : http://www.ucos-ii.com/.

Les temps d'exécution de la plupart des services de **µC/OSII** sont constants et déterministes. Ceci veut dire que le temps d'exécution ne dépend pas du nombre des tâches exécutées dans l'application. Ce RTOS est très rapide, son Scheduler ne contient que quatre lignes de code C!

3. *Les fonctions de µC/OS*

Les fonctions de base du noyau temps réel sont les suivantes :

- OSinit(); initialisation globale du noyau
- OSTaskcreate (pointeur sur le programme utilisateur, pointeur données, adresse haut de la pile, priorité); permet de définir une tâche. Le pointeur "données" permet de faire passer des données à une tâche lors de son initialisation. La priorité est définie
- telle que plus le nombre est faible, plus le degré de priorité est élevé.
- OSStart(); permet de créer au moins une tâche en appelant OSTaskCreate();

Le fonctionnement correct de µC/OS exige qu'une tâche doive obligatoirement appeler une fonction (un service du noyau) afin de :

- faire attendre la tâche d'un délai de n coups d'horloge (OSTimeDly).
- attendre un sémaphore.
- attendre un message d'une autre tâche ou d'une ISR (Interrupt Service Routine : il peut s'agir d'une fonction du noyau qui permet de modifier le statut d'une tâche

Rôle du fonction	Syntaxes du fonction
Initialisation	**OSinit, OSStart**
La gestion des tâches	**OStaskCreate, OSTaskDel, OSTaskDelReaq, OSTaskChangePrio, OSTaskSuspend, OSTaskResume, OSSchedLock, OSSchedUnlock**
La gestion du temps	**OSTimeDly, OSTimeDlyResume, OSTimeSet, OSTimeGet**
La gestion des sémaphores	**OSSemCreate, OSSemAccept, OSSemPost, OSSemPend, OSSemInit**
La gestion des boites aux lettres	**OSMboxcreate, OSMboxAccept, OSMboxPost, OSMboxPend**
La gestion des files d'attente	**OSQCreate, OSQAccept, OSQPost, OSQPend**
La gestion d'interruption	**OSIntEnter, OSIntExit**

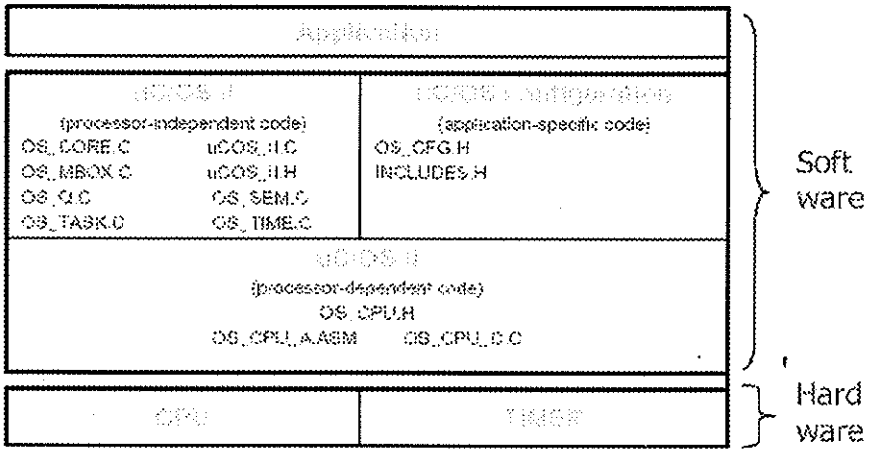
comme le degré de priorité d'une tâche).

- suspendre l'exécution de cette tâche.

Les fonctions du noyau commencent toutes par les lettres « OS » (Operating System), les lettres suivantes OS définissent les familles de fonctions de µC/OS II. La liste des fonctions de µC/OS II est la suivante :

Nous pouvons résumer la structure du système d'exploitation µC/OS II par le schéma récapitulatif suivant qui représente l'architecture formée par la partie logicielle : le code de

l'application et les services de l'OS ; et la partie matérielle : le processeur Nios (CPU et Timer)



4. Configuration du processeur NIOS

La Société Altera a développé le concept Excalibur, un ensemble de solutions et d'outils qui permettent d'intégrer de façon souple sur un même réseau logique un CPU Risc, des mémoires, des périphériques et de la logique utilisateur. Le processeur Risc Nios est disponible sous forme d'une description RTL, donc facilement adaptable par le concepteur à sa cible. Le processeur ainsi personnalisé sera ensuite « fondu » dans la logique programmable d'un FPGA au même titre que les autres éléments du système visé. Le développement d'une application passe par la personnalisation du cœur de processeur Nios suivie du choix des périphériques et de l'allocation mémoire. Le concepteur est guidé grâce à l'interface graphique appelée MegaWizard. L'architecture du processeur Nios est une machine Risc reposant sur un pipeline à quatre étages. A partir de cette base, il est possible de configurer le CPU dans plusieurs directions suivant les impératifs de l'application. La largeur des données sera de 16 ou 32 bits. L'option 32 bits rend possible la manipulation de mots de cette longueur pour le transfert de la mémoire vers les registres, et inversement, lors des opérations arithmétiques. Une fois l'utilisateur fixé sur sa personnalisation du cœur, l'ensemble des paramètres choisis est transmis au compilateur pour que ce dernier puisse prendre en compte la taille des données à manipuler, les caractéristiques de l'unité arithmétique et logique du cœur, le nombre de contextes à gérer sans avoir à faire appel à des procédures de sauvegarde en mémoire interne.

La dernière étape consiste à choisir la famille de FPGA (Apex, Acex ou Flex) dont les caractéristiques en taille de matrice et en architecture conviennent aux éléments définis pour l'application. La génération des sources VHDL ou Verilog est lancée par la commande « Generate ». Une synthèse en arrière-plan fournit automatiquement une netlist Edif compréhensible par les outils de BackEnd, Quartus et MaxPlus+II. En dernier lieu, tout l'environnement logiciel sera crée à l'image du matériel.

5. Mise en place du µC/OS-II

- **Modification des fichiers de compilation et d'édition de liens.**

Le noyau temps réel µC/OS est écrit en langage C (COSMIC SOFTWARE 4.5). Seuls les portages spécifiques au microprocesseur (ici le Nios 32) ou microcontrôleur sont écrits en langage d'assemblage. Le travail a consisté en premier lieu à adapter le noyau µC/OS au processeur Pentium III et dans une deuxième étape à le porter sur le processeur Nios d'ALTERA.

- **Programmes pour µC/OS II**

µC/OS-II est fourni avec un programme **test.c** qui permet de créer 3 tâches : une tâche de démarrage (AppStartTask) qui va créer 2 autres tâches (task#1 et task#2) lesquelles vont être mises en compétition avec des degrés de priorité différentes (la tâche de démarrage a le plus haut degré de priorité).

La figure suivante illustre simplement l'ordonnancement des 3 tâches. L'Ordonnanceur peut continuer ou suspendre une tâche grâce aux signaux « continue » et « suspend » représentés ici par des flèches en pointillés.

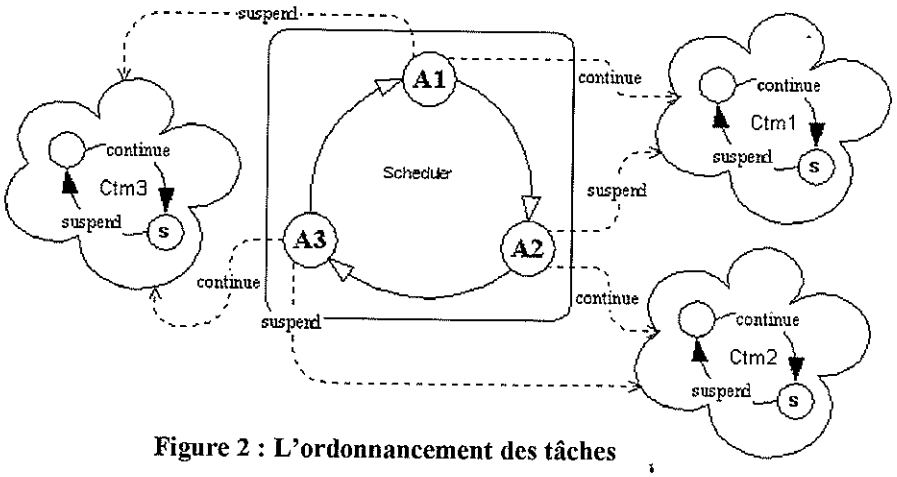


Figure 2 : L'ordonnancement des tâches

Lorsqu'une tâche est suspendue, la suivante est continuée. Le schéma montre comment sont ordonnancées les tâches mais ne précise pas comment fonctionne la communication entre les tâches ou avec les périphériques.

- **Portage du µC/OS II sur le Nios**

Le processeur Nios doit avoir les performances suivantes :

Processeur 32 bits avec 128 registres, interruptions permises et les périphériques : ROM GERMS Monitor ; UART ; Timer avec Irq =17 ; 256 Kb SRAM externe.

Il faut construire le répertoire de travail sous le répertoire crée par l'outil sopc_builder (outilde logiciel Quartus) ensuite copier tous les fichiers sources de µC/OS II et compiler avec la commande « make ». Après le chargement sur la carte Excalibur l'exécution est obtenu par la commande : nios-run.exe main.srec.

6. Exemple d'application : Détection de mouvement sur un fond d'image fixe

Pour simuler l'application détection de mouvement sur un fond d'image fixe afin de l'exécuter avec le système d'exploitation µC/OS-II, nous avons commencé par la décomposer en un nombre des tâches comme le montre la figure suivante (figure 3) :

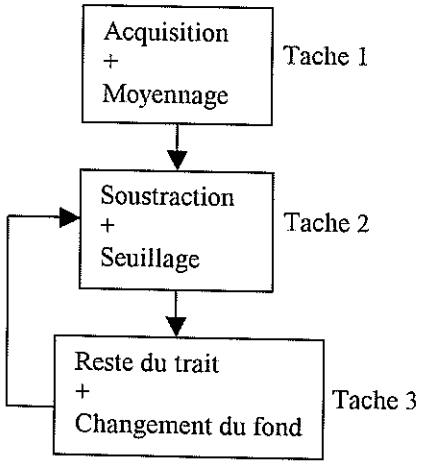


Figure 3 : Décomposition de l'application

Pour la version multitâche, nous avons lancé la simulation pour 100 itérations. Nous avons obtenu 33 itérations avec changement de l'image du fond et 67 sans changement de l'image du fond.

Le temps total est 924 ticks donc 1 itération dure en moyenne 9.24 ticks.

OV (over-head) (100 itérations) = 924 – (6*67 +33 * 8,002) = 257.934

(avec 0.002 c'est le temps mis pour le changement de l'image du fond)

Donc OV (1 itération) = 2.579 ticks = 0.1432s

Pour la version séquentielle le temps d'exécution total (dans le pire de cas) est 6.002 ticks

Donc la différence d'exécution entre une version Multi-tâches et une version séquentielle est :

9.24 – 6.002 = **3.238 ticks**.

Entre autre, et pour mieux préciser l'OV du système d'exploitation µC/OS-II, nous avons simuler d'autres exemples d'applications simples. Nous citons par exemple l'exécution d'un million d'itérations de deux tâches qui exécute chacune 1000 000 fois les instructions de sémaphore (P,V) dont la durée moyenne vaut 2 ticks d'horloge.

Le temps d'exécution totale vaut 28 ticks d'horloge.

Donc l'over-head dû à la commutation des tâches sera : OV = 28 – 3 – 2 * 2(il y a 2 tâches).

OV = 21 ticks d'horloge pour 1000 000 d'itérations.

Pour une seule itération (incrémentation du contenu de la variable + son affichage)

OV (1 itération) = 21 10^{-6} ticks avec : 18 ticks = 1s

D'où OV (1 itération) = 7/6 µs.

- ✓ **Analyse théorique :**

Pour la simulation de l'application détection de mouvement sur un fond d'image fixe, nous avons utilisé les fonctions de gestion de sémaphores et de mail box que nous pouvons avoir pour chacun de ces fonctions un intervalle du temps d'exécution donné par le concepteur:

La fonction OSSemPend : Min = 0.1714 µs et Max = 1.7428 µs.

La fonction OSSemPost : Min = 0.22857 µs et Max = 1.5142 µs.

La fonction OSMboxPend : Min = 0.2857 µs et Max = 1.8571 µs

La fonction OSMboxPost Min : 0.1714 µs et Max = 1.5142 µs

Pour notre application la valeur total Min sera égal 2.1141 µs et la valeur total Max sera égal 16.5136 µs.

Donc l'Over-Head de µC/OS-II sur cette application doit appartenir à l'intervalle [2.11 ; 16.51] µs.

L'application utilise ces fonctions dans les trois tâches de la façon suivante :

Tâche 1 : P (sem1) , MboxPost

Tâche 2 : MboxPend, V (sem1), P (sem2), V (sem3), MboxPost.

Tâche 3 : MboxPend, P(sem3), V(sem2), avec une boucle de recherche d'un nombre aléatoire qui précise s'il y a un changement du fond ou pas.

Le temps total d'un million d'itération est de 52 ticks donc $52/1000\,000 = 52\ 10^{-6}$ ticks = 2.88 µs.

Conclusion :

Nous avons présenté dans ce papier le portage du système d'exploitation temps réel : µC/OS II sur le processeur NIOS 32 et une étude de cas : la gestion de l'application détection du mouvement sur un fond d'image fixe. L'apport des services d'un noyau temps réel est considérable surtout pour les applications qui présentent un taux de parallélisme assez important.

Nous souhaitons comme perspective de ce travail, concevoir un système d'exploitation temps réel pour architectures embarquée distribué c'est-à-dire que certains services vues « critiques » (exemple le scheduler) vont être mappés sur les unités matérielles de l'architecture.

Références :

[1] S. Bilavarn, Exploration Architecturale au Niveau Comportementale- Applications aux FPGAs. Thèse de doctorat de l'université de Bretagne Sud, Février 2002.

[2] F.GHAFFARI, M.AUGUIN, M.BENJEMAA. Partitionnement d'applications à temps d'exécution variable sur architectures reconfigurables. JFAAA'2002 Monastir, TUNISIE 16 – 18 Décembre 2002.

[3] F.GHAFFARI, Etude du Partitionnement Logiciel/Matériel d'applications à distribution variable de charge de calcul, Rapport de stage de DEA effectué à I3S université de Nice Sophia-Antipolis, 2001/2002.

[4] Z. Peng, and K. Kuchcinki, "An Algorithm for Partitionning of Application Specific Systems", Proc.European Design & Test Conference (EDAC-ETC-EuroASIC), IEEE CS Press, February 1993.

[5] Nios Embedded Processor Getting Started Version 1.1 user Guide, ALTERA March 2001.