Estimation du temps d'exécution des systèmes sur puce temps réel

Kaïs Loukil[1,2],Yassine Aoudni[1,2],Guy Gogniat[2],Mohamed Abid[1],Jean Luc Philippe[2]

[1] Ecole Nationale des Ingénieurs de Sfax Laboratoire CES Sfax, Tunis
[2] Université de Bretagne Sud Laboratoire LESTER Lorient, France
Kais_loukil@yahoo.fr

Résumé: l'article suivant présente une méthode d'estimation du temps d'exécution d'une application temps réel dans le cadre des systèmes sur puce. La méthode proposée est basée sur une approche mixte statique/dynamique. L'objectif est d'estimer le surcoût ajouté par le système d'exploitation temps réel. Partant d'une application sans noyau temps réel, on estime le temps d'exécution de l'application complète avec un système d'exploitation temps réel. Notre méthode peut être utilisée pour évaluer différentes solutions d'implémentation pour une application temps réel. L'implémentation de l'algorithme de la synthèse d'image 3D sur différentes cibles architecturales a été choisie pour valider la méthode proposée. Les résultats obtenus ont permis de déduire l'apport du système d'exploitation temps réel et le surcoût induit afin d'étudier son intérêt lors de l'intégration d'une application temps réel.

I. Introduction

Les systèmes sur puce ont subi des évolutions telles que l'intégration de plusieurs processeurs, accélérateurs et coprocesseurs dans une seule puce qui devient de plus en plus classique. En plus, et du fait de la complexité croissante des applications embarquées, de la présence de fortes contraintes temps réel, de la limitation des ressources disponibles, tant en mémoire qu'en énergie et en puissance de calcul, et du fait également de la pression exercée par le marché sur ces produits, l'usage de systèmes d'exploitation temps réel (RTOS[1]) est devenu nécessaire dans les systèmes embarqués [3].

Cependant, et bien que le système d'exploitation temps réel soit couramment employé dans les systèmes embarqués spécifiques, celui-ci peut entraîner certains inconvénients en terme de coût, consommation et performances [9][14].

D'autre part, les systèmes temps réel sont soumis à plusieurs contraintes temps réel. Ce qui a donné naissance à des outils d'estimation de performance des systèmes temps réel afin de vérifier le respect des contraintes temps réel.

---

[1] Real Time Operating System

Nous nous intéressons dans ce papier à étudier un modèle permettant de déterminer le surcoût dû à l'utilisation d'un système d'exploitation temps réel. Cette méthode peut être utilisée pour calculer le temps d'exécution d'une application temps réel dans le pire cas, ou pour évaluer différentes solutions d'implémentation d'une application temps réel. Les résultats obtenus permettent de déduire l'apport du système d'exploitation temps réel et le surcoût induit afin d'étudier son intérêt lors de l'intégration d'une application temps réel.

La section suivante rappelle les principales méthodes d'estimation. La troisième section présente le flot du modèle généré ainsi que les facteurs intervenant sur la variation du temps d'exécution d'une application temps réel. Les résultats expérimentaux et l'interprétation seront résumés dans la section quatre. Nous finissons ce papier par les conclusions et quelques perspectives.

II. Etat de l'art

Les méthodes d'estimation que l'on trouve dans la littérature peuvent être classées dans trois catégories : statiques, dynamiques et mixtes [1].

Méthode statique : l'estimation de performance d'une solution est le résultat d'une analyse statique d'une spécification, (exemple : analyse de chemins dans une spécification de flots de contrôle).

Méthode Dynamique : les mesures de performance d'une solution sont le résultat d'une analyse dynamique d'une spécification, (exemple : simulation).

Méthode Mixte dynamique/statique : C'est l'utilisation de quelques éléments des deux approches précédentes pour l'analyse de performance d'une solution.

Les approches statiques sont en général très rapides, mais elles présentent une précision moyenne vu la distance qui sépare l'implémentation et la spécification [1]. Les approches dynamiques sont plus précises mais en contre partie plus lentes vu le temps pris pour l'obtention du modèle à simuler et le temps de la simulation. Ces inconvénients limitent l'utilisation de ces deux méthodes. La plupart des outils d'estimation existant utilisent donc des méthodes mixtes pour bénéficier des avantages des deux méthodes précédentes. Dans la suite on va présenter quelques méthodes utilisées pour l'estimation du temps d'exécution d'une application.

La méthode décrite dans [4] se base sur des analyses statiques (calcul du temps d'exécution basé sur le code assembleur) et dynamiques (profilage). Alors que pour [6] [5] ils calculent des métriques séparées pour le logiciel, le matériel et la communication, puis, ils utilisent ces métriques dans des équations particulières. Pour la partie logicielle, ils calculent le temps d'exécution dans le pire cas en utilisant des techniques d'analyse de chemin. [13][8] procèdent à des estimations de performance en utilisant des techniques d'estimation du temps d'exécution à bas niveau pour le logiciel, le matériel et la communication. Les auteurs de [12] attaquent le problème d'un point de vue générique. Ils analysent, au niveau système, l'interaction entre les différents processus en donnant le meilleur et le pire délai pour chacun d'entre eux. Ensuite, en partant d'un graphe acyclique représentant les dépendances de données entre les processus, ils calculent le temps d'exécution, dans le pire cas, pour le système

entier. Dans [2], ils décrivent une méthode d'estimation mixte statique/dynamique. Elle est faite en deux étapes :

- Pré-estimation : Un profilage de la description du système est réalisé pour obtenir des temps d'exécution pour différents niveaux (processus, bloc de base, communication).
- Estimation en ligne : Les résultats obtenus durant la phase de pré-estimation sont utilisés dans des expressions complexes pour le calcul de la performance globale du système.

La plupart de ces outils offre une bonne estimation pour la partie logicielle ; mais, pour la partie matérielle, on trouve toujours des valeurs estimées différentes des valeurs réelles : ce qui influe sur l'estimation du temps d'exécution du système complet vu la distance qui sépare la spécification du système de l'implémentation hardware. En plus les travaux cités sont basés sur des techniques d'estimation par analyse abstraite de haut niveau (analyse des chemins critique).Nous proposons dans ce papier une méthode mixte (statique/dynamique) d'estimation du temps d'exécution d'une application temps réel. Cette méthode se base sur un prototype du système pour l'estimation des performances de la partie matérielle et une analyse statique de l'application pour estimer le temps pris par le logiciel. Cette méthode peut être utilisée pour calculer le temps d'exécution d'une application temps réel dans le pire cas afin de vérifier si le système respecte les contraintes temporelles qui lui sont imposées ou non et pour évaluer différentes solutions d'implémentation d'une application temps réel à travers différentes routines offertes par le système d'exploitation temps réel. Dans la section suivante on présentera les étapes qui ont conduit à la mise en place de notre modèle.

III. Approche d'estimation du temps d'exécution d'une application temps réel

Dans un système temps réel, plusieurs facteurs peuvent influer sur le temps d'exécution. Parmi ces facteurs on cite :

- L'architecture cible (mono multiprocesseur) avec ou sans accélérateurs et coprocesseurs
- Service du système d'exploitation temps réel utilisé
- Temps de communication entre les différents modules du système (HW/SW, SW/SW)

Dans la suite de cette section on présente les facteurs pris en compte et les étapes qui ont conduit à la mise en place du modèle.

III.1. Principe de la méthode

L'exploration de l'espace de solutions est une composante primordiale dans un flot de conception conjointe matérielle/logicielle. Le problème à résoudre dans le flot de conception des systèmes complexes consiste à trouver la meilleure solution du sys-

tème incluant le découpage fonctionnel du système, la détermination des protocoles de communication et le placement/ ordonnancement, etc [1].

Le modèle qu'on propose permet facilement aux concepteurs d'explorer différentes solutions pour une application écrite avec les services d'un système d'exploitation temps réel (différents types de communications entre tâches, mécanismes de synchronisations ….) sans avoir recours à chaque fois à réécrire l'application avec les services voulus

Notre idée de départ se base sur la réalisation d'un prototype qui contient tous les modules matériels. En utilisant ce prototype on peut calculer le temps pris par l'application écrite sans utilisation des services du système d'exploitation temps réel. Donc, en partant du temps d'exécution d'une application non temps réel on peut estimer le temps de l'application temps réel en lui ajoutant le temps pris par chaque service utilisé et le temps de communication entre les différentes tâches de l'application et une certaine valeur due à l'utilisation du système d'exploitation temps réel.

On se propose en premier lieu de définir un modèle qui puisse déterminer l'effet de l'utilisation d'un système d'exploitation sur le temps d'exécution de l'application. En second lieu, on construira une base de données qui contiendra tous les services qu'on peut utiliser d'un système d'exploitation temps réel, ainsi que leurs temps d'exécution. A chaque fois qu'on emploie un service de ce système d'exploitation temps réel dans notre application, on ajoute le temps approprié au temps déjà calculé. La figure 1 illustre les étapes de notre approche d'estimation du surcoût d'un système d'exploitation temps réel. Partant d'une description de l'architecture de notre système, on construit un prototype du système. Ensuite, on exécute le code de l'application sur ce prototype et on mesure son temps d'exécution. Puis en appliquant le modèle construit et en utilisant le graphe de séquences de l'application, on peut estimer le temps de l'application si elle est écrite avec les routines d'un système d'exploitation temps réel.

Dans cette section, on va présenter en premier lieu les différents facteurs influant sur le temps d'exécution d'une application temps réel. En second lieu nous détaillons les différentes étapes faites pour la mise en place du modèle. On termine par la présentation de la plateforme utilisée ainsi que les tests réalisés pour la génération du modèle.

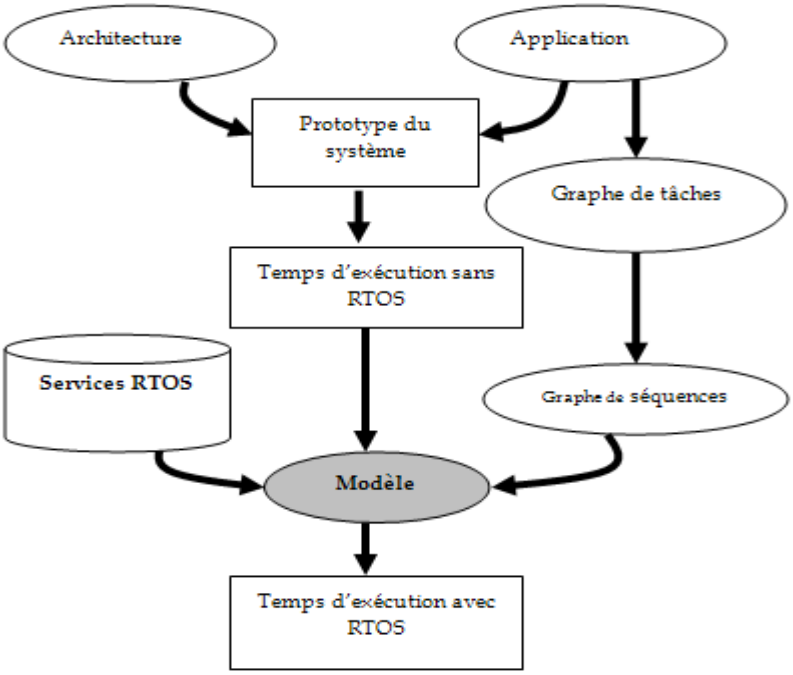

Fig.1. Approche d'estimation

Afin de dégager un modèle d'estimation du temps d'exécution d'une application temps réel, des analyses et des mesures ont été effectuées sur une plateforme à base de la technologie FPGA de chez ALTERA et un système d'exploitation temps réel embarqué MicroC/OS-II [7] [10] [11].

III.2. Evaluation de l'effet du MicroC/OS-II sur le temps d'exécution d'une fonction

Afin de pouvoir générer un modèle qui puisse déterminer l'effet de l'utilisation d'un système d'exploitation temps réel sur le temps d'exécution d'une application, on a procédé aux étapes suivantes :

- On a écrit une fonction qui fait un traitement quelconque. Cette fonction a été exécutée sur une plateforme monoprocesseur et on a pris son temps d'exécution sans utiliser un système d'exploitation temps réel.
- On a pris le même code de la fonction et on a mesuré son temps d'exécution sur la même plateforme, mais dans une application temps réel. Cette dernière se compose uniquement de la tâche qui contient le code de la fonction sans utiliser des routines offertes par notre système d'exploitation temps réel (dans cette étape, on ne mesure pas le temps de création de la tâche et d'activation des services du système d'exploitation temps réel mais le temps d'exécution de la portion du code qui exécute la même fonction déjà mesurée à l'étape précédente).

Le tableau 1 illustre les résultats trouvés lors de l'exécution de différentes applications sur notre plateforme qui se compose essentiellement du processeur NIOS II et du système d'exploitation MicroC/OS-II.

Tableau1. Mesure du temps d'exécution avec et sans système d'exploitation temps réel

	Temps sans RTOS(tics)	Temps avec RTOS (tics)
App1	90294	91106
App2	449559	451645
App3	897117	901089
App4	4477789	4495777
App5	8956136	8990554
App6	44779001	44950710
App7	89556865	89897197
App8	447781676	449484289
App9	895561972	898968551

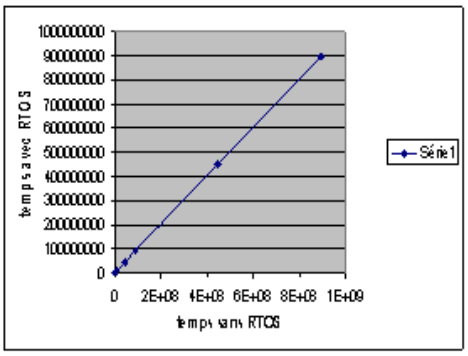

Fig.2. Courbe d'estimation

- A partir des mesures déjà effectuées dans les étapes précédentes, on a construit la figure 2. On constate que l'ensemble des points forme une droite linaire d'équation Y=1.0038X (X étant le temps de l'application sans RTOS, Y est le temps de la même application exécutée dans une seule tâche). Cela est dû au fait que chaque système d'exploitation temps réel possède des tâches système qui interviennent d'une façon périodique dans l'exécution de l'application.

Donc, en utilisant le graphe déjà construit, on peut déterminer le temps d'exécution de n'importe quelle application temps réel, tout en sachant son temps d'exécution sans système d'exploitation temps réel, bien évidemment sans utiliser les services offerts par le système d'exploitation temps réel.

III.3. Mesure du temps pris par les services du système d'exploitation temps réel

Une application écrite en utilisant les routines d'un système temps réel se compose essentiellement d'un ensemble de tâches. Ces tâches utilisent les différents services offerts par le système d'exploitation temps réel, pour gérer la communication et la synchronisation entre elles afin de réaliser la fonction globale de l'application.

On peut classer les services d'un système d'exploitation temps réel en deux groupes :

- Des services qui permettent d'une part la création des différentes tâches, mécanismes de synchronisation et de communication, et l'initialisation du système d'exploitation temps réel; et d'autre part, le démarrage de l'application temps réel. Généralement ces services n'entraînent pas de changement de contexte.
- Des services de communication et de synchronisation. Généralement appelés dans le code des tâches à des moments bien déterminés pour réaliser la fonction globale du système. L'appel de ces services peut causer parfois des changements de contexte.

Pour le premier groupe, on constate que le temps pris par n'importe quel service est indépendant du contexte là où il est appelé, puisqu'ils n'entraînent pas de changements de contexte. Par conséquent, ce temps restera le même peu importe le moment d'utilisation. Alors que les services du deuxième groupe sont plus complexes puisqu'ils exigent un ré-ordonnancement du système et peuvent entraîner des changements de contexte. Donc, il faut mesurer le temps pris par ces services dans les deux cas :

- Appel du service mais pas de changement de contexte : dans ce cas, on mesure le temps pris par l'appel du service et l'exécution de l'instruction qui le suit.
- Appel du service avec un changement de contexte : dans ce cas, on mesure le temps pris par l'appel du service approprié et l'exécution de la première instruction de la nouvelle tâche qui va être exécutée.

Le tableau 2 présente les mesures de quelques services offerts par MicroC/OS-II :

Tableau2. Temps pris par les services du MicroC/OS-II

Services du MicroC/OS-II	**tics**
Création d'une tâche OSTaskCreateExt	9756
Fonction OS_Start	827
Création d'un mailbox OSMboxCreate	679
Création d'un message queue OSQCreate	1565
Création d'un drapeau d'évènement OSEventFlagCreate	419
OSEventFlag sans changement de contexte	876
OSEventFlag avec changement de contexte	4464
OSMboxPost sans changement de contexte	854
OSMboxPost avec changement de contexte	3418
OSQPost sans changement de contexte	912
OSQPost avec changement de contexte	3966

III.4 . Formalisation du modèle

On considère une application qui consomme « n » cycles d'horloge lors de son exécution sur la plateforme Altera sans utiliser les routines du MicroC/OS-II.

Pour pouvoir estimer le temps de cette application lorsqu'elle est exécutée en utilisant les services d'un système d'exploitation temps réel, elle sera décomposée en un ensemble de tâches pour réaliser la fonction globale du système. A partir du graphe de tâches construit, on doit extraire le diagramme de séquences qui décrit la succession des différentes tâches dans le pire cas ainsi que les routines du système d'exploitation temps réel utilisées pour assurer la synchronisation et la communication entre elles.

III.5. Mise en équation

Pour calculer le temps global de l'application écrite avec les routines du système d'exploitation temps réel, il faut :

Déterminer la nouvelle valeur du temps d'exécution de l'application en utilisant le modèle.

En utilisant le graphe de l'application, ajouter, à chaque fois qu'on utilise un service du système d'exploitation temps réel, le temps approprié, à partir du tableau déjà mesuré.

Le modèle proposé peut se récapituler dans l'équation (E1) :

$$Ntr = N \times 1.0038 + \sum_{i=n}^{i=0} T_i(S_i) \qquad (1)$$

- Ntr : nombre de tics de l'application temps réel.
- N : nombre de tics de l'application sans système d'exploitation temps réel.
- Ti(Si) : nombre de tics du service i déterminé à partir du tableau construit.
- n : nombre de services utilisés.

Ainsi le modèle est mis en place. La section suivante présente les différentes mesures effectuées pour la validation du modèle.

IV. Expérimentation

Afin de valider le modèle proposé, on utilise comme application la synthèse d'images 3D sur différentes cibles architecturales.

IV.1. Application de traitement d'images 3D (Pipeline 3D)

Le pipeline 3D est l'ensemble des étapes nécessaires pour la création et la visualisation d'une image 3D. Cette chaîne est décomposée en un ensemble d'opérations nécessaires pour afficher un objet 3D observé à partir d'une position et avec une orientation donnée. Une mise en forme est présentée dans la figure 3.

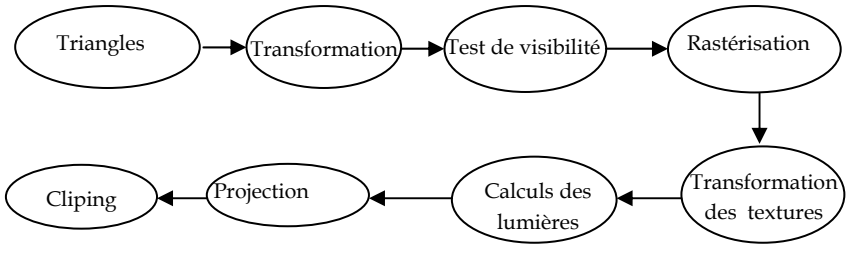

Fig.3. pipeline graphique 3D

IV.2. Modèle de tâche :

Cette application a été décomposée en 11 tâches qui coopèrent entre elles (figure 4) Les mécanismes de communications et de synchronisations entre les différentes tâches du système ont été choisis arbitrairement pour la validation du modèle.

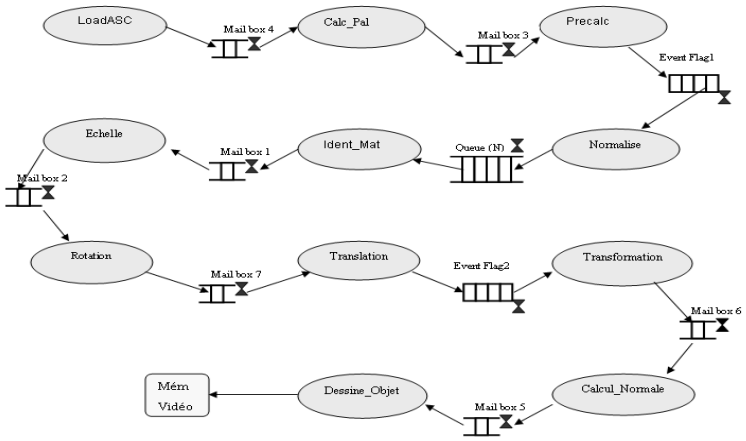

Fig.4. Diagramme de séquences

IV.3. Conception de l'architecture de prototypage:

Afin de mettre en place un système multiprocesseur, plusieurs types d'architectures sont envisagés [1]:

- Les systèmes à mémoire partagée: Ce type de mémoire présente l'avantage de permettre un partage immédiat des données, facilitant la programmation. Mais cette solution coûte cher, ce qui limite le nombre de processeurs pouvant être ajoutés sur une même mémoire.
- 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 directe sur celle des autres processeurs. Cela suppose donc de mettre en place une communication explicite entre les processeurs.
- 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 grâce à un réseau. Cela permet, dans une certaine mesure, de tirer les avantages des deux précédentes architectures et d'en réduire les inconvénients.

Pour pouvoir choisir l'une des solutions proposées il a fallu étudier les caractéristiques de la plateforme de prototypage et du bus qui assure la communication entre les différents processeurs.

IV.3.1. Bus Avalon

C'est le bus utilisé par Altera. Il 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 est un bus multi maîtres simultanés. Les maîtres peuvent accéder simultanément à leurs esclaves et en cas de besoin les maîtres peuvent échanger des données à travers une mémoire partager. Généralement l'accès à cette mémoire est géré par le système d'exploitation utilisé.

Comme solution on va utiliser un module d'un RTOS implémenté en hardware (mutex) pour assurer la gestion des accès aux mémoires partagées entre les différents processeurs [15].

IV.3.2.Le Mutex :

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 [15].

IV.3.3.Architecture multiprocesseur proposer

Suite à l'étude des caractéristiques de l'environnement de conception nous proposons une plateforme qui 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 fourni avec l'environnement d'Altera. Puisque le bus avalon est simultané multi-maître on a proposé d'utiliser pour chaque processeur deux mémoires, l'une est propre pour lui et l'autre est partagée avec les autres processeurs en écriture. Tous les messages qui lui sont destinés sont mis dans cette mémoire et c'est lui seul qui peut lire son contenu.

Après avoir conçu notre architecture multiprocesseur, nous avons ajouté l'ensemble des coprocesseurs et d'accélérateurs spécifiques à la synthèse d'image 3D. Ainsi notre plateforme de prototypage est prête figure 5.

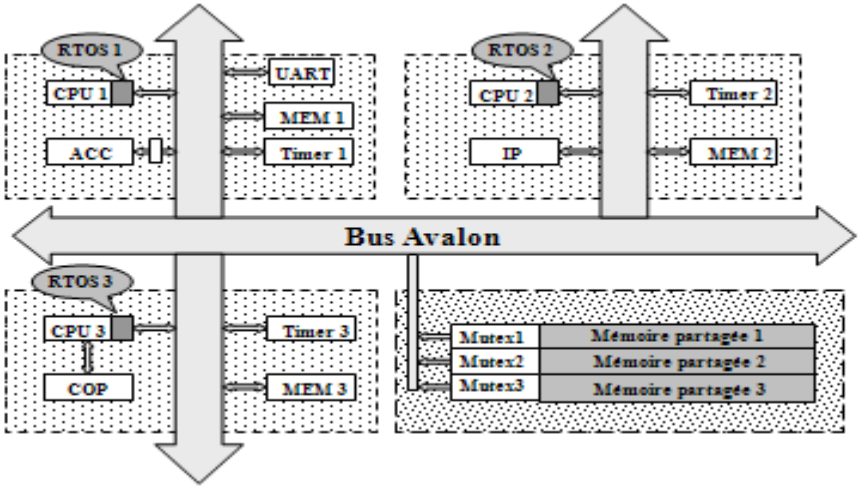

Figure 5 : architecture proposée

IV.4. Mesures du temps d'exécution

Toutes les mesures faites visent l'application de synthèse d'images 3D. Cette application a été testée sur différentes cibles architecturales (monoprocesseur, coprocesseurs, accélérateurs, multiprocesseur). A chaque fois on mesure le temps d'exécution par prototypage et on détermine la valeur obtenue en appliquant le modèle mis en place pour déterminer l'erreur du modèle.

tableau3. Mesures et résultats

	1 cpu	**1cpu+coproc**	**1cpu+acc**	**2 cpu**	**2cpu + coproc**
Total ALUTS	3505 (7%)	4676 (9%)	4254 (8%)	9020 (18%)	9153 (18%)
Temps d'exécution sans RTOS	1066889330	716576082 (**-32 ,83%**)	707680249 (**-33,66%**)	712248263 (**-33,24%**)	558288052 (**-47,67%**)
Temps d'exécution avec RTOS	1071095628	720223958	707872710	717162554	570376431
Temps d'exécution avec modèle	1074048767	719451189	710521551	715106924	560561644
Erreur du modèle	**0.27%**	**0.1%**	**0.37%**	**0.28%**	**1.72%**

En premier lieu l'application de synthèses d'images 3D a été exécutée sur différentes cibles architecturales sans l'utilisation d'un système d'exploitation et des temps d'exécution ont été pris. Cette valeur comprend le temps de communication entre le processeur et ses périphériques et la communication inter processeurs (pour cette raison ces valeurs ne sont pas tenues en compte dans notre modèle). En second lieu cette application a été décomposée en un ensemble de tâches qui coopèrent entre elles tel que décrit dans la figure 4 et on a mesuré le nouveau temps d'exécution de l'application sur chaque architecture. En troisième lieu on a estimé le temps d'exécution en utilisant le modèle et on a terminé par la comparaison entre les valeurs estimées et celles obtenues par exécution, pour déterminer l'erreur du modèle. Le tableau 3 récapitule les résultats obtenus. On remarque que notre modèle offre un taux d'erreurs faible (inférieur à 2%) et ce pour plusieurs architectures testées et met en évidence l'utilisation des mécanismes de communication inter processeurs.

V. Conclusion

Le présent article a abordé les problèmes liés à l'estimation de performance des systèmes sur puce. Nous avons développé et illustré une nouvelle méthode d'estimation du temps d'exécution d'une application temps réel. En effet notre méthode se base sur une approche mixte statique/dynamique. Nous avons présenté toutes les étapes qui nous ont conduit à la validation de la méthode proposée sur différentes cibles architecturales en utilisant un environnement de conception conjointe logicielle/matérielle et le noyau temps réel MicroC/OS-II.

Cette méthode permet l'estimation du temps d'exécution d'une application temps réel.

Le modèle a été formulé en se basant sur une palteforme d'Alétra à base de FPGA. Toutefois, la même démarche peut être appliquée sur d'autre plateforme pour la mise en place d'un modèle similaire. En effet, des travaux sur une plateforme Xilinx sont en cours pour la validation de l'approche d'estimation présentée.

Par ailleurs, Le modèle proposé présente des résultats intéressants : un taux d'erreurs faible. Toutefois, il peut être raffiné et ce en tenant compte de la taille de la mémoire cache et le nombre de défauts de cache qui peut influer sur le temps d'exécution de l'application. Ceci représente la suite des travaux futurs.

REFERENCES

1. A .Baghdadi, exploration et conception systématique d'architectures multiprocesseur mono-puces dédiées à des applications spécifiques, thèse PhD, Mai 2002, TIMA France.
2. D.Gajski, F.Vahid, S.Narayan and J.Gong System-level Exploration with SpecSyn. Design Automation Conference, Juin 1998.
3. D.Lyonnard, S.Yoo, A.Baghdadi, A.A.Jerraya: Automatic Generation of Application-Specific Architectures for Heterogeneous Multiprocessor System–on-Chip. DAC 2001.
4. H.J.Eikerling, W.HARDT, J.Gerlack, W.Rosenstiel: A Methodology for Rapid Analysis and optimization of Embedded Systems. International IEEE Symposium and workshop on ECBS, D-friedrichshafen, Mars 1996
5. J.Grode and J.Madsen Performance Estimation for Hardware/Software Codesign using Hierarchical Colored Petri Nets. Proceedings of Jigh Performance Computing'98, in Special Session on Petri Net Applications and HPC, Boston, Avril 1998.
6. J.henkel and R. Emst, High-level Estimation Techniques for usage in hardware/software codesign. Asia and south Pasific Automation Conference Yokohama, Japan, Fevrier 1998
7. J.J.Labrosse, "MicroC/OS-II, the Real-Time Kernel", Second Edition.
8. J.Liu, M.Lajolo and A.Sangiovanni-Vincentelli, Software Timing Analysis Using HW/SW Cosimulation and Instruction Set Simulator. International Workshop on Hardware-Software Codesign, Mars 1998.
9. L.Gauthier, Génération de système d'exploitation pour le ciblage de logiciel multitâches sur des architectures multiprocesseurs hétérogènes dans le cadre de systèmes embarqués, thèse PhD, décembre 2001, TIMA France.
10. P. Mabilleau Systèmes en temps réel, GEI 2002.
11. Site du noyau uCO/S : http://www.ucos-ii.com/.2007
12. T-Y.Yen and W.Wolf, Communication Synthesis for Distributed Embedded Systems. International Conference on Computer-Aided Design,1995.
13. S. Rouxel Modélisation et Caractérisation de Plates-Formes SoC Hétérogènes : Application à la Radio Logicielle, thèse PhD, décembre 2006, UBS
14. A. Morton W. M. Loucks A Hardware/Software Kernel for System on Chip Designs ACM Symposium on Applied Computing 2004
15. Site d'altera : http:// www.altera.com 2007