République Tunisienne
Ministère de l'Enseignement Supérieur,
de la Recherche Scientifique et des
Technologies de l'Information et de la
Communication

Université de Sfax
École Nationale d'Ingénieurs de Sfax

Cycle de Formation d'Ingénieurs
dans la Discipline
Génie Informatique

ST-EN07/00

Projet de Fin d'Etude
N° d'ordre: 2014 / DIMA-05

MEMOIRE

présenté à

L'Ecole Nationale d'Ingénieurs de Sfax
(Département de Génie Informatique et de Mathématiques Appliquées)

en vue de l'obtention

du Diplôme National d'Ingénieur en *Génie Informatique*

par

Amal CHAKER

Etude et implémentation d'une approche de gestion de la consommation d'énergie utilisant l'outil de prototypage virtuel « Platform Architect » de Synopsys.

soutenu le 15 Septembre 2014, devant la commission d'examen:

Mme.	**Mouna BAKLOUTI**	Président
Mme.	**Sonda CHTOUROU**	Examinateur
M.	**Maher BEN JEMAA**	Encadrant
M.	**Michel AUGUIN**	Encadrant
Mme.	**Hend AFFES**	Encadrant

Etude et implémentation d'une approche de gestion de la consommation d'énergie utilisant l'outil de prototypage virtuel « Platform Architect » de Synopsys.

Amal CHAKER

الخلاصة

إن استهلاك الطاقة عند الرقاقات الالكترونية ذو أهمية متزايدة نظرا لتعقد أنظمة التشغيل بها و تطور تصاميمها. لمواجهة هذه المشكلة ظهرت العديد من التقنيات للتحكم في استهلاك الطاقة مثل « clock gating » و« power gating ».

إن استخدام هذه التقنيات في تصميم هذه الأنظمة يوفر إمكانيات هامة في التقليص من استهلاك الطاقة. و في هذا السياق فان الأهداف الرئيسية لهذا المشروع يتمثل في تصميم نموذج افتراضي يشغل تطبيق « video capture » ثم تصور تسلسل المهام في هذا التطبيق مع مرور الوقت و ذلك في هدف تقييم تأثير اختيارات تنفيذ المهام المختلفة على الهيكل المقترح من ثم دراسة و تنفيذ استراتيجية أو أكثر من اجل تحسين التحكم في استهلاك الطاقة.

Résumé

La consommation d'énergie des systèmes sur puces (SoC) est de plus en plus importante vu la complexité des fonctionnalités intégrées et l'évolution des architectures. Face à ce problème, plusieurs techniques de gestion de la consommation d'énergie ont été définies comme « Power Gating » et « Clock Gating ». L'utilisation de ces techniques, pour la conception des systèmes sur puces (SoC) offre des possibilités intéressantes de réduction de la consommation d'énergie. Dans ce contexte, les objectifs principaux de ce stage consistent à modéliser une plateforme virtuelle fonctionnelle exécutant une application « Video_Capture » de Intel puis visualiser le séquencement des tâches de cette application au fil du temps afin d'évaluer l'impact des choix d'implémentation des différentes tâches sur l'architecture proposée. Ensuite, étudier et implémenter une ou plusieurs stratégies de « Power Management » en vue d'optimiser la consommation d'énergie des ressources de traitement de cette plateforme.

Abstract

Power consumption of Systems-on-Chip (SoC) is increasingly important, due to the complexity of integrated features and the evolution of architectures. In order to solve this problem, several techniques for managing energy consumption were defined, for example « Power Gating » and « Clock Gating ». Using these techniques for Systems-on-Chip design, the energy consumption can be reduced remarkably. In this context, the main objectives of this internship is to model a functional virtual platform for running the « Video_Capture » Intel application, then visualize the sequencing of tasks in this application over the time, which is aiming at assessing the impact of implementation choices of different tasks on the proposed architecture.

In order to optimize the energy consumption of processing resources of the platform, one or more strategies of « Power Management » are studied and implemented.

المفاتيح: الإلكترونية الرقاقات , power gating الطاقة استهلاك في التحكم ,افتراضي نموذج clock gating

Mots Clés: Systèmes sur puces (Soc), Power Gating, Clock Gating, plateforme virtuelle, Power Management.

Keywords: System-on-Chip (SoC), Power Gating, Clock Gating, virtual platform, Power Management.

Remerciements

J’ai eu une occasion très agréable de m’intégrer au sein d’une équipe synchrone qui travaille en coopération professionnelle.

En premier lieu, je tiens à exprimer ma sincère gratitude envers M. Michel Auguin le directeur de recherche CNRS qui m’a très bien accueilli, qui m’a mis dans de très bonnes conditions de travail et qui m’a apporté son appui et son encouragement tout au long de ma période de stage.

J’adresse mes remerciements à M. Maher Ben Jemaa le directeur de département d’Informatique et Mathématique Appliquées à l’ENIS qui m’a accordé sa confiance et qui m’a suivi pendant l’avancement de mon projet.

J’exprime ma très haute considération à Mme. Hend Affes qui a consacré son temps pour répondre à mes questions, qui m’a guidé dans l’approche du projet et qui m’a donné une certaine rigueur et méthode de travail.

Je voudrais aussi remercier ma famille et mes amis, pour leur soutien inconditionnel.

Enfin je remercie tous les membres du jury d’avoir accepté de participer à ma soutenance.

Sommaire

I. Introduction:

I.1 Introduction générale:

« Les systèmes embarqués nous entourent et nous envahissent littéralement, fidèles auposte et prêts à nous rendre service. Ils sont donc partout, discrets, efficaces dédiés à ceà quoi ils sont destinés. Omniprésents, ils le sont déjà et le seront de plus en plus ».

Patrice Kadionik de l'ENSEIRB

L'originalité de cette citation insiste sur le déploiement considérable des systèmes embarqués dans notre vie quotidienne. Il est intéressant donc de constater que nous nous servons de plus d'une dizaine de systèmes embarqués chaque jour souvent sans s'en rendre compte.

La conception de tels systèmes embarqués (System on Chip : SoC) est alors un grand défis que doivent relever les concepteurs afin de répondre à une série de contraintes comme la sûreté, la tolérance aux fautes, la robustesse, la fiabilité, la surface, et la consommation énergétique.

Dans ce stage, nous nous intéressons à la contrainte de consommation énergétique de ces systèmes. Ainsi, le travail a été réalisé au sein du laboratoire d'Electronique, Antennes et Télécommunications **(LEAT)** pendant une durée d'environ 6mois.

Ce stage se situe dans le cadre du projet **ANR HOPE** (Hierarchically Organized Power/Energy management).Ce projet **HOPE** vise à étudier et développer une approche pour la conception au niveau transactionnel d'architecture power de SoC (System on Chip) où l'énergie, la puissance, la température et la performance sont les principales contraintes.

Les travaux portent sur :

- La modélisation abstraite et les techniques associées pour explorer des solutions offrant de bons compromis performance/power/température.
- La validation au niveau T

LM de la ou des solutions précédentes en considérant le système complet : architectures matérielle, logicielle et power

- Proposer une approche de hiérarchisation du power management

Les travaux amont du projet HOPE s'appuieront sur des environnements de développement s'appuyant sur le prototypage virtuel fournis par les partenaires du projet : Synopsys, Magillem et Docea Power.

Le but du stage est donc d'utiliser les outils fournis pour modéliser et valider, dans ces types de plateformes, les aspects non fonctionnels, particulièrement la gestion de l'énergie consommée « ou power », conjointement avec les aspects fonctionnels.

I.2 Présentation du laboratoire:

Le Laboratoire d'Electronique, Antennes et Télécommunications (LEAT) est une unité mixte qui regroupedifférents activités de recherche qui sont organisés en 3 thématiques :
-La thématique MCSOC (Modélisation, Conception Système d'Objets Communicants)
-La thématique CMA (Conception et Modélisation d'Antennes)
-La thématique S2DS2A (Systèmes de Détection, D'imagerie et Systèmes Antennaires Associés)

Fin septembre 2012, le LEAT a déménagé au sein du campus SophiaTech qui devient un véritable moteur d'animation de la technopole. Le bâtiment Forum dans lequel est hébergé le LEAT accueille également plusieurs pôles de compétitivité, de nombreuses associations et des plateformes technologiques (Conception CIMPACA, Télécoms).

II. Problématique:

II.1 Optimisation de la consommation d'énergie dans un système sur puce:

Depuis des années, la consommation en énergie est devenue un problème crucial dans la conception des circuits électroniques dans différents domaines (avionique, domotique, automobile, etc…) qui augmentent en nombres de transistors et en fonctionnalités intégrées de plus en plus complexes.

Les nouvelles applications telles que les ordinateurs portables, les télécommunications sans fil et les applications multimédias possèdent la plus forte croissance du secteur électronique, en particulier de l'électronique mobile.

Les projections dans le futur montrent que cette croissance doit continuer tout en garantissant une durée de vie importante des batteries d'alimentation.

Également, plus que jamais, nous avons le besoin d'optimiser la consommation d'énergie de ces systèmes embarqués qui devient dans nos jours une priorité majeure et un grand défi lors de leur conception. Ce stage s'inscrit dans ce contexte de gestion de l'énergie consommée par un système sur puce. Le tableau 1 montre l'impact de l'évolution et des tendances des systèmes sur puces modernes sur la consommation d'énergie [7].

Tableau 1. Les tendances de multimédia agissant sur la consommation d'énergie

Tendances	Impacts sur la consommation d'énergie
Migration vers des hautes résolutions vidéo (HD)	Exige une plus grande efficacité au niveau des performances et de l'énergie
Les nouvelles applications telles que les jeux 3D, multimédia, navigation web, e-mail…	Les applications gourmandes en puissance qui drainent plus rapidement la batterie. Utilisation plus active des smartphones par les utilisateurs ➔ besoin d'une durée de vie de la batterie plus longue.
Un dispositif multi-services qui assure les communications, l'informatique et le divertissement …	Plus de transistors dans une seule puce ➔ plus de consommation d'énergie.

Comme il est décrit dans le tableau ci-dessus, les grands impacts des nouvelles tendances sur la consommation d'énergie représente le défi le plus critique rencontré de nos jours : avoir une petite taille et intégrer des fonctionnalités de plus en plus complexes, mais aussi avoir une batterie de longue durée. Pour relever ce défi, plusieurs techniques ont été développées dans le but d'optimiser la consommation d'énergie dans un système sur puce.

II.2 Les techniques de gestion d'énergie:

L'autonomie de la batterie est une performance clé et un critère de qualité pour n'importe quel appareil mobile à grande consommation publique. Le nombre croissant de processeurs avec leurs fréquences élevées, l'augmentation de la taille des écrans LCD, de la précision des caméras et la multitude de radios et de capteurs sont des facteurs conduisant à une consommation totale de l'énergie pouvant être au-delà de ce qui est acceptable par le consommateur. Réduire leur consommation énergétique est par conséquent devenu un enjeu important, notamment pour augmenter leur autonomie.

Seulement la bonne gestion de l'énergie au sein de ces dispositifs nous permet de réduire la consommation totale d'énergie (technique de power management).Cette consommation d'énergie est divisée en **énergie statique** et **énergie dynamique**.

Plusieurs stratégies sont adaptées et exploitées qui permettent considérablement d'optimiser la consommation de l'énergie tout en jouant sur la réduction de la fréquence de fonctionnement (puissance dynamique) et de la tension d'alimentation (puissances statique et dynamique) afin de concevoir des systèmes de faible puissance.

Puissance dynamique et puissance statique:

La consommation totale d'énergie pour une conception de SoC est la somme de la puissance dynamique et la puissance statique.

$$P_{total} = P_{stat} + P_{dyn}$$

- **Consommation dynamique:**

La fréquence et la tension d'alimentation des processeurs sont liées.

La consommation dynamique du processeur dépend à la fois de la fréquence et de la tension d'alimentation utilisées.

Elle est dissipée lors de la commutation des composants et dépend donc de la fréquence f du processeur.

Elle peut être approximée selon la relation :

$$P_{dyn} = C * f * V_{dd}^2$$

Où C correspond à la capacité de sortie du circuit ,V_{dd} à la tension d'alimentation et f à la fréquence.

$C= P_{dyn, ref} / (V_{dyn,ref}^{2 *} F_{dyn,ref})$ tels que :

-$P_{dyn, ref}$est une constante qui représente la valeur de puissance dynamique de référence.

- $V_{dyn,ref}$est une constantequireprésente la valeur de tension de référence.

- $F_{dyn,ref}$est une constante qui représentela valeur de fréquence de référence.

Les solutions permettant de réduire la consommation dynamique dans les circuits s'attachent par conséquent à réduire la fréquence de fonctionnement et de manière conjointe la tension d'alimentation. Afin d'atteindre une faible puissance dynamique, plusieurs approches sont appliquées. Elles comprennent l'approche de **« clock gating »**[9].

- **Consommation statique:**

La consommation statique des processeurs est en grande partie due aux courants de fuite.

Dans un circuit idéal, cette consommation statique est nulle mais, en réalité, un courant de fuite existe et est responsable d'une consommation énergétique non négligeable. Ce courant de fuite provient du fait que les transistors composant le circuit ne sont pas parfaits.Elle peut être approximée selon la relation :

$$P_{stat} = (P_{stat,ref} / V_{stat,ref}) * V$$

Tels que :

-V correspond à la tension d'alimentation.

- $P_{stat,ref}$ est une constante qui représente la valeur de puissance statique de référence.

- $V_{stat,ref}$est une constantequireprésente la valeur de tension de référence.

Plusieurs solutions matérielles existent pour réduire la consommation statique. Parmi ces solutions, l'approche de **« Power Gating »**[9].

Ainsi et au niveau logiciel, les principaux travaux de recherche sur la minimisation de la consommation énergétique ont visé à exploiter la possibilité matérielle de changer ces paramètres à travers la proposition des algorithmes d'ordonnancement de la fréquence et de la tension au sein d'un circuit.Parmi ces techniques on cite les suivants :

II.2.1. *Clock gating*

Le « clock gating » est utilisé aujourd'hui par la plupart des conceptions SoC (System On Chip) vu qu'elle est considérée comme une technique efficace pour économiser de l'énergie dynamique.C'est une méthode qui active et désactive les signaux d'horloge en fonction de l'activité du bloc, elle consiste donc à inhiber l'horloge sur les blocs qui sont inactifs.

Par conséquent, en appliquant le clock gating sur un bloc nous supprimons la puissance dynamique consommée.

II.2.2. *Power gating*

Comme la puissance statique a une influence sur la consommation totale du SoC (System On Chip), l'utilisation des techniques de réduction de la consommation de puissance statique est fortement nécessaire.La stratégie de base du « power gating » est de fournir au moins deux modes de puissance: un mode inactif et un mode actif. L'objectif donc est de passer entre ces deux modes dans le temps approprié et de la façon appropriée afin de maximiser les économies d'énergie tout en minimisant l'impact sur les performances.

Cette technique consiste à couper l'alimentation en tension temporairement (tension nulle) à des blocs sélectifs qui ne sont pas actifs pendant un certain temps : passer ces blocs au mode inactif. Par conséquent, en appliquant le « power gating » sur un bloc nous supprimons la puissance statique consommée.Lorsque ces blocs nécessitent d'être réactivés, ils seront remis au mode actif avec une pénalité en temps pour effectuer cette transition.

II.2.3.*DVFS (Dynamic Voltage Frequency Scaling)*

C'est un système de gestion d'énergie « power management » qui permet d'optimiser conjointement les performances et la consommation d'énergie pour les SoC [4].L'idée principale est de réduire la tension d'alimentation et la fréquence de fonctionnement pour un composant lorsqu'il n'est pas nécessaire de le faire fonctionner à sa puissance de traitement maximum.

Il s'agit alors de piloter à la fois la fréquence et la tension des composants pour optimiser leur fonctionnement.Cela conduit à des importantes économies dans la consommation d'énergie, statique et dynamique à la fois.

En raison de la dépendance quadratique de la tension avec la puissance dynamique la diminution de la tension d'alimentation est un moyen qui a un fort effet sur la réduction de la puissance dynamique.

Nous pouvons définir ainsi des modes de fonctionnement des composants tels que hautes performances ou faible consommation. En remarquant que pour réaliser le passage d'un mode à un autre il y a un ordre à suivre.

En effet, quand nous augmentons les performances il faut d'abord augmenter la tension puis la fréquence et quand nous diminuons les performances il faut suivre le procédé inverse.

III. Méthodes et outils utilisés:

III.1) Niveau de modélisation utilisé:

L'utilisation de plusieurs niveaux de modélisation offre l'avantage de disposer d'un outil de simulation et d'estimation des performances dès les premières phases de conception. Plusieurs travaux de recherches proposent une classification des niveaux d'abstraction, en allant du niveau le plus haut vers le niveau le plus bas.

Ainsi, dans notre travail, nous avons adopté la classification présentée dans la figure 1 qui résume les différents niveaux d'abstraction pour la description d'un SoC[11], [8].

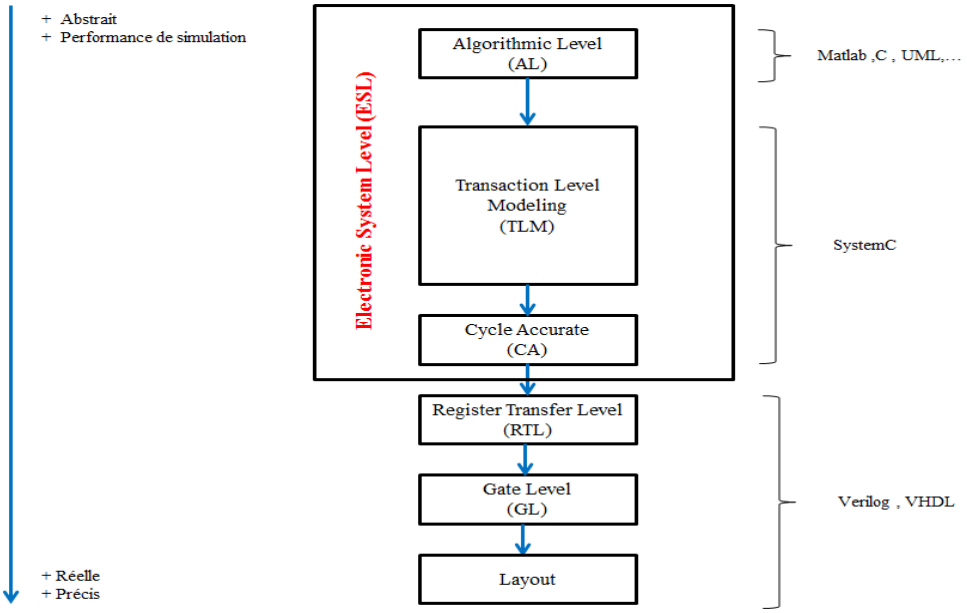


Figure1. Les différents niveaux d'abstraction dans le flot de conception d'un SoC

Niveau AL (Algorithmic Level):

A ce niveau de la description, nous disposons seulement de la description de l'application sous forme algorithmique. Les modèles à ce niveau sont décrits en utilisant de langages de description haut niveau tels que Matlab, C ou C++. Mais la description de l'architecture matérielle du système n'est pas définie. Ces modèles sont ensuite analysés et vérifiés du côté fonctionnel afin de diviser efficacement l'application en tâches matérielles et logicielles (le partitionnement).En outre, les techniques d'ingénierie à base de modèles (les machines de Moore/Mealy) et les langages (Unified Modeling Language « UML ») sont

largement utilisés à ce niveau afin de préciser et de valider une application.Ces techniques et langages permettent à la fois de modéliser le logiciel et le matériel après la phase de partitionnement. La fonctionnalité complète du système est ensuite validée en effectuant une co-simulation des deux parties ensemble.

Niveau transactionnel: TLM (Transaction Level Modeling):

TLM est une approche de modélisation haute niveau, fondée sur des langages de programmation haut niveau comme SystemC afin de décrire un prototype virtuel (VP) de la partie conception du matériel.Ce niveau définit de nouveaux concepts permettant de supporter la description du matériel et du logiciel au niveau transactionnel.

Principalement, nous identifions les fonctions de transport décrites dans les modules de communication qui servent à convoyer les transactions entre les initiateurs et les cibles et vice versa. Ces transactions se présentent sous formes de requêtes de commande ou bien de requêtes de réponse. A titre d'exemple, pour initier une opération de lecture ou d'écriture, des appels fonctions de type read () ou write () sont utilisés. Ces fonctions sont transmises à travers les ports des composants connectés à des canaux de communication (appelés aussi Channels).

Niveau CAL (Cycle Accurate Level):

Au niveau CAL, le modèle est décrit précisément du point de vue temps d'exécution. Il permet donc d'ajouter la notion de l'horloge, à vrai dire, il décrit ce qui se passe à chaque top d'horloge.

Niveau RTL (Register Transfer Level):

La modélisation au niveau RTL correspond à la description de l'implémentation physique du système sous forme de blocs élémentaires (bascules, multiplexeurs, UALs, registres, etc.) et à utiliser la logique combinatoire pour relier les entrées/sorties de ces blocs. Les langues utilisées ici sont le VHDL et Verilog.

Niveau GL (Gate level):

Ce niveau fait une abstraction de nombreux détails en se concentrant uniquement sur la description des portes logiques (AND, OR,..) et leurs connexions.

Niveau Layout:

C'est la description la plus précise d'une puce dans laquelle chaque emplacement d'un transistor est précisément connu et soigneusement vérifié.

Les techniques de gestion d'énergie présentées dans la section (II.2 Les techniques de gestion d'énergie) comme « **power gating** », « **clock gating** » et « **DVFS** » peuvent être modélisées à ces différents niveaux d'abstraction : niveau physique (Layout), niveau portes logiques, niveau RTL (Register Transfer Level) et niveau ESL (Electronic System Level) regroupant les niveaux AL et TLM.

Cependant, évaluer en simulation l'efficacité d'une stratégie de gestion d'énergie basée sur ces techniques, appliquée à un système complet, peut prendre un temps de simulation considérable si le niveau de modélisation utilisé est le niveau RTL ou les niveaux plus détaillés portes logiques et physique.

Ainsi le niveau ESL est le plus adapté pour effectuer cette évaluation même si les résultats obtenus sont a priori les moins précis à ce niveau de modélisation. Par ailleurs, il est montré que le niveau système est celui où il y a le plus d'opportunités pour obtenir des gains en énergie élevés.

La figure-2 montre la tendance vers l'utilisation du niveau ESL dans l'optimisation d'énergie.

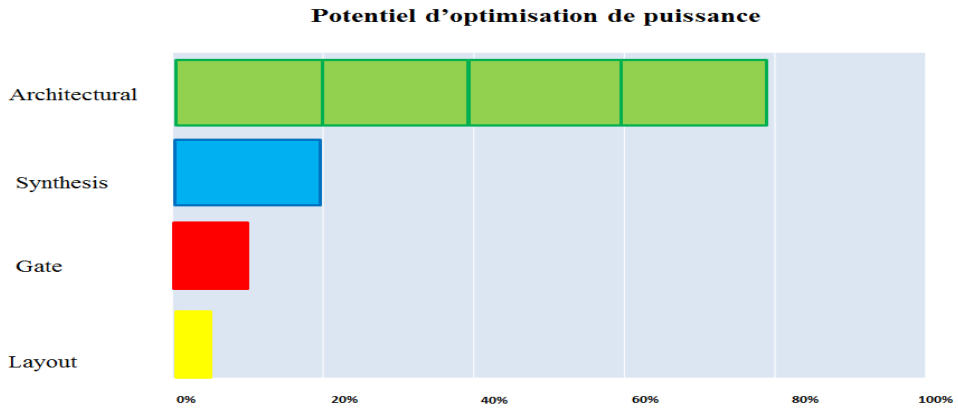


Figure2. Les Opportunités d'optimisation de la consommation d'énergie à chaque niveau d'abstraction (source : LSI Logic[3]).

L'avantage du niveau ESL réside dans la conception de modèles transactionnels lesquels sont par la suite partitionnés et implémentés grâce à de nombreux mécanismes d'abstraction et de réutilisation, il permet ainsi de combiner dans une même simulation le

software et le hardware et donc d'évaluer les gains en énergie lorsque le système complet est pris en compte sur un ou des cas d'utilisation « use cases ». Par conséquent, nous pouvons mesurer la puissance des différentes configurations pour voir ce qui se traduit dans le meilleur compromis entre les exigences de puissance et de performance.Ici, nous voyons clairement qu'il nous permet d'accélérer le processus de conception, de simulation et de vérification et donc de développement du système. Dans la suite seul le niveau **ESL** est considéré en se basant sur des outils de prototypage virtuel.

III.2 Outil utilisé:

L'un des partenaires industriels du projet **HOPE**, **Synopsys**, a fourni un outil de **prototypage virtuel** « **Platform Architect** ».

Pourquoi le prototypage virtuel ?

Le prototypage virtuel permet, avant la réalisation matérielle, de montrer une réalité virtuelle tendant à représenter l'objet à réaliser le plus fidèlement possible c'est-à-dire construire l'architecture virtuellement pour avoir une estimation de l'application et faire des simulations et des tests virtuels.Donc il rend plus facile et plus rapide les modifications éventuelles sans avoir recours à de gros frais de matériels.

Par conséquent, il permet un gain de temps non négligeable quant aux tests de fonctionnalités et d'intégration environnementale.

Platform Architect MCO (Multicore Optimization):

C'est un environnement graphique qui fournit aux concepteurs de systèmes, des outils de simulation d'analyse de performance et des méthodes SystemC TLM afin de garantir une conception efficace et une optimisation des systèmes multi-cœurs et des architectures de SoC.

Ainsi, avec cette plateforme, les concepteurs des systèmes explorent et optimisent le partitionnement matériel-logiciel et la configuration de l'infrastructure des systèmes sur puce dans le but d'atteindre les meilleurs performances du système.

Cet outil considère une architecture modélisée au niveau transactionnel (modèle abstrait en SystemC-TLM) dans le but d'obtenir des simulations rapides et de permettre le développement et la validation du logiciel applicatif qui devra être exécuté par le système réel. Ce modèle d'architecture au niveau transactionnel est appelé plateforme virtuelle, il permet de valider les aspects fonctionnels du système.

Un ensemble de vues et de traces est fourni par cet outil qui permet d'analyser le comportement des composants de l'architecture exécutant les tâches de l'application étudiée et suivre leur consommation de « Power » [12].

Ils sont disponibles à partir des instances des VPUs (Virtual Processing Unit) dans la plateforme. (Un VPU est un modèle abstrait d'un composant de l'architecture matérielle. Il peut s'agir d'un CPU ou d'un périphérique ou d'un composant matériel spécifique).

Un VPU dispose de différents paramètres génériques qui permettent de le particulariser dans cette perspective.

Les analyses MCO enregistrent des informations sur l'état des tâches dans le système et les ressources qu'ils utilisent au fil du temps (« running », « waiting », « ready », « created »..) de telle sorte que nous pouvons étudier et optimiser l'exécution des tâches sur les ressources disponibles.

Vues d'analyse de « Power »

Le PAM « Power Analysis Model », suite à son affectation au VPU associé, permet de suivre les états d'activités du VPU au fil du temps et de calculer sa consommation de puissance.

Ainsi, pour chaque PAM (Power Analysis Model), nous pouvons obtenir les résultats suivants :

Tableau 2. Les vues d'analyses fournies par le PAM:

Les vues d'analyses	**Description**
Power State	Décrit les états d'activités du VPU associé (ACTIVE, IDLE,..).
Total Power	La consommation de puissance totale calculée par le PAM du VPU associé.
Dynamic Power	La consommation de puissance dynamique calculée par le PAM du VPU associé.
Leakage Power	La consommation de puissance statique calculée par le PAM du VPU associé.
Voltage	La valeur de tension associée au VPU
Frequency	La valeur de fréquence associée au VPU

III.3 Langages de programmation utilisés:

III.3.1. *SystemC:*

SystemC est un langage de description matériel (HDL : Hardware Description Langage) permettant une modélisation de haut niveau des systèmes sur puce (SoC), que ce soit au niveau matériel ou logiciel.C'est une plate-forme de modélisation composée de librairies de classes C++ et d'un noyau de simulation qui introduisent des nouveaux concepts

(par rapport au C++ classique) afin de supporter la description du matériel et ses caractéristiques inhérentes comme la concurrence et l'aspect temporel. Ces nouveaux concepts sont implémentés par des classes C++ tels que les modules, les ports, les signaux, les FIFO, les processus (threads et méthodes), etc., permettant ainsi de faire la conception à différents niveaux d'abstraction.

Depuis décembre 2005, SystemC est standardisé auprès de l'IEEE sous le nom d'IEEE 1666-2005.

Et au cours de ces dernières années, SystemC a suscité un intérêt de plus en plus grand auprès des chercheurs et des industriels relativement à d'autres langages vu qu'il possède des caractéristiques intéressantes pour la description de l'architecture à haut niveau.

TLM « Transaction Level Modeling »:

La modélisation au niveau transactionnel (TLM) « Transaction Level Modeling » est une approche de haut niveau (ESL) pour la modélisation des systèmes numériques où les détails de communication entre les modules sont séparés des détails de la mise en œuvre des unités fonctionnelles ou de l'architecture de communication. TLM nous permet donc de travailler à haut niveau, ainsi nous pouvons former (virtuellement) une plateforme et effectuer des simulations et des vérifications [5].

Pourquoi le TLM?

-besoin d'une abstraction plus élevée (plus haut niveau).

-Augmenter la vitesse de simulation.

-Réduire l'effort de modélisation.

-Assurer une exploration efficace de l'architecture [7].

III.3.2. *C++:*

Le **C++** est un langage de programmation permettant la programmation sous de multiples paradigmes comme la programmation procédurale, la programmation orientée objet et la programmation générique. C'est l'un des langages de programmation les plus populaires, avec une grande variété de plateformes matérielles et de systèmes d'exploitation, ce langage est fortement utilisé dans la programmation des systèmes embarqués.

III.3.3.*Langage TCL (Tool Command Language)*

C'est un langage de script qui s'inspire principalement des langages C, Shell… Il est donc multiplateforme, puissant, conçu pour être facilement étendu ou inclus dans une application.

IV. Travail réalisé:

En se basant sur un cas d'utilisation fourni par Intel, partenaire du projet HOPE, constitué par un graphe de tâches qui décrit le séquencement des tâches de l'application, notre système se compose d'une architecture matérielle, d'un ou de systèmes d'exploitation (ordonnanceur), du cas d'utilisation programmé au-dessus du système d'exploitation et d'une stratégie de gestion d'énergie.

La figure 3 définit les démarches suivies dans notre travail.

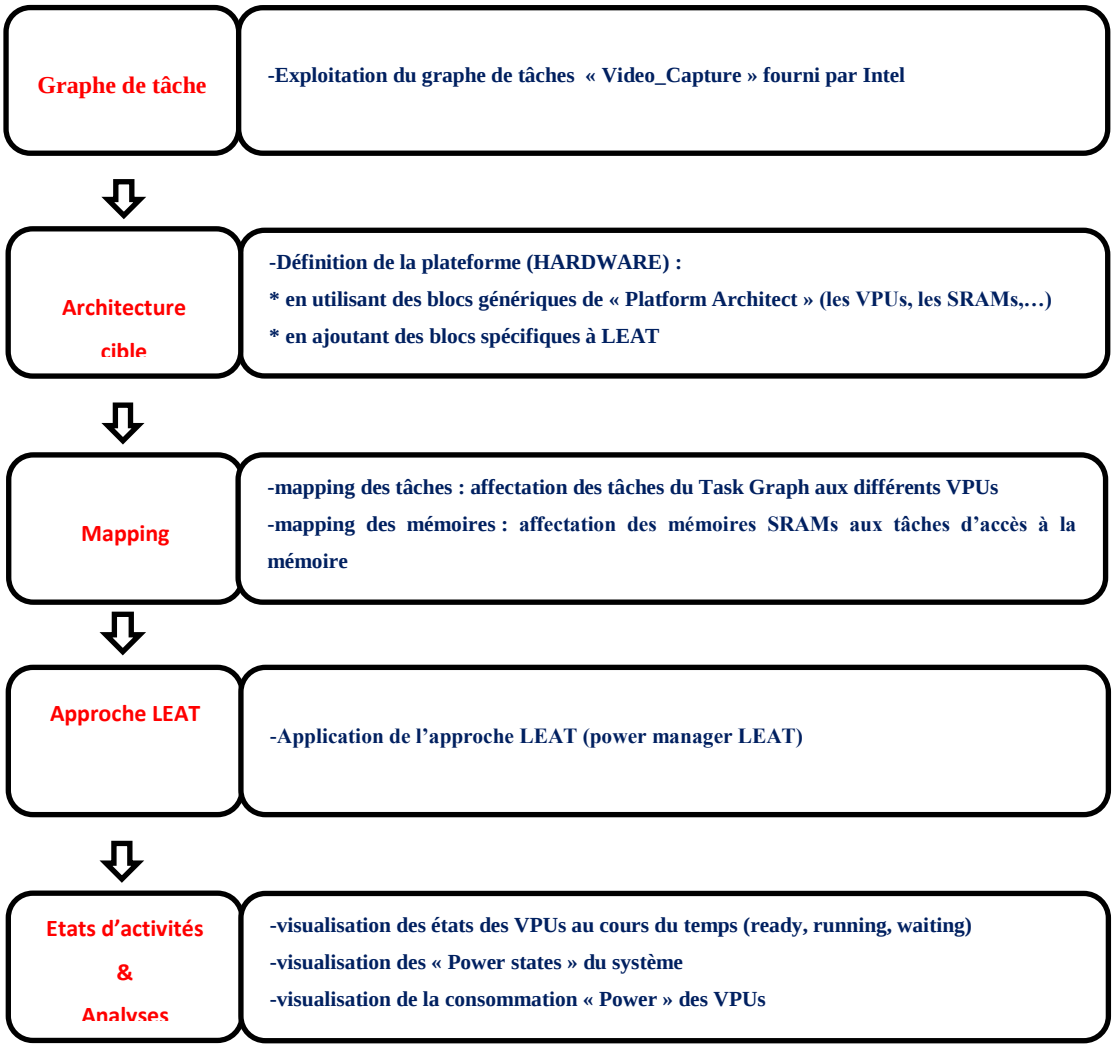


Figure3.*Principales parties du stage*

L'objectif est donc de représenter tous ces éléments et d'évaluer une ou plusieurs stratégies de gestion d'énergie pour ce système, comme cela est indiqué dans la figure 3.

De manière concrète nous considérons un graphe de tâches relatif à un use case décrivant l'application. L'architecture cible doit être déterminée en fonction de ses capacités à exécuter l'application suivant les performances souhaitées et à être gérée de manière efficace en énergie. Dans la suite, le mapping doit être réalisé de façon à optimiser l'ordonnancement des tâches sur les ressources de traitement disponibles dans l'architecture tout en respectant les contraintes de temps et de séquencement fixées par l'application. Nous reprenons dans la suite en détails ces différents points.

IV.1 Exploration du graphe de tâches:

Les applications embarquées comme le « Video_Capture » connaissent de plus en plus de succès auprès du grand public. Ces applications sont caractérisées par une consommation très importante en termes d'énergie qui diminue d'autant l'autonomie attendue par l'utilisateur.

Ainsi, cette consommation du système doit être gérée soigneusement. Cette gestion est particulièrement délicate pour les applications multimédias car le besoin en ressource processeur peut varier énormément d'une tâche à une autre. De plus, les applications multimédias sont des applications temps réels pour lesquelles un retard d'exécution de quelques dixièmes de millisecondes a un impact sur la qualité du traitement.

Pour ce faire, **Intel**, l'un des partenaires industriels de ce projet, nous a fourni un document qui décrit l'application de « **Video_Capture** ». Ce document consiste en un graphe de tâches « Task Graph » spécifique à l'outil **Platform Architect** de **Synopsys**. Il est présenté sous forme de tableaux qui listent l'ensemble des tâches de l'application avec une précision de quelques paramètres fonctionnels. Vu la confidentialité de ce graphe de tâches, aucune description sur le fonctionnement et sur l'architecture de cette application n'a été accordée.

Donc, la première étape, pour mener à bien cette étude, consiste à faire des analyses afin de comprendre le fonctionnement global et le séquencement des tâches de cette application.

Dans le cadre de cette étude, nous sommes censés proposer un mapping optimisé pour ce graphe de tâches; en effet, ce mapping permet de définir l'architecture utilisée c'est-à-dire les composants sur lesquels on exécute les différents tâches de l'application et d'obtenir des simulations sur l'ordonnancement de ces tâches au cours du temps et par conséquent réaliser des estimations de performances de bonnes précisions.

VI.1.1. Définition de l'application « Video Capture »:

Cette application représente une suite d'actions de capture d'images successives et d'audio, un ensemble de traitements, de stockages, et de transmissions, puis une reconstruction d'une séquence d'images fixes représentant des scènes en mouvement.

La figure 4 décrit le flot de tâches de l'application « Video_Capture » déduit à partir du graphe de tâches proposé par **Intel**.

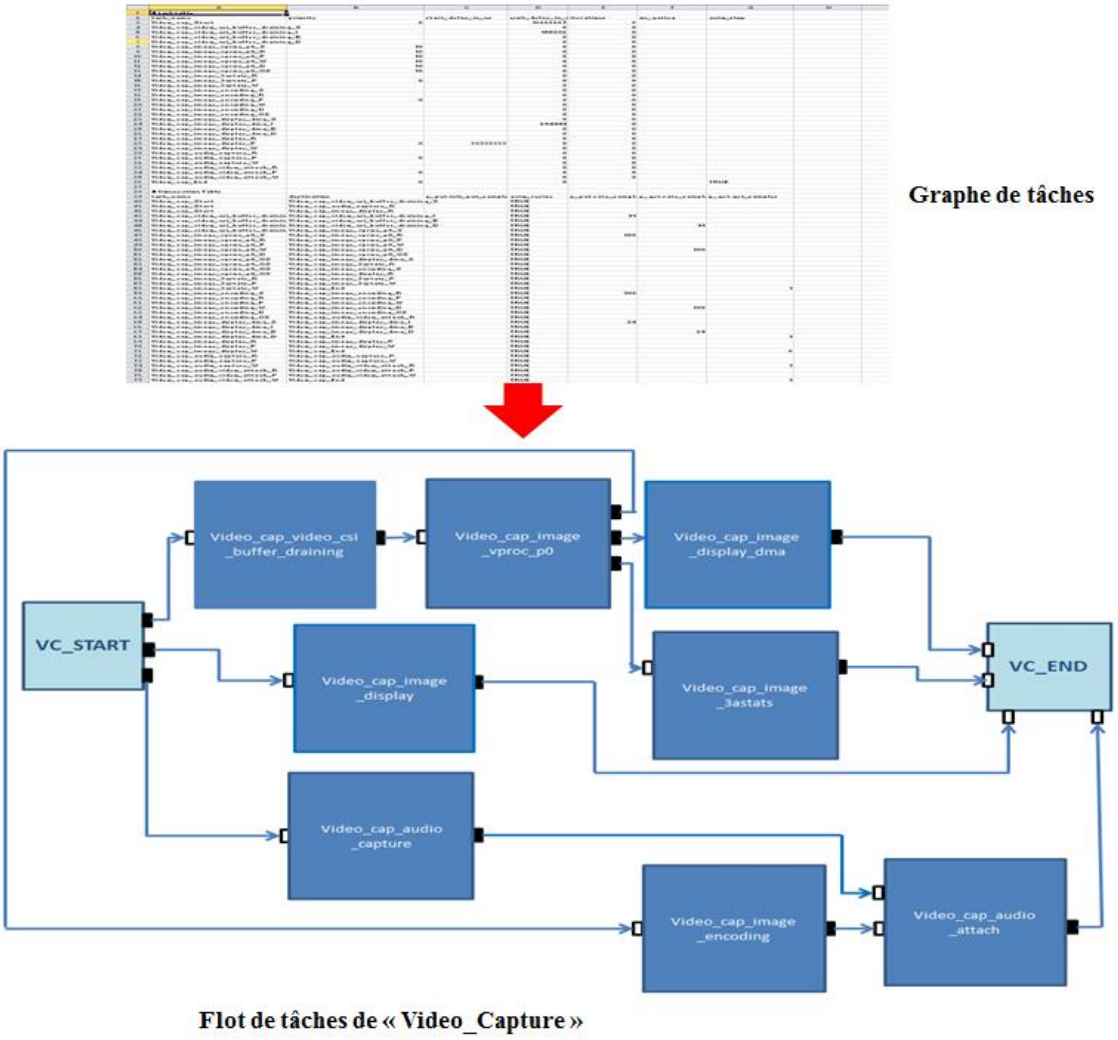


Figure 4. Flot de tâches de « Video_Capture »

Définissons dans un premier temps les éléments et les paramètres utilisés pour décrire un graphe de tâches.

VI.1.2.Définition du graphe de tâches spécifique à Synopsys:

C'est un ensemble de tâches qui communiquent les unes avec les autres par l'intermédiaire de jetons représentant la production et la consommation de données. Il est présenté sous forme d'un fichier CSV (Comma-separated values)constitué d'un ensemble de tableaux qui permettent de définir les propriétés pour chaque tâche en utilisant des paramètres bien spécifiques [12].

Dans un environnement complexe de System-on-Chip (SoC), les graphes de tâches sont utilisés pour abstraire l'application afin d'explorer de manière fiable les performances du système.

VI.1.3.Tableaux de spécification du graphe de tâches:

Tableau de définition des tâches:

Il liste l'ensemble des tâches du graphe de tâches tel que chaque ligne du tableau précise les propriétés d'une tâche de ce graphe de tâches.Ce tableau est identifié par la colonne nommée « **task_name** ». Pour chaque tâche, il définit un ensemble de paramètres :

- Iterations:

Il vaut 0 pour toutes les tâches sauf pour la tâche source (c'est la tâche qui déclenche l'ensemble de l'application). Dans le graphe de tâches « **Video_Capture** », le nombre d'itérations associés à la tâche VC_Start=8, c'est-à-dire que l'ensemble des tâches successives du « **Video_Capture** » va se déclencher 8 fois.

- start_delay_in_ns:

Spécifie le temps de début d'exécution « start_time » en nanosecondes de la 1ère invocation de la tâche concernée par rapport à l'instant initial.

- wait_delay_in_ns:

Spécifie le temps d'attente en nanosecondes avant chaque nouvelle invocation de la tâche concernée.

Tableau de connexion des taches:

Représente les dépendances entre les tâches du graphe de tâches en précisant le nombre de jetons à produire et à consommer entre deux tâches successives. Il permet alors de décrire l'ordre partiel d'exécution des tâches.

Ce tableau est identifié par la colonne nommée « **task_name** ». Pour chaque tâche, il définit un ensemble de paramètres :

- destination: indique la tâche qui suit la tâche « **task_name** » dans l'ordre d'exécution.
- p_get.get_samples : nombre de jetons qui doivent être consommés par la tâche destination pour son activation
- p_put.rate_samples: nombre de jetons produits par la tâche source pour chaque activation.
- p_get.rate_samples: nombre de jetons qui doivent être consommés par la tâche source pour son activation.

Tableau de fonction:

Il existe deux tableaux de fonctions : un associé aux tâches de traitement et l'autre est associé aux tâches d'accès à la mémoire .Ils sont identifiés par la colonne nommée « **task_name** ».

➢ Pour une tâche de traitement, on définit le paramètre suivant :

- Processing cycles: représente le nombre de cycles pendant lesquels la ressource de traitement associée à la tâche est occupée.

➢ Pour une tâche d'accès à la mémoire, il définit les paramètres suivants :

- read: c'est une variable booléenne qui indique s'il y'a un accès en lecture (true) ou s'il y'a un accès en écriture (false).
- memory_name: Indique l'ID symbolique de la mémoire fonctionnelle ciblée
- total_data_size_in_bytes: indique la quantité maximale en octets des activations après laquelle les accès débutent de nouveau à l'adresse locale 0x0.
- data_size_per_activation_in_bytes: Indique la quantité d'octets transférée par activation.
- burst_size_in_bytes : Indique la quantité maximale d'octets transférée à travers les ports.

Tableau de mémoires:

Ce tableau est identifié par la colonne nommée « **memory_name**». Chaque ligne du tableau précise les propriétés d'une telle mémoire telle que :

- addr_width : Indique la taille de la mémoire.

VI.1.3.Description globale des tâches de l'application:

Vu la confidentialité du Task Graph « Video_Capture » de Intel, aucune description sur la fonctionnalité des tâches n'a été accordée. Cependant d'après les noms des fonctions et la description d'applications similaires comme celle décrite dans [10] nous pouvons identifier le comportement suivant :

-video_cap_video_csi_buffer_draining : comme nous exécutons les traitements en mode pipeline il faut vider le buffer dès que l'image est lue depuis la caméra.

-Video_cap_audio_capture : la partie audio de la vidéo est capturée depuis le microphone afin d'être ensuite placée dans le stream.

-Video_cap_image_vproc_p0 : correspond au traitement sur l'image, par exemple la conversion de format RAW vers YUV.

-Video_cap_image_encoding : représente la compression suivant une norme de type H264 par exemple.

-Video_cap_image_display_dma : permet d'afficher en temps réel l'image capturée pour contrôle.

-Video_cap_image_3astats : réalise les traitements statistiques pour l'auto-focus, l'exposition automatique et la balance automatique des blancs.

-Video_cap_audio_attach : insère dans le bit-stream l'audio.

-Video_cap_image_display: permet l'affichage de l'image capturée, ce traitement s'effectue en parallèle avec Video_cap_image_display_dma

Les tâches de la figure 4 sont en réalité constituées de sous-tâches telles que représentées sur les figures (5 et 6) pour quelques blocs du graphe de tâches « Video_Capture ».

Les autres blocs du graphe de tâches sont détaillés dans l'annexe A.

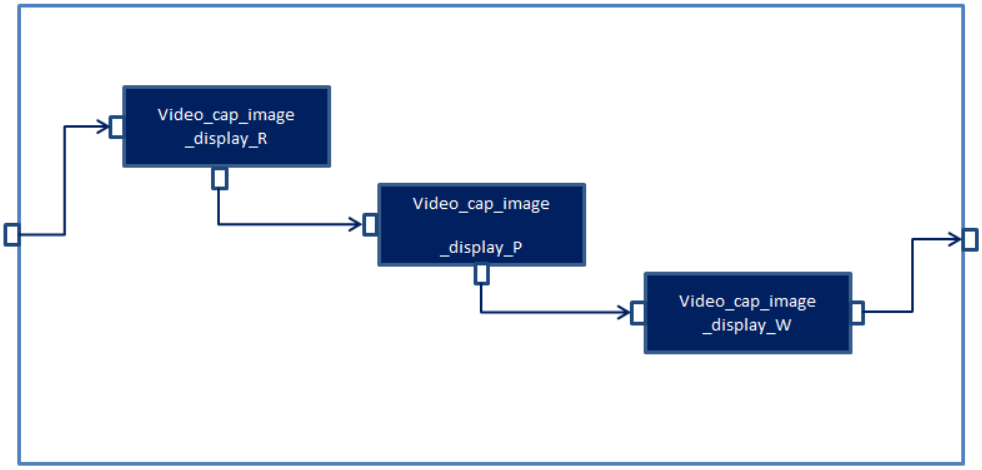


Figure5. Bloc: Video_cap_image_display

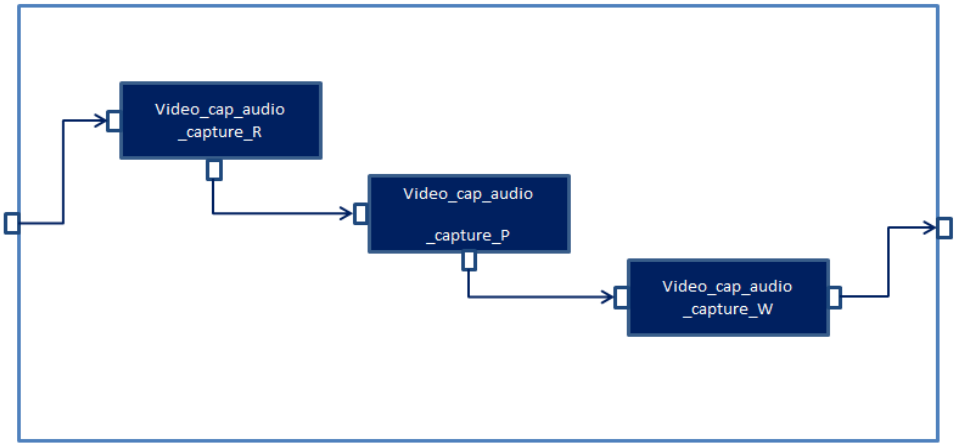


Figure6. Bloc: Video_cap_audio_capture

Dans la suite de ce paragraphe, nous définissons les sous tâches des blocs présentés dans les figures 5 et 6 et le reste des tâches est défini dans l'annexe B.

- Les tâches du bloc « Video_cap_image_display »:

- Video_cap_image_display_R: c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

- Video_cap_image_display_P: c'est une tâche de traitement.

Elle dispose d'un processing_cycles=3333334 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

-Video_cap_image_display_W: c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire.

- Les tâches du bloc « Video_cap_audio_capture »:

- Video_cap_audio_capture_R: c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

- Video_cap_audio_capture_P: c'est une tâche de traitement.

Elle dispose d'un processing_cycles= 200000 définissant son temps d'exécution.

- Video_cap_audio_capture_W: c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire.

IV.2Définition de la plateforme (Architecture fonctionnelle):

Dans cette partie, nous définissons l'architecture sur laquelle les tâches de l'application « **Video_Capture** » sont exécutées.

La figure-7montre une vue globale des composants orientés traitement de l'architecture utilisée au niveau du mapping qui sont décrits dans la suite de ce paragraphe.

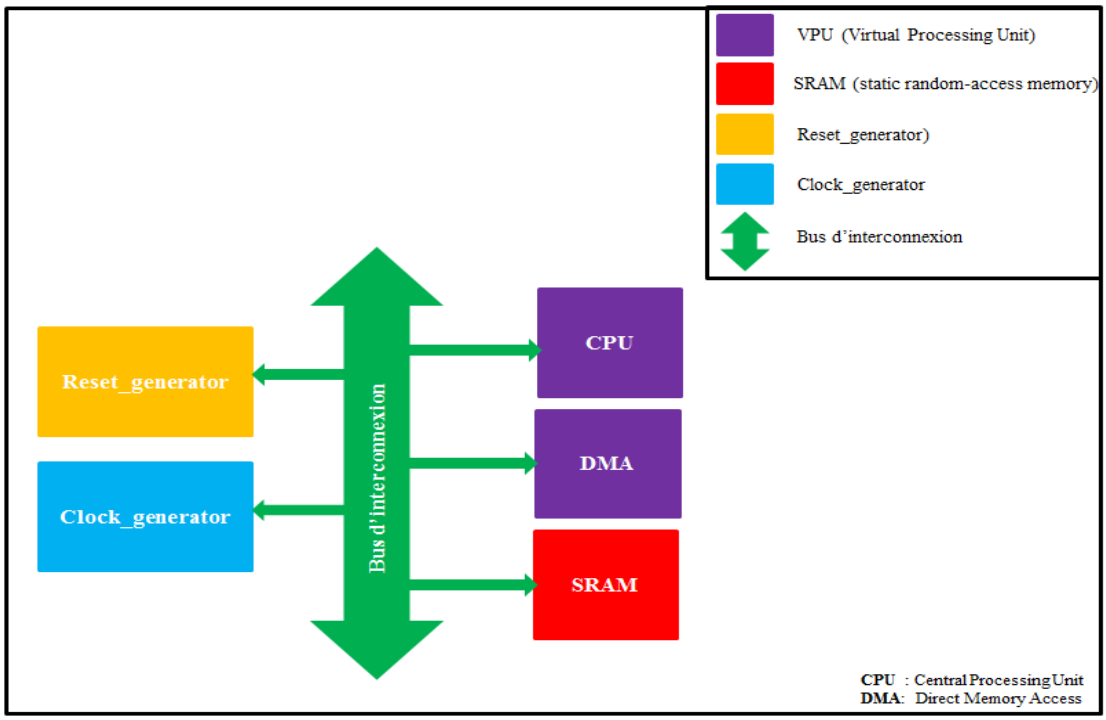


Figure 7. Les composants orientés traitement dans l'architecture cible.

Clock generator (générateur d'horloge):

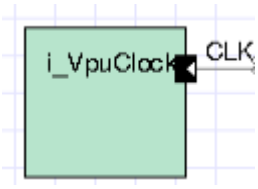


Figure 8: Générateur d'horloge

Le module « Clock Generator » présenté dans la figure 8est un circuit intégré utilisé pour générer un signal d'horloge configurable.Ce composant est instancié de la bibliothèque « **GenericIPlib** » fourni par Platform Architect[12].

Le tableau 3 décrit les ports du module « Clock Generator ».

Tableau 3. Description des ports associés au module « Clock Generator »

Nom du port	**Description**	**Type**	**Type de données**
CLK	C'est un signal d'horloge configurable. Ce port est une exportation de la scml_clock_object.	Out_Master	sc_export<sc_signal_inout_if<bool>>

Reset generator (générateur de réinitialisation):

Le module « Reset Generator » génère une impulsion de remise à zéro configurable.

La figure 9 montre le module « Reset Generator ».

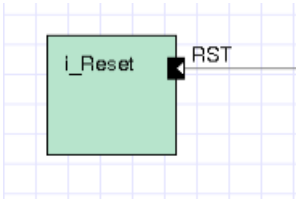


Figure 9.Générateur de réinitialisation

Ce composant est instancié de la bibliothèque « **GenericIPlib** » fourni par Platform Architect[12].Le tableau suivant décrit les ports du module « Reset Generator ».

Tableau 4. Description des ports associés au module « Reset Generator »

Nom du port	**Description**	**Type**	**Type de données**
RST	L'impulsion « Reset » générée	Out - Master	sc_out<bool>

VPU (Virtual Processing Unit) : unité de traitement virtuel:

Un VPU est un modèle abstrait d'un composant de l'architecture matérielle. Il peut s'agir d'un CPU ou d'un périphérique ou d'un composant matériel spécifique. C'est une ressource de traitement pour un certain nombre de tâches tel que ses tâches sont contrôlées par son propre gestionnaire de tâches. Notre choix de travail était d'attribuer un CPU (VPU) pour chacune des tâches de traitement et un VPU de type DMA pour les tâches d'accès à la mémoire (lecture ou écriture). La figure suivante montre le module « VPU».

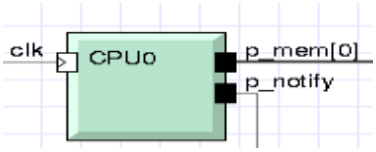


Figure 10.VPU (Virtual processing unit)

Ce composant est instancié de la bibliothèque « **SCML_TM_VPU** » fourni par Platform Architect [12].Le tableau suivant décrit les ports du module « VPU ».

Tableau-5. Description des ports associés au module « VPU »

Nom du port	Description
CLK	C'est un signal d'horloge. Il spécifie la période d'horloge attribuée au gestionnaire de tâches du VPU.
p_mem	C'est un tableau de ports mémoire. Ce nombre de ports est configurable.
p_notify	C'est un port de notification qui renvoi des entiers 0 ou 1 pour indiquer l'état actuel du VPU : -0 pour inactif -1 pour actif

Dans notre travail, nous nous intéressons seulement au port de notification.

SRAM (Static Random-access Memory):

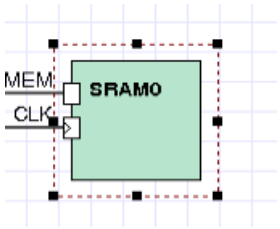


Figure 11.SRAM (Static Random-access Memory).

C'est une mémoire vive statique et rapide. Les données stockées dans cette mémoire sont conservées tant que la tension d'alimentation est présente. Cette mémoire est utilisée dans le mapping des tâches qui effectuent un accès à la mémoire.

Ce composant est instancié de la bibliothèque « **SYSTEM_LIBRARY** » fourni par Platform Architect [12]. Le tableau suivant décrit les ports du module « SRAM ».

Tableau 6. Description des ports associés au module « SRAM »

Nom du port	Description
MEM	Port d'interface mémoire
Clk	C'est un signal d'horloge. Il spécifie la période d'horloge attribuée à la mémoire.

Bus d'interconnexion:

Il assure la liaison entre les différents composants de l'architecture.Ce composant est instancié de la bibliothèque « **SYSTEM_LIBRARY** » fourni par Platform Architect [12].

IV.3 Réalisation du mapping (abstraction de l'application vers l'architecture):

Dans le cadre de la création d'un mapping, nous nous sommes basés sur la notion **d'AAA (Adéquation Algorithme Architecture)** afin de déterminer une architecture adaptée à l'application de « Video_Capture » représentée par un ensemble de tâches qui échangent les données[2],[6],[15].

IV.3.1.Adéquation Algorithme Architecture:

La méthodologie AAA d'Adéquation Algorithme Architecture consiste à étudier en même temps les aspects algorithmiques et architecturaux en prenant en compte leurs interactions, en vue d'effectuer une abstraction optimisée de l'algorithme (l'ensemble de tâches de l'application) vers une architecture adéquate tout en réduisant le nombre de composants de l'architecture (ressources de traitement) , en assurant le parallélisme des tâches et leurs exécutions dans l'ordre spécifié par l'application étudiée et en respectant les contraintes temps réel et d'embarquabilité. La figure suivante donne une vue globale sur la méthodologie AAA.

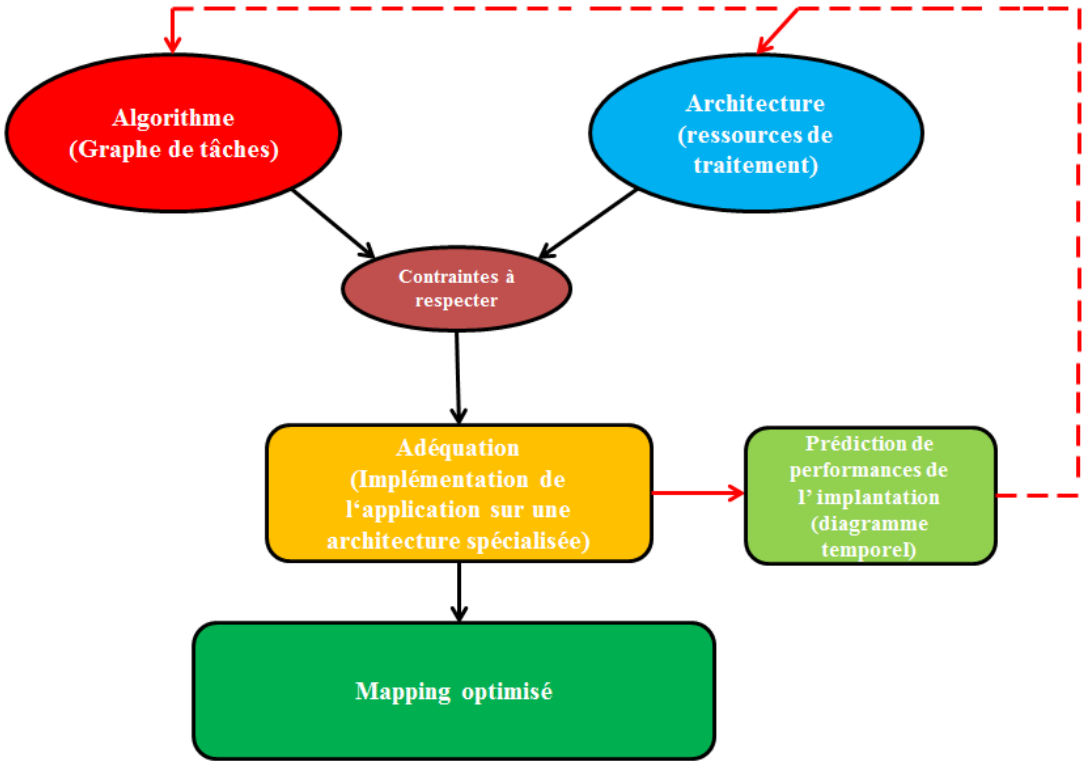


Figure 12.Méthodologie AAA (Adéquation Algorithme Architecture)

IV.3.2.Abstraction de l'application vers l'architecture:

Figure 13. Problème du mapping

Le problème à résoudre dans sa globalité est un problème d'assignement, d'affectation et d'ordonnancement. Comme est décrit dans la figure 13, ceci consiste principalement à assigner et ordonnancer les tâches et les communications de l'application sur les ressources d'architecture cible de telle façon que les objectifs spécifiés sont atteints.

A vrai dire, notre objectif est de réaliser le « **scheduling** » le plus optimisé des tâches de l'application sur une architecture avec un nombre minimal de VPUs tout en assurant une réduction de la consommation d'énergie de ces composants.

Nous sommes donc amené à respecter les contraintes de temps dans le séquencement des tâches , à considérer les tâches qui doivent s'exécuter en parallèle ,vérifier la condition qu'une tâche ne doit être allouée que sur un seul VPU et prendre en considération qu'un VPU ne peut exécuter qu'une seule tâche à la fois. Pour cela, nous avons suivi une approche multi objectifs basée sur la technique d'optimisation dans le but de résoudre notre problème.

En effet, nous avons effectué une étude pour modéliser ce problème d'optimisation suivant une approche par programmation linéaire en nombres entiers. Cependant pour présenter les contraintes d'ordonnancement, de mapping et de la fonction objective qui vise à minimiser l'énergie, nous avons traité un exemple simple constitué d'un nombre réduit de tâches. Cet exemple simple a été entièrement traité mais la taille du problème est devenue rapidement importante malgré ce graphe de tâches réduit.

Aussi avec le graphe de tâches de l'application « Video_Capture » intégrant toutes les sous tâches il n'était pas possible pendant la durée limitée du stage d'aller plus en avant dans cette étude, ainsi nous avons choisi d'opérer par simulation dans la partie optimisation de d'énergie. L'étude et la résolution du problème de programmation linéaire effectué sur l'exemple de graphes de tâches réduit sont présentées dans l'annexe C.

IV.3.3.Description du problème de mapping:

La phase de mapping ou l'association « application/matérielle » constitue la phase la plus importante et la plus critique dans le flot de conception et de développement des SoC (System on chip). Elle consiste à placer les éléments d'une application (les tâches) sur l'architecture (les VPUs) tout en essayant d'atteindre les objectifs et de vérifier les contraintes qui ont un impact direct sur les performances du système.La figure 14décrit ces différents contraintes et objectifs rencontrés dans la réalisation du mapping.

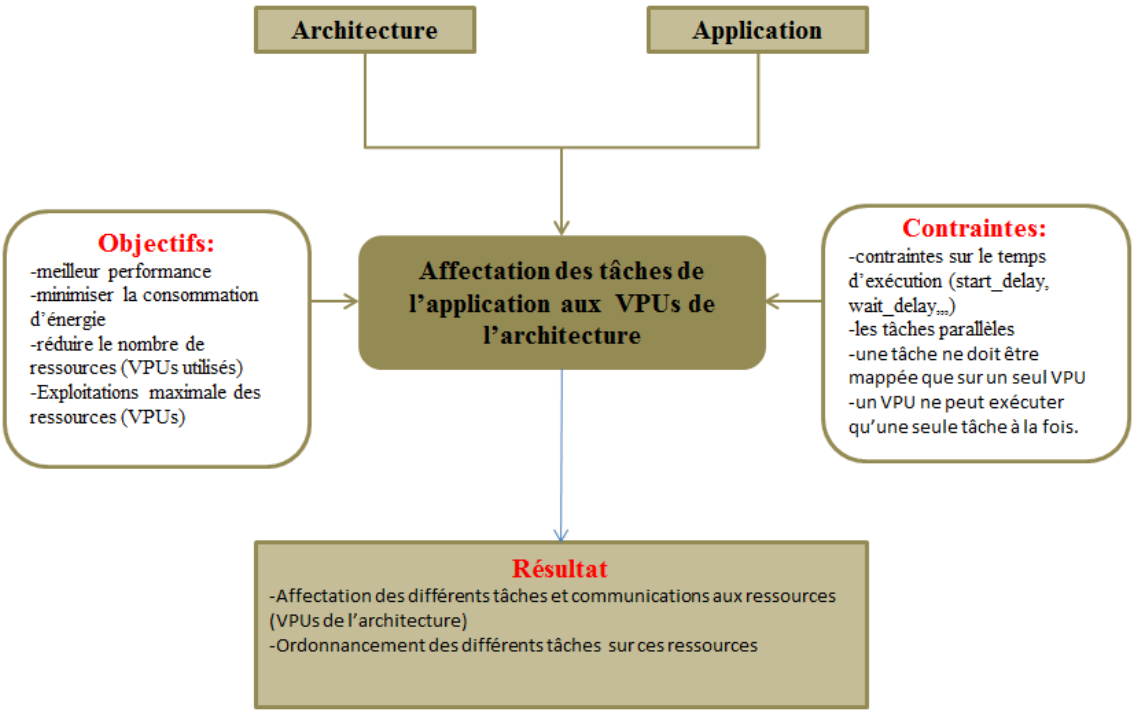


Figure 14. Description détaillée du problème du mapping

C'est dans ce cadre que s'insère notre travail. Il consiste à proposer le placement optimisé d'une application intensive « Video_Capture » sur les différents éléments (VPUs) de notre architecture cible.

La solution obtenue est basée sur des essais consécutifs, des analyses des résultats et des réunions avec l'industriel qui grâce à leurs remarques nous a aidé à apporter des modifications à nos propositions et à améliorer notre travail sur le mapping.

IV.3.4.Réalisation du mapping:

Dans cette partie, nous détaillons la démarche suivie qui nous a amené à trouver le meilleur placement des tâches parmi toutes les tentatives effectuées.

Il existe deux parties dans le mapping :

1) Mapping des mémoires : répartition des mémoires SRAMs entre les tâches de lecture et d'écriture de l'application.

2) Mapping des tâches : affectation des tâches de l'application aux ressources de traitement (VPUs).

Mapping des mémoires:

Le mapping des mémoires SRAMs se base sur les points suivants :

- Plusieurs tâches peuvent lire parallèlement depuis la même mémoire SRAM.

-Les tâches d'écriture qui s'exécutent en parallèle ne peuvent pas être associées à la même SRAM.

-Plusieurs tâches de lecture peuvent être affectées à la même SRAM.

Le tableau suivant montre la répartition des mémoires SRAMs entre les tâches de lecture et d'écriture de l'application « Video_Capture » telle que la colonne « memory_target » indique la mémoire de la plateforme à laquelle la fonction mémoire (« memory_name ») est mappée et les colonnes « vpu_name.driver » spécifient le « driver » à utiliser pour accéder à la mémoire associée pour chaque VPU.

Tableau 7. Mapping des mémoires

Memory_name	Addr_width	Memory_target	CPU0.driver	CPU1.driver	CPU2.driver	DMA0.driver
wr7	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr6	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr5	17	SRAM4/SRAM4	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr4	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd1	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr3	17	SRAM1/SRAM1	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr2	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
wr1	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd6	17	SRAM2/SRAM2	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd7	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd4	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd5	17	SRAM3/SRAM3	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd2	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER
rd3	17	SRAM0/SRAM0	CPU0_DRIVER	CPU1_DRIVER	CPU2_DRIVER	DMA0_DRIVER

La figure 15montre l'emplacement des mémoires SRAMs entre les tâches d'écriture et de lecture du graphe tâches (en raison de visibilité, seulement les tâches d'accès à la mémoire sont présentées dans la figure).

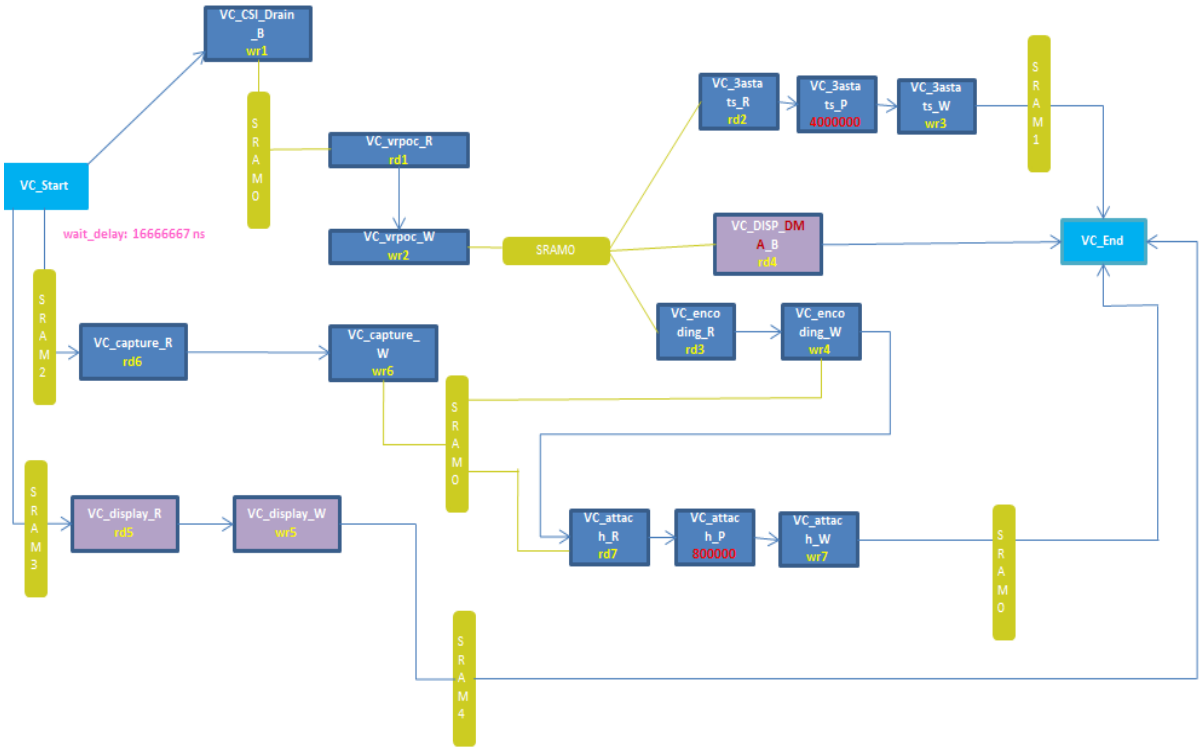


Figure 15. Mapping des mémoires

Mapping des tâches:

L'idée initiale était d'associer à chaque bloc de tâches qui sont dans le même contexte un VPU. La répartition des VPUs entre les tâches de l'application est présentée dans le tableauet la figure qui suivent :

Tableau 8. Mapping initial des tâches de « Video_Capture »

Nom de tâche	**Mapping : VPU_Associé**
Video_cap_Start	CPU0
Video_cap_video_csi_buffer_draining_S	CSI
Video_cap_video_csi_buffer_draining_I	CSI
Video_cap_video_csi_buffer_draining_B	CSI
Video_cap_video_csi_buffer_draining_D	CSI
Video_cap_image_vproc_p0_S	ISPO
Video_cap_image_vproc_p0_R	ISPO
Video_cap_image_vproc_p0_P	ISPO
Video_cap_image_vproc_p0_W	ISPO
Video_cap_image_vproc_p0_D	ISPO
Video_cap_image_vproc_p0_OS	ISPO
Video_cap_image_3astats_R	3astats
Video_cap_image_3astats_P	3astats
Video_cap_image_3astats_W	3astats
Video_cap_image_encoding_S	encoding
Video_cap_image_encoding_R	encoding
Video_cap_image_encoding_P	encoding
Video_cap_image_encoding_W	encoding
Video_cap_image_encoding_D	encoding
Video_cap_image_encoding_OS	encoding
Video_cap_image_display_dma_S	DISPDMA
Video_cap_image_display_dma_I	DISPDMA
Video_cap_image_display_dma_B	DISPDMA
Video_cap_image_display_dma_D	DISPDMA
Video_cap_image_display_R	DISPLAY
Video_cap_image_display_P	DISPLAY
Video_cap_image_display_W	DISPLAY
Video_cap_audio_capture_R	CAPTURE
Video_cap_audio_capture_P	CAPTURE
Video_cap_audio_capture_W	CAPTURE
Video_cap_audio_video_attach_R	MEDIA
Video_cap_audio_video_attach_P	MEDIA
Video_cap_audio_video_attach_W	MEDIA
Video_cap_End	CPU0

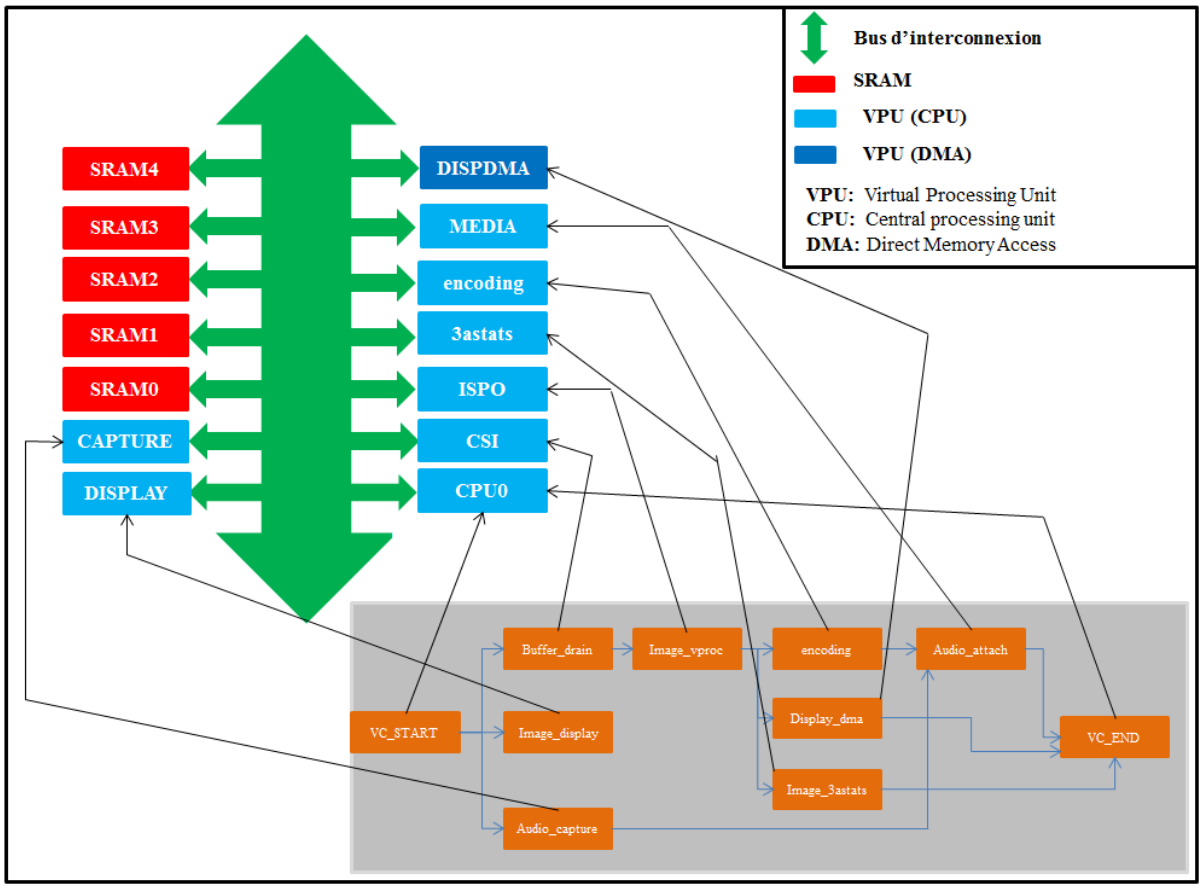


Figure 16. Mapping initial des tâches de « Video_Capture »

En analysant les résultats de la simulation présentés dans la figure 17tels que les couleurs rouge , vert et jaune correspondent respectivement aux états « WAITING » « RUNNING » et « READY » , nous remarquons que les tâches sont mal réparties entre les VPUs, à vrai dire il y a une mauvaise exploitation des VPUs utilisés .En effet, il est bien clair qu'il y a des VPUs qui sont toujours en état « RUNNING » alors qu'il y a d'autres VPUs qui sont la plupart du temps en état « Waiting » . Cela implique que le nombre de VPUs utilisé n'est pas optimal.

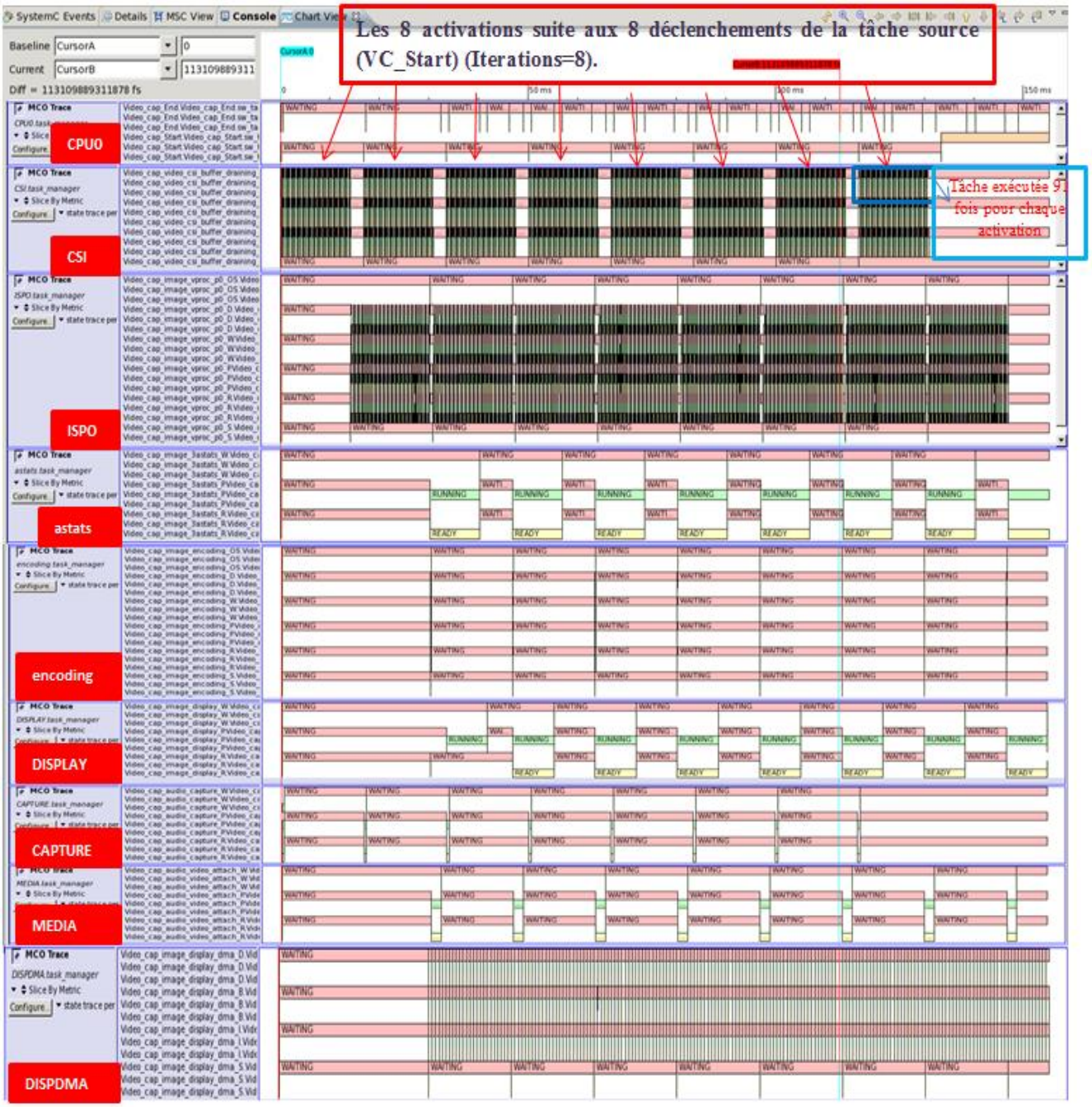


Figure 17. Exécution des VPUs au fil du temps

La figure ci-dessous montre les traces d'analyses associées aux VPUs suite au mapping effectué du graphe de tâche « Video_Capture » à l'architecture dédiée.

-La première trace dans la figure est celle du VPU « CPU0 » qui exécuteles tâches VC_Start et VC_End du graphe de tâche.

Nous pouvons voir clairement l'invocation de la tâche VC_Start huit fois comme indiqué dans le paramètre itérations dans tableau de description des tâches.

Aussi, cette tâche est exécutée en un temps nul. En effet c'est la tâche qui déclenche l'exécution de l'ensemble des tâches de l'application donc son rôle n'est pas un traitement

d'une opération,elle est simplement une tâche « stimulus » de l'ensemble des tâches de l'application.

-La deuxième trace dans la figure est celle du VPU « ISPO » qui exécute les tâches du bloc « Video_cap_image_vproc_pO ».Un zoom sur l'exécution de ce VPU montre que la tâche Video_cap_image_vproc_pO _P s'exécute 100 fois consécutivement à chaque fois qu'elle est invoquée, ce nombre d'exécutions est due au nombre de jetons qu'elle consomme.

-La sixième trace dans la figure est celle du VPU « DISPLAY » qui exécute les tâches du bloc « Video_cap_image_display ».Pour la tâche Video_cap_image_display_P, elle s'exécute huit fois à la suite de chaque invocation de la tâche « VC_Start ».

-Les tâches de l'application sont exécutées dans le bon ordre et au temps approprié.

-La fin de l'exécution de toutes les tâches de l'application n'est pas limitée par le prochain VC_Start.

➔ La solution proposée pour remédier à cette mauvaise répartition était alors d'optimiser le nombre de VPUs de l'architecture et par conséquent de maximiser le temps d'exécution des VPUs.

Suite à des modifications, des analyses des résultats obtenus et des réunions avec l'industriel nous avons considéré le mapping suivant qui est représenté dans le tableau et la figure qui suivent :

Tableau 9. Mapping final des tâches de « Video_Capture »

Nom de tâche	**Mapping : VPU_Associé**
Video_cap_Start	CPU0
Video_cap_video_csi_buffer_draining_S	Pas de mapping
Video_cap_video_csi_buffer_draining_I	Pas de mapping
Video_cap_video_csi_buffer_draining_B	Pas de mapping
Video_cap_video_csi_buffer_draining_D	Pas de mapping
Video_cap_image_vproc_p0_S	Pas de mapping
Video_cap_image_vproc_p0_R	Pas de mapping
Video_cap_image_vproc_p0_P	CPU1
Video_cap_image_vproc_p0_W	Pas de mapping
Video_cap_image_vproc_p0_D	Pas de mapping
Video_cap_image_vproc_p0_OS	CPU1
Video_cap_image_3astats_R	Pas de mapping
Video_cap_image_3astats_P	CPU2
Video_cap_image_3astats_W	DMA0
Video_cap_image_encoding_S	Pas de mapping
Video_cap_image_encoding_R	Pas de mapping
Video_cap_image_encoding_P	CPU1
Video_cap_image_encoding_W	Pas de mapping
Video_cap_image_encoding_D	Pas de mapping
Video_cap_image_encoding_OS	CPU1
Video_cap_image_display_dma_S	Pas de mapping
Video_cap_image_display_dma_I	Pas de mapping
Video_cap_image_display_dma_B	Pas de mapping
Video_cap_image_display_dma_D	Pas de mapping
Video_cap_image_display_R	DMA0
Video_cap_image_display_P	CPU2
Video_cap_image_display_W	DMA0
Video_cap_audio_capture_R	DMA0
Video_cap_audio_capture_P	CPU0
Video_cap_audio_capture_W	DMA0
Video_cap_audio_video_attach_R	Pas de mapping
Video_cap_audio_video_attach_P	CPU0

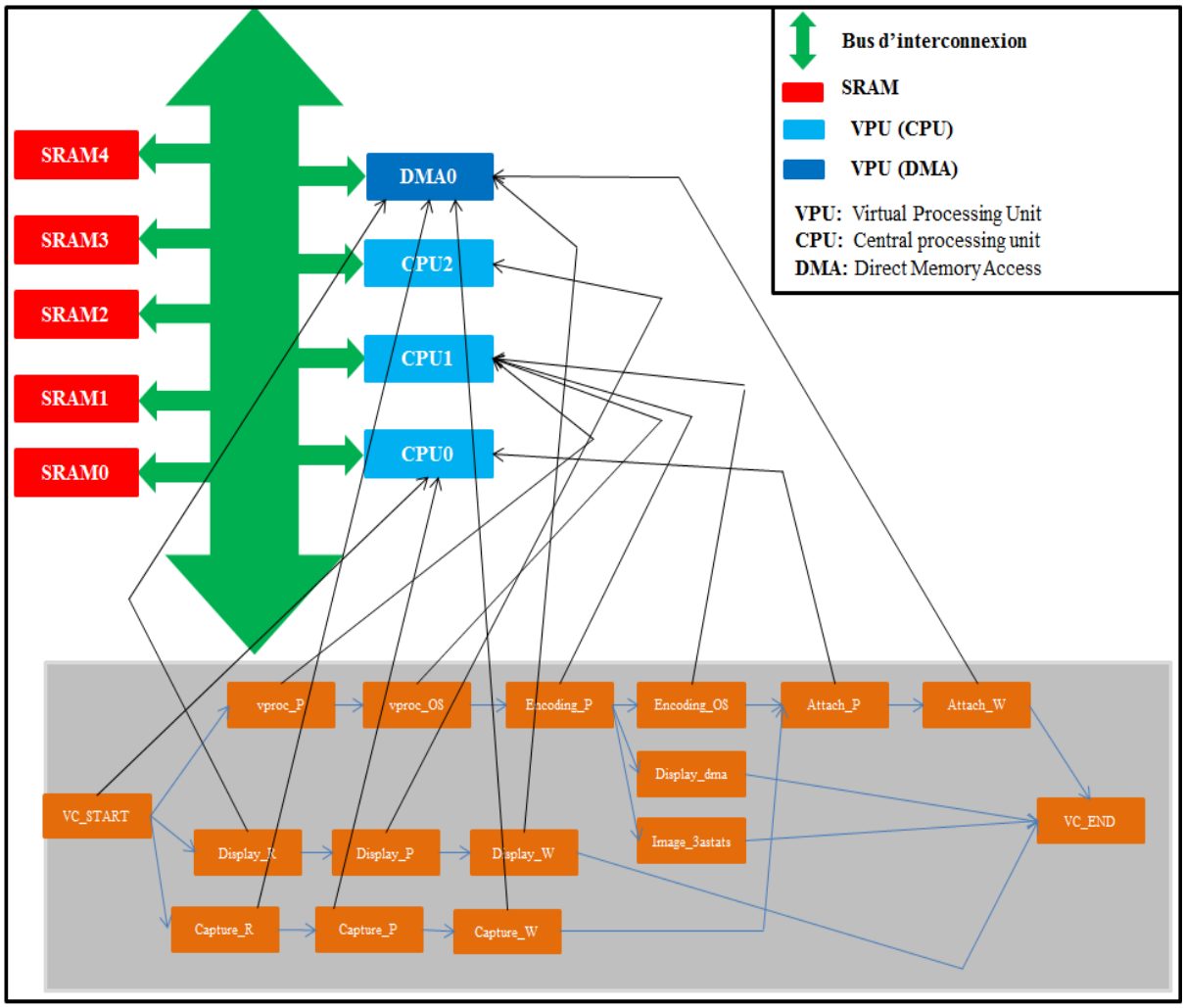


Figure 18. Mapping final des tâches de « Video_Capture »

Ce mapping est basé sur les points suivants:

-Une opération de traitement est associée à un CPU.

-Une opération d'accès à la mémoire est associée à un DMA.

-Une tâche de traitement qui a un processing_cycles nul n'est pas mappée.

-Deux tâches d'accès à la mémoire de type « mem_function » qui font des opérations de lecture et d'écriture sur la même SRAM ne sont pas mappées.

Alors nous avonseffectué le mapping suivant :

-la tâche « Video_cap_audio_capture_R » fait une lecture depuis la mémoire « SRAM2 » donc nous lui associons un DMA0.

- la tâche « Video_cap_image_display_R » fait une lecture depuis la mémoire « SRAM3 » donc nous lui associons un DMA0.

-la tâche « Video_cap_image_3astats_W » fait une écriture sur la mémoire « SRAM1 » donc nous lui associons un DMA0.

-la tâche « Video_cap_audio_video_attach_W » fait une écriture sur la mémoire « SRAM1 » donc nous lui associons un DMA0.

-la tâche «Video_cap_image_display_W » fait une écriture sur la mémoire « SRAM4 » donc nous lui associons un DMA0

-Les tâches d'accès à la mémoire sont toutes affectées à la même DMA0 vu qu'elles ne s'exécutent jamais en parallèle.

-Les tâche « Video_cap_video_csi_buffer_draining_S », « Video_cap_video_csi_buffer_draining_I »,« Video_cap_video_csi_buffer_draining_D », « Video_cap_image_vproc_p0_S »,« Video_cap_image_vproc_p0_D »,« Video_cap_image_ encoding_S »,« Video_cap_image_encoding_D », « Video_cap_image_display_dma_S », « Video_cap_image_display_dma_I », « Video_cap_image_display_dma_D » ont un processing_cycles=0 donc ces tâches ne sont pas mappées.

-Les tâches « Video_cap_image_vproc_p0_P », « Video_cap_image_vproc_p0_OS », « Video_cap_image_3astats_P », « Video_cap_image_encoding_P », « Video_cap_image_en coding_OS », « Video_cap_image_display_P », « Video_cap_audio_capture_P » et « Video_cap_audio_video_attach_P » ont un processing_cycles non nul donc ces tâches sont mappées sur des CPUs tel que leur répartition est effectuée de façon à ne pas associerau même CPU des tâches qui s'exécutent en parallèle.

-Les deux tâches « Video_cap_video_csi_buffer_draining_B » et « Video_cap_image_vproc_p0_R » font respectivement une écriture et une lecture sur la même « SRAM0 » donc ces tâches ne sont pas mappées.

-La tâche « Video_cap_image_vproc_p0_W » fait une écriture sur la mémoire « SRAM0 » et les tâches « Video_cap_image_encoding_R », « Video_cap_image_3astats_R », et « Video_cap_image_display_dma_B » font une lecture depuis la même mémoire « SRAM0 » donc ces tâches ne sont pas mappées.

-La tâche « Video_cap_audio_capture_W » fait une écriture sur la mémoire « SRAM0 » et les tâches « Video_cap_image_encoding_R » et « Video_cap_audio_video_attach_R » font une lecture depuis la même mémoire « SRAM0 » donc ces tâches ne sont pas mappées.

Avec ce mapping, nous avons obtenu le résultat représenté dans la figure 19.

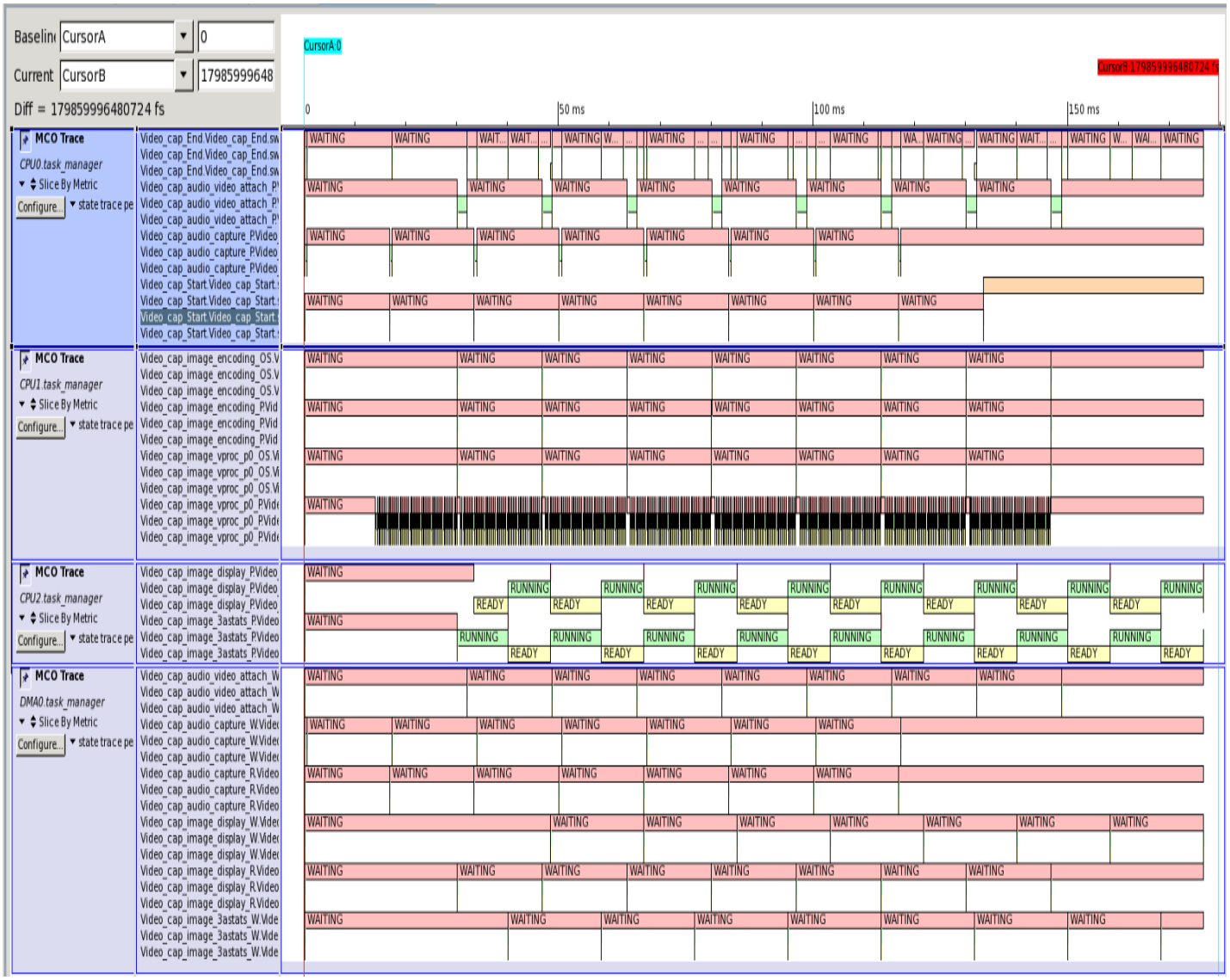


Figure 19. Exécution des VPUs au fil du temps

Cette figure montre bien une amélioration au niveau de la répartition des tâches entre les VPUs de l'architecture cible, en effet :

-le nombre de VPUs est clairementréduit.

-il y a maximisation du temps d'exécution des différents VPUs utilisés dans l'architecture, donc ces VPUs sont bien exploités au cours du temps.

-il y a respect des contraintes de temps (start_delay,wait_delay) des tâches fournies dans la description du graphe de tâche « Video_Capture »

-il y a respect de l'ordre d'ordonnancement des tâches exigé par le tableau de connexion du graphe de tâches.

-il y a respect des dépendances exigées entre les tâches (nombre de jetons consommés ou produits pour chaque activation d'une tâche).

Apres une discussion et une analyse du résultat obtenu, nous avons considéré que ce mapping répond bien aux exigences et aux contraintes fixées au début du travail. Par conséquent, nous avons adapté ce mapping pour la suite des travaux à effectuer dans ce stage.

La figure 20 montre une vue d'ensemble décrivant le séquencement des tâches suivant le tableau de connexionsdu graphe de tâches « Video_Capture », le choix de répartition des mémoires SRAMs entre les tâches d'accès à la mémoire et l'affectation des différents tâches aux VPUs de l'architecture cible.

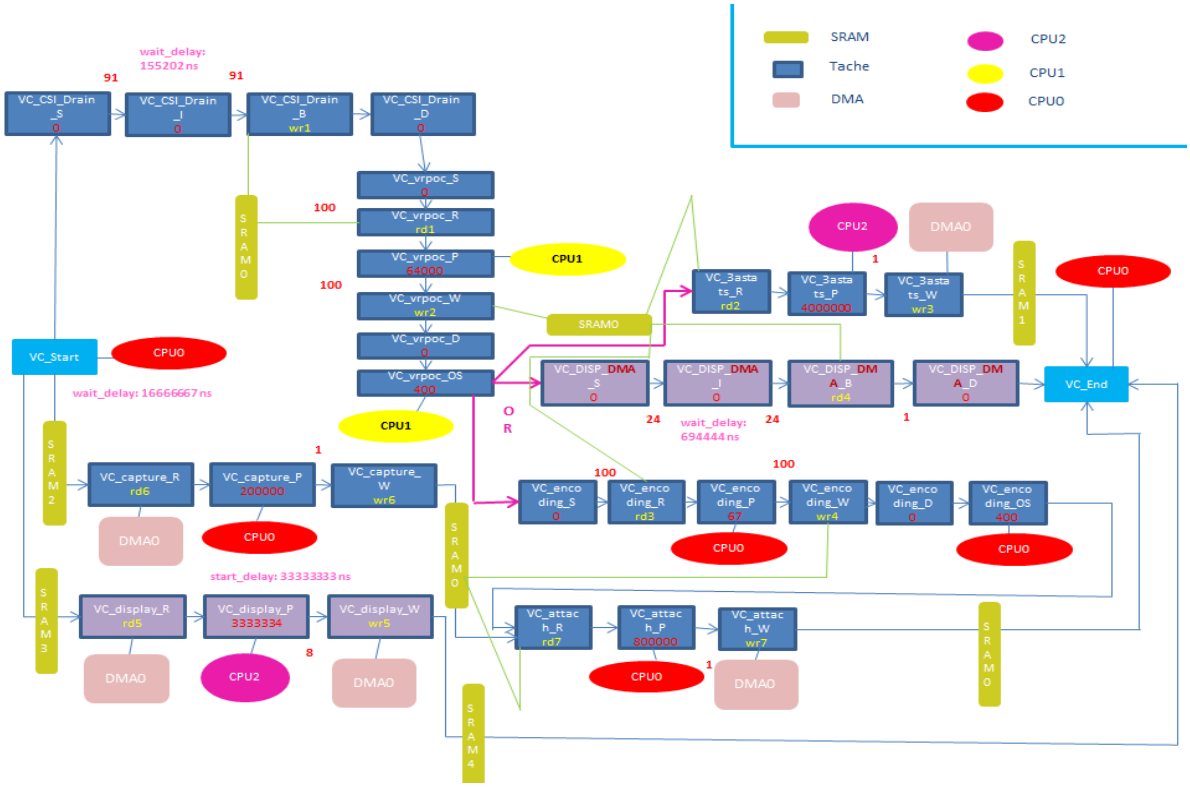


Figure 20. Mapping des tâches et des mémoires

IV.4 Approche de power management:

Une technique simple de gestion de la consommation d'énergie est introduite par Synopsys. Elle consiste à suivre le changement d'état composant par composant,c'est-à-dire dès qu'un compostant change d'état pour passer à l'état inactif « **IDLE** », nous lui appliquons une faible tension et une faible fréquence qui assure une réduction de la puissance statique et de la puissance dynamique respectivement. Cette technique est appelée « **DVFS** » (Dynamic Voltage and Frequency Scaling).Dès que le composant revient à l'état actif « **ACTIVE** », des valeurs appropriées de tension et de fréquence seront appliquées. Cette méthode a bien assuré

une variation sur la puissance totale consommée par chaque VPU de l'architecture mais nous voyons clairement que nous nous trouvons dans une situation où le VPU consomme de l'énergie tout le temps même s'il est dans un état inactif « **IDLE** » de façon prolongée.

Une idée pour remédier à cette lacune est d'inhiber la puissance dynamique et la puissance statique du VPU inactif tout en respectant les contraintes de temps de son fonctionnement et en évitant les pénalités qui peuvent impacter sur ses performances. Notre but est donc d'appliquer non seulement le « **clock gating** » mais d'ajouter aussi le « **power gating** ».

Pour ce faire, la méthode proposée consiste à construire un historique qui liste les transitions successives entre les états des VPUs de l'architecture durant la simulation.

Ce travail va être appliqué sur le graphe de tâche « **Video Capture** » qui est mappé sur une architecture dédiée pour ce mapping, donc une analyse de l'ordonnancement de cette application permettra de construire dynamiquement l'historique des états. En effet, à chaque changement d'état de n'importe quel VPU de l'architecture, nous nous trouvons face à deux cas possibles :

-soit un nouvel état qui sera créé et qui indique l'état actuel de tous les VPUs de l'architecture

-soit un état déjà rencontré qui sera référencé à l'état déjà existant dans la liste des états.

Nous définissons ici la notion d'état fonctionnel présenté sous forme d'un vecteur d'entiers de dimension égale au nombre de VPUs à gérer, donc chaque élément du vecteur prend soit la valeur « **0** » qui indique l'état « **IDLE** » soit la valeur « **1** » qui indique l'état « **ACTIVE** » du VPU associé.

Un suivi détaillé de l'enchainement des tâches exécutées sur les différents VPUs de l'architecture a été effectué afin de vérifier l'obtention d'un historique qui répond bien à l'ordonnancement réel de l'application.

La figure 21 montre le graphe des états obtenu à partir de la construction dynamique de l'historique avec l'architecture issue du mapping précédent.

Dans ce graphe, un état par exemple noté « E1 1001 » indique que pour cet état CPU0 est actif (bit de poids faible du vecteur de bits) ainsi que le DMA0 (bit de poids fort) alors que CPU1 et CPU2 sont inactifs.

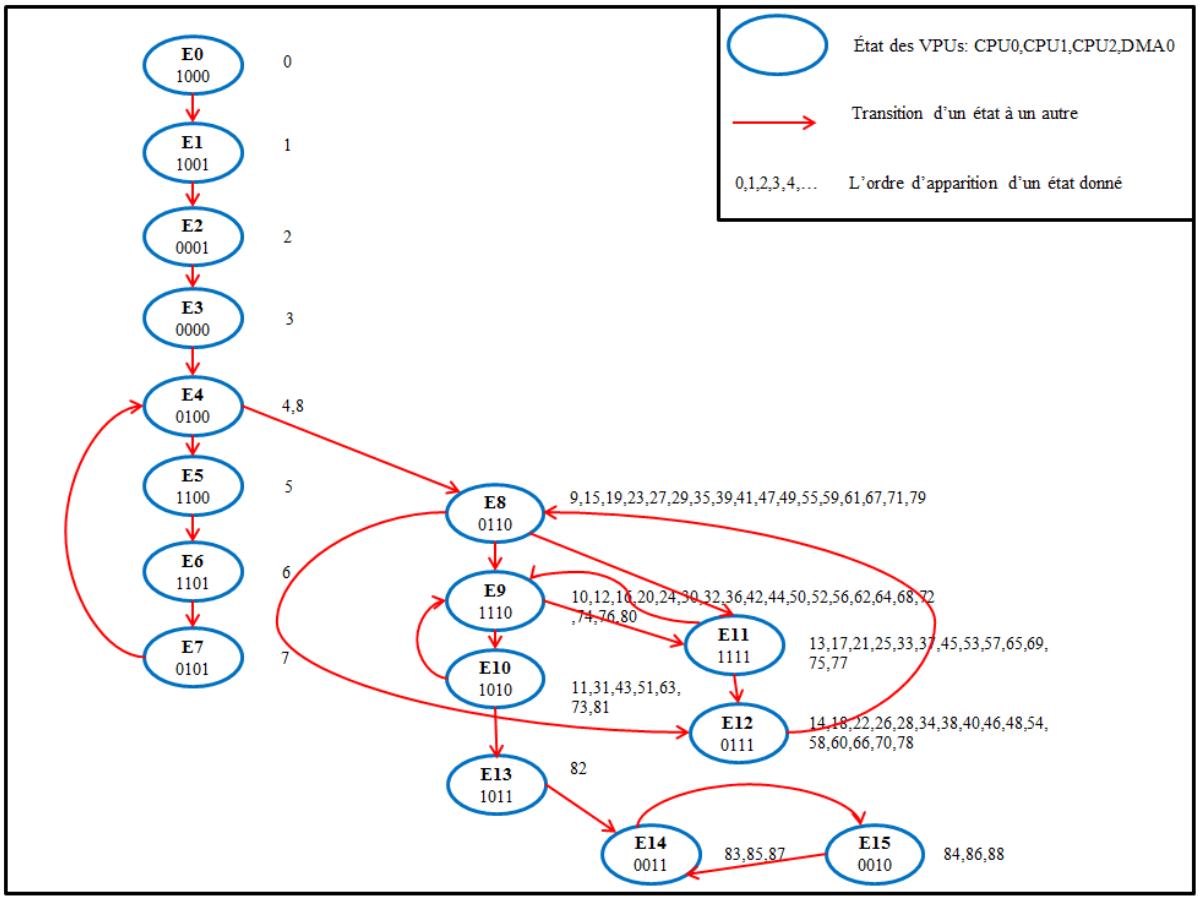


Figure 21. Graphe d'états du « Video_Capture »

Ainsi, à partir de l'historique d'états construit, nous associons à chaque état fonctionnel, la durée pendant laquelle il reste stable. Le but sera ensuite d'appliquer une approche de power management temps réel qui s'appuie sur cet historique et les relations entre ces états pour prédire à partir d'un état actuel la possibilité de la réduction de la consommation des VPUs qui sont inactifs « **IDLE** ».

Pour ce faire, nous avons modifié l'architecture sur laquelle nous avons effectué le mapping en ajoutant des composants orientés power (« clock divider », « Power manager »,..).

Nous détaillons ainsi ces différents composants :

PMU (Power Management Unit) : historique_leat:

Le « **power manager** » (ou gestionnaire d'énergie) est une entité globale qui regroupe toutes les techniques qui permettent de gérer l'énergie du système. Le « **power manager** » comme est décrit dans la figure 22 contient:

-Un port d'entrée de notification qui lui permet d'être averti à propos les changements d'état de chaque bloc VPU.

-Un port de sortie de contrôle qui permet de fournir les valeurs (« **idle_divider** » et « **active_divider** »). Ces valeurs sont :

- ✓ Idle_divider : 0 pour un module inactif «**IDLE** »→ appliquer le « **clock gating** »
- ✓ active_divider: 1 pour un module actif « **ACTIVE** »

-Un port de sortie de tension qui permet de fournir les valeurs de tension (« **idle_voltage** », « **off_voltage** » et « **active_voltage** »). Ces valeurs sont :

- ✓ idle_voltage : 0.8V pour un module inactif « **IDLE** »→ appliquer le « **voltage_scaling** »
- ✓ off_voltage : 0V pour un module inactif « **IDLE** » → appliquer le « **power gating** »
- ✓ Active_voltage: 1.1V pour un module actif « **ACTIVE** »

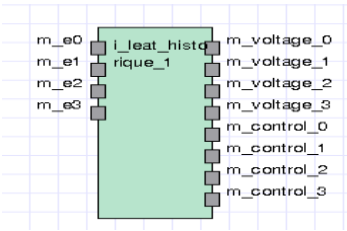


Figure 22. Power manager

Ce module est implémenté afin de remplacer le « **power manager** » classique de Synopsys et appliquer notre approche de power managementque nous sommes en train de définir dans cette partie.Nous avons associé à ce composant des paramètres spécifiques à son fonctionnement qui sont présentés dans le tableau suivant.

Tableau-10. Paramètres spécifiques du « power manager »

Paramètre	**Valeur associée**	**Description**
power_up_delay_in_ns	2000	temps de transition de « off_voltage » vers « active_voltage »
power_down_delay_in_ns	400	temps de transition de « active_voltage » vers « off_voltage »
voltage_scaling_up_delay_in_ns	1000	temps de transition de « idle_voltage » vers « active_voltage »
voltage_scaling_down_delay_in_ns	400	temps de transition de « active _voltage » vers « idle_voltage »
idle_divider	0	Diviseur d'horloge d'un module en état « IDLE »
active_divider	1	Diviseur d'horloge d'un module en état « ACTIVE »
off_voltage	0	Tension (V) utilisé pour un module en état « IDLE »
idle_voltage	0.8	Tension (V) utilisé pour un module en état « IDLE »
active_voltage	1.1	Tension (V) utilisé pour un module en état « ACTIVE »

Clock Divider: Programmable_Clock:

Un diviseur d'horloge fournit une horloge de sortie suite à une division effectuée de l'horloge d'entrée par le facteur de division affecté à son port de contrôle (« **idle_divider** » pour le module en état « **IDLE** » et « **active_divider** » pour le module en état « **ACTIVE** »). Ainsi, l'horloge produit permet de calculer la fréquence de sortie associée au VPU approprié. Suivant le facteur de division associé, le « **clock_divider** » fournit :

-soit un « **gated_clock** » qui est une horloge inactive.

-soit un « **divided_clock** » qui est une horloge divisée par ce facteur de division.

La figure suivante montre le module « **clock_divider**».

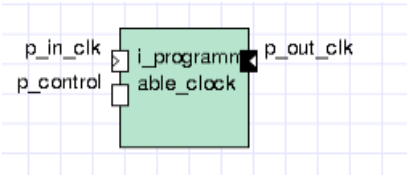


Figure 23. Clock Divider

Ce composant est instancié de la bibliothèque « **MCOPower** » fourni par Platform Architect[12].

PAM (Power Analysis Model):

Afin de suivre les transitions entre les états d'activités « **power states** » et qualifier la consommation de puissance des différents VPUs au fil du temps, une activation du «PAM » est nécessaire pour chaque VPU.

Pour chaque instance du PAM, nous devons définir un PDD « **Power Domain Description** » qui inclut des informations qui servent aux calculs de la puissance statique et dynamique du VPU pour ces différents états d'activités « **power states** ».

Pour ce faire, nous avons ajouté des composants spécialisés pour chaque VPU :

-Les composants PCI et PMIC qui sont reliés au port voltage approprié (sortie du composant leat_historique).

Ce sont des simples composants qui sont ajoutés afin de pouvoir affecter à chaque « **PAM** » (Power Analysis Model) du VPU associé la valeur de tension appropriée pour le calcul de la puissance consommée.

-Un composant state pour lequel on associe l'instance du PAM.

Ce composant est ajouté afin de pouvoir suivre les transitions des états d'activité pour chaque VPU. Nous associons ainsi pour chaque module « state » un « PAM ».

Ainsi, les mesures sont effectuées à l'aide d'une caractérisation des états d'activité « **power states** » (les états « **IDLE** » et « **ACTIVE** »).

Ces changements des « **power states** » sont déclenchés à partir d'évènements**TLM** (Transaction Level Model) : les signaux SystemC. C'est-à-dire, à chaque fois que la valeur du signal est modifiée, une transition de « **power states** » est déclenchée.

Donc, grâce aux signaux SystemC, le « **PAM** » connait dynamiquement l'état d'activité courant du VPU associé. Alors, les deux valeurs qu'un signal peut prendre sont :

- Une valeur « **1** » pour indiquer l'état « **ACTIVE** » du VPU.
- Une valeur « **0** » pour indiquer l'état « **IDLE** » du VPU.

Ensuite, différentes entrées doivent être fournies au « **PAM** » pour le calcul de la consommation d'énergie, ceci se fait suite à une implémentation des scripts TCLspécifiés pour chaque VPU qui permettent l'activation des « **PAM** » sur les composants appropriés , la spécification des entrées de tension et de fréquence et la définition des transitions des états d'activités ,ces scripts sont lancés au début de la simulation. Les entrées définies au « **PAM** » sont les suivantes :

- Ref_freq_dyn_power : la valeur de fréquence (de référence) fournie pour le calcul de la puissance dynamique.
- Ref_voltage_dyn_power: la valeur de tension (de référence) fournie pour le calcul de la puissance dynamique.
- Ref_voltage_leakage_power: la valeur de tension (de référence) fournie pour le calcul de la puissance statique.
- Actual_Voltage : la valeur de tension actuelle fournie par le composant leat_historique qui est associée au VPU approprié.
- Actual_Frequeny : la valeur de fréquence actuelle associée au VPU approprié : cette valeur est calculée à partir de la période d'horloge « clock_period » fournie par le diviseur d'horloge « clock_divider ».

Par conséquent, le « **PAM** » à partir de ces valeurs va calculer la puissance dynamique, la puissance statique et la puissance totale consommées par chaque VPU dans ces différents états d'activités « **power states** » en utilisant des méthodes prédéfinies correspondant aux formules du chapitre II.2.

Lors de la simulation, toutes les données tracées telles que les états d'activités, la consommation de puissance ainsi que les fréquences et les tensions sont rassemblés dans la même base de données qui stocke également d'autres traces comme le séquencement des tâches de l'application qui s'exécutent sur les VPUs de l'architecture. Cela permet une analyse efficace et une corrélation entre la répartition des exécutions des VPUs, les transitions entre les états d'activités « **power states** » et les consommations dynamiques et statiques.

PCI et PMIC:

Ces composantssont déjà présentés dans la description dumodule « **PAM** ».

State:

Ce composant est déjà présenté dans la description du module « **PAM** ».

La figure 22 montre une vue globale de l'architecture construite et une description des différents interactions du « power manager » avec les autres composants de l'architecture.

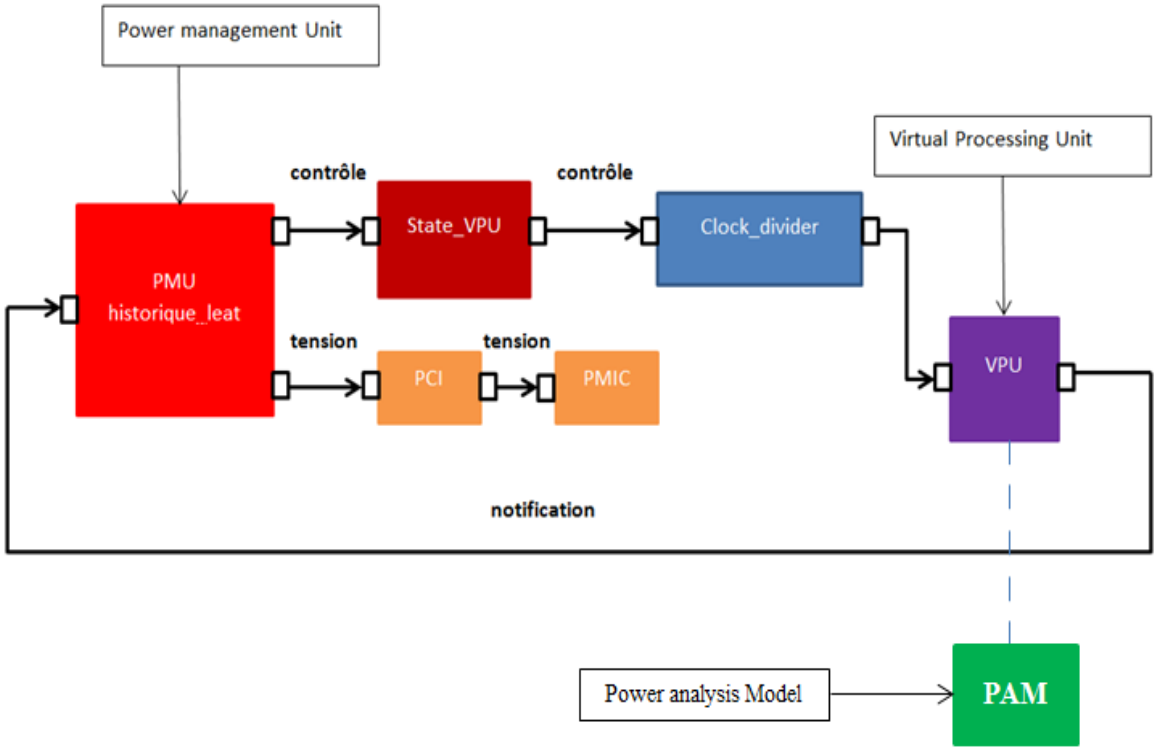


Figure 22. Architecture de « power management »

Les interactions entre le « **power manager** » et les différents composants de l'architecture peuvent être interprétées comme suit :

-Le VPU informe en permanence durant la simulation le « **power manager** » d'une occurrence de changement de son état d'activité:

- Soit il envoie une valeur de notification « **0** » pour informer le « **power manager** » qu'il est en état actif « **ACTIVE** ».
- Soit il envoie une valeur de notification « **1** » pour informer le « **power manager** » qu'il est en état inactif « **IDLE** ».

-Basé sur les entrées de notification reçues par le VPU à travers son port de notification (port d'entré), le « **power manager** », gère et contrôle l'alimentation du VPU en tension et en fréquence à travers ses ports de sortie.

Donc, il fournit à travers son port de contrôle (port de sortie) un facteur de division de la période d'horloge :

- Si le VPU est actif « **ACTIVE** », il associe au facteur de division la valeur de « **active_divider** ».
- Si le VPU est inactif « **IDLE** », il associe au facteur de division la valeur de «**idle_divider** », par conséquent, appliquer le « **clock gating** ».

Ce port de contrôle est relié au bloc « **clock_divider** » qui à son tour fournit l'horloge divisée c'est-à-dire la nouvelle valeur de fréquence au VPU approprié.
Cette valeur de fréquence sera alors utilisée par le « **PAM** » pour estimer la puissance consommée par le VPU.
Aussi, Il fournit à travers son port de tension (port de sortie) la valeur de tension appropriée à l'état actuel du VPU :

- Si le VPU est actif « **ACTIVE** », il affecte à la tension une valeur de « **active_voltage** ».
- Si le VPU est inactif «**IDLE** » :
- soit il affecte à la tension une valeur de « **off_voltage** », par conséquent appliquer le « **power gating** »
- soit il affecte à la tension une valeur de « **idle_voltage** »,par conséquent appliquer le « **voltage_scaling** ».

Cette valeur de tension est utilisée uniquement comme entrée du « **PAM »** pour les calculs de la puissance consommée du VPU.

Notre approche s'appuie sur la supposition qu'un état qui est récemment rencontré a une forte chance de se répéter prochainement.

Donc, en parcourant dynamiquement l'enchainement des états fonctionnels, nous pouvons remarquer qu'un état donné peut avoir un ou plusieurs états successeurs avec des durées de stabilité différentes, ce qui est évident étant donné qu'un VPU peut exécuter différents tâches qui possèdent des temps de traitement différents.

De cette façon, pour un nouvel état, il est possible d'appliquer le « **clock gating** » (puissance dynamique nulle) étant donné que les pénalités d'activation et de désactivation de l'horloge sont négligeables dans ce cas. Par contre, on ne peut rien prédire pour la possibilité d'effectuer le « **power gating** » ou le « **voltage scaling** ».

Pour un état déjà rencontré, étant donné que c'est une technique temps réel, nous nous basons donc sur la prédiction des temps d'inactivation à partir des exécutions précédentes. Il est donc strictement obligatoire de respecter toutes les échéances des tâches exécutées sur le

VPU afin d'assurer une bonne prédiction d'où vient l'idée de prendre le minimum des durées de stabilité de chaque état. En effet, cet état existant dans l'historique possède un ou plusieurs états successeurs avec des durées de stabilité différentes, ainsi nous choisissons pour cet état actuel le minimum de ces temps pour le comparer ensuite avec une constante de temps appelée « **break_event_time** » en admettant que cette constante est la somme des temps de transition de « **off_voltage** » vers « **active_voltage** » (« **power_ up _delay** ») et vice versa (« **power_down_delay** »).

En tenant compte de la comparaison entre le temps minimum calculé et le « **break_event_time** » nous pouvons déterminer si l'intervalle de temps pendant lequel un VPU reste inactif « **IDLE** » est suffisamment grand pour absorber les pénalités des transitions entre « **IDLE** » et « **ACTIVE** » et par la suite appliquer le « **power gating** » ou si seulement le « **clock gating** » peut l'être.

A vrai dire, deux situations seront rencontrées, nous proposons dans la suite de ce paragraphe unefigure explicative(figure 23)afin de bien comprendre la démarche suivie :

-Soit nous trouvons que le minimum des temps calculé est supérieur au « **break_event_time** », alors à côté du « **clock gating** » ,nous pouvons appliquer le « **power gating** » sur le VPU en état inactif « **IDLE** » pour une durée égale à ce temps minimum moins le « **break_event_time** » c'est à dire couper l'alimentation en tension temporairement (tension nulle) à ce VPU pendant un certain temps.

Ensuite, après cette durée de temps le VPU doit être réveillé, il sera donc mis au mode actif « **ACTIVE** » en lui fournissant une tension non nulle (« **active_voltage** »), de même nous lui affectons une fréquence appropriée au mode actif « **ACTIVE** ».

Dans le cas, où nous trouvons que ce VPU reste « **IDLE** » dans tous les états successeurs, ce VPU ne sera pas réveillé dans cet état.

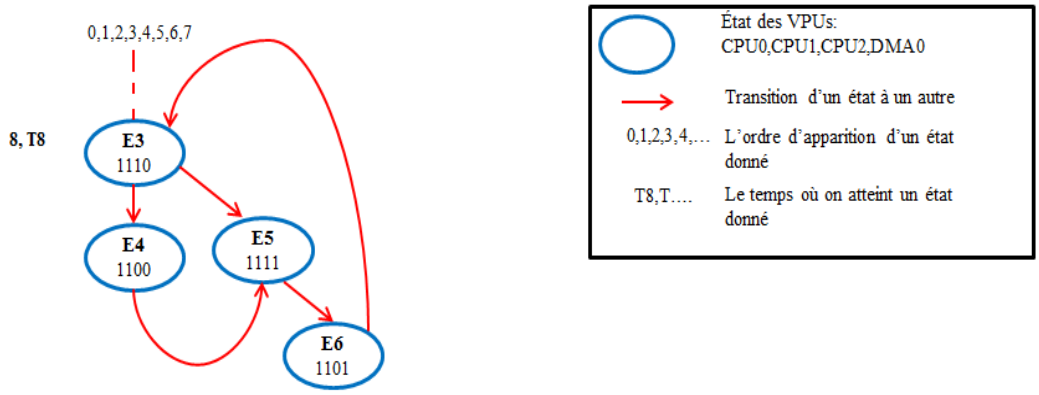
-Soit nous trouvons que le minimum des temps calculé est inférieur au « **break_event_time** », c'est dire le « **power gating** » ne peut pas être appliqué, dans ce cas, une deuxième comparaison est nécessaire entre ce temps minimum et la constante de temps appelée « **voltage_scaling_time** », en admettant que cette constante est la somme des temps de transition de « **idle_voltage** » vers « **active_voltage** » (« **voltage_scaling_up_delay** ») et vice versa (« **voltage_scaling_down_delay** »).

Ce qui nous mène à deux autres situations :

1) Le cas où le minimum des temps choisi est supérieur au « **voltage_scaling_time** », alors il est possible d'effectuer le « **voltage_scaling** » à côté du « **clock gating** » pour le VPU en état inactif « **IDLE** » pour une durée égale à ce temps minimum moins le « **voltage_scaling_time** ». C'est-à-dire que nous lui soumettons à une tension plus faible (« **idle_voltage** »). Ensuite, après cette durée nous devons augmenter cette valeur de tension en « **active_voltage** », de même,nous lui affectonsune fréquence appropriée au mode « **ACTIVE** ».

Dans le cas, où nous trouvons que ce VPU reste « **IDLE** » dans tous les états successeurs, ce n'est plus obligatoire de lui réaugmenter les valeurs de tension et de fréquence.

2) Le cas où le minimum des temps calculé est inférieur au « **voltage_scaling_time** », ici, seulement le « **clock gating** » peut être appliqué au VPU inactif « **IDLE** ».



Nous supposons que nous avons déjà rencontré les états E3,E4,E5 et E6 avec un nombre d'occurrences égale à 1 pour chacun d'eux, arrivant de nouveau à l'état E3 au temps T8 , nous sommes dans le cas où nous disposons d'un état déjà existant , donc nous allons suivre la démarche décrite dans le paragraphe précédent:
En parcourant l'historique des états , nous trouvons que l'état E3 a 2 successeurs E4 et E5 avec des temps de stabilité respectives s4 et s5 tel que s4 représente le temps de transition de l'état E3 vers l'état E4 et s5 est le temps de transition de l'état E3 vers l'état E5.
Dans ce cas , nous calculons le minimum entre ces deux temps de stabilité.
Supposons que s4= min (s4,s5) , donc le temps s4 sera pris comme temps de comparaison.
Nous comparons alors s4 avec la constante « **break_event_time** »:
- Si s4> « **break_event_time** » → appliquer le « power gating » à côté du « clock gating » sur le VPU « IDLE » (DMA0) de l'état E3,

En vérifiant l'état du DMA0 dans les successeurs E4 et E5 , nous trouvons qu'à l'état E5 le DMA0 devient « ACTIVE » donc il doit être réveillé après la durée de (s4- « **break_event_time** »)
- Si s4< « **break_event_time** » → ce n'est pas possible d'appliquer le « power gating » , 2 situations sont rencontrées:
  - si s4> « **voltage_scaling_time** » → appliquer le « voltage scaling » à côté du « clock gating » sur le DMA0 puis réaugmenter les valeurs de tension et de fréquence après une durée de (s4- « **voltage_scaling_time** »),
  - si s4< « voltage_scaling_time » → seulement le « clock gating » peut être appliqué sur le DMA0

Figure 23. Exemple explicatif

Après l'affectation des « PAM » aux différents VPUs de l'architecture, nous visualisons les vues de **Voltage**, **Frequency**, **Power States**, **Dynamic_power**, **Leakage_power** et **Total_power**.

L'analyse des résultats obtenus à l'aide de ces vues a montré une réduction au niveau de la consommation, nous obtenons ainsi une puissance statique nulle lors de l'application du « **power gating** » et une puissance dynamique nulle lors de l'application du « **clock gating** ».

Mais l'idée de prendre toujours le temps minimal entre les successeurs a introduit un réveil très rapide du VPU après avoir appliqué le « **power gating** ». Ce qui ne permet pas de maximiser le maintien d'une consommation faible d'un VPU dans son état d'activité « **IDLE** ».Ainsi, nous avons pensé à utiliser une autre méthode afin de retarder le temps de réveil des VPUs.

En effet, cette méthode consiste à calculer pour chaque état actuel le nombre d'occurrences des passages consécutifs pour tous ces successeurs, initialiser le nombre d'occurrences à zéro pour tous les successeurs de l'état actuel et à chaque fois que nous passons par un successeur, nous incrémentons par un, son nombre d'occurrences et nous affectons la valeur zéro pour les nombres d'occurrences associés aux autres successeurs. Par conséquent, si nous obtenons des nombres d'occurrences inférieurs à deux pour tous les successeurs, dans ce cas nous ne pouvons pas prédire qu'il y a un parmi les successeurs qui a une forte chance de se reproduire dans le futur proche donc nous suivons la méthode de calcul du minimum. Par contre, si l'un des successeurs vérifie bien un nombre d'occurrences supérieur ou égal à deux, dans ce cas, nous pouvons anticiper que ce successeur a une grande probabilité de se reproduire prochainement,nous prenons donc le temps de stabilité associé à ce successeur pour l'étude de la possibilité d'appliquer le « **power gating** » ou le « **voltage scaling** ».

Malheureusement, cette méthode ne peut pas être efficace dans notre cas parce qu'en parcourant le graphe des états obtenu, nous avons trouvé qu'un état successeur ne peut jamais dépasser le nombre de passages successifs de deux, donc nous allons avoir toujours une fausse prédiction de l'état suivant, à vrai dire même si nous trouvons un successeur qui a un nombre d'occurrences égale à deux, ce successeur ne peut jamais réapparaitre dans le prochain état.

Une idée pour optimiser l'approche était d'ajouter une condition à vérifier entre l'état actuel et ses successeurs avant de commencer la comparaison entre les temps des successeurs.

En effet, pour chaque état actuel, nous effectuons un tri croissant de ses successeurs selon le temps de stabilité. Ainsi, en parcourant le tableau des successeurs trié, nous vérifions si l'état actuel se conserve en faisant son union avec l'état successeur, donc :

-si la condition est vraie, dans ce cas, nous sommes sûrs que les VPUs en état IDLE pour l'état actuel vont rester IDLE dans l'état successeur donc ce successeur ne sera pas pris en compte alors nous vérifions la condition avec le successeur suivant dans le tableau trié.

-si la condition n'est pas vraie, dans ce cas, le temps de stabilité associé à ce successeur sera pris en compte comme temps de comparaison au lieu du temps minimum soit avec le « **break_event_time** » soit avec le **« voltage_scaling_time »** pour décider la possibilité d'appliquer le « **power gating** » ou le « **voltage scaling** » et nous arrêtons le parcourt.Ainsi si la condition est vérifiée pour tous les successeurs, nous pouvons directement appliquer le « **power gating** » sans être obligé de faire le réveil du VPU.

En appliquant cette technique de « **power management** » sur l'application « Video_Capture » avec l'architecture du mapping adaptée, nous obtenons les résultats suivants.

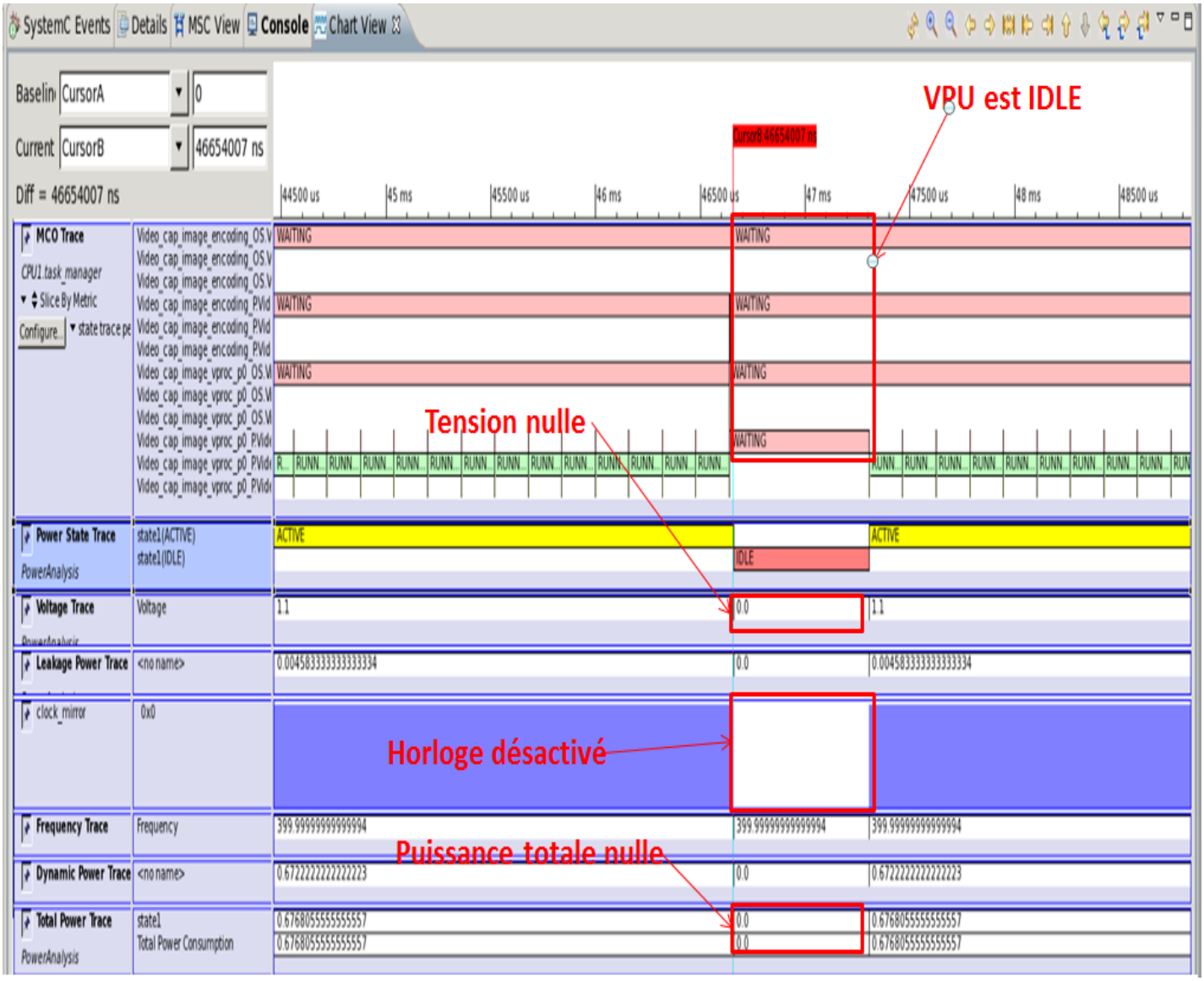


Figure 24. Consommation « power » du CPU1

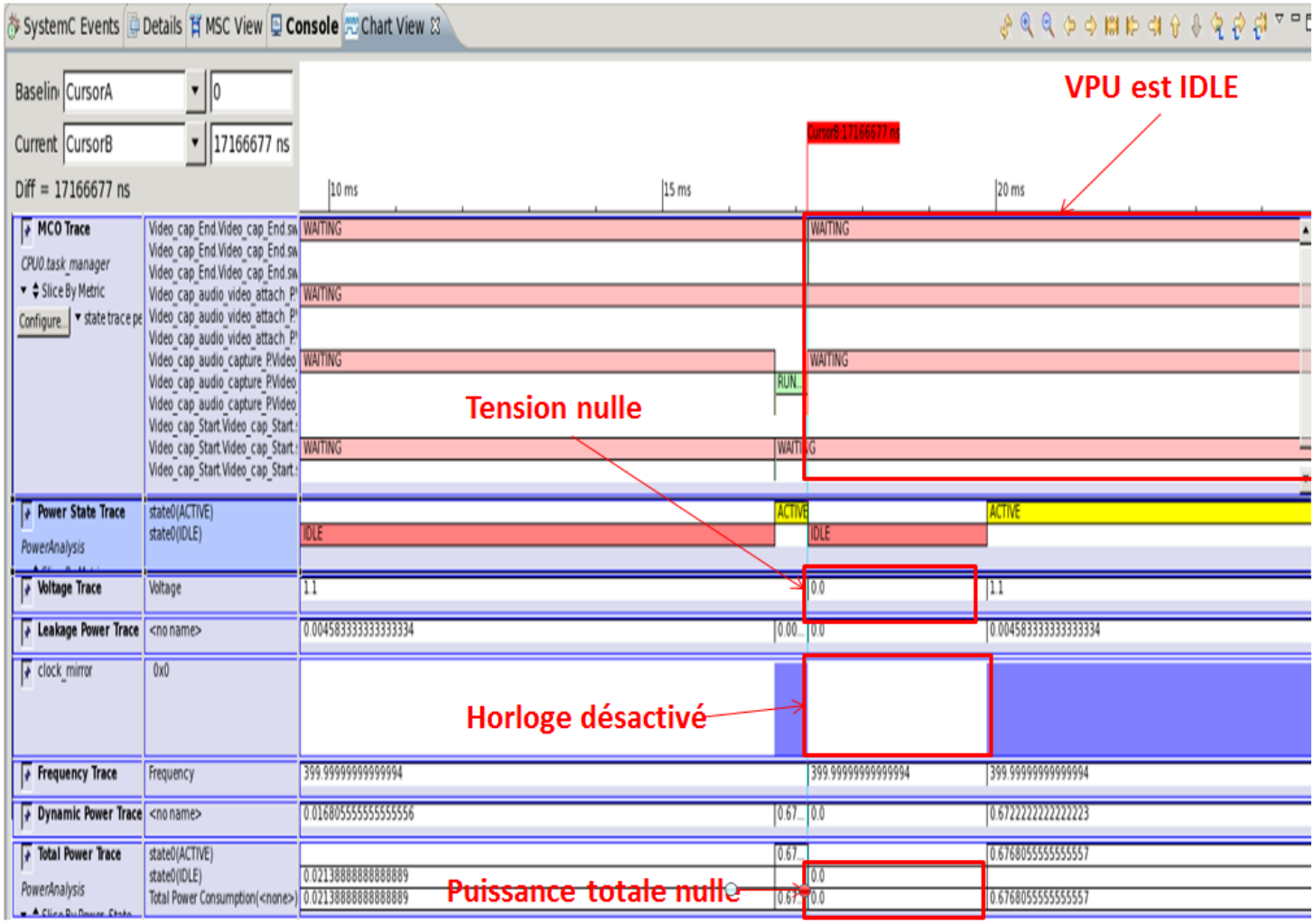


Figure 25. Consommation « power » du CPU0

Les figures 24 et 25montrent la consommation en puissance des VPUs « CPU1 » et « CPU0 ».Il est bien visible que pour « CPU1 » (figure 24), nous avons appliqué le « **power gating** » (tension nulle) et le « **clock gating** » (désactivation de l'horloge) pour toute la période pendant laquelle il est resté en état inactif « **IDLE** ».

Par contre, pour « CPU0 » (figure 25), nous avons appliqué aussi le « **power gating** » et le « **clock gating** » mais, bien que « CPU0 » soit resté en état inactif « IDLE » nous l'avons réveillé en lui fournissant une tension non nulle et en activant son horloge c'est à dire que le temps de réveil calculé pour ce VPU est trop tôt. Par conséquent cela peut influer négativement sur la consommation en énergie de ce VPU vu qu'il va consommer de l'énergie dès qu'il passe en état « ACTIVE ».

Ainsi, nous avons pensé à optimiser notre approche en modifiant la stratégie suivie dans le but de mieux estimer le temps de réveil pour chaque VPU.

La nouvelle idées'appuie sur la supposition que pour un état actuel, nous supposons que son successeur qui était récemment rencontré a une plus forte chance de se répéter dans le futur proche par rapport à ses autres successeurs.Dans ce cas,pour vérifier si nous pouvons effectuer le « **power gating** » ou le « **voltage scaling** » à un VPU donné, nous considérons le temps de stabilisation associé au successeur le plus récent comme temps de comparaison avec le « **break_event_time** » ou avec le « **voltage_scaling_time** ».

Les figures suivantes montrent les résultats de l'application de cette stratégie :

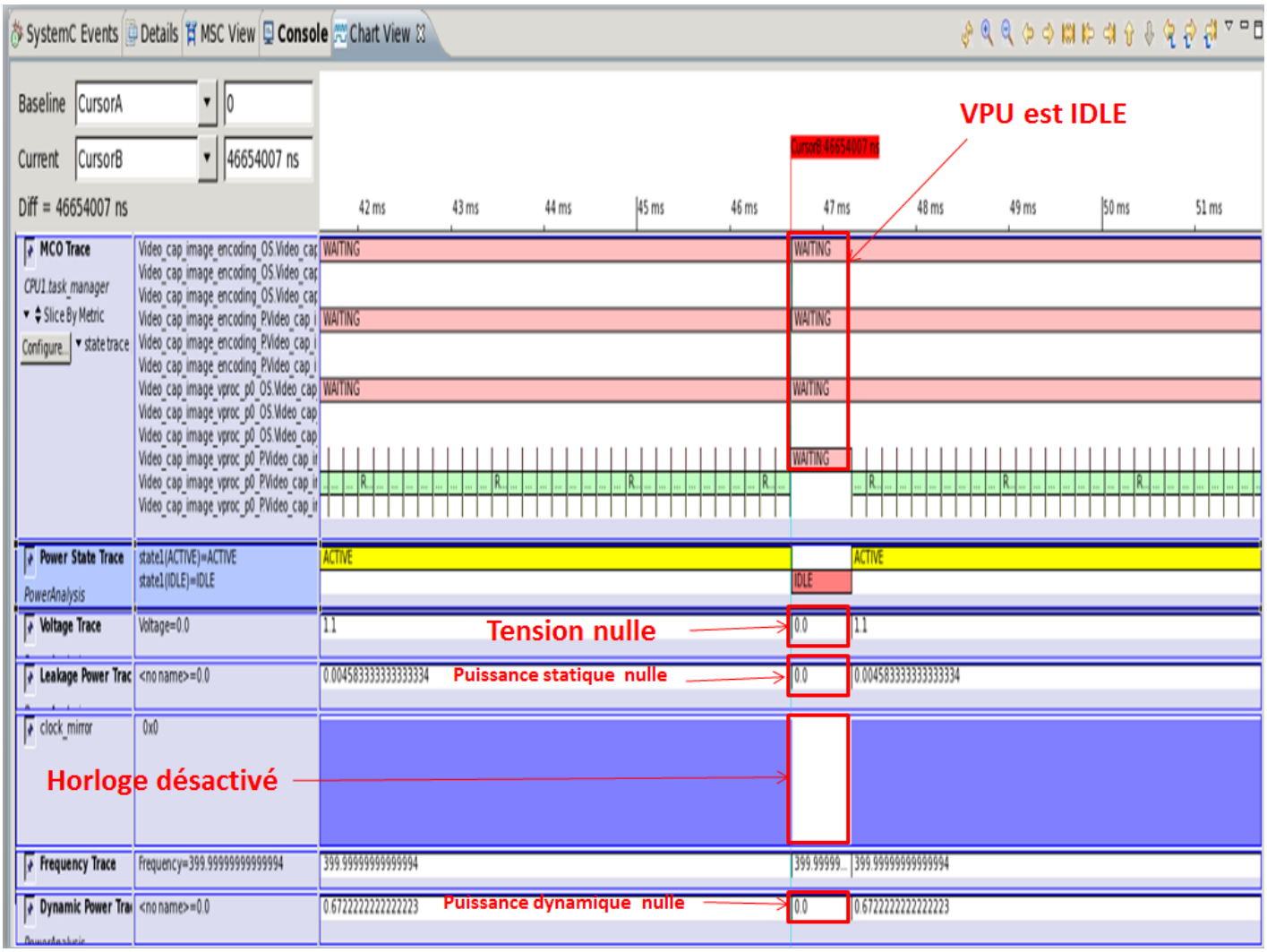


Figure 26. Consommation « power » du CPU1

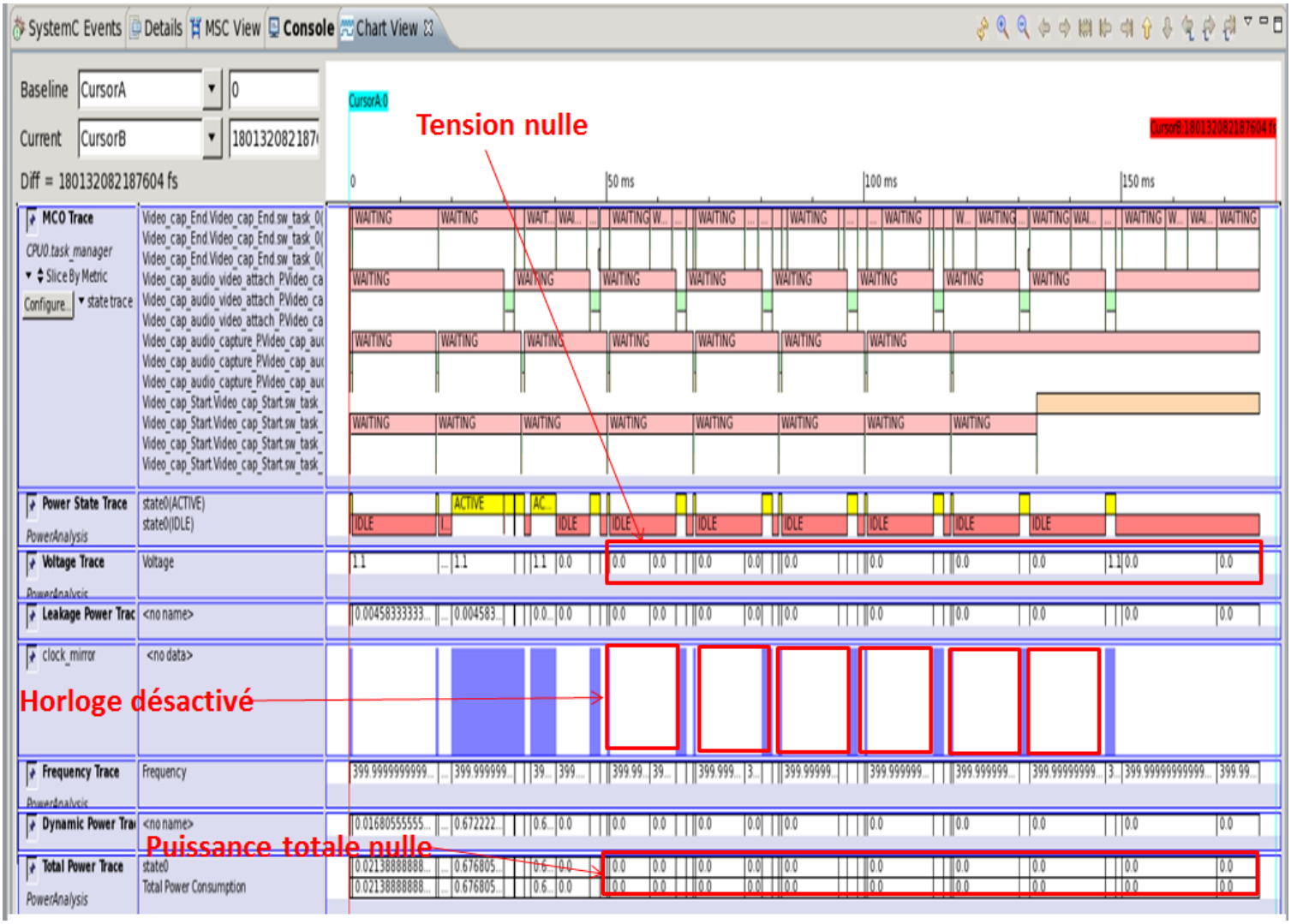


Figure 27. Consommation « power » du CPU0

Comme le montre la figure 27, à partir d'un certain temps, le VPU est maintenu sous une tension nulle et une horloge désactivée pendant toute la période où il est en état inactif « **IDLE** ». En effet cette stratégie est une sorte d'apprentissage qui suite aux exécutions successives des tâches sur le VPU étudié, augmente la chance d'avoir une vraie prédiction de l'état suivant.

Par conséquent les valeurs de tension et d'horloge fournies au VPU seront adaptées régulièrement selon son état d'activité tout en respectant les contraintes de temps réel et de séquencement des tâches.

IV.5 Comparaison;

Dans cette partie, nous nous intéressonsà comparer le résultatobtenu en consommation d'énergie suite à l'application de notre approche de « power management » avec celle obtenu par le « power manager » spécifique à Synopsys.

En se basant sur les figures présentées ci-dessous, nous comparons la consommation d'énergie du VPU « CPU0 » de notre architecture.

Figure 28. Consommation d'énergie du CPU0 suite à la méthode de Synopsys

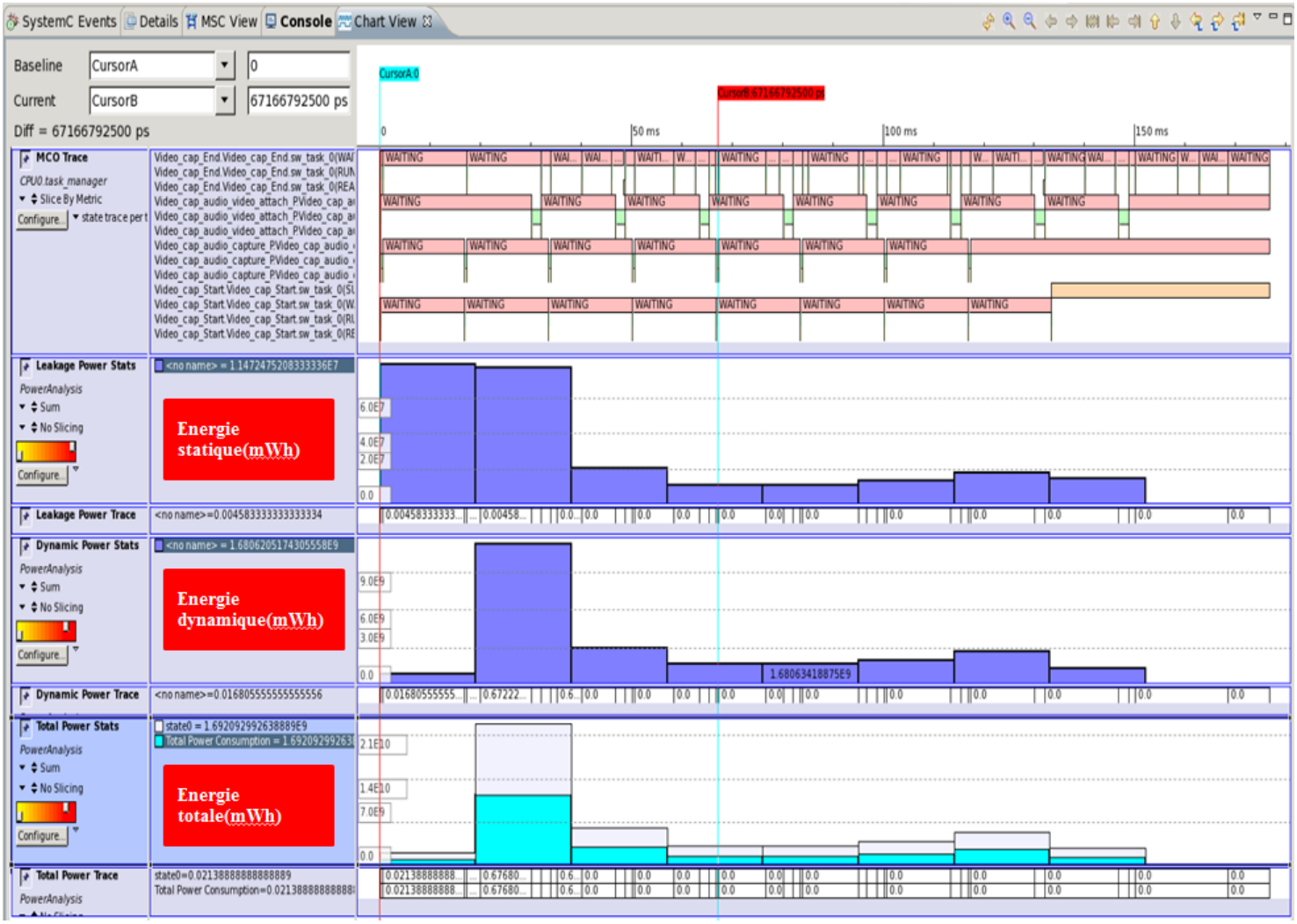


Figure 29. Consommation d'énergie du CPU0 suite à l'approche de LEAT

Les figures ci –dessus représentent les énergies statique (mWh), dynamique (mWh) et totale (mWh) du composant CPU0 suite à l'application de deux méthodes de « **power management** ».

Le calcul de l'énergie consommée est effectué pendant chaque période de 19 ms.

Pendant cette période ce composant passe par l'état actif « ACTIVE » et l'état inactif « IDLE » donc il aura des valeurs d'énergie différentes, par la suite, la valeur considérée pour cet intervalle sera la somme de ces valeurs calculées.

Nous avons effectué des calculs de la consommation totale d'énergie de tous les composants du système pour chacun des deux méthodes de power management. Ainsi, une comparaison peut être réalisée entre ces deux méthodes.

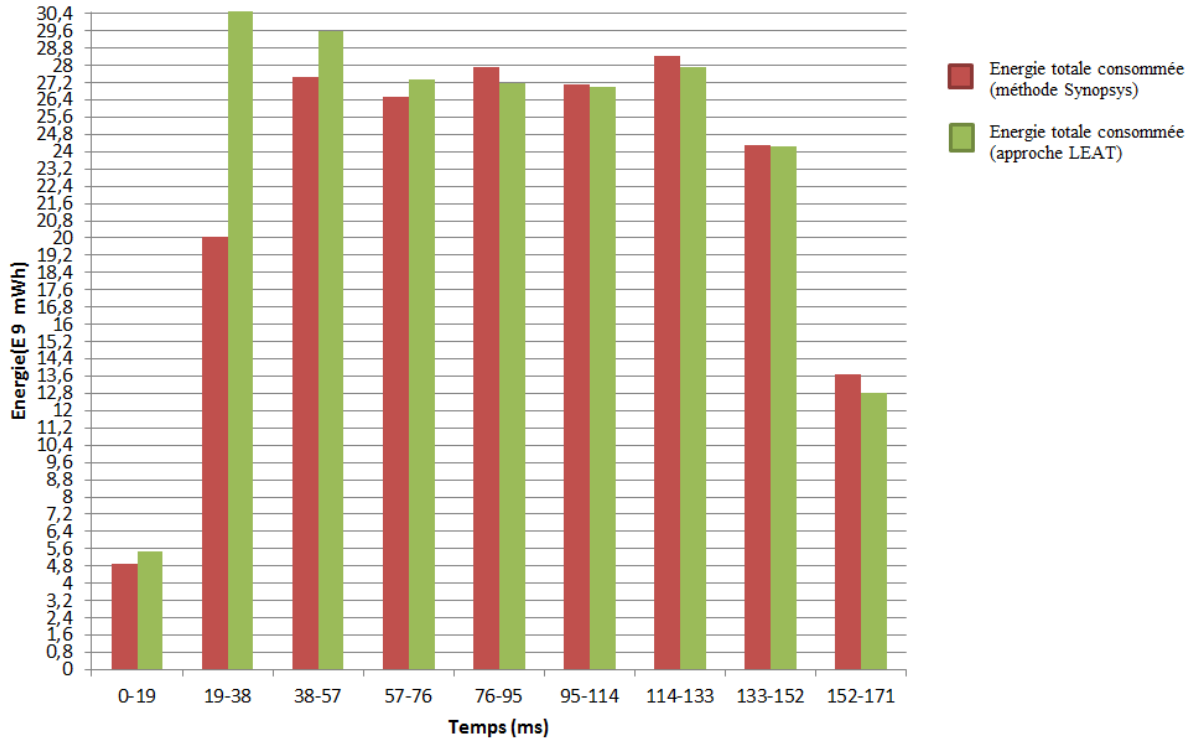


Figure 30. Comparaison des résultats de consommation d'énergie obtenus par la méthode de **Synopsys** et l'approche de **LEAT**

En se basant sur le diagramme de consommation d'énergie présenté dans la figure 30, nous pouvons déduire les résultats suivants :

-Au début de la simulation (de 0 à 76 millisecondes), nous remarquons que la consommation d'énergie obtenue par notre approche est visiblement supérieure à celle obtenue par la méthode suivie par Synopsys. Cela s'explique par le fait que notre approche se base sur une méthode d'apprentissage qui dure quelques millisecondes lorsque le système n'est pas encore la connaissance de la façon dont il traite les états d'activités des VPUs c'est-à-dire que même si nous décidons de mettre un tel VPU en « **Power down** » : lui appliquer des valeurs de tensions et de fréquence appropriées à son état inactif « **IDLE** », le réveil de ce VPU est rapidement effectué. Donc il y aura une probabilité que ce VPU sera traité comme s'il était resté en état actif « **ACTIVE** » tandis qu'il est encore dans un état fonctionnel inactif « IDLE ».

Par conséquent, nous aurons des valeurs de consommation d'énergie supérieures à celles obtenues par la méthode de Synopsys qui traite les VPUs dès le début de la simulation jusqu'à sa fin d'une façon à faire varier ses valeurs de fréquences et de tension suivant la notification reçue :

- si la notification reçue d'un tel VPU indique qu'il en état actif « ACTIVE », des valeurs « active_voltage » et « active_divider » lui seront attribuées.

- si la notification reçue d'un tel VPU indique qu'il en état inactif « IDLE », des valeurs « idle_voltage » et « idle_divider » lui seront attribuées.

-A partir de l'instant 76 millisecondes, nous commençons à avoir des valeurs légèrement réduites par rapport à celles obtenues par la méthode de Synopsys, ceci s'explique par le fait que le système est bien commencé à construire l'historique des états, c'est-à-dire qu'à partir d'un état actuel nous anticipons son état successeur et suivant cet état prédit nous étudions la possibilité de mettre les VPUs qui sont en état inactif « IDLE » en « **power down** ».

Ainsi, une application de « Video_Capture » peut durer quelques minutes donc nous pouvons au fil du temps remédier la grande consommation en énergie au tout début de la simulation.

-Bien que notre approche ne montre pas une grande réduction d'énergie par rapport à la méthode de Synopsys, notre technique utilisée offre plus de garantie. En effet, pour la méthode de Synopsys elle n'effectue pas une vérification du temps avant de mettre un VPU en « power down » c'est-à-dire qu'elle ne vérifie pas si le temps où ce VPU va rester « IDLE » est vraiment suffisant pour le mettre en « **power down** » sans causer des pénalités. Ceci s'explique par la façon dont Synopsys traite les changements des états des VPUs. En effet, leur méthode ne permet pas de faire cette vérification vu qu'elle réagit et prend la décision à propos des valeurs attribuées après avoir été notifiée de l'état du VPU. De cette façon en augmentant les pénalités en temps de transitions, cette méthode ne sera plus valide vu qu'elle va perturber le temps d'exécution des tâches de l'application. A l'inverse, notre approche vise à prédire le prochain état avant qu'il ne soit produit, de cette façon nous pouvons effectuer la vérification avec le temps de stabilisation du passage de l'état actuel à l'état prédit pour décider si nous pouvons mettre un tel VPU en « power down », comme ça nous pouvons éviter ces pénalités. Aussi grâce à cette prédiction, nous gagnons le temps émis pour le passage d'un état à un autre d'un VPU.

V. Conclusion et perspectives

Le travail effectué lors de ce projet permet de mieux comprendre et de formaliser notre approche adoptée pour la gestion de l'énergie appliquée sur une architecture dédiée à une application « Video_Capture » caractérisée par une consommation très importante en termes d'énergie.

L'idée était de réaliser une modélisation de la gestion de l'énergie à travers l'implémentation d'un « power manager » en se basant sur des hypothèses restrictives.
Nous avons donc travaillé sur un outil de prototypage virtuel permettant de simuler notre modèle de manière assez rapide et adaptée au contexte.Pour ce faire, nous avons suivi un apprentissage et manipuler des exemples afin de se familiariser avec l'outil « Platform Architect MCO ».Cette plateforme offre des librairies qui nous permettent de travailler avec des composants qui sont déjà implémentés pour modéliser l'architecture à étudier.
Ces différents composants sont implémentés par le langage SystemC TLM,un apprentissage et une manipulation de quelques exemples en SystemC TLM était donc indispensable dans le but depouvoir analyser le fonctionnement de ces composants, de les adapter à notre contexte de travail et par la suite les bien exploiter.

Dans un premier temps, nous avons effectué une étude sur l'application « Video_Capture », dans ce contexte, nous avons discuté avec les représentants de Intel afin d'acquérir des informations et de clarifier les notions non évidentes.
Dans un deuxième temps, nous étions censés de répartir d'une façon optimale les tâches de notre application sur un nombre réduit de ressources de traitements virtuelles qui composent l'architecture cible de notre système.Pour cela, nous avons appliqué une approche d'optimisation par programmation linéaire pour le mapping et l'ordonnancement d'un graphe de tâches constitué d'un nombre réduit de tâchesdans le but d'étendre par la suite la modélisation de ce problème d'optimisation afin de résoudre notre problème.

Cependant, il n'était pas possible pendant la durée limitée du stage d'aller plus en avant dans cette étude, ainsi nous avons choisi d'opérer par simulation dans la partie optimisation d'énergie.

Dans un troisième temps, nous avons commencé à étudier et à implémenter notre approche de « power management » sur notre système. Pour cefaire,il était nécessaire de créer de nouveaux composants en SystemC-TLM pour modifier la plateforme Synopsys en vue d'intégrer le nouveau « power manager » proposé.

Ensuite, nous avons développé ce « power manager » dans « Platform Architect MCO »et nous avons effectué une étude en simulation afin d'analyser son comportement et effectuer des modifications qui améliorent sa méthodologie de gestion d'énergie.

Ainsi, les résultats obtenus par le biais des simulations permettent de bien mettre en évidence le travail de modélisation qui a été effectué durant ce projet. Cela nous permet d'augmenter la confiance que nous pouvons avoir dans la possibilité d'optimiser plus encore notre approche afin d'améliorer encore les résultats obtenus, étendre notre étude de la consommation d'énergie nous semble donc prometteuse.

En effet, certaines améliorations peuvent être apportées à ce travail. Nous pouvons effectuer des études sur la succession des états d'activités du système, trouver une solution qui permet de repérer la boucle répétitive d'un certain nombre d'états d'activités et apprendre à notre système à connaitre cette succession puis anticiper à travers cet apprentissage la décision la plus probable, cela pourrait augmenter la probabilité d'avoir une prédiction correcte et donc avoir une meilleure efficacité de gestion de l'énergie.

Table des figures

Liste des tables

Bibliographie

[1]http://www.cl.cam.ac.uk/research/srg/han/ACSP35/documents/TLM_2_0_presentation.pdf

[2]http://documents.irevues.inist.fr/bitstream/handle/2042/4459/000ED.PDF+TEXTE.pdf?sequence=1

[3]http://www.electronicsweekly.com/news/design/eda-and-ip/mentor-graphics-sees-esl-route-to-power-optimisation-2010-09/

[4]http://www.emo.org.tr/ekler/035226640b6b89f_ek.pdf

[5]http://ens.ewi.tudelft.nl/Education/courses/et4351/SystemC-TLM-14v1.pdf

[6]http://www.esiee.fr/~info/a2si/rapport1999-2003/node127.html

[7]http://leat.unice.fr/ECoFaC2012/presentations/Auguin.pdf

[8]http://ftp.linux62.org/~glibersat/etatArt.pdf

[9]http://www.research.ibm.com/haifa/conferences/hvc2008/present/The_challenges_of_low_power_design.pdf

[10]Alberto Aguirre, Aziz Umit Batur, Gregory Hewes, Ibrahim Pekkucuksen, Narasimhan Venkatraman, Fred Ware, Buyue Zhang, Embedded stereoscopic 3D camera system for mobile platforms, ICASSP, 25-30 March, Kyoto, 2012

[11]Ons Mbarek, An Electronic System Level Modeling Approach for the Design and Verification of Low-Power Systems-on-Chip

[12]Documentation fournie par Synopsys

[13]RALF NIEMANN, PETER MARWEDEL, An Algorithm for Hardware/Software Partitioning Using Mixed Integer Linear Programming, Ralf Niemann and Peter Marwedel

[14]Salamy, H.; Electr. Eng , Texas State Univ , San Marcos , TX, USA , Aslan , Methukumalli ; TASK SCHEDULING ON MULTICORES UNDER ENERGY AND POWER CONSTRAINTS, 2013 26th IEEE Canadian Conference Of Electrical And Computer Engineering (CCECE)

[15]Y. SOREL, « Massively Parallel Systems with Real Time Constraints: the Algorithm Architecture Adequation methodology »,in: Proc. of Massively Parallel Computing Systems, the Challenges of General-Purpose and Special-Purpose Computing Conference,Ischia Italy, May 1994.

Les annexes

Annexe A : les sous tâches des différents blocs du graphe de tâches « Video Capture »

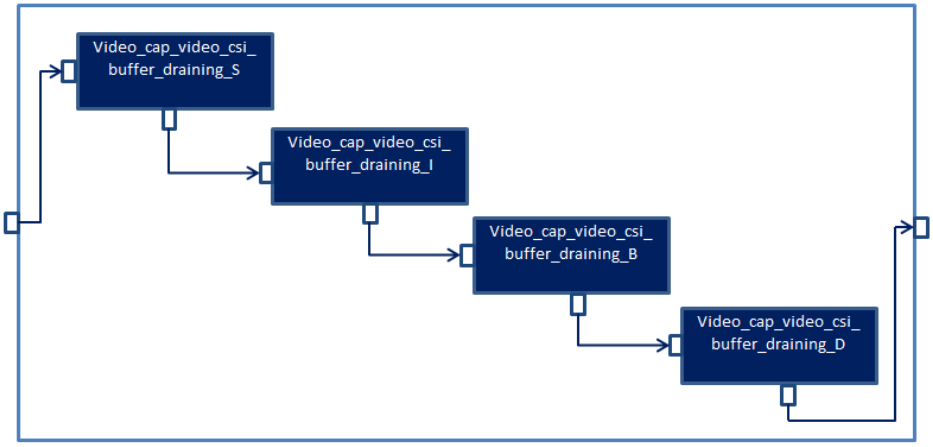


Figure 31.Bloc: Video_cap_video_csi_buffer_draining

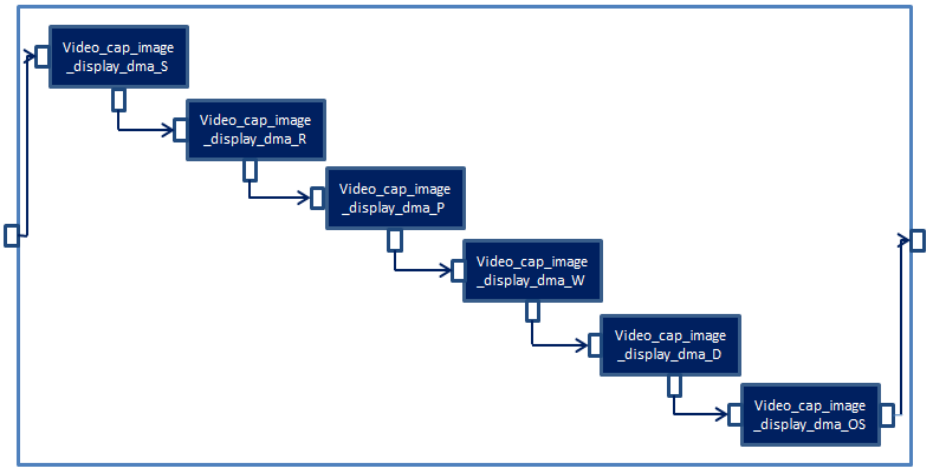


Figure32. Bloc: Video_cap_image_display_dma

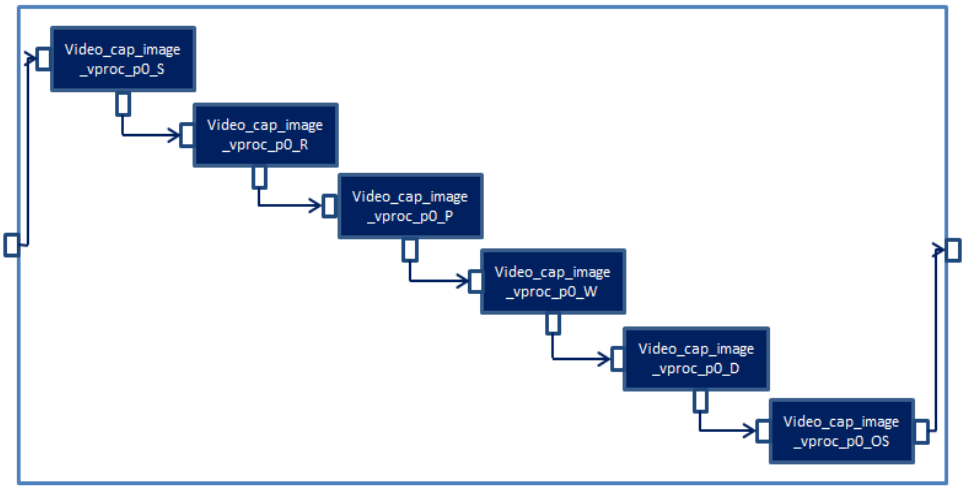


Figure33. Bloc: Video_cap_image_vproc_pO

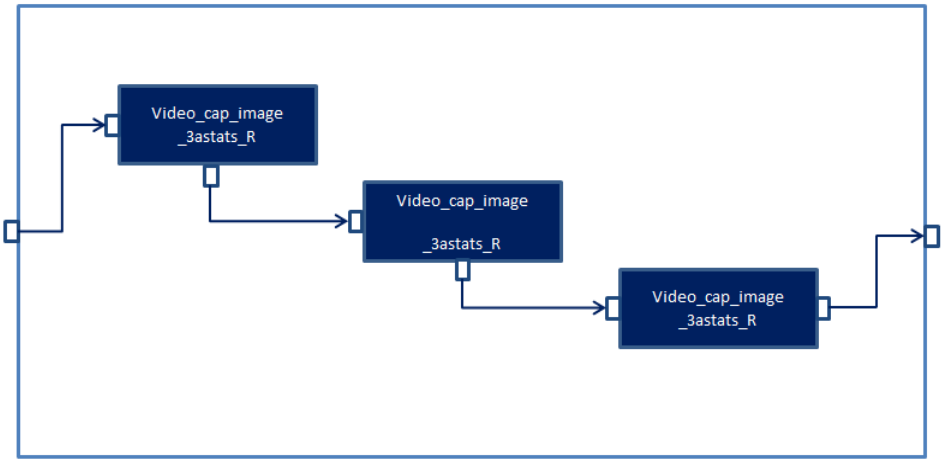


Figure34. Bloc: Video_cap_image_3astats

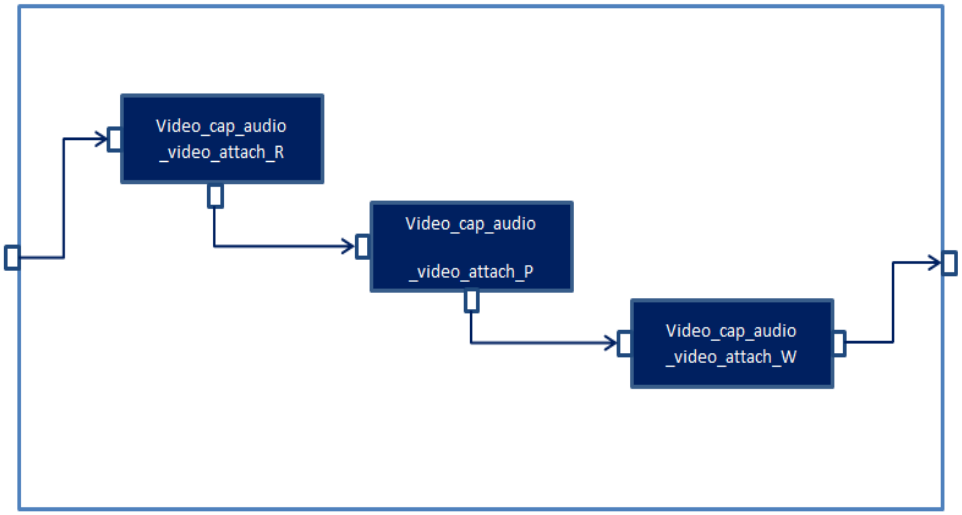


Figure35 Bloc: Video_cap_audio_video_attach

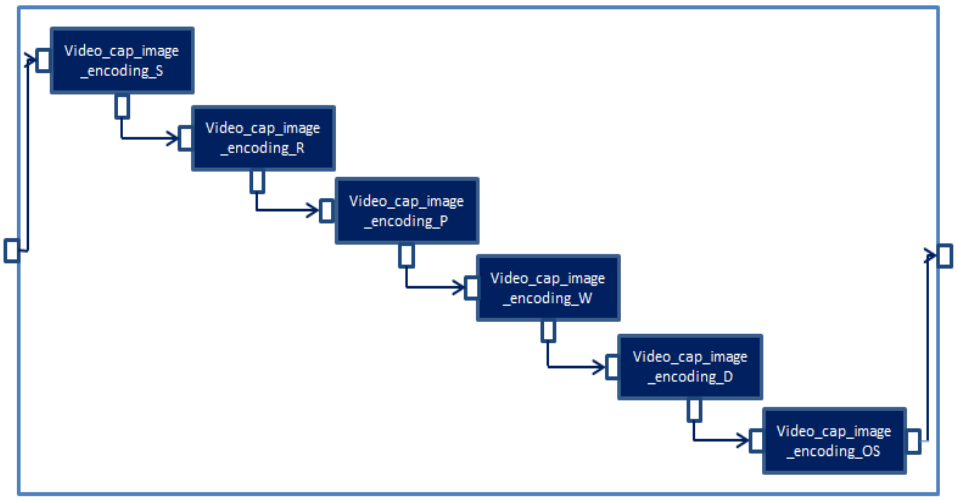


Figure36. Bloc: Video_cap_image_encoding

Annexe B : Description des différents tâches du graphe de tâches « Video Capture »

- La tâche source de l'application

Video_cap_Start : c'est la tâche qui déclenche l'exécution de l'ensemble des tâches de l'application. Cette tâche dispose d'un wait_delay=16666667 ns, d'un nombre d'itérations=8 et d'un temps d'exécution nul. C'est-à-dire qu'elle se déclenche 8 fois, la 1ère fois au temps 0 ns et puis après chaque durée de 16666667 ns.

Cette tâche à un temps d'exécution nul implique que les tâches qui la suivent directement dans le tableau de connexions s'exécutent au même temps de son déclenchement.

- Les tâches du bloc « Video_cap_video_csi_buffer_draining»

-Video_cap_video_csi_buffer_draining_S: c'est une tâche « Start » de traitement, elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul. Elle produit 91 jetons par activation.

- Video_cap_video_csi_buffer_draining_I : c'est une tâche de traitement. Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Video_cap_video_csi_buffer_draining_B: C'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire. Elle consomme 91 jetons par activation.

- Video_cap_video_csi_buffer_draining_D : c'est une tâche « Done » de traitement.

Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Les tâches du bloc « Video_cap_image_vproc_pO»

- Video_cap_image_vproc_p0_S: c'est une tâche « Start » de traitement. Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul. Elle produit 100 jetons par activation.

- Video_cap_image_vproc_p0_R : c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

-Video_cap_image_vproc_p0_P : c'est une tâche de traitement. Elle dispose d'un processing_cycles=64000 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

-Video_cap_image_vproc_p0_W : c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire. Elle consomme 100 jetons par activation.

-Video_cap_image_vproc_p0_D : c'est une tâche « Done » de traitement.

Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Video_cap_image_vproc_p0_OS : c'est une tâche de traitement.

Elle dispose d'un processing_cycles=400 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

Les tâches du bloc « Video_cap_image_3astats»

- Video_cap_image_3astats_R : c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

- Video_cap_image_3astats_P : c'est une tâche de traitement. Elle dispose d'un processing_cycles=4000000 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

-Video_cap_image_3astats_W : c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire.

- Les tâches du bloc « Video_cap_image_encoding»

- Video_cap_image_encoding_S : c'est une tâche « Start » de traitement, elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul. Elle produit 100 jetons par activation.

- Video_cap_image_encoding_R: c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

- Video_cap_image_encoding_P: c'est une tâche « Start » de traitement.

Elle dispose d'un processing_cycles= 67 donc un temps d'exécution non nul.

-Video_cap_image_encoding_W: c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire. Elle produit 100 jetons par activation.

- Video_cap_image_encoding_D : c'est une tâche « Done » de traitement. Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Video_cap_image_encoding_OS : c'est une tâche de traitement.

Elle dispose d'un processing_cycles=400 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

Les tâches du bloc « Video_cap_image_display_dma»

-Video_cap_image_display_dma_S: c'est une tâche « Start » de traitement. Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul. Elle produit 24 jetons par activation.

- Video_cap_image_display_dma_I: c'est une tâche « Start » de traitement.Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Video_cap_image_display_dma_B: c'est une tâche de type « mem_function », elle effectue une opération de lecture de mémoire. Il consomme 24 jetons par activation.

- Video_cap_image_display_dma_D: c'est une tâche « Done » de traitement. Elle dispose d'un processing_cycles=0 c'est-à-dire qu'elle a un temps d'exécution nul.

- Les tâches du bloc « Video_cap_audio_video_attach»

- Video_cap_audio_video_attach_R: c'est une tâche d'accès à la mémoire, elle effectue une opération de lecture de mémoire.

- Video_cap_audio_video_attach_P: c'est une tâche de traitement.

Elle dispose d'un processing_cycles= 800000 donc le temps d'exécution est obtenu en multipliant cette valeur par la période d'horloge de l'unité de traitement qui exécute cette tâche.

-Video_cap_audio_video_attach_W: c'est une tâche d'accès à la mémoire, elle effectue une opération d'écriture de mémoire.

- La tâche Sink (elle ne dispose pas de sorties)

Video_cap_End : cette tâche est exécutée uniquement pour mettre fin à la simulation après qu'un montant spécifique de jetons est reçu :

- Soit après la réception d'un jeton qui indique la fin d'exécution des tâches Video_cap_image_display_dma.
- Soit après la réception d'un jeton qui indique la fin d'exécution des tâches Video_cap_image_3astats.
- Soit après la réception d'un jeton qui indique la fin d'exécution des tâches Video_cap_audio_attach.

Annexe C : Problème de programmation linéaire pour une application simple

Nous disposons d'une application constitué d'un ensemble de trois tâches (A= $\{T_1, T_2, T_3\}$) représentée dans la figure …

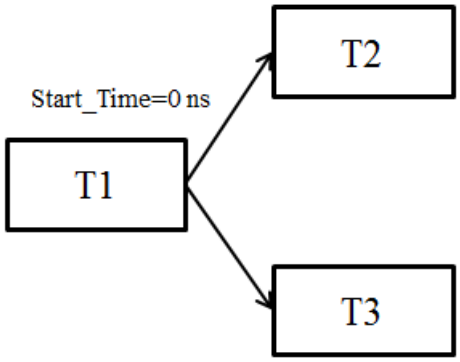


Figure 37. Application simple

L'ensemble de ses tâches représente les contraintes suivantes :

-La tâche T_1 est la tâche source qui déclenche toute l'application, elle commence au temps 0ns.

-La tâche T_2 doit commencer directement après la fin de la tâche T_1.

-La tâche T_3 doit commencer directement après la fin de la tâche T_2.

-La tâche T_2 et T_3 sont exécutées en parallèles.

-La tâche T_2 doit terminer son exécution avant un deadline précis.

-La tâche T_3 doit terminer son exécution avant un deadline précis.

L'architecture cible est constituée de deux VPUs : VPU_0 et VPU_1 tels que :

-VPU_0 dispose de deux fréquences de fonctionnement : f1 et f2.

-VPU_1 dispose d'une seule fréquence de fonctionnement : f.

Problématique

Nous visons à faire un mapping des tâches de l'application sur les VPUs de l'architecture de façon à déterminer un ordonnancement optimal pour un système multiprocesseur et que toutes les tâches soient exécutées avant les deadlines respectives tout en respectant le séquencement des tâches.

Contraintes

-minimiser le nombre de VPUs utilisés

-maximiser le temps d'exécution des VPUs

- les tâches concurrentes qui doivent être mappées sur les différents VPUs disponibles sans l'application

-les tâches sont ordonnées d'une façon à les faire exécuter le plus possible en parallèle

-suivre l'ordre d'exécution des tâches (séquencement des tâches).

-prendre en considération les tâches qui s'exécutent en parallèle : les tâches en parallèles ne peuvent jamais être mappées sur un même VPU

-une tâche est mappée sur un et un seul VPU

→ Il s'agit d'un Problème **NP-Complet**

Fonction objective

Maximiser le temps d'exécution de l'ensemble de tâches

Variables du problème

Pour résoudre ce problème, nous avons définis un ensemble de variables déclarées comme des entiers :

E_i : temps de fin d'exécution de la tâche T_i

S_i= temps de début d'exécution de la tâche T_i

Y_{ik} : c'est une variable binaire tel que :

- Y_{ik}= 1 si la tâche T_i est exécutée sur le VPU_k
- Y_{ik}= 0 sinon

F_{ik} : la fréquence fk associée au VPU_i

n_i= nombre de cycles associé au VPU_i

T_{ijk} : le temps d'exécution de la tâche T_i sur le VPU_j avec la fréquence F_{jk}

F_{ijk} : c'est une variable binaire tel que :

- F_{ijk}=1 si la tâche T_i est exécutée sur un VPU_j avec la fréquence Fj_k
- F_{ijk}=0 sinon

Equations du problème

-Une tâche T_i ne peut être exécutée que sur un seul VPU en même temps.

Cela se traduit par l'équation : la somme des Y_{ik}=1 tels que :

i=1..3 : numéro de la tâche, k=1..2 : numéro du VPU

-Une tâche T_i ne peut être exécutée que sur un seul VPU_j avec une seule fréquence F_{jk}

Cela se traduit par l'équation : Y_{ik}= la somme des F_{ijk} <=1tels que :

i : numéro de la tâche (constant), j : numéro du VPU (constant) et k varie selon le nombre de fréquences associées au VPU_j

-Le temps d'exécution d'une tâche T_i sur le VPU_j avec la fréquence F_{jk} se traduit par l'équation suivante : $T_{ijk}= n_j * F_{jk}$

-Le temps de fin d'exécution d'une tâche T_i se traduit par l'équation suivante :

$E_i= S_i$ + la somme des (Y_{ij} x T_{ijk}) tels que :

j=1..2 : numéro du VPU et k varie selon le nombre de fréquences associées au VPU_j

-La tâche i doit s'exécuter suite à la fin de la tâche j.

Cela se traduit par l'équation : $S_i=E_j$

-La tâche T_i a un deadline **d** à ne pas dépasser

Cela se traduit par l'équation : $E_i<=d$