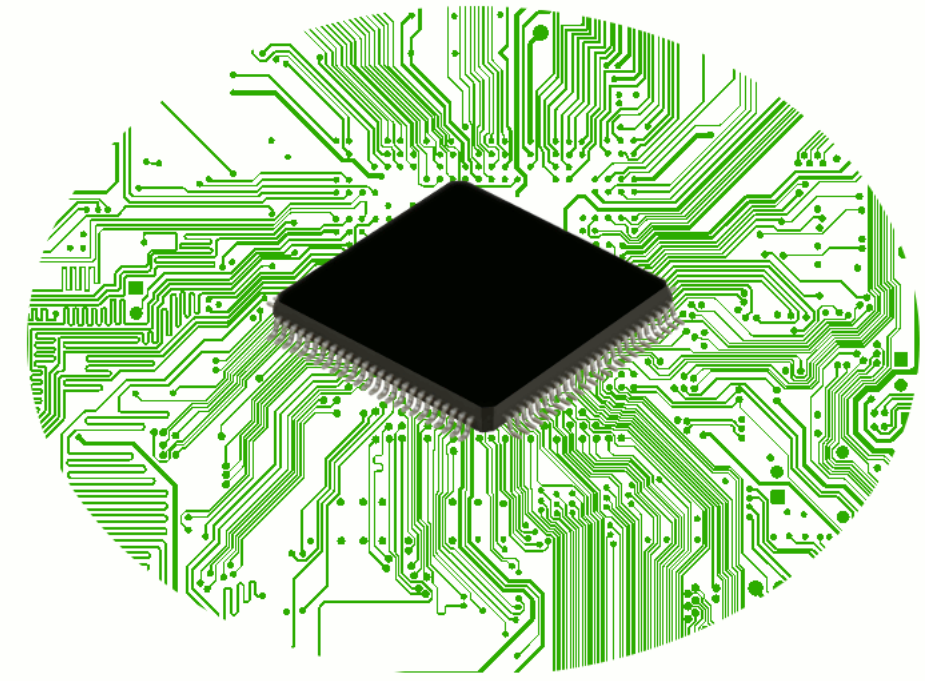
Commande faible coût pour une réduction de la consommation d'énergie dans les systèmes électroniques embarqués

Sylvain Durand

PhD Thesis

REDUCTION OF THE ENERGY CONSUMPTION IN EMBEDDED ELECTRONIC DEVICES WITH LOW CONTROL COMPUTATIONAL COST



Sylvain DURAND

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Automatique et Productique**

Arrêté ministérial : 7 août 2006

Présentée par

Sylvain DURAND

Thèse dirigée par **Nicolas MARCHAND** et **Daniel SIMON**

préparée au sein du centre de recherche **INRIA Grenoble Rhône-Alpes**, du laboratoire **GIPSA-Lab, département automatique** et de l'école doctorale **Électronique, Électrotechnique, Automatique et Traitement du Signal**

Reduction of the energy consumption in embedded electronic devices with low control computational cost

Thèse soutenue publiquement le **17 janvier 2011**, devant le jury composé de :

Mazen ALAMIR, Président
Directeur de Recherche CNRS, GIPSA-Lab (Grenoble, France)
Patrick BOUCHER, Rapporteur
Professeur des Universités, Supélec (Paris, France)
Maurice HEEMELS, Rapporteur
Professor, Eindhoven University of Technology (Eindhoven, Pays-Bas)
Arben CELA, Examinateur
Maître de Conférences, ESIEE (Paris, France)
Jose-Fermi GUERRERO-CASTELLANOS, Examinateur
Profesor Investigador, BUAP (Puebla, Mexique)
Suzanne LESECQ, Examinatrice
Ingénieure de Recherche, CEA-Leti (Grenoble, France)
Laurent FESQUET, Invité
Maître de Conférences, TIMA (Grenoble, France)
Daniel SIMON, Directeur de thèse
Chargé de Recherche INRIA, INRIA Rhône-Alpes (Grenoble, France)
Nicolas MARCHAND, Co-Directeur de thèse
Chargé de Recherche CNRS, GIPSA-Lab (Grenoble, France)

à Cécile.

Aknowledgments

This thesis arose in part of research that has been done within the NeCS group in the INRIA Rhône-Alpes research center and the control department of the GIPSA-Lab, in Grenoble. This experience would not have been possible without the support of many people. This page is dedicated to them.

First of all, I greatly indebted my two supervisors, Nicolas Marchand and Daniel Simon, for the confidence they accorded to me. Their encouraging and personal guidance have provided a good background for the present thesis.

I am also grateful to the jury members for the precious time they have spent for me. Thanks to Prof. Mazen Alamir, for the interest he gave to this work by agreeing to be the president of my jury. I am also thankful to the reviewers, Prof. Maurice Heemels and Prof. Patrick Boucher. Their constructive suggestions on the thesis are really appreciated for me. My thanks also go to Dr. Arben Cela for his commitment to take part in my jury.

Many thanks to Dr. Laurent Fesquet for his wide contribution in my research works, for his patience when explaining the micro-electronics aspects or the asynchronous paradigm, and his understanding on the control issues. I also thank him for accepting the invitation to be in my jury committee. Furthermore, I wish to extend my thanks to all those who have helped me in the ARAVIS project. In particular Hatem and Anne-Marie who contribute in merging my work in a low and high level respectively. I do not forget the numerous WP2 meetings too, with Carolina, Carlos, Yvain, Sylvain, Fabien, and all other people from TIMA, CEA-Leti, INRIA and STMicroelectronics. I also thank Brigitte from the LJK for her collaboration in the TATIE project.

I would like to express my deep regards to Dr. J-Fermi Guerrero Castellanos. I will not forget how I was well received during "mi estancia en Puebla", his availability at the university to discuss on valuable ideas and his friendly attitude the weekend to visit his beautiful country. Thanks also for agreeing to deem my work and for his courage when being present in my jury committee in video conference at three in the morning in Mexico. I do not forget his family too, especially his father Prof. Willy and his wife Esmeralda. I also thank Jonathan and the other Mexican students for their attempt to get my Spanish. Eventually, my special thanks to Betsa and her family for their warm welcome in Pachuca and their friendship.

I wish to thank Prof. Suzanne Lesecq for giving me the opportunity of a post-doc in order to test my control strategies on a practical nanometric SoC. I am grateful for her acceptance to take part in my jury committee too.

During this period, I have worked with many other colleagues for whom I have great regard and I wish to extend my warmest thanks to all those who have helped me from a professional and personal point of view. Especially to Luc for our great discussions, Émilie, Gabriel, Lara, Nicolas, Valentina, Alexandre, Alain, Kateřina, Florine, Federica, Jonathan, Riccardo, Wenjuan and other people from the NeCS team, Antoine, Lizeth, Charles, Corentin, Irfan and others from the GIPSA-Lab.

I am also thankful to the secretaries, librarians and any administrative and technical support, in particular Élodie, Myriam and Florence in INRIA, Marie-Thérèse, Patricia, Virginie and Olivier in GIPSA-Lab, for helping and assisting me in many different ways.

My special thanks to my family and friends who support me through difficult period and for providing to me their friendship, entertainment and attention. In particular, I would like to thank Sébastien and Cyril who encouraged me in this direction. I also thank my brother Nicolas for accommodating me when I had a party in Grenoble. Finally, I wish to thank Cécile, Patricia, Denise and Suzanne who hand-made the "pot de thèse" which was perfect.

Lastly, my deep lovely thanks go to my wife Cécile for her emotional supports and her continual and confident encouragement. I dedicate this work to her.

Table of contents

Summary of the thesis
(in French)

Chapter

A Problématiques de recherche

De nos jours, les systèmes embarqués sont de plus en plus présents et nous accompagnent au quotidien. Nous les retrouvons ainsi dans les téléphones, PDAs ou autres assistants électroniques, dans les véhicules et même dans les maisons avec la domotique florissante. Cette omniprésence entraîne une course à la miniaturisation. Les circuits électroniques sont également étudiés de manière à minimiser leur consommation et ainsi augmenter leur durée de vie. C'est dans ce contexte que s'inscrit cette thèse puisque nous avons travaillé à réduire à la fois ***i)*** la consommation d'énergie des circuits électroniques miniaturisés et ***ii)*** le coût de calcul du contrôleur dans les systèmes en boucle fermée. Ces deux points constituent les deux parties principales de la thèse.

Réduction de la consommation d'énergie

Il est possible de réduire la consommation d'énergie dans une puce électronique en diminuant la tension d'alimentation et/ou la fréquence d'horloge mais ceci a pour conséquence de diminuer la vitesse de fonctionnement du circuit en contrepartie. Une solution consiste toutefois à réduire la consommation lorsque les tâches à exécuter sont peu nombreuses, et augmenter la puissance dès qu'une charge de calcul plus importante doit être traitée. Une certaine stratégie de commande est ainsi nécessaire afin de gérer dynamiquement un tel compromis. Dans cette thèse, une architecture en boucle fermée permet cette approche. Deux actionneurs sont ainsi commandés afin d'alimenter le circuit en tension et en fréquence. Une commande prédictive rapide permet alors de calculer une consigne de vitesse à suivre de manière à ce que la consommation d'énergie soit minimisée tout en garantissant un bon fonctionnement du système. Ce principe est tout d'abord appliqué à un système monocœur puis le concept est ensuite généralisé à un système multicœur, où plusieurs processeurs fonctionnent ensemble sur un même domaine d'alimentation. Des résultats de simulation permettent finalement de mettre en évidence une forte réduction de la consommation d'énergie dans les deux cas.

Outre le compromis énergie-performance, un second aspect doit également être pris en compte dans les puces électroniques, notamment en ce qui concerne les circuits en technologie nanométrique. En effet, les performances après fabrication de telles puces ne peuvent plus être complètement prédites à cette échelle (alors que les mêmes circuits n'avaient aucun problème

dans les technologies précédentes). Ce phénomène, appelé variabilité du procédé de fabrication, est la principale cause de défauts dans les puces. Il résulte en une certaine incertitude qui est introduite dans le fonctionnement du circuit et les stratégies de commande implémentées devront donc prendre en compte ces dispersions technologiques. Un point important de la thèse est justement de proposer des lois de commande robustes, dont une notamment qui n'est basée sur aucun paramètre du système.

Le résumé en français de ces travaux est ensuite présenté dans le chapitre B.

Réduction du coût de calcul du contrôleur

La commande d'un système conduit souvent à ajouter une partie logicielle et/ou matérielle supplémentaire. Cela peut éventuellement poser problème dans les systèmes embarqués qui ont d'importantes contraintes, notamment en terme de ressources. Pour cette raison, il devient important de trouver des solutions afin de réduire le coût de calcul du contrôleur. C'est dans cet optique que les systèmes asynchrones font leur apparition. Contrairement à un système classique qui calcule la commande à chaque instant d'échantillonnage (constants et périodiques), l'échantillonnage asynchrone est quant à lui déclenché par des événements (lorsque le signal de mesure varie *suffisamment* de la consigne à suivre par exemple). Ce principe permet ainsi de réduire le nombre d'échantillons et, par conséquent, d'économiser des ressources CPU, tout en garantissant de bonnes performances. Néanmoins, peu de travaux existent dans ce domaine et des outils qui prennent en compte la nature événementielle du système à commander manquent. Dans cette thèse, nous nous intéressons au schéma asynchrone en proposons des contrôleurs PID dont le calcul de la commande est déclenché par franchissement de niveaux du signal de mesure. Une autre analyse propose ensuite des contrôleurs à retour d'état asynchrones où les événements sont basés sur une fonction de Lyapunov. Enfin, des résultats de simulation, mais surtout expérimentaux, permettent de valider l'intérêt de telles commandes et surtout le gain en coût de calcul.

Ces travaux sont résumés dans le chapitre C.

Chapter

B

Gestion du compromis énergie performance

Ce chapitre résume la première partie de la thèse concernant la gestion du compromis énergie-performance dans les circuits électroniques.

B.1 Context et motivations

L'évolution des techniques de fabrication des semi-conducteurs est telle que le nombre de transistors rapporté à la surface de la puce ne cesse d'augmenter. Les techniques actuelles permettent ainsi de commercialiser des circuits en technologie nanométrique, ce qui permet d'augmenter considérablement le nombre de fonctions électroniques sur une même puce. Cependant, certains effets physiques parasites - sans importance à plus grande échelle - deviennent prépondérants à l'échelle sub-micrométrique. Un de ces effets est la variabilité du procédé de fabrication qui est la principale cause de défauts et de délais dans les circuits intégrés. Ce phénomène introduit une incertitude dans le fonctionnement du circuit : alors qu'un circuit est prévu pour fonctionner à une certaine fréquence d'horloge, les performances de la puce fabriquée varient grandement. En d'autres termes, alors qu'une partie du circuit fonctionnera avec les performances attendues, une seconde partie fonctionnera moins bien, voire une troisième ne fonctionnera pas du tout. Ce comportement est illustré sur la figure 1. Finalement, les architectures doivent être complètement repensées et adaptées afin de pallier ces contraintes de dispersion technologique, en incluant notamment des boucles de commande qui deviennent indispensables. Ainsi, les différentes zones de la puce seront commandées afin d'exécuter les tâches les plus critiques sur les parties fonctionnant normalement et les tâches de fond sur les parties fonctionnant moins bien. Les parties ne fonctionnant pas seront quant à elles connues et non utilisées.

Une solution pour arriver à ce comportement est introduite dans le chapitre 1. Elle consiste entre autres à appliquer le paradigme "globalement asynchrone localement synchrone" (GALS). Ce concept revient à diviser la puce en plusieurs domaines de fréquence, où chaque domaine fonctionne de manière synchrone par rapport à une horloge qui lui est propre alors que les différents domaines sont asynchrones entre eux. Finalement, alors qu'une architecture classique est régit par le temps mis pour parcourir le plus long chemin électrique existant dans toute la puce (appelé le chemin critique), une architecture GALS va transposer le problème pour chaque

domaine. Les variations technologiques sont ainsi réduites. De plus, une gestion dynamique de la puissance est facilitée. Une telle méthode est primordiale pour espérer commander le compromis énergie-performance de la puce. L'approche consiste à ajuster ensemble la tension et la fréquence - en respectant certaines règles - afin de minimiser la consommation énergétique tout en garantissant que les tâches à traiter soient terminées à leur échéance temporelle. Différentes stratégies de commande sont ainsi proposées dans la première partie de cette thèse. Un système monocœur est tout d'abord étudié, l'activité d'un seul système électronique (un processeur par exemple) est alors régulée, puis une extension est ensuite proposée pour un système multicœur où plusieurs processeurs fonctionnent ensemble sur un même domaine d'alimentation.

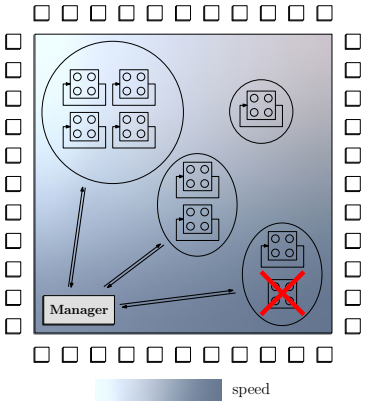


Figure 1: Phénomène de variabilité technologique dans les circuits en technologie nanométrique.

Ces travaux ont été réalisés dans le cadre du projet ARAVIS dont les objectifs sont décrits dans la sous-section 1.4.2.

B.2 Le système monocœur

Le chapitre 2 traite du compromis énergie-performance dans les systèmes monocœurs. Alors qu'un processeur fonctionne généralement avec une tension d'alimentation et une fréquence d'horloge nominales et constantes, il peut être intéressant de pouvoir faire varier dynamiquement ces deux grandeurs. Diminuer la puissance d'alimentation permet en effet d'abaisser la consommation d'énergie. Ceci n'est toutefois pas sans conséquence puisque la capacité de calcul du processeur est également réduite. Un compromis entre consommation et performance doit donc être établit. Une architecture en boucle fermée, telle que proposée sur la figure 2, permet une gestion dynamique de ce compromis. Deux actionneurs (un oscillateur en anneau et un Vdd-hopping) fournissent respectivement la fréquence d'horloge $f_{clk}(t)$ et la tension d'alimentation $V_{dd}(t)$ au système électronique (dénommé *device* sur le schéma). Ces deux actionneurs sont ensuite commandés par un contrôleur qui calcule les niveaux de fréquence $f_{level}(t)$ et de tension $V_{level}(t)$ les plus adaptés pour que le système consomme le moins possible, tout en garantissant que la charge de calcul à exécuter soit correctement traitée. Pour cela, la vitesse de calcul $\omega(t)$ du système électronique est mesurée afin de la comparer à une certaine consigne $ref(t)$. Cette architecture est présentée plus en détail dans la section 2.1.

En fait la référence $ref(t)$ est fournie par le système d'exploitation pour chaque tâche T_i à traiter. Elle se décompose en deux signaux, à savoir ***i)*** *la charge de calcul* qui correspond au nombre d'instructions à exécuter $\Omega_i(t)$ et ***ii)*** *l'échéance temporelle* ou deadline $\Delta_i(t)$. La laxité $\Lambda_i(t)$ est également intéressante. Elle correspond au temps restant avant la fin de la tâche et peut être obtenue facilement à partir de l'échéance temporelle.

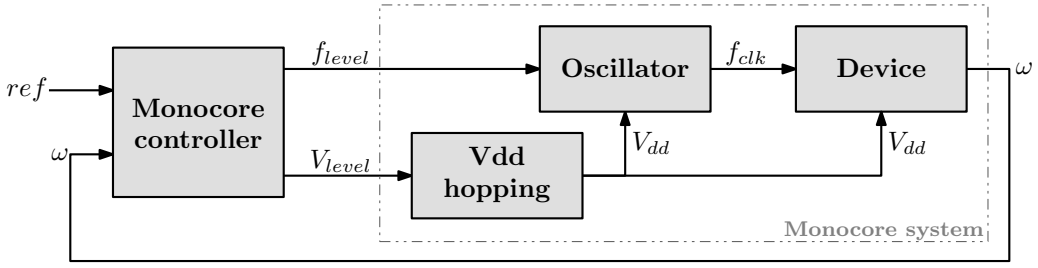


Figure 2: Architecture du système monocœur.

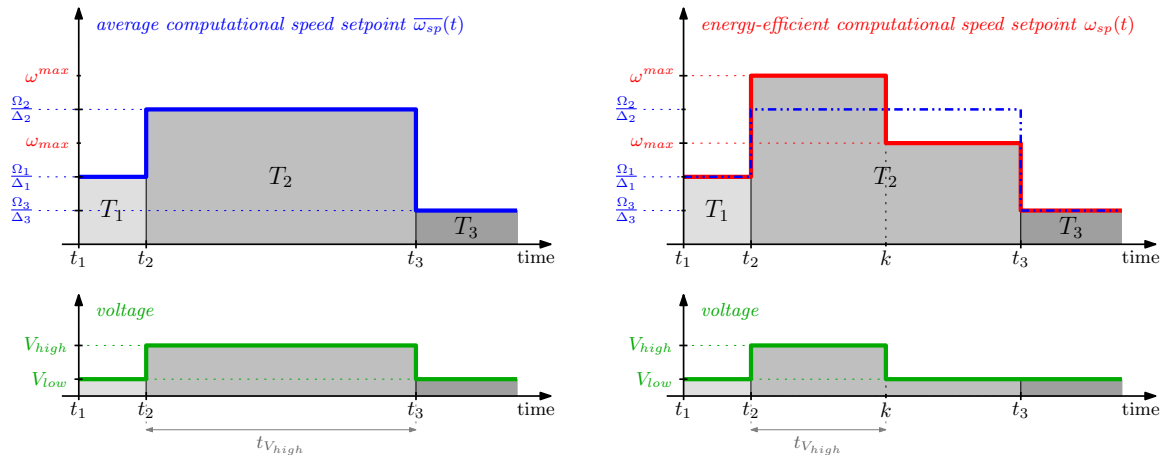
Dans un premier temps, le signal de commande envoyé au Vdd-hopping sera un signal discret à deux valeurs, correspondant à deux niveaux de tension possibles, i.e. V_{low} et V_{high}. Le signal délivré à l'oscillateur, quant à lui, sera une fréquence désirée $f(t)$ qui évolue de manière continue, et non un niveau de fréquence discret comme décrit plus haut.

B.2.1 Commande intuitive de la fréquence et du niveau de tension

Une stratégie de commande intuitive, proposée dans la section 2.2, consiste en un simple suivi de consigne. Les signaux de commande sont ainsi calculés de manière à ce que la vitesse de calcul du système $\omega(t)$ suive une consigne de vitesse moyenne $\overline{\omega_{sp}}(t) = {}^{\Omega_i(t)}/_{\Delta_i(t)}$, cette référence correspondant à la valeur nécessaire pour que la tâche se termine exactement à son échéance temporelle. Un bon suivi de consigne garantira ensuite que la tâche est correctement exécutée. Les niveaux de fréquence et de tension sont calculés indépendamment, puis finalement ajustés afin que le chemin critique soit respecté.

B.2.2 Commande de la vitesse de calcul

Une seconde approche est ensuite proposée dans la section 2.3 afin de réduire davantage la consommation énergétique. La méthode précédente permet déjà une réduction puisque la puissance d'alimentation est réduite pour traiter une faible charge de calcul. Cependant, une meilleur consigne peut diminuer davantage le fonctionnement à tension haute, la tension étant le paramètre le plus consommant en terme d'énergie du fait d'une relation (quasi) quadratique. La figure 3 illustre ce principe. Dans la version intuitive - figure B.3(a) - une tâche est exécutée tout le temps à tension haute lorsque la consigne moyenne est supérieure à la vitesse maximale possible à tension basse, notée ω_{max}. C'est notamment le cas pour la tâche T_2. En revanche, la seconde approche - figure B.3(b) - va permettre de la traiter d'une part à tension haute (puisqu'il faudra utiliser la tension haute de toute façon pour espérer terminer la tâche avant sa deadline) puis ensuite à tension basse, ce qui va réduire d'autant la consommation. Pour arriver à ce comportement, nous proposons que le système fonctionne à vitesse maximale lorsqu'il est alimenté à tension haute de manière à réduire le temps de ce fonctionnement pénalisant en terme de consommation. Ensuite, comme la tâche sera en avance par rapport à une consigne moyenne pendant la première phase du traitement (du début de l'exécution de la tâche t_2 jusqu'à un certain temps k sur la figure), elle pourra fonctionner ensuite à une vitesse inférieure à ω_{max} - et donc à tension faible - pour la fin du traitement (de l'instant k à t_3). Le temps de passage de tension haute à la tension basse - l'instant k - est donc nécessaire. Or il est difficilement déterminable a priori. C'est pourquoi nous proposons d'utiliser une boucle de commande afin de prédire le temps minimum de fonctionnement à tension haute. Cette seconde boucle va maintenant calculer une consigne de vitesse meilleure en terme d'énergie, notée $\omega_{sp}(t)$, à partir des références $\Omega_i(t)$ et $\Delta_i(t)$ fournies par le système d'exploitation pour chaque tâche à traiter.



(a) Construction intuitive

(b) Construction efficace en terme d'énergie

Figure 3: Comparaison de différentes manières d'obtenir une consigne de vitesse et son impact sur la consommation énergétique.

La présence d'échéance temporelle et d'horizon de temps pour compléter la tâche nous conduit naturellement vers une commande prédictive. Dans notre cas, l'horizon de temps est contractant, ce qui signifie que le temps restant diminue au cours de l'exécution d'une tâche, jusqu'à atteindre son échéance. Puis un nouvel horizon apparaît avec une nouvelle tâche à traiter. En outre, la commande prédictive est souvent lourde à mettre en œuvre du fait d'une optimisation requise. Cependant, l'approche que nous utilisons par la suite, appelée *commande prédictive rapide*, consiste à prendre en compte la structure du système à commander afin de simplifier la loi de commande. Deux architectures utilisant ce principe sont proposées dans la thèse. La première calcule une consigne de vitesse en bouclant le système avec la vitesse mesurée. Ainsi, la prédiction tient compte de ce qui a déjà été exécuté. La seconde boucle le système avec la consigne elle-même afin de réduire davantage la consommation d'énergie. Cependant, elle a l'inconvénient d'être plus complexe à implémenter. Seule la première approche sera détaillée dans ce résumé en français.

Commande prédictive rapide

Comme expliqué précédemment, le but de cette commande est de minimiser le temps de fonctionnement à tension haute. Pour cela, le contrôleur va calculer dynamiquement - pour chaque tâche T_i à traiter - si le système doit fonctionner à tension haute et fréquence maximale ou si la tension basse suffira pour terminer la tâche juste à sa deadline, compte tenu de ce qui a déjà été fait. Le problème d'optimisation résultant équivaut à minimiser le temps de fonctionnement à tension haute $t_{V_{high}}$ de manière à ce que le nombre d'instructions réellement exécutées (c'est à dire l'intégrale de la vitesse de calcul mesurée $\omega(t)$ sur le temps d'exécution de la tâche $\Delta_i(t)$) corresponde à la consigne $\Omega_i(t)$ donnée par le système d'exploitation. Ceci peut être exprimé mathématiquement comme suit :

$$\min \; t_{V_{high}} \qquad \text{s.t.} \quad \int_{\Delta_i(t)} \omega(t) \; dt = \Omega_i(t)$$

Bien que ce critère d'optimisation permette de résoudre le problème de prédiction, il est toutefois trop complexe pour être implanté sur un circuit électronique. Heureusement, la solution en boucle fermée conduit à une expression plus simple et plus rapide à calculer. Il suffit en fait

de connaître ***i)*** *la charge de calcul que le processeur doit traiter* et ***ii)*** *combien de temps on dispose pour le faire.* La vitesse nécessaire pour terminer la tâche exactement à son échéance - dénommée la vitesse prédite $\delta(t)$ par la suite - est ensuite facilement obtenue comme étant le rapport entre *ce qu'il faut faire* moins *ce qui a déjà été fait* (c'est à dire *ce qu'il reste à faire*) et *le temps qu'il reste avant d'atteindre la deadline de la tâche.* Ceci s'exprime ainsi :

$$\delta(t_{k+1}) = \frac{\Omega_i(t_k) - \sum_{t_i}^{t_k - t_i} \omega(t_k)}{\Lambda_i(t_k)}$$

La consigne de vitesse $\omega_{sp}(t)$ est ensuite déduite directement à partir de la valeur de $\delta(t)$. Si la prédiction indique que la vitesse requise pour terminer la tâche est supérieure à la vitesse maximale à tension basse, c'est à dire si $\delta(t) > \omega_{max}$, alors il est nécessaire de fonctionner à tension haute et à fréquence maximale, c'est à dire à vitesse maximale ω^{max}. En revanche, dès que $\delta(t)$ devient inférieure à ω_{max}, alors le système peut passer à tension basse et ce niveau de tension sera suffisant pour terminer la tâche (si les consignes $\Omega_i(t)$ et/ou $\Delta_i(t)$ ne changent pas en cours d'exécution de la tâche). La relation suivante résume ce principe :

$$\omega_{sp}(t_k) = \begin{cases} \omega^{max} & \text{if} \quad \delta(t_{k+1}) > \omega_{max} \\ \delta(t_{k+1}) & \text{otherwise} \end{cases}$$

Finalement, une consigne de vitesse faible consommation est ainsi calculée. De plus, la tâche à traiter sera correctement exécutée puisque, si le système est plus lent que prévu, alors la consigne sera dynamiquement ajustée du fait de la contre réaction. De même, les retards éventuels dans les changements de tension et/ou de fréquence sont déjà pris en compte et le calcul des variables de commande est donc simplifié.

Notons que les vitesses maximales ω_{max} et ω^{max} sont directement mesurées dans un premier temps. Elles sont ainsi simplement obtenues en cadençant le système électronique à fréquence maximale, lorsque ce dernier est alimenté respectivement à tension basse et à tension haute.

Commande de la fréquence et du niveau de tension

La consigne de vitesse de calcul $\omega_{sp}(t)$ fournie par la commande prédictive permet de réduire le temps de fonctionnement à tension haute. Cette consigne étant différente de celle utilisée dans la première stratégie, à savoir la consigne moyenne $\overline{\omega_{sp}}(t)$, le contrôle de la fréquence et du niveau de tension doit par conséquent être modifié. Un simple correcteur intégral suffit pour le calcul de la fréquence. On ajoute ensuite un système d'anti-windup afin de prendre en compte les limitations de l'oscillateur, ce qui conduit à :

$$\begin{aligned} \varepsilon(t_k) &= \omega_{sp}(t_k) - \omega(t_k) \\ \sigma(t_k) &= \frac{\omega(t_k)}{f(t_{k-1})} \\ f(t_k) &= f(t_{k-1}) + T_s \cdot \frac{1}{\sigma(t_k)} \cdot K \cdot \varepsilon(t_k) - T_s \cdot K_a \cdot \Big(f(t_{k-1}) - f_{sat}(t_{k-1}) \Big) \end{aligned}$$

Concernant le niveau de tension, le système doit fonctionner à fréquence maximale lorsqu'il est alimenté avec la tension haute. Une fonction hystéresis, représentée sur la figure 4, est alors utilisée afin de faire la concordance entre les deux variables. Ainsi, le niveau de tension haut est automatiquement appliqué dès que la fréquence est supérieure à la fréquence maximale possible à tension haute, i.e. $F_{V_{low}max}$, c'est à dire :

$$V_{level}(t_k) = \begin{cases} V_{level\_high} & \text{if} \quad f(t_k) \geq F_{V_{low}max} \\ V_{level\_low} & \text{otherwise} \end{cases}$$

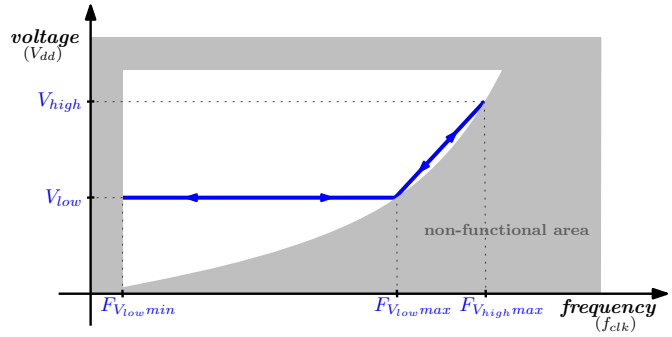


Figure 4: Hystéresis utilisée pour ajuster le niveau de tension en fonction de la fréquence calculée, afin de minimiser le temps de fonctionnement à tension haute.

B.2.3 Commande complètement discrète

Afin de coller complètement avec l'architecture introduite plus haut sur la figure 2, une dernière loi de commande est développée dans la section 2.4. Elle permet d'étendre la précédente stratégie à des niveaux discrets de fréquence (alors qu'elle variait continuement avant) et de généraliser le principe à un nombre plus important de niveaux de tension (alors que seul deux niveaux étaient possible avant). Désormais, M niveaux de tension et N niveaux de fréquence sont donc possibles. La généralisation est assez simple puisque le problème peut être ramené aux deux niveaux qui encadrent la vitesse de calcul moyenne. Il faut ensuite minimiser le temps de fonctionnement du niveau supérieur tout en garantissant que la tâche pourra être correctement terminée avec le niveau inférieur. Le calcul des variables de commande $f_{level}(t)$ et $V_{level}(t)$ est également simplifié puisqu'ils sont maintenant directement déduits de la valeur de la vitesse prédite $\delta(t)$. De plus, la consigne de vitesse à suivre $\omega_{sp}(t)$ n'est plus nécessaire, ce qui simplifie grandement la loi de commande.

Du fait de ce fonctionnement discret, il est possible qu'une tâche soit terminée bien avant son échéance si le seul niveau de fréquence possible est beaucoup plus important que celui requis. C'est pourquoi nous avons ajouté une décision supplémentaire, basée sur le principe dit du *clock-gating*. Il est ainsi possible de "désactiver" l'horloge du circuit lorsque la tâche est terminée afin de réduire davantage la consommation.

Une autre amélioration est d'estimer dynamiquement les vitesses maximales possibles pour chaque niveau de tension (précédemment notée ω_{max} et ω^{max} dans le cas où deux niveaux de tension étaient possibles). Leur valeur est en effet nécessaire dans la loi de commande pour prédire l'instant de changement du niveau de tension. Nous avons ainsi proposé d'utiliser une moyenne pondérée de la vitesse mesurée, ce qui peut se résumer par :

$$\tilde{\omega}_m(t_k) = (1-\nu)\cdot\tilde{\omega}_m(t_{k-1}) + \nu\cdot\omega(t_k)$$

où $\tilde{\omega}_m(t)$ est l'estimation de la vitesse $\omega_m(t)$, et $0 \leq \nu \leq 1$ est la valeur de pondération. Ce dernier paramètre a notamment besoin d'être borné afin de ne pas rendre le système instable. Finalement, cette stratégie est très robuste aux dispersions technologiques présentes dans les circuits nanométriques puisque la loi de commande ne repose sur aucun paramètre du système.

Pour terminer sur la gestion du compromis énergie-performance dans un système monocœur, quelques simplifications sont suggérées dans la section 2.5 et une rapide analyse de stabilité du système est décrite dans la section 2.6.

B.3 Le système multicœur

La gestion du compromis énergie-performance de plusieurs processeurs fonctionnant ensemble sur une même puce est également étudiée, dans le chapitre 3 de cette thèse. L'architecture multicœur - représentée sur la figure 5 - est en fait une généralisation de l'architecture monocœur précédente (figure 2). Le principe est quasiment identique. Le contrôleur calcule dynamiquement les niveaux de fréquence $f_{level}(t)$ et de tension $V_{level}(t)$ à envoyer aux actionneurs. X systèmes doivent maintenant être commandés, ce qui signifie autant de références $ref^X(t)$ et de signaux de mesure $\omega^X(t)$ que de circuits. Cependant, la commande des différents systèmes ne peut se faire de manière indépendante puisqu'ils fonctionnent tous avec la même tension d'alimentation et la même fréquence d'horloge. Une certaine liberté est toutefois possible grâce aux ratios de fréquence $\rho^X(t)$ qui autorisent un système à être cadencé moins vite que l'horloge globale. Cette architecture est plus détaillée dans la section 3.1. Plusieurs extensions des différentes stratégies de commande monocœur sont ensuite développées dans la section 3.2. Finalement, une seconde architecture multicœur est proposée dans la section 3.3, où les différents processeurs fonctionnent cette fois avec leur propre fréquence d'horloge. Ainsi, X oscillateurs cadencent désormais le circuit, quand un seul Vdd-hopping continue d'alimenter l'ensemble.

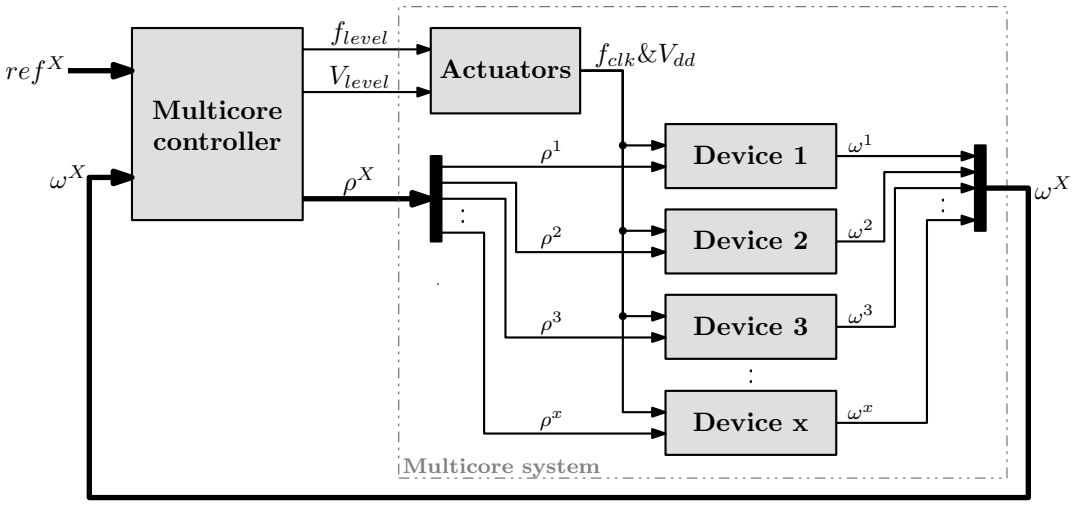


Figure 5: Architecture du système multicœur.

Notations:

- $\rho^X(t)$ (indice en majuscule) indique qu'il existe X signaux $\rho(t)$, à savoir un pour chaque circuit de la puce.
- $\rho^d(t)$ (indice en minuscule) est utilisé pour référer au signal $\rho(t)$ correspondant au circuit d, où $d \in \{1, 2, \ldots, X\}$.
- $\rho^x(t)$ est utilisé pour le signal $\rho(t)$ du dernier circuit.

B.4 Résultats de simulation

Afin de valider nos solutions pour une gestion dynamique du compromis énergie-performance, différents tests de simulation sont réalisés dans le chapitre 4. Ils sont exécutés sous *Matlab/Simulink*. Les résultats montrent que l'utilisation de la commande prédictive rapide permet de réduire énormément la consommation d'énergie, que ce soit pour les systèmes monocœurs

ou multicœurs, sans pour autant augmenter le coût de calcul du contrôleur. De plus, la loi de commande s'adapte automatiquement lorsque la consigne donnée par le système d'exploitation pour chaque tâche à traiter change, même si cette consigne est modifiée pendant l'exécution de la tâche. En outre, une commande complètement discrète permet de réduire davantage la consommation et le coût de calcul, tout en assurant une forte robustesse aux dispersions technologiques qui sont présentes dans les circuits en technologies nanométriques. Ce dernier point était un objectif important à atteindre dans cette thèse.

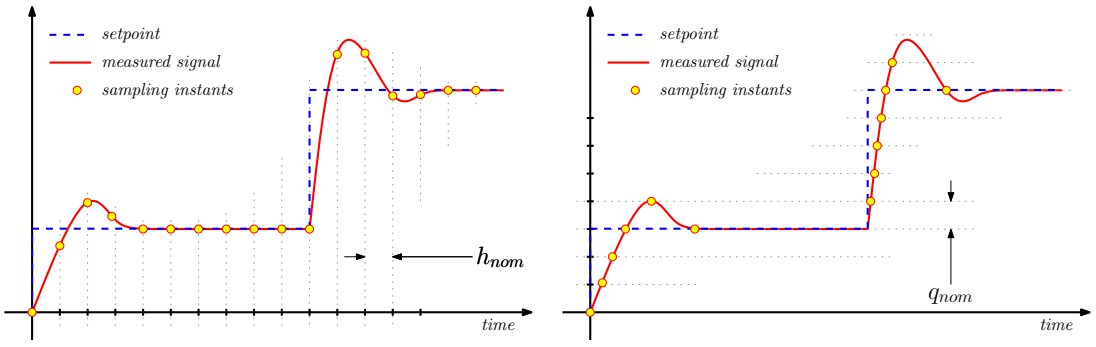
Chapter

C Commande déclenchée par événements

Ce chapitre résume la seconde partie de la thèse traitant de la commande déclenchée par événements.

C.1 Context et motivations

L'approche classique pour calculer une loi de commande en temps discret consiste à échantillonner le système uniformément en temps, avec une période d'échantillonnage h_{nom} constante. La loi de commande est alors calculée et mise à jour périodiquement, c'est à dire à chaque instant $t_k = k \cdot h_{nom}$. Ce principe, illustré sur la figure C.1(a), correspond au cas synchrone dans le sens où tous les échantillons sont synchrones. D'un autre côté, de récents travaux s'intéressent à un échantillonnage asynchrone où les instants d'échantillonnage sont déclenchés par des événements, comme par exemple lorsque le signal de mesure franchit un certain niveau d'amplitude $q_j = j \cdot q_{nom}$. Cette seconde approche, représentée sur la figure C.1(b), est introduite dans le chapitre 5.



(a) Échantillonnage classique en temps

(b) Échantillonnage en amplitude (déclenché par des événements)

Figure 1: Alors que le système à commandé est échantillonné uniformément en temps dans le cas classique, le système est échantillonné en amplitude dans le cas asynchrone.

Une technique asynchrone permet théoriquement de réduire le nombre d'échantillons, et par conséquent d'économiser des ressources CPU du contrôleur. Ce nouveau schéma peut toutefois être compliqué à mettre en œuvre, en particulier parce que le paradigme synchrone est bien ancré dans les mœurs et qu'il est donc difficile de s'en affranchir. Les architectures de traitement numérique doivent être complètement repensées et adaptées. C'est notamment un des objectifs du projet TATIE dont les objectifs sont décrits dans la sous-section 5.2.3. Dans cette thèse nous nous intéressons plus particulièrement aux méthodes de commande. Dans un premier temps, les contrôleurs *proportionnel intégral dérivé*, très utilisés dans l'industrie, sont étudiés. Un contrôleur plus général est étudié ensuite avec un retour d'état dynamique asynchrone.

C.2 Commande PID et détection d'événement par franchissement de niveau

Le contrôleur PID est une architecture très répandue, notamment dans l'industrie, c'est pourquoi nous avons cherché dans un premier temps à obtenir une version asynchrone de ce dernier. Cette démarche est détaillée dans le chapitre 6. Un tel contrôleur calcule différentes composantes à partir de l'erreur de mesure, à savoir les termes *proportionnel*, *intégral* et *dérivé*, puis somme ces composantes afin de corriger le système. Cependant, alors qu'un contrôleur classique met à jour le signal de commande à chaque instant t_k, un contrôleur dont l'échantillonnage déclenché par des événements ne va calculer la commande que lorsque l'erreur de mesure franchit un certain niveau d'amplitude. *Karl-Erik Årzén* a proposé un tel mécanisme pour une commande PID. Le principe, décrit dans l'article [10], se compose de deux parties : ***i)*** *un détecteur d'événements* utilisé pour la détection de franchissement de niveau et ***ii)*** *le contrôleur* à proprement parlé qui met à jour le signal de commande. Cette architecture est représentée sur la figure 2. Alors que la première partie est échantillonnée en temps avec la période d'échantillonnage h_{nom} (la même que celle utilisée pour le contrôle classique), le calcul de la commande est quant à lui déclenché par des événements. Des requêtes sont ainsi générées lorsque le signal de commande doit être mis à jour. Les intervalles d'échantillonnage du contrôleur asynchrone, notés $h(\cdot)$, ne sont donc plus constants mais sont désormais définis par le temps séparant deux requêtes. L'instant courant d'échantillonnage asynchrone est dénoté t_a par la suite.

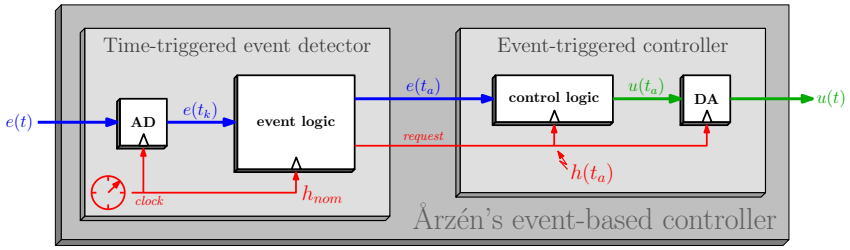


Figure 2: Architecture du contrôleur asynchrone proposé par Årzén.

Différentes façons de générer une requête existent. Dans l'approche initiale proposée par Årzén, un événement est déclenché ***i)*** lorsque l'erreur relative franchit un certain seuil, i.e. $abs\big(e(t_a) - e(t_{a-1})\big) > q_{nom}$, ou ***ii)*** si la période d'échantillonnage maximale est atteinte, i.e. $h(t_a) \geq h_{max}$. Le fonctionnement est présenté dans les sous-sections 6.2.1 et 6.2.2. La dernière condition est ajoutée afin de respecter le théorème d'échantillonnage de Nyquist-Shannon. Elle n'est toutefois pas nécessaire du fait du fonctionnement événementiel du nouveau contrôleur. Nous proposons donc de l'enlever.

Auparavant, nous proposons d'améliorer le contrôleur d'Årzén, concernant plus particulièrement la méthode de discrétisation utilisée pour obtenir le terme intégral de la commande. La méthode d'Euler explicite permet de pré-calculer le terme intégral pour l'échantillon suivant. Cela nécessite néanmoins de connaître a priori la période d'échantillonnage, ce qui n'est pas possible dans le cas asynchrone (car contrairement au cas classique, le paramètre $h(t)$ varie). Il faut donc préférer la méthode d'Euler implicite qui calcule le terme intégral en fonction de ce qui s'est passé avant.

C.2.1 Commande sans limite de sécurité

Comme expliqué plus haut, nous proposons d'enlever la condition $h(t_a) \geq h_{max}$ introduite par Årzén afin de simplifier le schéma asynchrone. Au final, une requête est donc seulement générée lorsque les performances du système sont dégradées, c'est à dire lorsque l'erreur est importante. Le fait d'enlever la condition nécessite néanmoins de prendre quelques précaution car l'intervalle entre deux échantillons peut maintenant augmenter indéfiniment en absence d'événement (lors du régime permanent). D'un autre côté, l'erreur peut également devenir importante lorsque la consigne change. Or, le terme intégral de la commande est fonction du produit entre ces deux grandeurs, i.e. $h(\cdot)e(\cdot)$ (appelé le gain intégral par la suite, noté $he(\cdot)$), ce qui peut conduire à un gain énorme et par conséquent une trop forte compensation. En d'autres termes, des effets indésirables de dépassements importants de la consigne vont se produire. En observant de plus près le régime permanent, nous avons constaté qu'il peut en fait être découpé en deux phases, comme illustré sur la figure 3. Durand la première, le système est stable et l'erreur est donc inférieure au seuil de détection (sinon un événement est généré et le régime permanent n'est alors pas atteint). L'intervalle augmente pendant ce temps. Ensuite, dès que la consigne varie l'erreur devient importante mais, en revanche, cela dure seulement le temps de la détection d'un nouvel événement, à savoir h_{nom}.

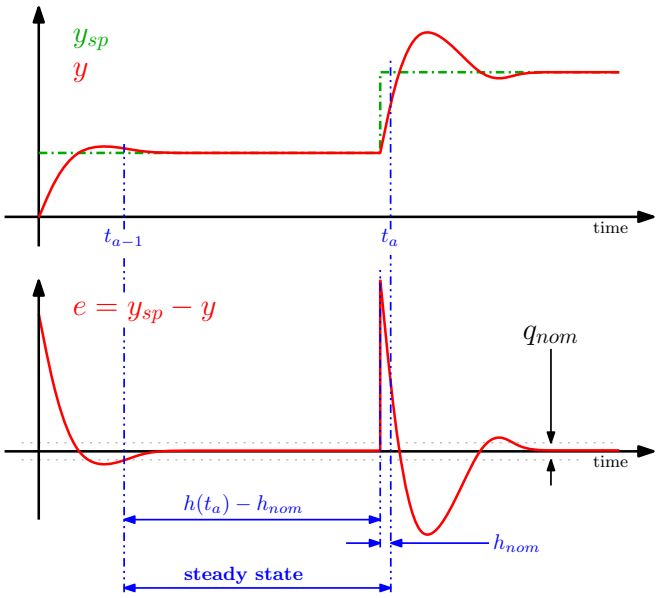


Figure 3: Décomposition du régime permanent dans le cas d'une commande déclenchée par événement.

Au final, la somme des deux phases conduit à borner le gain intégral tel que :

$$he(t_a) \leq \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$$

Dans cette expression, seul l'un ou l'autre des deux paramètres est grand dans le produit entre $h(\cdot)$ et $e(\cdot)$ et, de ce fait, l'erreur du système ne sera pas sur-compensée. Ceci est expliqué plus en détail dans la sous-section 6.2.3. A partir de cette analyse, nous proposons ensuite de nouvelles stratégies de commande PID asynchrone. Les six algorithmes peuvent être résumés comme suit. Il est bon de noter que l'approche utilisée est assez similaire à celle utilisée dans un mécanisme anti-windup, où l'erreur due à la saturation des actionneurs est compensée.

1. *Algorithme seulement sans la condition de limite de sécurité :*
   Comme expliqué plus haut, si la condition est enlevée sans aucune précaution, alors des dépassements importants de la consigne vont apparaître après un long régime permanent.

2. *Algorithme avec saturation du gain intégral :*
   Nous proposons de borner la gain intégral afin de réduire son impact dans la loi de commande, puisqu'en fait, seul $h(\cdot)$ ou $e(\cdot)$ est grand dans le produit $he(\cdot)$. A partir de cette constatation, le gain intégral devient $he_{sat}(t_a) = \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$.

3. *Algorithme avec facteur d'oubli exponentiel sur la période d'échantillonnage :*
   Puisque l'intervalle d'échantillonnage $h(\cdot)$ n'est plus limité, nous proposons d'appliquer un facteur d'oubli afin de limiter son impact, notamment lorsque le régime permanent dure longtemps. Ainsi, sa valeur devient $h_{exp}(t_a) = h(t_a) \cdot exp\Big(\alpha \cdot \big(h_{nom} - h(t_a)\big)\Big)$.

4. *Algorithme hybride :*
   Nous proposons ensuite d'appliquer le facteur d'oubli sur la période d'échantillonnage et la saturation du gain intégral, mélangeant ainsi les deux précédentes stratégies. Cela conduit à $he_{hybrid}(t_a) = \big(h_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$.

Les algorithmes proposés permettent de réduire le gain intégral dans la loi de commande. Cependant, certains peuvent être difficiles à implémenter sur un système disposant de faible ressources, notamment en ce qui concerne la fonction exponentielle $h_{exp}(\cdot)$. Pour cette raison, nous proposons d'améliorer les algorithmes concernés, en remplaçant l'exponentielle par une fonction linéaire définie par morceaux moins coûteuse en terme de calcul.

5. *Algorithme avec facteur d'oubli exponentiel sur la période d'échantillonnage et faible coût d'implémentation :*
   Dans cette stratégie, le facteur d'oubli exponentiel devient $he^i_{exp}(t_a)$.

6. *Algorithme hybride et faible coût d'implémentation :*
   De même, le facteur d'oubli exponentiel faible coût est remplacé dans l'algorithme hybride, ce qui amène le gain intégral à $he^i_{hybrid}(t_a) = \big(h^i_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$.

Finalement, il est proposé dans cette thèse de changer la condition de détection de franchissement de niveau. L'erreur absolue plutôt que relative est ainsi préférée, i.e. $abs\big(e(t_a)\big) > q_{nom}$. De cette manière, le nombre d'échantillons va inévitablement augmenter pendant les périodes de transition mais, en revanche, il sera certain que le régime permanent est atteint lorsque l'erreur est inférieure à q_{nom}, ce qui n'était pas forcément vérifié avant.

C.2.2 Commande avec condition d'échantillonnage minimum

Dans le but de réduire davantage le nombre d'échantillons, une condition d'échantillonnage minimale est introduite, i.e. $h(t_a) \geq h_{min}$. Ainsi un événement ne sera possible que si le temps écoulé depuis la dernière mise à jour du signal de commande est supérieur à la valeur h_{min}. Ce principe est détaillé dans la sous-section 6.2.4.

C.2.3 Commande avec échantillons supplémentaires

Enfin, dans le but de réduire l'erreur statique qui est inévitable dans le schéma asynchrone, nous proposons d'ajouter quelques échantillons supplémentaires après le régime transitoire. Ainsi, la marge d'erreur sera diminuée. Ces échantillons sont espacés d'une période h_{extra} et on arrête d'ajouter des échantillons dès que l'erreur passe en dessous d'un nouveau seuil de détection q_{min}. Ceci est développé plus en détail dans la sous-section 6.2.5.

C.2.4 Résultats de simulation

Les différentes stratégies de commande PID proposées conduisent à une forte diminution du nombre d'échantillons nécessaires pour suivre une certaine consigne. Des résultats de simulation montrent dans la section 6.6 qu'une réduction de plus de 80 % est atteinte alors que les performances du système sont toujours assurées. Cependant, ces résultats ne sont basés sur aucune preuve réelle de stabilité, comme expliqué dans la section 6.4.

C.3 Retour d'état et échantillonnage déclenché par une fonction de Lyapunov

Un contrôleur à retour d'état dynamique est également très pratique. Il consiste à multiplier la sortie du système avec un certain gain, et d'appliquer ensuite le résultat comme la nouvelle entrée du système. De plus, de nombreux outils méthodologiques existent pour ce type de commande, notamment pour prouver la stabilité en utilisant la théorie de Lyapunov (voir la sous-section 7.2.1 pour plus d'informations sur ce sujet). Pour cette raison, nous avons choisi un retour d'état basé sur un échantillonnage déclenché par événement, afin de prouver qu'un système peut être stable même si la commande n'est pas mise à jour pendant un long moment. Dans ce cas, les requêtes ne sont plus générées par franchissement de niveau du signal de mesure. Nous allons plutôt utiliser une fonction de Lyapunov qui va déterminer les instants d'échantillonnage. Ces travaux sont développés dans le chapitre 7.

C.3.1 Échantillonnage déclenché sur fonction de Lyapunov

Le principe a été introduit par *Manel Velasco et al.* dans l'article [68]. L'idée est de déclencher un événement lorsque le système atteint un certain niveau d'énergie (défini par une fonction de Lyapunov donnée). Le comportement attendu est représenté sur la figure 4 pour un système à deux états, i.e. x_1 et x_2. La trajectoire du système franchit ainsi plusieurs ellipses dans le plan (x_1, x_2), qui définissent une région avec un niveau d'énergie constant de plus en plus proche du point d'équilibre. La mise à jour de la commande est effectuée seulement lorsqu'une courbe est franchie de l'extérieur vers l'intérieur, ce qui est un comportement stable d'un point de vu de Lyapunov puisque l'énergie diminue. Pour arriver à ce comportement, l'échantillonnage basé sur Lyapunov est déclenché lorsque :

$$V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big)$$

où V est une fonction candidate de Lyapunov. Le *facteur de gain énergétique* η défini ainsi la fréquence des événements, puisqu'une petite valeur va conduire à de large intervalles entre deux échantillons. Par construction, la séquence d'échantillonnage est stable selon Lyapunov si $0 < \eta < 1$. Cependant, la stabilité du système n'est pas encore assurée. En effet, si la trajectoire du système augmente avant d'avoir franchi le niveau d'énergie suivant, alors aucun événement ne sera généré et le système va diverger. Il faut donc garantir une séquence infinie d'échantillons. Cela est également proposé par *Manel Velasco et al.* mais conduit à un algorithme très lourd

qui nécessite d'être exécuté hors-ligne. Cet algorithme permet néanmoins d'assurer la stabilité en calculant le facteur de gain énergétique minimum η^*. Ce paramètre garantit alors que le prochain niveau sera franchit, en posant $\eta^* < \eta < 1$. L'ensemble de ces travaux et rappelé dans la sous-section 7.2.2.

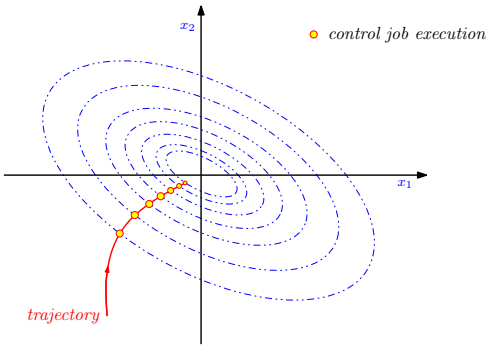


Figure 4: Comportement du mécanisme d'échantillonnage déclenché par une fonction de Lyapunov pour un système à deux états.

Une première amélioration est proposée dans cette thèse afin d'éviter que le système ne diverge, lorsque la consigne change par exemple ou lorsqu'une perturbation apparaît. Cela se traduit par le fait que la fonction de Lyapunov augmente et rien n'était prévu pour traiter ce cas dans les travaux existants. Nous ajoutons donc la condition suivante :

$$\begin{aligned} &\Delta V(t_a) > 0 \\ &\text{avec} \quad \Delta V(t_a) = V\big(x(t_a)\big) - V\big(x(t_{a-1})\big) \end{aligned}$$

C.3.2 Simplification du mécanisme d'échantillonnage

Dans un soucis d'alléger la loi de commande, nous proposons de relaxer la restriction faite sur η pour assurer une séquence d'échantillonnage infinie, i.e. $\eta^* < \eta < 1$. En fait, comme expliqué plus haut, l'algorithme utilisé pour calculer le facteur de gain énergétique minimum η^* est très lourd à mettre en œuvre en terme de calculs. Ce paramètre dépend du système à commander et de la fonction de Lyapunov choisie. De plus, la méthode proposée est très conservative puisque la valeur calculée est appliquée durant tout le fonctionnement du système alors que, en fait, la séquence est stable pour un grand nombre de gain inférieurs à η^*. C'est pourquoi nous proposons de faire varier dynamiquement le facteur de gain énergétique, qui devient donc $\eta(t)$. La condition de déclenchement d'un événement est par conséquent modifiée :

$$V\big(x(t_a)\big) = \eta(t_{a-1}) \cdot V\big(x(t_{a-1})\big)$$

avec $0 < \eta(\cdot) < 1$ par construction. Quatre nouveaux algorithmes sont finalement proposés dans la sous-section 7.2.3 :

1. *Diminution légère/augmentation brutale du facteur de gain énergétique :*
   Le premier algorithme consiste à diminuer $\eta(t)$ plus que permis initialement dans la condition $\eta^* < \eta < 1$. Cependant, une telle relaxation peut rendre le système instable. Dès que cela se produira, le facteur de gain énergétique est aussitôt remis à sa valeur stable η^*.

2. *Amélioration pour un fonctionnement complètement en-ligne :*
   La valeur du facteur de gain énergétique minimum est toujours nécessaire dans le premier algorithme, ce qui implique d'exécuter une partie de code hors-ligne pour obtenir η^*. Afin

de s'affranchir de cela, nous proposons de remplacer sa valeur par la borne supérieure dans la condition de stabilité, à savoir $1 - \varepsilon$. Le facteur de gain pourra ainsi être réduit, et lorsque le système deviendra instable alors il retournera à cette valeur haute. Cela va inévitablement augmenter le nombre d'échantillons mais a l'avantage d'être complètement exécutable en-ligne.

3. *Diminution/augmentation légère du facteur de gain énergétique :*
   Plutôt que de remettre le facteur de gain énergétique à une valeur très haute lorsque le système devient instable, nous proposons de l'augmenter doucement. Le système devrait être stabilisé moins vite mais le nombre d'échantillons devrait en revanche diminuer.

4. *Variation plus formelle du facteur de gain énergétique :*
   En se basant sur les précédents algorithmes, nous proposons finalement de formaliser notre idée de relaxation en utilisant le facteur de gain énergétique comme un état interne du contrôleur, i.e. $\dot{\eta}(t)$. Après discrétisation, $\eta(t_a)$ est finalement ajusté en fonction de la variation de la fonction de Lyapunov $\Delta V(t_a)$ - qui a l'avantage d'être déjà calculé pour la détection des événements - et de la période d'échantillonnage $h(t_a)$.

Les trois derniers algorithmes peuvent être exécutés complètement en-ligne. Ils autorisent le système à devenir instable mais réagiront en conséquence lorsque cela se produira.

C.3.3 Commande avec condition d'échantillonnage minimum

Comme précédemment avec le cas du contrôleur PID, nous ajoutons une condition afin qu'un événement ne soit possible que si un certain temps s'est écoulé depuis la dernière mise à jour du signal de commande, i.e. $h(t_a) \geq h_{min}$. Ceci est détaillé dans la sous-section 7.2.4.

C.3.4 Résultats de simulation

Des résultats de simulation sur un système à double intégrateur montrent dans la section 7.5 que le nombre d'échantillons est grandement réduit sans pour autant sacrifier les performances du système.

C.4 Résultats expérimentaux

Les différentes stratégies de commande asynchrone que nous proposons ont été testées en simulation. Elles ont ainsi permis de valider nos intuitions de départ. Nous proposons ensuite de les tester sur un système réel afin de prendre en compte les erreurs qui peuvent exister (erreurs de modélisation, erreurs dues aux bruits de mesure, etc...). Le banc de test utilisé est un pendule inversé de la société FeedBack Incorporated. Il est constitué d'un chariot sur lequel est placé le pendule inversé. Un moteur permet alors de déplacer le chariot dans le but de stabiliser l'ensemble. Ce système est très instable puisque les bras du pendule retombent dès que la commande n'est plus appliquée. Une commande asynchrone, qui laisse la possibilité au contrôleur de ne pas mettre à jour le signal de commande pendant un long moment, peut donc être risqué. Toutefois, les résultats obtenus montrent le contraire puisque nous obtenons un système qui fonctionne aussi bien qu'avec la méthode classique - sinon mieux - avec un nombre d'échantillons est très réduit. Ces résultats sont présentés dans le chapitre 8.

General introduction

Research problematic and proposed solutions

This document synthesizes the three-year PhD work realized under the supervision of Daniel SIMON and Nicolas MARCHAND, in both the INRIA Rhône-Alpes[1] research center and the control department of GIPSA-lab[2], in Grenoble, France. Actually, the demand of low-power electronic devices in all embedded and miniaturized applications becomes more and more important and encourages companies to develop new versions of the existing components. Our work is in line with these requirements since we propose different solutions to ***i)*** reduce the energy consumption in electronic devices and ***ii)*** soften the computational cost of the controller in closed-loop architectures.

Reduction of the system energy consumption

An electronic device with a variable processing power is known to be a promising solution for energy savings. Classical systems usually run with nominal and constant power supply (in voltage and frequency), but it might be interesting to make these parameters dynamically varying in order to control the energy consumption of the system. However, decreasing the power of the device leads to run more slowly in return, which directly impacts the computational performance. This is clearly an energy-performance tradeoff which needs to be managed. A solution is to save energy when the computational load of the processor becomes low and increase the processing power back as soon as an important computation has to be done. As a result, a control strategy is required to monitor the activity of the system and decide the control variables. In this thesis, we propose a closed-loop architecture which allows such a dynamical scheme. Two actuators respectively provide the supply voltage and the clock frequency to the circuit. A fast predictive control law then consists in providing an energy-efficient setpoint to track, while guaranteeing that the computational load to treat is correctly executed. This is applied to a monocore system - where a single processor runs on the chip - in a seminal analysis, before extending the principle to a multicore system where several devices work together with the same power supply. Finally, the proposed approaches clearly give an important reduction of the energy consumption in both schemes.

Another aspect also occurs in electronic chips with the upcoming of nanometric technologies. In fact, the system performance after fabrication is not fully predictable anymore since new

[1] INRIA Rhône-Alpes, one of the eight research centers of the French national institute for research in computer science and control: `http://www.inrialpes.fr/`

[2] GIPSA-lab, laboratory in image, speech, signal and control of Grenoble: `http://www.gipsa-lab.inpg.fr/`

problems - which did not influence the circuit at a higher scale - become very annoying at a sub-micrometric scale. The process variability is notably one of the leading causes for chip failures and delayed schedules. This phenomenon introduces an uncertainty about how the system will perform. Although a circuit is designed to run at a nominal clock frequency, the fabricated implementation may vary far from this expected performance. As a result, the control strategies have to be robust to this inherent dispersion. Eventually, our last contribution is highly robust to tackle variability since it is not based on any parameters of the system.

Reduction of the control computational cost

Controlling a device implies to add some extra hardware and software materials. This could be a problem in embedded systems with high constraints, such as low allocated resources for instance. For this reason, it becomes essential to find some solutions to reduce the control computational cost. In this sense, many reasons are motivating event-driven system and, in particular, because more and more systems with asynchronous needs are encountered. Although a time-triggered framework simplifies the design and analysis in sampling the system uniformly in time, it results in a conservative usage of resources since the control law is computed and updated every periodic time instants, that is at the same rate regardless it is really required or not. On the other hand, event-based sampling (also called asynchronous) allows resource-aware implementations of the control law enforcing some events only when the measured signal *sufficiently* changes. This scheme theoretically enables to reduce the number of samples and, consequently, to save computations in the control law. This behavior has hence to be considered in embedded low-power systems where a significant power consumption reduction would be very appreciated. However, this original approach requires to develop new methodological tools which take into account the even-driven nature of the system to control.

Actually, the different scientific communities begin to be very active in this field. In control theory for instance, *Karl-Erik Årzén* proposed in 1999 an event-based PID controller. The basic setup consists in two parts: a time-triggered event detector used for level-crossing detection and an event-triggered controller which calculates the control signal. Whereas the first part runs with a constant sampling period - that is the same than for the corresponding conventional time-based controller - the second part is driven by some requests sent by the event detector. In the original proposal, a request is activated either when the measurement changes and crosses a given level or if a maximal sampling period is achieved. The second condition was added for stability reasons in order to fulfill the condition of Nyquist-Shannon sampling theorem: a new control signal is performed when the amount of time elapsed since the last sample exceeds a certain limit. This clearly shows how it is difficult to untie the well-established synchronous paradigms because, in fact, such a safety limit condition is no more consistent thanks to the level detection mechanism. Consequently, we develop in this thesis some new event-based algorithms without safety limit condition anymore. Finally, the proposals lead to a noticeable reduction of the mean control computational cost (in simulation and practice), and besides, the performance of the closed-loop system is also improved.

Whereas the previous approach updates the control law when required from a performance point of view, it could be interesting to enforce events with respect to the controlled system stability. Some theoretical tools would thus be useful to prove that an event-driven scheme can correctly control a system, even if the system is not sampled during a long amount of time. Using a Lyapunov function to decide when updating the control law is such a solution, for instance, since the Lyapunov theory is powerful in stability analysis. Actually, the Lyapunov sampling mechanism introduced by *Manel Velasco et al.* in 2009 enforces events only when the system trajectory reaches a given value. A state-feedback strategy then updates the control signal

with respect to the resulting requests. However, the existing work relies on the heavy off-line computation of a parameter which is not acceptable in embedded devices since the key point in such systems is to reduce the computational cost. A solution relaxing the event condition is hence proposed here, which leads to a fully on-line algorithm while still ensuring the stability of the system.

Eventually, this thesis covers different domains. Firstly, a skill on micro-electronics field is needed in order to model the different systems and to be able to tackle the different problems which occur in low-scale technologies. An awareness in computing is also interesting since some tricks will be developed to program the different control algorithms and implement them in real-time testbeds. The least knowledge is the most important. It concerns the control theory which is the key point of all contributions in this thesis.

Structure of the thesis

As previously explained, the thesis focuses on reducing both the energy consumption and the control computational cost in embedded electronic systems. Due to the dissimilar natures of the objectives, the thesis is composed of two parts:

Part I deals with the energy-performance tradeoff in embedded electronic circuits. Actually, a voltage scalable processor makes possible to reduce the energy consumption controlling the power supply (the voltage and frequency), but this gain decreases the computational activity of the processor in return. Furthermore, a high dispersion phenomenon could occur in nanometric chips (such as those of the study case) and a certain robustness is expected. Therefore, a control loop is required to decide the strategy to adopt in order to reduce the energy consumption while guaranteeing some good computational performance. The context and the motivations of this work are introduced in chapter 1. Then, a closed-loop architecture is depicted in chapter 2 where a single processor has to be controlled, and several control laws are also developed. This seminal scheme is finally extended in chapter 3 to a global structure where different devices have to work together. The devices are thus power supplied with the same voltage and clock frequency. At the end, some simulation results are performed in chapter 4 in order to highlight the gain achieved thanks to our different proposals. The robustness to parameter variations is also demonstrated.

Part II is devoted to reducing the computational cost of the controller using an asynchronous sampling mechanism. Contrary to the classical time-triggered scheme which calculates the control signal in a periodic fashion, an asynchronous (or event-based) control strategy updates the control signal only when the measured signal *sufficiently* changes. This allows to highly reduce the number of samples and directly impacts the computational cost of the system. The framework is introduced in chapter 5 together with the context of this research topic. Then, a simple event-based scheme is studied in chapter 6. The PID control setup is used in this first work where some events are enforced with respect to the measured system error. Different strategies are developed and some simulation results confirm the interest of such an approach. In chapter 7, the principle is extended to a state-feedback control law using a Lyapunov function to decide when updating the control signal. Finally, some experimental results are performed in chapter 8 on linear and nonlinear real-time systems.

Eventually, some conclusions and perspectives are drawn at the end of the thesis.

Part I

Energy-performance tradeoff in electronic systems

Chapter

1 Context and motivations

The embedded electronic devices intend nowadays to be more and more efficient with important low-power constraints and yet, this is clearly an energy-performance tradeoff which needs to be wisely studied. Actually, with the upcoming of micro and nanometric technologies, the number of transistors which can be fitted onto a chip continuously increases and the circuits are now composed of millions of transistors. This allows to deal with some complex functions in a minimum amount of time. On the other hand, these chips have reached some hardware limits in terms of power consumption, computational efficiency and fabrication yield. The forthcoming generations hence require several drastic technological evolutions. Furthermore, the system performance after fabrication is not fully predictable anymore since new problems - which did not influence the circuit at a higher scale - appear, notably the process variations which become huge at the chip scale. This implies to develop new strategies for energy management which are robust to the uncertainty of the chip. Therefore, some solutions including control loops become essential and have to be considered.

This chapter starts introducing micro and nano-electronics in section 1.1. A survey of different problems facing designers over the nanometric era is then detailed in section 1.2 while some solutions are suggested in section 1.3. At the end, a framework to make such systems robust to process variability using feedback control loops is developed in section 1.4. Different control strategies will be proposed in the next chapters based on that. One could note that this work was done within the ARAVIS project context - see subsection 1.4.2 - which aims at giving architectural solutions for manufacturing some integrated circuits in technology in limit of scalability where high technological variability could occur into a single chip. However, the whole proposal can be transposed to more general electronic systems and the robustness to process variability could be applied to any uncertainties.

1.1 Micro and nano-electronics

Microelectronics is a subfield of electronics which, as the name suggests, is related to the study and manufacture of electronic components at a very small scale, that is the micrometer (denoted μm and corresponding to $10^{-6}\,m$). In order to have an idea on the order of magnitude, a micrometer is about thirty times thinner than the thickness of a human hair. Some miniaturized components are made from semiconductor material (such as Silicium) using different technologies in such a way that classical components have their microelectronic equivalents. In particular, this is the case of the *transistor* which is the main component in all logical operations since several applications are possible. Thus, a transistor can control its output in proportion to the input signal, that is, it can act as an amplifier. Alternatively, it can be used to turn the current "on" or "off" in a circuit, such as an electrically controlled switch. Indeed, the transistor is composed of three electrodes - labeled *source*, *drain* and *gate* - which could easily make the component passing or blocking: no current is provided without any supply voltage at the gate, but as soon as a positive voltage occurs an electrical current flows between the source and the drain. This principle is illustrated in figure 1.1. Note that a transistor which is designed in a $10\,\mu m$ technology means the length of the gate of the transistor is $10\,\mu m$.

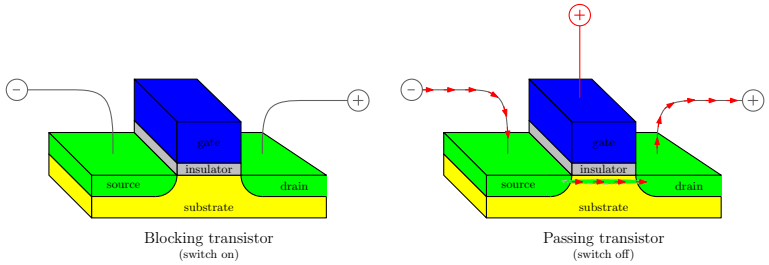


Figure 1.1: Transistor principle: the electrically controlled switch case.

The manufacture of semiconductors also allows to integrate several electronic functions on only one surface. These so-made circuits are called *chips* or *integrated circuits*. Contrary to the first electronic circuits which were an amount of several components connected together by conductive wires, the connections in integrated circuits are designed before the fabrication in order to use a shared support in Silicium. Finally, the chip is less bulky and less expansive. Moreover, it integrates more components and, consequently, more functions could be realized. For these reasons, the integrated circuits are present everywhere: they are in each room of the house (TVs and DVD players, household electrical appliances, etc...), in mobile phones or in automotive, and even in the wallet with the smart card on the credit cards.

The most advanced integrated circuit is the well-known *microprocessor*, the core of every computers, which contains millions of transistors in order to deal with complex operations in a minimum amount of time. The evolution of the manufacturing techniques for these circuits is so important that the number of transistors which can be fitted onto a chip continuously increases. The "*Moore's law*" - from the name of its author, Gordon Moore, the Intel co-founder - incidentally predicts that this value will double approximately each eighteen months. Finally, that is surprisingly accurate since 1971 which is the date where the first microprocessor was manufactured. However, the ratio between the number of transistors and the surface of the chip increases more slowly for few years. That is due to some power dissipation difficulties which

Note that this section was established with the help of the French web site on nanosciences and nanotechnologies (`http://www.nanomicro.recherche.gouv.fr/`) and some articles from the free encyclopedia Wikipedia (`http://fr.wikipedia.org/`)

avoid a faster frequency whereas the components are smaller. Nevertheless, the parallelism is a solution which consists in multiplying the number of processors on a single chip. Furthermore, the notion of *System on Chip* - afterwards denoted SoC - recently appeared and refers to a single chip embedding a full system. A system on chip can hence integrate one or several microprocessors, some memories, some interface peripherals, or all other components required to realize an expected electronic function. Moreover, in order to facilitate the design of a SoC, a lot of semiconductor Intellectual Property cores exist. Denoted IP-CORE or IP-BLOCK, such a block is a reusable unit of logic, cell, or chip layout design that is the intellectual property of one party. These IP-cores can then be used as some building blocks within chip designs or logic designs.

Finally, the chips are even smaller and smaller and the current techniques of fabrication lead to commercialize some circuits in $90\,nm$ technology or smaller. This means that a new leap was taken since the length of the gate is now close to a nanometer (denoted nm and corresponding to $10^{-9}\,m$). But some parasitic physical phenomena, which do not influence the circuit at a higher scale, become very annoying at a sub-micrometric scale. As a result, the scalability - or Moore's law - is not persistent anymore. The next section lists these problems whereas some solutions are proposed in sections 1.3 and 1.4.

1.2 Problems in nanometric technologies

With the upcoming nanotechnologies, the integrated system performance after fabrication will not be fully predictable. Indeed, the embedded integrated systems have reached some limits in terms of power consumption, computational efficiency and fabrication yield. The forthcoming generations hence require several drastic technological evolutions. Notably, performance estimation and management are today some key points in the new integrated systems and control loops become essential in nanometric circuits. Thus, for example new strategies for a global energy management have to be used to meet energetic and real-time constraints. Some solutions, such as dynamic voltage and frequency scaling techniques (DVFS), have to be considered for instance. They have been explored and have shown significant energy savings while meeting performance requirements (see subsection 1.3.1 for further details). Nevertheless, we are still facing nowadays to some intrinsic problems with the nanometric technologies.

Process variability: The variability refers to the unpredictability, inconsistency, unevenness, and changeability associated with a given feature or specification. Therefore, variability has become one of the leading causes for chip failures and delayed schedules. In nanometric design flows, this is associated with design modes, power states, environmental conditions, manufacturing steps, and the behavior of devices and interconnects. This is why the process variability affects the entire physical design environment, from power management, through timing and signal integrity closure to manufacturability. A major problem facing the computer and semiconductor industries is the increasing amount of CMOS process variability (note that the complementary metal oxide semiconductor is the most commonly used technology to manufacture electronic components and integrated circuits). Thus, variability in low-level circuit parameters - such as transistor gate length and gate oxide thickness - complicates the system design by introducing an uncertainty about how a fabricated system will perform [55]. Although a circuit or chip is designed to run at a nominal clock frequency, the fabricated implementation may vary far from this expected performance. In other words, a part of the circuit could run with the expected computational speed whereas another part may run more slowly and another might not run at all. This phenomenon is illustrated in figure 1.2 where a light speed gradient indicates a well-running area and a darker gradient is used for a worst-running area. This

process variability becomes one of the main problems in nanotechnologies and some tricks are hence required to manage its impact on the circuit manufacturing.

Leakage power: As voltage levels scale downward with geometries, the threshold voltages must also decrease to gain performance advantages of the new technology. This reduction in threshold voltages has led to an exponential increase in leakage current in transistors, while thinner gate oxides have led to an increase in gate leakage current as well. Moreover, at $65nm$ and below the leakage power accounts for a significant portion of the total power in high-performance designs [52], therefore its management is essential in the design process of integrated circuits. The leakage power is a growing concern in the overall design process. Unlike dynamic power - which can be managed by reducing switching activity - the leakage power effect exists as long as the power is on. That is why current process technologies are pushing designers to consider new design methods to reduce this issue.

Yield: Early on, the integrated circuit design rules were absolute and finite. The path to yield was fairly simple: comply with all the design rules and yield would follow. Designers did not need to worry too much about what happened in the fabrication after tape-out [51]. However, in the nanometric era the game has changed and a yield success is much harder to achieve because of the increased number and complexity of variables affecting manufacturability. The designer's strategy must hence shift from simple design rule compliance to the definition and design of the optimal layout for the highest yield.

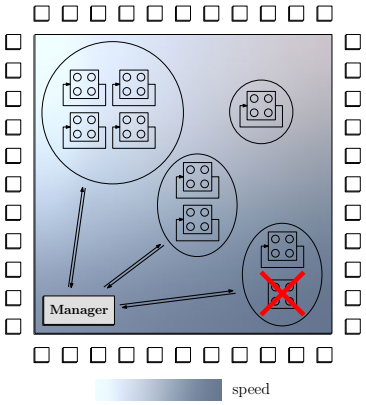


Figure 1.2: Technological variability: representation of such a phenomenon which could appear in sub-micrometric integrated circuits.

1.3 Suggested solutions

As the problems in nanometric technologies are numerous and important, the computational architectures have to be completely (re)-designed from scratch and adapted to compensate the technological constraints. Different solutions exist. For instance, some power management techniques are presented in subsection 1.3.1 and a special focus on the energy-performance tradeoff done in subsection 1.3.2. A parallelization paradigm is then introduced in subsection 1.3.3.

1.3.1 Power management techniques

Three main sources of power consumption exist in CMOS circuits, as explained in [24], which can be sorted into ***i)*** *a dynamic consumption* due to the electrical gate switching, that

is $P_{switching}(t)$, and ***ii)*** *a static consumption* induced by the short-circuit and leakage currents, that are $P_{short-circuit}(t)$ and $P_{leakage}(t)$ respectively. This yields

$$\begin{aligned} P(t) &= P_{switching}(t) + P_{short-circuit}(t) + P_{leakage}(t) \\ &= K_{dyn} \cdot f_{clk}(t) \cdot V_{dd}(t)^2 + K_{sc} \cdot f_{clk}(t) \cdot V_{dd}(t) + K_{leak} \cdot V_{dd}(t) \end{aligned} \tag{1.1}$$

where $P(.)$ is the total power consumption while $V_{dd}(.)$ *is the supply voltage* and $f_{clk}(.)$ *is the clock frequency* which both supply the device. These variables evolves with respect to time. On the other hand, K_{dyn}, K_{sc} and K_{leak} are some constant parameters which depend on the electronic circuit. The static and dynamic power consumption can be reduced (and consequently the energy consumption). This is explained in the following subsections.

Static power management

As the gate length and threshold voltage are scaled down in today's technology chips (see section 1.2 for further details), the leakage power is a significant contributor to the total power. Several techniques can be applied at the circuit level to reduce this leakage power, including multi-threshold libraries, power gating and variable body biasing [39, 70]. Furthermore, since in advanced technologies the associated processor leakage has an important contribution to the system energy consumption, a sleep mode management can be required to put useless processors into a sleep mode and hence limit their static power consumption.

Dynamic power management

For current integrated circuits, the average power consumption and the energy dissipation are dominated by the switching power defined in equation (1.1). That arises from the charging and discharging of the load capacitance. Finally, the previous relation becomes

$$\begin{aligned} P_{avg}(t) &\propto C \cdot f_{clk}(t) \cdot V_{dd}(t)^2 \\ E(t) &\propto C \cdot V_{dd}(t)^2 \end{aligned} \tag{1.2}$$

where $P_{avg}(.)$ is the average power consumption, $E(.)$ is the energy dissipation and C is the load capacitance. This relation suggests that minimizing the load capacitance, reducing the supply voltage or slowing the clock frequency can reduce the power. However, while the load capacitance can only be affected during the chip design - for example by minimizing the on chip routing capacitances and reducing the external components access - a voltage scalable processor and power controllable peripheral devices make possible to reduce the power and control it by a dedicated digital hardware or by an operating system (OS). For instance, the OS can control the processor frequency and its voltage and/or put the device in low-power sleep states. This is called the dynamic power management in [41]. The power minimization can be achieved by resolving an off-line stochastic optimization problem but this is not always possible. Therefore the optimization has to be performed on-line by a control system dedicated to the power management. Thus, some techniques will lead to run the processor slower at a reduced voltage according to the instantaneous computational demand.

As highlighted with equation (1.2), reducing the clock frequency and the supply voltage decreases the power consumption. The reduction is a quadratic function of the voltage and a linear function of the frequency. Moreover, only decreasing the voltage impacts on the energy. As a result, the dynamic voltage scaling (DVS) can be used to efficiently manage the chip energy consumption [29]. The supply voltage can be reduced whenever a slack - i.e. an amount of time - is available in the critical path - the longest electrical path a signal can travel to go

from one point of the circuit to another - but one has to take care that scaling the voltage of a microprocessor changes its speed as well. Indeed, the propagation delay of transistors $T_d(.)$ significantly increases as the voltage approaches the threshold voltage of the device V_t, as defined by

$$T_d(t) \propto \frac{V_{dd}(t)}{\Big(V_{dd}(t) - V_t\Big)^2}.$$

Therefore, controlling the voltage is a power-delay tradeoff: the power consumption decreases while the delay increases. Adapting the supply voltage is very interesting when possible but this implies the use of dynamic frequency scaling (DFS) to keep the system behavior correct. The addition of DFS to DVS is called the dynamic voltage and frequency scaling and DVFS results in simultaneously managing the frequency and the voltage. In fact, in many cases of application the only performance requirement is that the task meets a deadline, as depicted in figure 1.3(a). Such a scheme creates some opportunities to run the processor at a lower performance level and achieve the same perceived performance while consuming less energy. Indeed, figure 1.3(b) shows that decreasing the processor clock frequency reduces the power consumption but simply spreads the computation out over time, thereby consuming the same total energy than before. On the other hand, figure 1.3(c) shows that decreasing the voltage level as well as the clock frequency leads to achieve the expected goal of reduced energy consumption at an appropriate performance level [67]. To sum up, dynamic power management is an energy-performance tradeoff. Furthermore, the supply voltage and the clock frequency have to be controlled together in order to ensure the maximum delay over the critical path. Clearly, it is required to decrease the frequency before decreasing the voltage and, respectively, to increase the voltage before increasing the frequency. Finally, the DVFS method leads up to an important

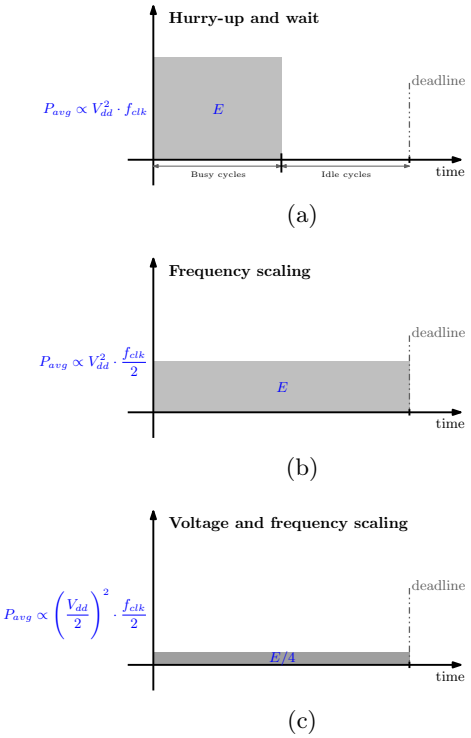


Figure 1.3: Dynamic power management: energy consumption vs. power consumption.

energy consumption reduction [53]. Moreover, it seems that most applications could run on a voltage scalable device [21].

Clock gating

Contrary to power gating - introduced above in static power management - which affects the leakage power, the clock-gating principle allows to decrease the dynamic power consumption. Clock gating is used on many circuits. To save power, clock gating support adds more logic to a circuit to prune the clock tree, thus disabling some portions of the circuitry in order their switching power consumption goes to zero [39]. This behavior eventually leads in "pausing" the clock frequency. That can hence be used if necessary to save energy when a task is performed before its deadline.

1.3.2 Focus on the energy-performance tradeoff

Several behaviors are known to minimize the energy consumption while guaranteeing good computational performance (see [36] for further details). Classically, each task has to be considered independently. Thus, when several tasks - with their own and different computational load - have to be executed, the voltage level has to be calculated for each one rather than considering a global scaling for all the tasks. Moreover, the execution time has to fit with the deadline regardless the chip runs with a continuously or a discretely varying voltage range. Indeed, in figure 1.4 one can see several possible behaviors to execute a given task with different possible voltage levels. In this example, $1000 M\,cycles$ have to be computed. The processor consumes $10\,nJ/cycle$, $25\,nJ/cycle$ and $40\,nJ/cycle$ when it is running at $2,5\,V$, $4\,V$ and $5\,V$ respectively, and the corresponding maximal frequencies are $25\,MHz$, $40\,MHz$ and $50\,MHz$. When only the maximum voltage level is possible - as this is the case for a classical processor (without dynamic voltage scaling) - the hurry-up and wait running is performed but leads to an important energy consumption, as shown in figure 1.4(a). On the other hand, in figure 1.4(b) a continuously varying voltage scaling allows using an unique level to fit with the deadline. The consumption is hence optimal and cannot be decreased anymore. However, the energy consumption can also be reduced with a discretely varying voltage scalable processor by using the available voltage levels to fit the task with its deadline. In this case, the lowest energy consumption is achieved using the two immediate neighbors of the optimal level. This is depicted in figure 1.4(c). Another essential rule is that selecting some suitable voltage levels leads to a drastic energy reduction even if the number of levels is very small. Anyway, a frequency range is available for each voltage level [54]. This can be useful to fit the task with its deadline, else, the clock-gating principle - presented in the previous subsection - could be an alternative solution.

A task scheduling can be a solution. In [30] for instance, a simple method consists in monitoring the total activity of the chip - without knowing task by task information - and applying a high voltage when the processor is busy or a low voltage when it is idle. In [20, 21], the different tasks are sorted into three possible throughput (the number of instructions to treat in a given amount of time): *compute intensive and short-latency processes* for some tasks which require the maximum performance and will be executed with a high voltage level, *background and high-latency processes* for some non-critical tasks which will be computed at a lower voltage and *processor idle*. The tradeoff between a maximal throughput and a minimal energy consumption is thereafter the key point. More sophisticated scheduling policies propose to integrate a DVFS scheduler and a feedback controller, as the one proposed in [76] where an EDF (earliest deadline first) scheduler is mixed with a PID (proportional integral derivative) controller. As a result, closed-loop control strategies are required to manage the energy-performance tradeoff in electronic devices and some new laws are developed in this sense in the following chapters.

Finally, the main idea of the different proposals consists in ***i)*** reducing the penalizing supply voltage so far as possible in order to minimize the energy consumption and ***ii)*** adapting the clock frequency to the computational load to fit the task with its deadline.

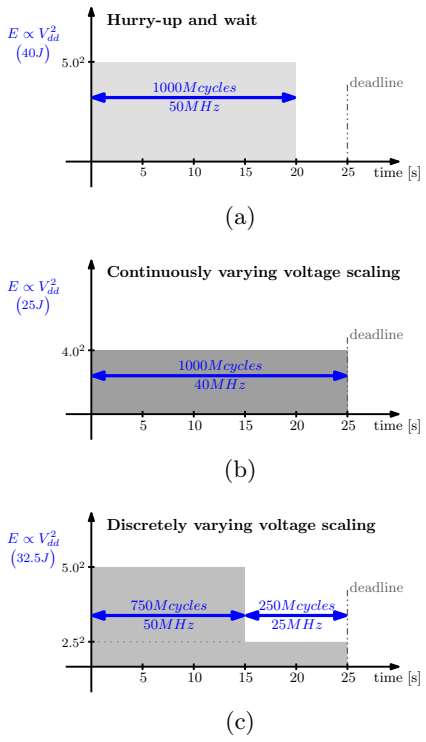


Figure 1.4: Task scheduling: an issue to minimize the energy consumption of a task fitting its execution time with the deadline.

1.3.3 Globally asynchronous locally synchronous paradigm

Embedded integrated systems have two means of implementation. Firstly, the conventional clocked circuits with their global synchronization - in which is facing the huge challenge of generating and distributing a low-skew global clock signal and reducing the clock tree power consumption of the whole chip - makes them difficult for implementation. Secondly, systems on chips, built with predesigned IP-blocks running at different frequencies, need to integrate all the IP-blocks into a single chip (this notion is presented in section 1.1). Therefore, the global synchronization tends to be impractical [28]. By removing the globally distributed clock, globally asynchronous locally synchronous (GALS) circuits provide a promising solution for a SoC design. Moreover, some GALS techniques allow to set independently the frequency and the voltage of each locally synchronous modules, making scaling far more convenient than with the standard synchronous approach. A dynamic power management - introduced in subsection 1.3.1 - is hence easier.

The GALS systems are chips split into multiple frequency domains, where each domain is synchronous with respect to its clock. The different domains are mutually asynchronous in that they may run at different clock frequencies. An asynchronous network on chip (ANoC) - a new approach to design the communication into a SoC - can be used in this way for instance, such as represented in figure 1.5. Thus, a GALS architecture can mitigate the impact of process and temperature variations, since a globally asynchronous system does not require that the global

frequency was dictated by the longest path delay of the whole chip (i.e. the critical path). Indeed, in this case each clock domain frequency is only determined by the slowest path in its domain.

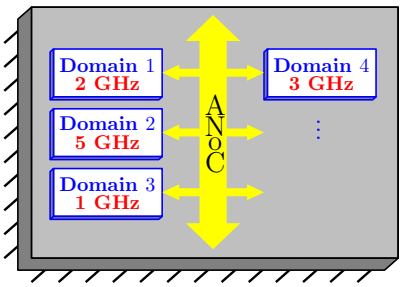


Figure 1.5: Globally asynchronous locally synchronous (GALS) architecture.

Eventually, using both a NoC distributed communication scheme and a GALS approach offers an easy integration of different functional units thanks to a local clock generation [38]. Moreover, it can allow better energy savings since each functional unit can easily have its own independent clock frequency and supply voltage. Hence, an architecture using NoC combined with a multi-power domain GALS system appears as a natural enabler for distributed power management systems as well as for local DVFS.

1.4 Handling the process variability

In a reconfigurable GALS system, the process variability and the fabrication yield can be improved by smartly removing some tasks over fault or some low performance frequency domains and assign them into other high performance ones. As each domain performance can be measured by a sensor, a global system manager is able to distribute the tasks over them. The task assignment takes into account the domain performance and the task processing load. The main target of this manager is then to ensure an overall chip performance. With this kind of approach, it is no more required to separately guaranty a performance for each domain which hence relaxes the fabrication constraints and permits a yield enhancement. Based on that, a solution is proposed in subsection 1.4.1 to manage the impact of process variability. An important conclusion is that control loops become essential in the electronic chips fabricated in nanometric technologies. Then, a study case is proposed in subsection 1.4.2 where a French project is depicted. It aims at providing some architectural solutions for manufacturing such circuits.

1.4.1 Essential feedback control loops in nanotechnologies

In a multiprocessor GALS system, one can choose to slow down some parts of the circuit while allowing some others to operate at the maximum frequency. This approach enables more energy saving opportunities than the conventional systems built around only one processor, and allows adapting the clock speed to the local process quality. Moreover, it has also been shown that multiple-clock design with voltage scaling is even better not only in terms of power and performance, but also in terms of variability [44]. As a result, building a system based on the implementation of hardware resources - whose performance is unpredictable at the fabrication time - requires having some global management strategies of the performance. This can be achieve by adapting the voltage/frequency in order to respect the real-time constraints of the application and the allocated energy budget. Finally, **we propose to use feedback control**

loops based on ***i)*** the measurement of the real local performance of the chip, ***ii)*** the actuation of the voltage/frequency variables and ***iii)*** the suitable hardware resource allocation for the execution of a task in the assigned time-energy budget. Then, the idea is the use of a DVFS mechanism with some task scheduling techniques to dynamically manage not only the energy budget but also the activity of the different frequency domains. This is possible thanks to some advanced closed-loop techniques. These techniques will allow an optimal regulation of the power supply according to the computational load and the load distribution in the various GALS processors. In order to compensate for the process variation due to the technology dispersion, and optimize the operation of the circuit, the dynamic voltage/frequency regulation should be self-adjustable with the variable loads and dispersion models and robust against process variability.

One of the main points of interest of the proposal is to handle the uncertainty (due to the process variability) of a given frequency domain - afterwards denoted a *cluster* - over a GALS system and also to reduce its energy-consumption by means of automatic control methods. Some activity sensors are supposed to be embedded in each processing unit. These sensors provide a real performance measurement of the different processing nodes after the fabrication process. This will be used afterwards by the operating system to distribute the tasks over different nodes. For instance, some background tasks will be assigned over idle nodes while fault circuits will be known and therefore not be used. This scheme means to be able to reschedule the tasks in each processing node, in order to meet the new assigned deadlines, and this will be achieved by controlling its voltage/frequency (which in accordance will control its consumption of energy). Eventually, the control system uses three overlapped control loops, applied in different architecture levels: control of the processing power (supply voltage and clock frequency), control of the energy consumption and control of the quality of the running application. Voltage, frequency and energy control loops are used in order to adapt the energy consumption and the process variability effect. The other control loop is needed to deal with the quality of service (QoS) at the application level, with the limitation of processing power and/or channel of communication and with some constraints in energy consumption. The operating system provides a set of information (required speed, number of instructions and deadline for each task to treat) that can be statically inserted into the code or dynamically computed at run time. There are also sensors embedded in each processing unit in order to provide some real-time measurements of the computational speed of the processor. Therefore, this information about the real-time requirements of the applications enables to create a computational load profile with respect to time. Consequently, using such a profile makes possible to apply a fine-grain power management allowing application deadlines to be met. The DVFS hardware part contains voltage/frequency converters (such as a DC-DC converter and a programmable clock generator). Then, a controller dynamically controls them to scale the supply voltage as well as the clock frequency in order to satisfy the application computational needs with an appropriate management strategy. Of course, the controllers should have a strong robustness against process variability for a correct system behavior. This closed-loop architecture is finally detailed in the following items.

A. Voltage and frequency control

Very low-level hardware systems control the actuators which supplied the different computational nodes, that are the *voltage controller* and the *frequency controller*, in such a way that the supply voltage $V_{dd}(t)$ and the clock frequency $f_{clk}(t)$ dynamically track a given reference: the voltage level $V_{level}(t)$ and the frequency level $f_{level}(t)$ respectively. These controllers lead to a stable power supply without non-desirable effect, some current peaks could notably appear during a sudden voltage level transition. This low control level is

drawn in figure 1.6 and fully detailed in [4, 74].

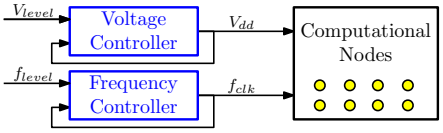


Figure 1.6: Essential control loops in chips: the voltage and frequency (low-level) control.

B. Energy-performance tradeoff control

A second feedback control loop then provides the references to the previous low control level: an *energy-performance controller* calculates the voltage level $V_{level}(t)$ and the frequency level $f_{level}(t)$ which minimize the energy consumption of the processors while guaranteeing some computational performance. Thus, the tasks to treat have to be correctly executed and within the given amount of time. This is decided from the error between the measured activity of the computational nodes - given by $\omega(t)$ which is the computational speed of the nodes - and a reference to track - given by $\omega_{sp}(t)$ which is the expected computational speed required to correctly execute the tasks - provided by the operating system for each task to treat. This is represented in figure 1.7.

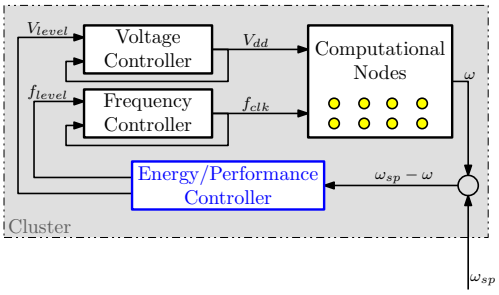


Figure 1.7: Essential control loops in chips: the energy-performance tradeoff (middle-level) control.

The two first control levels are present into a cluster. This corresponds to a frequency domain and more precisely to a power supply domain. Each cluster has its own voltage and frequency controllers which supply the different nodes. This architecture also involves an energy-performance controller by cluster. Finally, the last feedback control loop is placed outside the clusters in order to be able to manage all of them. The latter controller could be placed near the OS for instance.

C. Control of the applicative quality of service

The third controller has a global view of the whole set of clusters, as depicted in figure 1.8, firstly in order to allocate the different tasks to treat to the different computational nodes of each cluster, considering the capacity of each one. Thus, the critical tasks will be executed with the best nodes whereas the background ones will be performed with slower ones. Secondly, the controller calculates a quality of service in such a way that the running application fulfils some criteria. For example, in a video coding/decoding mechanism, the *QoS controller* could calculate the image resolution quality required to finish to watch the movie with the remaining power in the battery. Several criteria exist regarding a given application. The QoS controller, after having determined the criteria, ensures that the

running service $\lambda(t)$ is as good as the expected service $\lambda_{sp}(t)$. As a result, the last control level will modify the activity reference $\omega_{sp}(t)$ sent to the energy-performance controller (the middle-level control loop) when the quality changes. Such a control is depicted in [7].

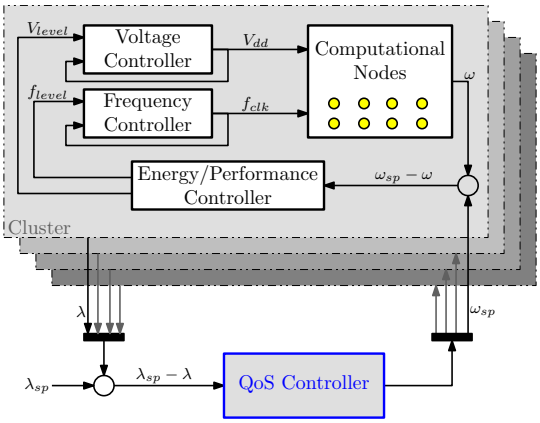


Figure 1.8: Essential control loops in chips: the applicative quality of service (high-level) control.

These control algorithms will allow an optimal control of the supply voltage regarding the computational load and a good allocation of the load into the different processors. The expected gain is to compensate the process variability intrinsic to a nanometric technology and, therefore, optimize the circuit. Sensing the computational activity becomes essential. Activity sensors hence play a critical role and must be selected carefully. In [74], an *instruction counter* is used to calculate the real speed of the processor in MIPS (million instructions per second) by incrementing the counter each time a new instruction has been executed and calculating the average value with respect to a reference clock. *Activity sensors* (current sensors) are also disseminated into each processing node to locally evaluate the process quality. They consist in an analog solution to monitor the activity of the system with respect to the amount of consumed current. At the end, the output of such a monitoring can be used as a tool to decide how to adjust the working conditions of the circuit and, therefore, get the best power-performance response.

These solutions - the three overlapped feedback control loops with some specific sensors to monitor the activity of the chip - have now to be developed and implemented. This is done within the ARAVIS project context which is now presented.

1.4.2 Study case: The ARAVIS project

The ARAVIS project (French acronym used for "advanced reconfigurable and asynchronous architecture for video and software radio integrated on chip) proposes to give some architectural solutions for manufacturing integrated circuits in a technology in limit of scalability. This is when the Moore's law (see section 1.1 for further information) is not applicable anymore due to the high technological variability of the parameters into a single chip. This project targets more particularly the computational platforms for embedded systems. Such architectures have to be flexible because they have to implement algorithms which are more and more numerous and customizable. This new demand hence transforms the fixed hardware-acceleration architectures into reconfigurable software architectures. The ARAVIS project proposes to combine three key technologies to make a system on chip in $32nm$ technology and give some parts of a solution for the sub-micrometric challenges. These three technologies are ***i)*** a multiprocessor framework, ***ii)*** a hardware/software asynchronous design and ***iii)*** a dynamical energy-performance control

based on advanced control techniques. Furthermore, two applications with a high industrial and marketing potential will finally run on a SoC ARAVIS demonstrator: video coding/decoding application and software radio application. Therefore, the SoC ARAVIS will be dynamically adapted by allocating the applicative load into the different processors of the chip, in order to control the process variability and the energy consumption.

To deal with such an ambitious industrial project with high technological constraints - in particular the accessibility to the $32nm$ technology - several partners offered to contribute within this project. Bringing together the different members of this consortium hence gives to the ARAVIS project a large experience and a real expertness in several and different knowledge domains, more particularly regarding asynchronous, multiprocessor and SoC architectures designing, software tool development for embedded and nomad applications, or advanced control techniques for complex systems.

STMicroelectronics[1] is a worldwide leader in semiconductors and has a high experience in designing and manufacturing embedded chips. The firm brings the ARAVIS project its skill in embedded systems on Silicium, from the specifications to the manufacturing and system validation.

CEA-Leti[2] is one of the most important applicative research center in Europe and is principally assigned in helping the industry by improving its competitiveness with some technological innovations. The main activities are based on micro and nano-electronics. As regards the project, the SCME department (service of conception for emerging microtechnologies) is expert for designing some complex SoCs and NoCs. Moreover, with the technological variability growing in nanotechnology chips, the department works more particularly on asynchronous system development and low-power techniques.

TIMA[3] is a public laboratory. The research topics cover the specification, design, verification, test, computer-aided tools and design methods for integrated systems, from basic analog and digital components to multiprocessor systems. Two research groups contributes within the ARAVIS project: SLS (system level synthesis) and CIS (concurrent integrated systems). The first one has a high experience in high-level hardware/software conception for complex SoCs and NoCs whereas the second one develops efficient tools for asynchronous circuits and systems. Both bring an expertness in designing and hardware/software integrating for embedded systems on asynchronous chips.

INRIA[4] is a research center which focus the information and communication sciences and technologies. It fosters technology transfer in computer science and control. Two project-teams provide a double skill within the project:

- Sardes (system architecture for reflective distributed computing environments) studies the construction of computer systems which can scale from resource-constrained embedded systems to cloud-based systems. This project-team hence brings its expertness on component-based operating systems.
- NeCS (networked controlled systems) has a large experience in developing dynamical control loops for complex systems with high constraints, such as computational and communication limited resources or other actuator/sensor specific constraints. Within the ARAVIS project, this project-team brings a huge expertness in control theory in order to ***i)*** reduce the process variability effects and ***ii)*** dynamically control the energy and the computations of the system.

The ARAVIS project is financed for a three-year period by the global competitive cluster MINALOGIC[5] which channels in a single physical location - in Grenoble, France - a range of highly-specialized skills and resources from knowledge creation to the development and production of intelligent miniaturized services for industry.

The first objective of the ARAVIS project is an increase of the yield of an advanced Silicium process, which means being able to use as many manufactured chips as possible. This boom is possible controlling ***i)*** *the configurability* of the platform as regards the failings, ***ii)*** *the programmability* in function of the effective computational power (all parts of the circuit could be allocated considering their own performance) and ***iii)*** *the adaptability* to technological variabilities and environmental/ageing variations (an application could run regardless the chip, without requiring any individual parametrization because of uncertainties nor reconfiguration after a certain amount of time). Finally, the ARAVIS project aims at concretely giving some solutions to the variability problematic and different realizations are expected in this sense.

★ **Proposal of a massively parallelized SoC architecture based on a set of multiprocessor clusters in $32nm$ technology.** The circuit is divided into several parts, denoted *clusters*, and each part itself is composed of several processors, denoted *computational nodes*. This architecture - represented in figure 1.9 - applies the GALS paradigm (previously defined in subsection 1.3.3). Each cluster has its own frequency domain - in order to independently work whatever the process variability is - and a cluster can communicate with another without any required synchronization, simply using an ANoC. On the other hand, the computational nodes have an unique clock frequency - the one of the corresponding frequency domain - which triggers the set of processors in such a way that they are all synchronized together into a cluster.

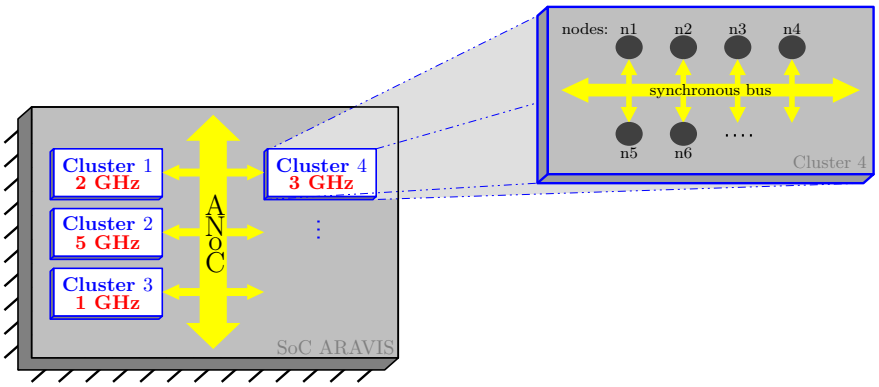


Figure 1.9: The SoC ARAVIS, a massively parallelized architecture.

★ **Proposal of new algorithms based on advanced control techniques to dynamically control the energy and the activity of the SoC.** The idea is to use an architecture including some feedback control loops in order to have a robust system. Such an architecture

[1]STMICROELECTRONICS: `http://www.st.com/`
[2]CEA-LETI, innovation for industry: `http://www.leti.fr/`
[3]TIMA, laboratory of techniques of informatics and microelectronics for integrated systems architecture: `http://tima.imag.fr/`
[4]INRIA, the French national institute for research in computer science and control: `http://www.inria.fr/`
[5]Competitive cluster Minalogic - an unique hybrid of micro and nanotechnologies and embedded software: `http://www.minalogic.com/`

allows for instance to control the behavior of a complex non-deterministic system regarding some specific dimensions. These well-known methodologies could then be adapted to control the non-predictable asynchronous systems with performance and consumption constraints which are quite close. Finally, three control levels were proposed into the SoC ARAVIS. They were introduced in subsection 1.4.1.

★ **Proposal of a prototype circuit in $32nm$ technology running with two complex applications.** The software radio and multimedia applications, more precisely video coding/decoding, will be implemented on this prototype in order to demonstrate the following advances:

- Contribution of the asynchronous logic in terms of robustness in this advanced submicrometric technology.
- Gain on the yield manufacturing thanks to a massively parallelized architecture.
- Energy reduction of the whole system thanks to the dynamical power and activity control.
- Relevancy of the proposed architecture for both types of application.

To summarize, some ambitious objectives are expected within the ARAVIS project. One of them is to control the power supply (voltage and frequency) of the different clusters of the SoC ARAVIS. The energy-performance tradeoff is hence a key point for each computational node. The voltage and frequency levels have to be dynamically controlled in order to treat a given computational load whatever the technological process variability. Therefore, the different points to study are:

1. Analysis and modeling of a computational node.
2. Conception of different control laws to minimize the energy consumption of a node while guaranteeing some computational performance.
3. Stability and robustness analysis.
4. Extension to the global activity to control a whole cluster.

These aspects will be analyzed more particularly in the following chapters: the monocore case is treated in chapter 2 whereas the multicore is detailed in chapter 3. Eventually, some simulation results are depicted in chapter 4 for both schemes.

Chapter

2 Control of the energy-performance tradeoff in monocore systems

As suggested in chapter 1, an electronic device with variable supply voltage and clock frequency allows energy savings. This is interesting, and more especially for embedded systems where this point is crucial. It enables to reduce the general speed of the circuit - and therefore its consumption - in order to fit with the computational needs, but this means to know the current load of the system. As a result, a closed-loop architecture is presented in this chapter for a single electronic device to control. This is firstly introduced in section 2.1. Several control strategies are developed in order to scale the processing power while guaranteeing good performance of the chip. An intuitive frequency and voltage control law is hence presented in section 2.2 while a more energy-efficient one is developed in section 2.3. A fully discrete scheme - where only a small number of voltage and frequency levels can be used - is also detailed in section 2.4. The last proposal is strongly robust to tackle variability - since it is not based on any parameters of the chip - and therefore suitable for $32\,nm$ technology or smaller implementations. Then, the control computational cost is reduced so far as possible in section 2.5 - using some simple tricks - in order to be suitable to embedded chips (where the resource allocation is often a hard constraint). At the end, an approximated stability analysis is performed in section 2.6 before synthesizing all the proposals in section 2.7. Eventually, one could note that some simulation results will be presented in chapter 4.

2.1 A single voltage scalable device to control

Whatever the electronic device - a processor, a computational node or a single system on chip - it usually runs with nominal and constant supply voltage and clock frequency. However, it could be interesting to make these parameters dynamically varying in order to be able to reduce the energy consumption. Thus, when the computational load of the processor becomes low, the power supply can be decreased to save energy, and increased back as soon as an important computation has to be done. Indeed, when the voltage and/or the frequency is decreased, the power consumption decreases too, with the effect that the device runs more slowly in return. This was detailed in section 1.3. Nevertheless, this is not a problem to correctly execute a low computational load, but a control strategy is required to know the current activity of the system. Therefore, in collaboration with the different partners of the ARAVIS project (see subsection 1.4.2 for further details), **we propose a closed-loop architecture which allows a dynamical control of the power supply (in voltage and frequency)**. This architecture is drawn in figure 2.1. Two actuators, the *oscillator* and the *Vdd-hopping*, respectively provide the supply voltage $V_{dd}(t)$ and the clock frequency $f_{clk}(t)$ to a single electronic system, denoted the *device*. These two actuators are then controlled by the *monocore controller* which calculates the most energy-efficient voltage level $V_{level}(t)$ and frequency level $f_{level}(t)$ while guaranteeing that the computational load to treat is correctly done. In order to deal with this energy-performance tradeoff, a feedback control loop is required: the measured computational speed $\omega(t)$ - relating the activity of the device - is monitored and compared with a given reference $ref(t)$ - provided by the operating system for each task to treat - and so is obtained the error which is then used by the controller to decide the control variables.

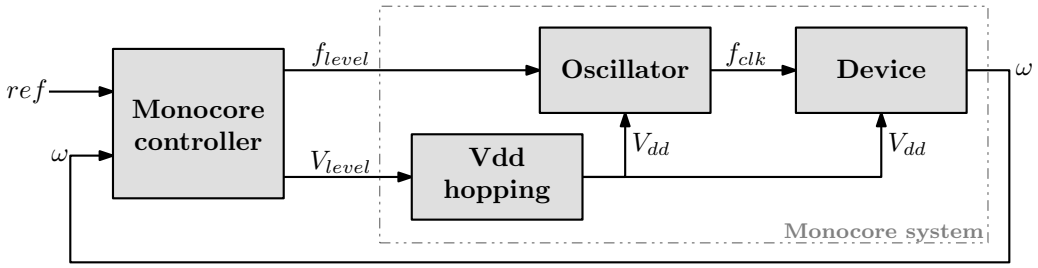


Figure 2.1: Architecture of the monocore system.

Before proposing some control strategies, each block has to be more detailed. Firstly, the electronic device behavior is explained in subsection 2.1.1. Then, the models of the two actuators are given in subsection 2.1.2: the Vdd-hopping and the oscillator are depicted in subsections 2.1.2.1 and 2.1.2.2 respectively. Finally, the control principle is shortly introduced in subsection 2.1.3.

One could note that the different variables vary with respect to time because their dynamics are needed in such a closed-loop architecture. Moreover, the *Laplace domain* is afterwards used - where s is the Laplace variable - because the frequency-domain is more convenient to manage some differential and integral equations in the transfer functions of dynamical systems.

2.1.1 The electronic device

As explained in the introduction, an electronic device usually runs at nominal supply voltage and constant clock frequency but these quantities will now dynamically vary in order to be able to reduce the energy consumption of the chip. Indeed, a dynamic voltage and frequency scaling (DVFS) mechanism could lead to important energy savings (regarding the running application) - this was notably explained in section 1.3 - and such a principle has to be adapted to the present

architecture. Different control strategies will be proposed in the next sections but the model of the system to controlled as firstly to be known. The electronic device under study is supplied with the voltage $V_{dd}(s)$ and triggered with the clock frequency $f_{clk}(s)$. These parameters are the two inputs of the system while the output is the resulting activity, i.e. the computational speed $\omega(s)$, as represented in figure 2.2.

Figure 2.2: Details of the architecture: the electronic device.

In order to build a behavioral model for a processing node, one has firstly to define the major sources of its power dissipation. This was summarized in subsection 1.3.1 with equation (1.1). Then, a fictive load that behaves more like a real loaded processor has been built in [74] using the power equation and representing an output current waveform with power dissipation variations. This finally gives the current variation and the processed number of instructions of the fictive processor which depends on the applied voltage and frequency values. As a result, the computational speed is obtained. This yields the curve plotted in figure 2.3, where the speed is drawn with respect to the frequency for two voltage levels; i.e. V_{low} and V_{high} respectively.

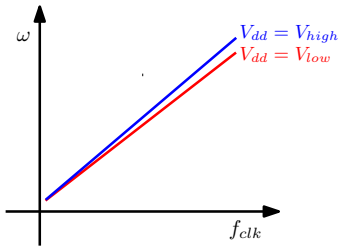


Figure 2.3: Behavior of the electronic device.

Finally, the model of the electronic device is a nonlinear relation which depends on the frequency and the voltage. Nevertheless, this can be written as a linear function with respect to only the frequency and where some parameters are function themselves of the voltage. This can be resumed as

$$\omega(s) = \alpha\Big(V_{dd}(s)\Big) \cdot f_{clk}(s) + \beta\Big(V_{dd}(s)\Big)$$

where $\alpha(.)$ and $\beta(.)$ are the varying parameters. These parameters can eventually be considered as some constant gains, that are α and β, due to the small impact of the voltage on the variables. They are positive. Moreover, β can be discarded because its value does not influence the speed so much and, anyway, α highly varies with respect to the temperature and the location on the chip (process variability). Finally, the model could be simplified. It becomes

$$\omega(s) \simeq \alpha \cdot f_{clk}(s) \tag{2.1}$$

2.1.2 The actuators

Two actuators control the electronic circuit: the Vdd-hopping and the oscillator which provide the supply voltage and the clock frequency respectively.

2.1.2.1 The Vdd-hopping to control the supply voltage

The application of DVS to a system requires the use of a source for supplying a variable processing power. This is a DC/DC converter in the present case: a circuit that converts a voltage source (of direct current) from one voltage to another. Two kinds of DC/DC converter can be used. The first class is a continuous converter which provides an accurate supply voltage, but with a weak efficiency. The second kind of converter is a digitally controlled step-converter (called Vdd-hopping converter) that has a better efficiency but discrete output values. Due to resource and space constraints in nanometric chips, the latter one is applied in our architecture. Thus, the Vdd-hopping mechanism allows to supply an electronic circuit with several possible voltage levels, with a given transition time and some dynamics that depend upon an internal control strategy. A model of this system can be represented by a simple block, as drawn in figure 2.4, where the supply voltage $V_{dd}(s)$ (the output) is function of the expected voltage level $V_{level}(s)$ (the input).

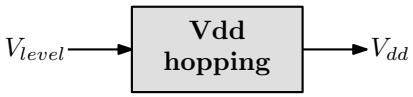


Figure 2.4: Details of the architecture: the Vdd-hopping.

The input of a M-voltage level mechanism belongs to a M-value set and such a Vdd-hopping hence provides the voltage V_m when $V_{level}(s) = V_{level\_m}$. We define $m \in \{1, 2, ..., M\}$ and $V_m > V_{m+1}$. Considering that this inner-loop is extremely fast with respect to the control loop considered in this chapter, the dynamics of the Vdd-hopping can be neglected. However, the complex transfer function between the supply voltage and the voltage level can be simply expressed as

$$V_{dd}(s) = \varphi\Big(V_{level}(s)\Big) \tag{2.2}$$

where $\varphi(.)$ depicts the Vdd-hopping dynamics. Nevertheless, firstly we consider a less complex Vdd-hopping scheme, initially detailed in [48, 4], where only two voltage levels are possible:

- The high voltage level, i.e. V_{high}, is the maximum possible voltage of the device. One could note that this value is the one used in common electronic systems where only one voltage level is available (systems without DVS mechanism).

- The low voltage level, i.e. V_{low}, is lower than V_{high} and will be used to reduce the energy consumption.

Thus, the supply voltage goes from V_{low} to V_{high} (respectively from V_{high} to V_{low}) when $V_{level}(s)$ becomes equal to V_{level_high} (respectively V_{level_low}). However, as explained before, the voltage transition is not instantaneous and a transition time exists: in [48] the voltage tracks an intuitive linear transition during the rising and falling times whereas a faster nonlinear transition is proposed in [4] (see references for further details). This second scheme is depicted in figure 2.5. Eventually, the function $\varphi(.)$ in equation (2.2) defines the dynamics of the chosen behavior.

Note that the two-level Vdd-hopping mechanism will be used in our first and second proposals of control strategies, in sections 2.2 and 2.3 respectively, before extending that to a M-level scheme in section 2.4.

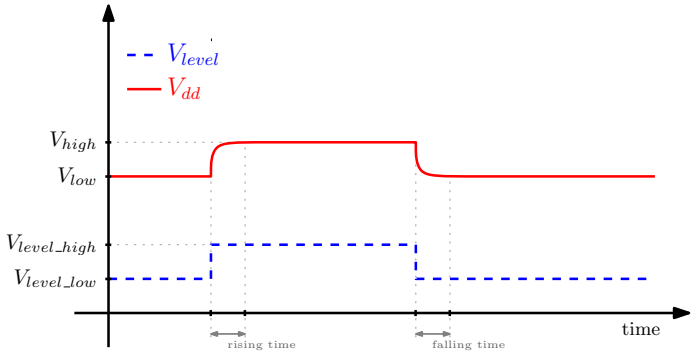


Figure 2.5: Behavior of the Vdd-hopping.

2.1.2.2 The oscillator to control the clock frequency

The application of DFS to a system requires the use of a source for generating some adjustable clocks. For example these clocks can be derived from analog a voltage controlled oscillator (VCO). However, VCOs have a limited operating range and require a stabilization time when changing the frequency. Another solution is to use a standard clock divider, but this makes the time resolution coarser, due to counting integer periods of the input frequency. In addition, they give regular time step which implies irregular frequency step (usually frequency step follows "$1/x$" curve). Self-timed rings are considered as a promising solution for generating clocks and will hence be used in our architecture. In [75] they are efficiently used to generate high-resolution timing signals. Their robustness against process variability in comparison to inverter rings is proven in [32]. The present oscillator is based on the behavior of programmable/stoppable oscillator depicted in [73], and finally, the oscillator provides the triggering clock signal to the electronic device. However, a so-fine grain is not necessary for the current analysis. Indeed, in fact one can directly deduce the computational speed of the device from the value of the clock frequency - and not the clock itself - since it is only function of the frequency, as defined in equation (2.1). The clock frequency $f_{clk}(s)$ is hence chosen as the output of the oscillator. As regards the inputs, they are the expected frequency level $f_{level}(s)$ and the supply voltage $V_{dd}(s)$, as proposed in [27]. This is represented in figure 2.6.

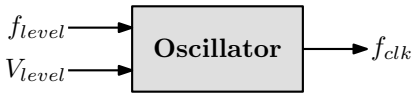


Figure 2.6: Details of the architecture: the oscillator.

While the relation between the clock frequency and the expected frequency is clear, the relation with the voltage is more indirect. In fact, the oscillator is supplied with this voltage which influences its behavior. Consequently, the clock frequency is reduced when the voltage decreases, and inversely. Finally, the oscillator model is

$$f_{clk}(s) = \gamma \cdot f(s) \cdot V_{dd}(s) \tag{2.3a}$$

where γ is a positive constant and $f(.)$ is the expected frequency. Moreover, this expected frequency depends on the frequency level, using such a look-up table mechanism for instance. This can be transposed as

$$f(s) = \psi\Big(f_{level}(s)\Big) \tag{2.3b}$$

where $\psi(.)$ depicts the relation between both variables.

Note that the clock frequency will be firstly continuously varying in the first proposed control strategies, in sections 2.2 and 2.3, before really using some discrete levels in section 2.4. Thus, a continuous behavior means an infinite number of frequency levels and in this case we set the expected frequency such that $f(s) = f_{level}(s)$. On the other hand, in a discrete case only some limited frequency values are possible, i.e. $f_{level}(s) = f_{level\_n}$, and switching from one to another can be considered as instantaneous. This scheme also leads to have a limited number of frequencies, i.e. $f(s) = f_n$. We define $n \in \{1, 2, ..., N\}$ and $f_n > f_{n+1}$, respectively $f_{level\_n} > f_{level\_n+1}$. Moreover, we choose $N \geq M$ because a frequency range is available for each voltage level and several frequency levels are thus possible for each voltage. This was explained in section 1.3 and one could refer to the introduction of section 2.4 for further details in this choice. Eventually, an example of the discrete behavior is drawn in figure 2.7 for $N = 5$.

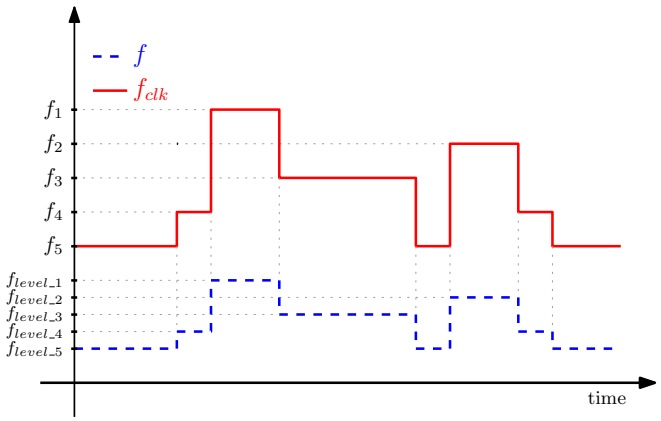


Figure 2.7: Behavior of the oscillator.

As explained in introduction, in a complete modeling the oscillator would provide the clock signal, denoted $clk(s)$ that is a square signal whose period is directly the inverse of the clock frequency $f_{clk}(s)$. Such a scheme is drawn in figure 2.8 in order to have an idea of the real running of the oscillator.

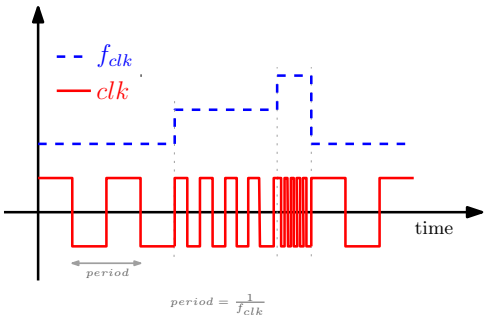


Figure 2.8: Real (not modeled) behavior of the oscillator.

2.1.3 The monocore controller

This is the main component in a closed-loop architecture. Indeed, the controller aims at calculating the control signals, i.e. the voltage level $V_{level}(s)$ and the frequency level $f_{level}(s)$ in the present case, which minimize the energy consumption while guaranteeing good computational

performance. A strategy is required to control the energy-performance tradeoff of the chip - denoted the *monocore system* - composed of the electronic device and the two actuators. This block merging was firstly highlighted in figure 2.1 (with the doted line) and eventually is gathered in figure 2.9.

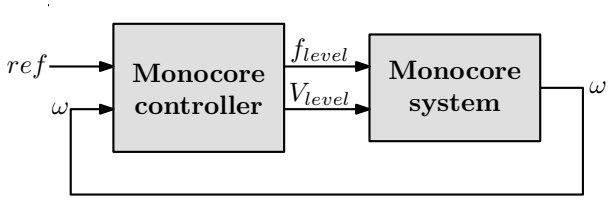


Figure 2.9: Details of the architecture: the monocore controller.

The complete system model is given by the relation $\omega(s) \simeq \alpha \cdot \gamma \cdot f(s) \cdot V_{dd}(s)$. This is obtained from the behaviors depicted above in equations (2.1), (2.2) and (2.3). It can finally be approximated by an affine function with respect to the expected frequency and the supply voltage, that is

$$\omega(s) \simeq \mu \cdot f(s) \cdot V_{dd}(s) \tag{2.4}$$

where $\mu = \alpha \cdot \gamma$ is a positive constant by construction.

Note that the controller intends to minimize the energy consumption and yet, two power sources exist in such a system:

$$P_{monocore\ system}(s) = P_{circuit}(s) + P_{hopping}(s) \tag{2.5}$$
$$\text{with} \quad \left| \begin{array}{l} P_{circuit}(s) = K_{dyn} \cdot f_{clk}(s) \cdot V_{dd}(s)^2 + K_{sc} \cdot f_{clk}(s) \cdot V_{dd}(s) + K_{leak} \cdot V_{dd}(s) \\ P_{hopping}(s) = K_{hopp} \cdot P_{circuit}(s) \end{array} \right.$$

- The main source of consumption comes from the CMOS technology of the electronic components (as explained in section 1.3) where the different parameters K_{dyn}, K_{sc} and K_{leak} belong to the circuit. Reducing the voltage and the frequency will hence decrease the energy consumption but with an increase of the delay in return.
- The Vdd-hopping is also consuming, in particular during the voltage transitions. This is notably detailed in [48]: the parameter K_{hopp} is higher during the transitions (about 20 %) than during the steady-state intervals (about 3 %) and a small number of voltage transitions will hence save the energy consumption.

As a result, the strategy to control the energy-performance tradeoff of a single device consists in reducing the energy consumption, minimizing the high voltage running time and the voltage transitions, while guaranteeing some good computational performance, fitting the tasks with their deadline thanks to the available frequency range. To do that, the two control variables are the voltage level $V_{level}(t)$ and the frequency level $f_{level}(t)$ which directly control the actuators. On the other hand, the current computational speed $\omega(t)$ (how many instructions are done per second) is measured and compared to the computational load reference $ref(t)$ (given by the operating system for each task to treat). Several strategies are then proposed. In section 2.2, a simple control of the frequency and the voltage level is thus detailed. Then, in sections 2.3 and 2.4, a predictive control law leads to save more energy by minimizing the high voltage running time, for a system with two and M voltage levels respectively.

2.2 Frequency and voltage level control

As explained in subsection 2.1.3, the objective is to reduce the energy consumption while guaranteeing good computational performance. This is apparently an energy-performance tradeoff. We previously defined the two control variables - in section 2.1 - and also explained that both the frequency and the voltage levels have to be controlled together. Indeed, as explained in section 1.3, the clock frequency has to be decreased before reducing the supply voltage - in order to ensure the maximum delay over the electronic critical path of the circuit - and inversely, the voltage has to be increased before growing the frequency. These rules have also to be applied to the control variables. As a consequence, **we propose to establish a frequency/voltage level control law in order to handle the energy-performance tradeoff**. The resulting architecture is drawn in figure 2.10, where the *frequency and voltage level controller* compares the *measured computational speed* $\omega(t)$ with a given reference. In this section, the reference is the *average computational speed setpoint* $\overline{\omega_{sp}}(t)$, that is the ratio between a given number of instructions to treat $\Omega_i(t)$ and the given amount of time to do that $\Delta_i(t)$ for each task T_i. This yields $\overline{\omega_{sp}}(t) = {}^{\Omega_i(t)}/_{\Delta_i(t)}$. One could note that the different parameters are piecewise defined since they are constant for a given task but change for each task. This will be explained in detail in subsection 2.3.1 where another and better reference is proposed. At the end, the difference between the measurement and the setpoint allows to calculate the *frequency level* $f(t)$ and the *voltage level* $V_{level}(t)$.

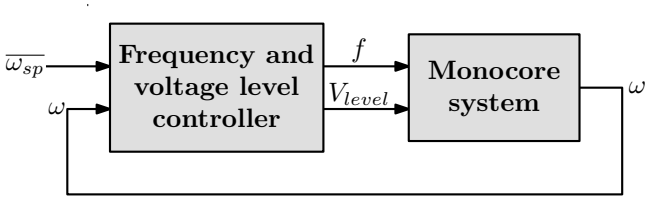


Figure 2.10: Closed-loop system: the frequency and voltage level controller.

In spite of the fact that the two control variables have to be calculated together, we deliberately divide the section into several parts: firstly, the frequency is calculated in subsection 2.2.1 in such a way that the measured speed tracks the setpoint and secondly, the voltage level is chosen in subsection 2.2.2 to reduce the energy consumption. The two variables are eventually adapted together in subsection 2.2.3 to ensure the critical path, before sending them to the actuators. At the end, a global algorithm summarizes all the control strategy in subsection 2.2.4.

Note that the frequency is continuously varying in this section, i.e. $f(t) = f_{level}(t)$, as explained in subsection 2.1.2.2. The *expected frequency* $f(t)$ is thus used in this section instead of the frequency level $f_{level}(t)$. Moreover, for the same reasons than explained in section 2.1, we afterwards use the Laplace domain to design the frequency control law.

2.2.1 Frequency control

The closed-loop system to control only the frequency is given in figure 2.11, where $C_f(s)$ is the *frequency controller* used to calculate the expected frequency $f(s)$ which will be sent to the system to control $\Sigma(s)$. The error between the reference and the measurement, that is $\varepsilon(s) = \overline{\omega_{sp}}(s) - \omega(s)$, is used in this way. The well-known closed-loop transfer function, denoted $H_{cl}(s)$, is deduced from this scheme and yields

$$H_{cl}(s) = \frac{\omega(s)}{\overline{\omega_{sp}}(s)} = \frac{C_f(s) \cdot \Sigma(s)}{1 + C_f(s) \cdot \Sigma(s)}$$

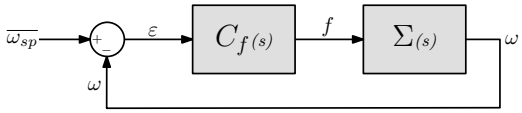
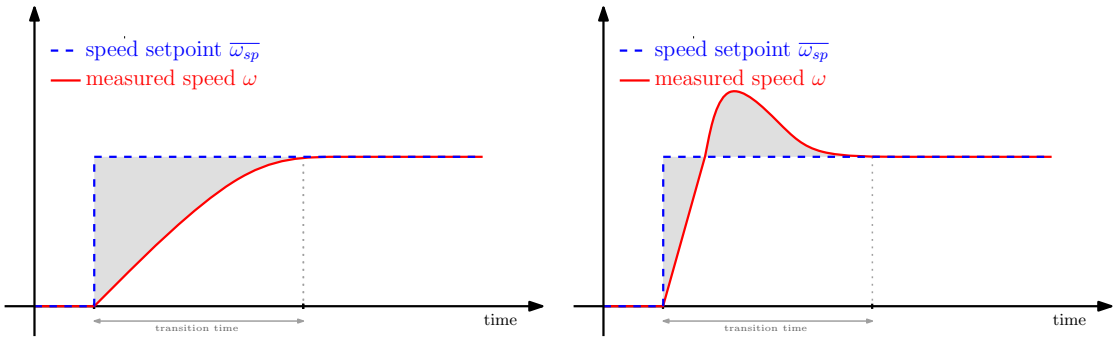


Figure 2.11: Details of the frequency control loop.

The transfer function of the controller $C_f(s)$ can now be calculated so that the system $\Sigma(s)$ correctly runs (executing the tasks to compute in the given amount of time). The current computational speed $\omega(s)$ - the output of the closed-loop system - has hence to track the speed setpoint $\overline{\omega_{sp}}(s)$ - the input - and a null static error between the output and the input is needed. This means $\varepsilon(s) = 0$ after a certain transition time. However, this constraint is not hard enough because the computational speed is lower than the one required during the transition intervals, as highlighted with the darken surface area in figure 2.12(a). Thus, even if after the transition time the measured computational speed tracks the given reference, finally the processor did not compute as many instructions as it would do to correctly execute the task. To solve this problem, the frequency controller has to compensate such a behavior by running faster during a given amount of time, and so will be made up on the delay. This is depicted with the darken area in figure 2.12(b). The area where the measured speed is below the reference has finally to be equal to the one where the measurement is above the speed setpoint in such a way that the average measured computational speed is finally equal to the speed setpoint. This intuitive principle can be mathematically translated as rendering null the integral of the static error.



(a) Null static error

(b) Null static error's integral

Figure 2.12: Frequency control: two possible methods to track a given computational speed setpoint.

To summarize, the frequency controller has to ensure that ***i)*** *the static error is null* in order the system output tracks the input during the steady-state intervals and ***ii)*** *the integral of the error is null* to offset the missed computations during the transition intervals. The error $\varepsilon(s)$ and the error's integral $E(s)$ are defined as

$$\begin{aligned}
\varepsilon(s) &= \overline{\omega_{sp}}(s) - \omega(s) \\
&= \overline{\omega_{sp}}(s) \cdot \Big(1 - H_{cl}(s)\Big) \\
&= \overline{\omega_{sp}}(s) \cdot \frac{1}{1 + C_f(s) \cdot \Sigma(s)} \\
E(s) &= \frac{1}{s} \cdot \varepsilon(s)
\end{aligned}$$

Splitting the frequency controller into a polynomial fraction, i.e. $C_f(s) = \frac{C_f^{num}(s)}{C_f^{den}(s)}$, leads to

$$\varepsilon(s) = \overline{\omega_{sp}}(s) \cdot \frac{C_f^{den}(s)}{C_f^{den}(s) + C_f^{num}(s) \cdot \Sigma(s)}$$

Then, the so-called "final value theorem" ensures the above ***i)*** and ***ii)*** conditions. This theorem, in mathematical analysis, is used to relate frequency domain expressions to the time domain behavior as time approaches infinity. It allows the time domain behavior to be directly calculated by taking a limit of a frequency domain expression, as opposed to converting to a time domain expression and taking its limit. In our case, we need to have

$$\begin{aligned} \lim_{t\to\infty} \varepsilon(t) &= \lim_{s\to 0} s \cdot \varepsilon(s) = 0 \\ \lim_{t\to\infty} E(t) &= \lim_{s\to 0} s \cdot E(s) = 0 \end{aligned}$$

Using the previous error expression and the Laplace transform of an unit step response, i.e. $\overline{\omega_{sp}}(s) = {}^1\!/_s$, the theorem results become

$$\begin{aligned} \lim_{s\to 0} \frac{C_f^{den}(s)}{C_f^{den}(s) + C_f^{num}(s) \cdot \Sigma(s)} &= 0 \\ \lim_{s\to 0} \frac{\frac{1}{s} \cdot C_f^{den}(s)}{C_f^{den}(s) + C_f^{num}(s) \cdot \Sigma(s)} &= 0 \end{aligned}$$

which finally allow to conclude on the parameters of the frequency controller $C_f(s)$ as follows

$$\begin{aligned} & C_f^{den}(s) = s^2 \cdot A(s) \\ & C_f^{num}(s) \cdot \Sigma(s) = B(s) \\ & \text{with} \left| \begin{array}{lll} A(s) = a_0 + a_1 \cdot s + a_2 \cdot s^2 + \dots & \forall & a_0, a_1, a_2, \dots \\ B(s) = b_0 + b_1 \cdot s + b_2 \cdot s^2 + \dots & \forall & b_0 \neq 0, b_1, b_2, \dots \end{array} \right. \end{aligned} \tag{2.6}$$

The transfer function of the system to control can be seen as a *varying gain* $\sigma(.)$ with respect to the expected frequency, i.e. $\Sigma(s) = \frac{\omega(s)}{f(s)} \simeq \sigma(s)$. This comes from the system model defined in subsection 2.1.3. Actually, the varying gain is function of the supply voltage since $\sigma(s) = \mu \cdot V_{dd}(s)$, from equation (2.4). Moreover, the system can be considered as a varying gain because the voltage slowly varies due to the Vdd-hopping principle (see subsection 2.1.2.1 for further details). Conclusively, we can hence choose some values for the different parameters of $A(.)$ and $B(.)$ described in equation (2.6), and one of the numerous possibilities is settled to define the frequency controller. This yields

$$\begin{aligned} C_f(s) &= \frac{b_0 + b_1 \cdot s}{a_0 \cdot \sigma(s) \cdot s^2} \\ &= \frac{1}{s} \cdot \frac{1}{\sigma(s)} \cdot \left(K_p + \frac{K_i}{s} \right) \qquad \text{with} \left| \begin{array}{l} K_p = {}^{b_1}\!/_{a_0} \\ K_i = {}^{b_0}\!/_{a_0} \end{array} \right. \end{aligned} \tag{2.7}$$

At the end, **the proposal is a simple proportional integral controller, with a varying gain, applied to the integral of the error and not directly to the error**. The parameters K_p and K_i are the proportional and the integral tunable gains respectively. A discrete-time

frequency controller is then deduced (using the backward difference approximation) where the frequency varies with respect to the measured error, that is

$$\begin{aligned} E(t_k) &= E(t_{k-1}) + T_s \cdot \varepsilon(t_k) \\ f_p(t_k) &= \frac{1}{\sigma(t_k)} \cdot K_p \cdot E(t_k) \\ f_i(t_k) &= f_i(t_{k-1}) + T_s \cdot \frac{1}{\sigma(t_k)} \cdot K_i \cdot E(t_k) \\ f(t_k) &= f_p(t_k) + f_i(t_k) \end{aligned} \tag{2.8}$$

where T_s is the sampling period and $E(.)$ *is the discrete integration of the error* $\varepsilon(.)$, while $f_p(.)$ and $f_i(.)$ are respectively the proportional and the integral parts of the PI controller.

One could note that the varying gain $\sigma(\cdot)$ is required in the frequency controller equation because in practice, nothing more than this gain can be measured - when measured - due to space and time dispersion. Indeed, an approximation of the controlled system equation - modeled in subsection 2.1.3 - is the varying gain $\sigma(s) = \mu \cdot V_{dd}(s)$, where μ is neither known nor directly measurable in practice. However, a trick consists in using the $\omega(.)$ and $f(.)$ signals - from the system transfer function definition, i.e. $\Sigma(s) = \frac{\omega(s)}{f(s)} \simeq \sigma(s)$ - which are themselves measurable. This yields

$$\sigma(t_k) \simeq \frac{\omega(t_k)}{f(t_k)}$$

where the current value of the speed is measured but the one of the frequency is not calculated yet. Consequently, **we propose to use the previous value of the frequency to calculate the varying gain**. The equation becomes

$$\sigma(t_k) \simeq \frac{\omega(t_k)}{f(t_{k-1})} \tag{2.9}$$

Finally, since the oscillator (introduced in subsection 2.1.2.2) can only provide a given frequency range, **we propose to apply an anti-windup mechanism** in order to prevent windup when the actuator is saturated. This consists in adding an extra term in the integral part of the control law previously defined in equation (2.8), which is

$$f_i(t_k) = \ldots - T_s \cdot K_a \cdot \Big(f(t_{k-1}) - f_{sat}(t_{k-1}) \Big) \tag{2.10}$$

where K_a is a tunable parameter while $f_{sat}(.)$ is the saturated value of $f(.)$ (calculated at the previous sampling time, regarding the frequency available range afterwards defined in the following subsection).

2.2.2 Voltage level control

The voltage level can be easily deduced when knowing the expected frequency. Indeed, the voltage scalable device can run with a given frequency range for a given voltage, and the minimum voltage level will reduce the energy consumption so far as possible, as explained in section 1.3. Basing our approach on this idea, a clock frequency range is hence available for each supply voltage level. This is drawn in figure 2.13.

- A functional area is defined for each voltage level - two levels in our case, that are V_{high} and V_{low} respectively - with a corresponding frequency range, while the electronic circuit does not work in the darken non-functional area.

- The minimal frequency F_{min} comes from the oscillator designing (see subsection 2.1.2.2 for further details). Whatever the voltage level the electronic device could run at this minimum frequency without malfunctioning, the system simply runs very slowly.
- On the other hand, the maximal frequencies have to ensure the maximum delay over the critical path of the circuit and, therefore, are function of the voltage level. Thus, the *maximal frequency at high voltage* $F_{V_{high}max}$ is calculated for a given circuit in order to ensure the maximum delay when this circuit is power supplied with V_{high} and this maximal frequency. Respectively, $F_{V_{low}max}$ *is the maximal frequency at low voltage*. By not overshooting this maximal frequencies when running at the corresponding voltage will assure that the computations are correct. A key point for the *voltage level controller* is hence to verify that.

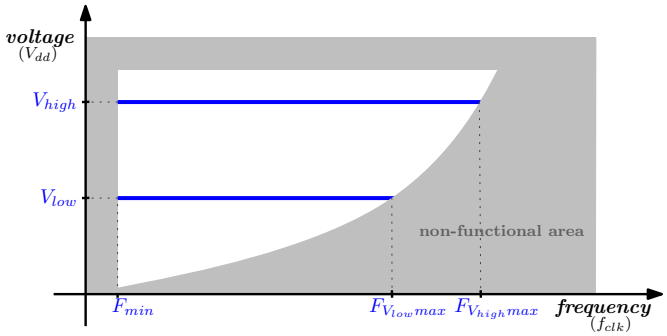


Figure 2.13: Voltage scalable device: representation of the possible frequency range for each voltage level.

We also explained in the state of the art (in section 1.3) that the high voltage level is the most energy consuming one. For this reason, **we propose to reduce the frequency range available at the high voltage level** in such a way that the system runs more often at the low voltage. The frequency range hence becomes

$$\begin{aligned} F_{V_{low}min} &\leq f_{clk}(t) \leq F_{V_{low}max} \quad \text{if} \quad V_{dd}(t) = V_{low} \\ F_{V_{high}min} &\leq f_{clk}(t) \leq F_{V_{high}max} \quad \text{if} \quad V_{dd}(t) = V_{high} \end{aligned} \tag{2.11}$$

where $F_{V_{low}min}$ *and* $F_{V_{high}min}$ *are the minimal frequencies at low and high voltage* respectively (note that $F_{V_{low}min}$ could still be equal to F_{min}, the minimal possible frequency). By directly using this new range definition, **we propose an hysteresis function to describe the direct relation between the frequency and the voltage**. This is drawn in figure 2.14. The depicted principle avoids to run at high voltage when the frequency is low, i.e. lower than $F_{V_{high}min}$, in order to reduce the energy consumption. However, one could note that the clock frequency $f_{clk}(\cdot)$ and the supply voltage $V_{dd}(\cdot)$ are represented on the figure, whereas the control variables are the expected frequency $f(\cdot)$ and the voltage level $V_{level}(\cdot)$. The principle has hence to be adapted to the control variables and yet, we know that the expected frequency is equal to the clock frequency, that is $f(t) = f_{level}(t)$ in this continuously varying frequency scheme. Moreover, the supply voltage slowly varies, as explained in subsection 2.1.2. Consequently, the relation between the variables which act to the actuators can be transposed to the control ones. Therefore, the voltage level directly varies with respect to the calculated frequency, using the hysteresis behavior depicted in figure 2.14. Thus, if the expected frequency becomes above a certain value $F_{V_{low}max}$ the voltage level to apply is the high one, i.e. V_{level_high}, and when the frequency goes below the $F_{V_{high}min}$ value the voltage level can go back to V_{level_low}. The

equation which summarizes this behavior is

$$V_{level}(t_k) = \begin{cases} V_{level\_high} & \text{if } f(t_k) \geq F_{V_{low}max} \text{ and } f(t_{k-1}) < F_{V_{low}max} \\ V_{level\_low} & \text{if } f(t_k) \leq F_{V_{high}min} \text{ and } f(t_{k-1}) > F_{V_{high}min} \\ V_{level}(t_{k-1}) & \text{otherwise} \end{cases} \quad (2.12)$$

A problem could still occur when going from one level to the other because of the critical path. As a result, we depict several strategies to control the voltage transitions in the following subsection.

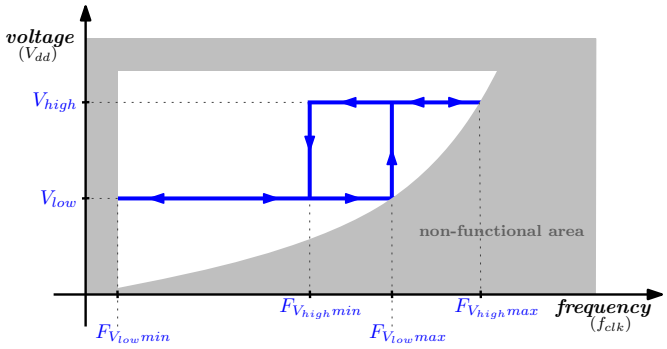


Figure 2.14: Voltage scalable device: a hysteresis function used to adjust the supply voltage with the clock frequency.

An important thing to remark in the previous paragraph is that the minimal frequency $F_{V_{low}min}$ and the maximal ones $F_{V_{low}max}$ and $F_{V_{high}max}$ directly comes from the electronic circuit whereas the $F_{V_{high}min}$ parameter is chosen by the control designer. Actually, this value defines the window's width of the hysteresis. It allows to reduce more or less the energy consumption while increasing the number of voltage transitions in return (which consume too, as explained in subsection 2.1.3). Indeed, with a value close to $F_{V_{low}max}$, the voltage often changes from V_{high} to V_{low} and so is increased the number of voltage switches, but the consumption is quite reduced. Inversely, moving away this parameter reduces the transition number but the energy is not saved as much as before. The choice of this parameter is hence not harmless. Eventually, $F_{V_{high}min}$ could be chosen regarding some specifications and/or the application to execute on the chip.

2.2.3 Guarantee of the maximum delay over the critical path

The Vdd-hopping mechanism requires a given transition time to go from one voltage level to another (as explained in subsection 2.1.2.1), and yet, the maximum delay over the critical path has to be guaranteed during the voltage transitions. Indeed, it is required to decrease the frequency before decreasing the voltage and, respectively, to increase the voltage before increasing the frequency (see subsection 1.3.2 for further details). The voltage transition could hence causes problems, in particular when the system goes from V_{low} to V_{high}. There is no really problem when the voltage is decreased because the frequency - which is already lower than $F_{V_{high}min}$ when the controller decides to change the voltage level thanks to equation (2.12) - works for both voltage levels and so is ensured the maximum possible delay. Moreover, the oscillator is power supplied by the voltage - as explained in subsection 2.1.2.2 - and the frequency varies with respect to that variable. As a result, **we propose different solutions to manage the rising voltage transitions** in the following subsections.

2.2.3.1 Frequency restriction during the voltage transitions

An intuitive solution is to lock the frequency at $F_{V_{low}max}$ as long as the high voltage level is not achieved during a rising transition. This extra constraint can be considered as a saturation. The resulting equation which summarizes that - transposing the principle to the expected frequency - is

$$f(t_k) = F_{V_{low}max} \quad \text{if} \quad f(t_k) \geq F_{V_{low}max} \quad \text{and} \quad V_{dd}(t_k) < V_{high} \tag{2.13}$$

This condition implicates to measure the supply voltage $V_{dd}(.)$. Actually, one only needs to know if the high voltage or low voltage is achieved. That is, a simple discrete signal can be sent from the Vdd-hopping rather than measuring the current value of the voltage. Thus, a signal - which defines the *voltage state* $V_{state}(.)$ - would be equal to a given value V_{state_high} when $V_{dd}(t) = V_{high}$, V_{state_low} when $V_{dd}(t) = V_{low}$ and $V_{state\_transition}$ otherwise, for instance. Eventually, the frequency restriction hence becomes

$$f(t_k) = F_{V_{low}max} \quad \text{if} \quad f(t_k) \geq F_{V_{low}max} \quad \text{and} \quad V_{state}(t_k) \neq V_{state\_high} \tag{2.14}$$

2.2.3.2 Voltage measurement for a maximum gain

An alternative to the previous proposal is to make the frequency varying during the voltage transitions. Consequently, we suggest that the frequency linearly varies between the maximal frequency at low voltage $F_{V_{low}max}$ and the maximal one at high voltage $F_{V_{high}max}$ during a rising transition (and inversely during a falling transition), as represented in figure 2.15. This behavior is possible because the non-functional area is convex and, therefore, the maximum delay over the critical path cannot be violated.

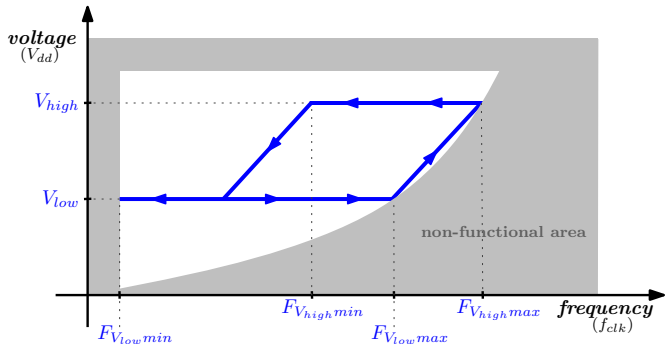


Figure 2.15: Voltage scalable device: improvement of the hysteresis function.

The previous constraint in equation (2.13) is hence modified in order to add the linear relation between both variables. It becomes

$$f(t_k) = a \cdot V_{dd}(t_k) + b \quad \text{if} \quad f(t_k) \geq F_{V_{low}max} \quad \text{and} \quad V_{dd}(t_k) < V_{high} \tag{2.15}$$

$$\text{with} \quad \left| \begin{array}{l} a = \dfrac{F_{V_{high}max} - F_{V_{low}max}}{V_{high} - V_{low}} \\ b = F_{V_{high}max} - a \cdot V_{high} \end{array} \right.$$

Of course the same thing can be done for the falling transitions - as represented on the figure - but that is not inevitably required. However, the effect in using that method is to maximize the computational job. Thus, as many instructions as possible are executed when the system runs at the penalizing high voltage (or at least a voltage higher than V_{low}). This will save energy

since the load will be reduced faster. Otherwise, the falling transition does not need a safety condition (as explained in the previous subsection).

Note that this second restriction on the frequency during the voltage transitions allows to optimize the computational speed of the device but needs the voltage measurement. Furthermore, this measurement is required anyway since it is applied in the control law in equation (2.15), and using the voltage state $V_{state}(t_k)$ - as previously suggested in subsection 2.2.3.1 - does not offer any interest here.

2.2.3.3 Self-management from the oscillator

The last proposal consists in an hardware solution where, in fact, the oscillator directly manages the voltage transitions on its own. Indeed, the oscillator (introduced in subsection 2.1.2.2) is power supplied by the supply voltage $V_{dd}(\cdot)$ and one can imagine that this device automatically restricts the frequency with respect to the voltage, as defined in equation (2.13), (2.14) or (2.15). In this case, ensuring the critical path does not have to be taken into account in the control law and, as a result, this highly simplify the algorithm. For this reason, this solution will be applied in the following control strategies (in section 2.3 and 2.4).

2.2.4 Control algorithm

Eventually, the control strategy consists in scaling the frequency in such a way that the measured computational speed tracks a given speed setpoint, to fit the tasks to treat with their

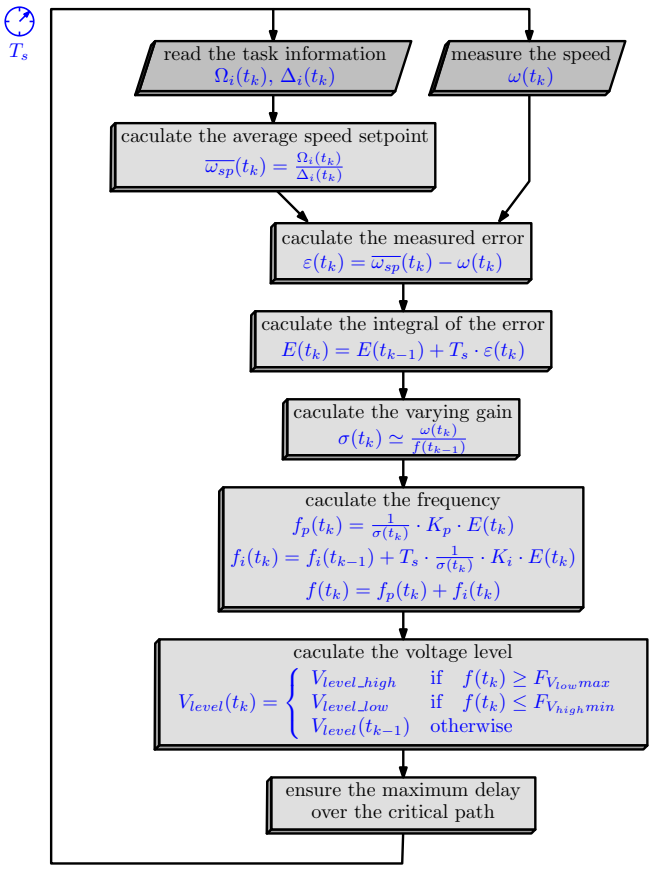


Figure 2.16: Algorithm: the frequency and voltage level controller.

deadline (see subsection 2.2.1 for further details). The lower voltage level is then applied as often as possible rearding a hysteresis function, that is as soon as the computational speed is low, in order to reduce the energy consumption since this parameter is very penalizing (see subsection 2.2.2). At the end, both control variables are adapted together and, if necessary, a restriction is set in order to ensure the maximum delay over the critical path (see subsection 2.2.3). The resulting algorithm is represented in figure 2.16.

2.3 Computational speed control

We previously presented, in section 2.2, a strategy to control the frequency and the voltage level of an electronic system. We explained in particular that, in order to reduce the energy consumption while guaranteeing some good computational performance, the penalizing high voltage running time has to be minimized while the measured computational speed has to track a given speed setpoint. However, in a first hand we used an intuitive reference whereas a more complex one would be better. For this reason, **we propose to improve the control architecture in order to reduce even more the energy consumption**. The structure of the *monocore controller* - initially introduced in section 2.1 and then detailed in section 2.2 for the intuitive control strategy - is hence modified and the new proposal can now be divided into two parts. These parts are also represented in figure 2.17.

The computational speed controller provides an energy-efficient computational speed setpoint. Thus from some task information - the number of instructions $\Omega_i(t)$ and the deadline $\Delta_i(t)$ (these variables are more precisely defined in the next subsection) - given by the operating system for each task to treat, this part of the controller calculates the best speed reference $\omega_{sp}(t)$ which minimizes the energy consumption while guaranteeing the computational performance.

The frequency and voltage level controller fits the computational speed $\omega(t)$ of the electronic system with the speed setpoint $\omega_{sp}(t)$ (calculated by the first part), by adapting the control variables, that are the frequency level $f_{level}(t)$ and the voltage level $V_{level}(t)$.

One could note that the frequency is still continuously varying in this section, that is $f(t) = f_{level}(t)$ as explained in subsection 2.1.2.2, this is why the expected frequency $f(t)$ is used afterwards - and in figure 2.17 - instead of $f_{level}(t)$.

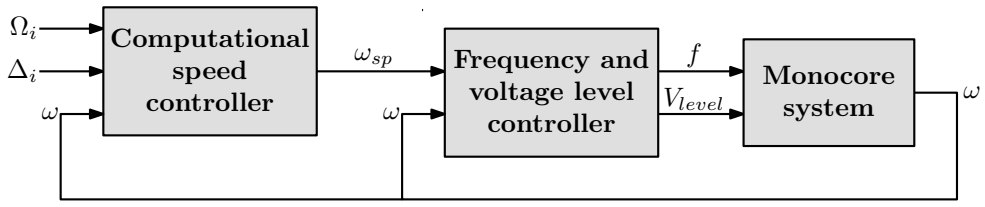


Figure 2.17: Closed-loop system: the computational speed controller (using the measured computational speed as a feedback).

Two control architectures are proposed: the first one where the dynamical feedback loop of the computational speed controller is realized with the measured computational speed $\omega(t)$, as shown on the previous figure, while in the second architecture the closed-loop is done thanks to the calculated speed setpoint $\omega_{sp}(t)$. We begin presenting some different speed setpoint buildings in subsection 2.3.1 while the two control schemes are detailed in subsection 2.3.2 and subsection 2.3.3 respectively. Then, the way to know the maximum computational speeds - required to build an energy-efficient setpoint - is explained in subsection 2.3.4. The frequency and voltage

level control law is adapted (from the intuitive control in section 2.2) to the new architectures in subsection 2.3.5. Finally, the general control algorithm is given in subsection 2.3.6.

2.3.1 Computational speed setpoint building

In order to calculate a *computational speed setpoint* $\omega_{sp}(t)$ (in number of instructions per second) some task information - backwards denoted $ref(t)$ - are required. Indeed, for each task T_i to treat the operating system provides some data to the controller regarding the computational load: the computations to treat and the amount of time before the end of the task, also called the *number of instructions* $\Omega_i(t)$ and the *deadline* $\Delta_i(t)$ respectively. In fact, the remaining available time to execute the task - that is the *laxity* $\Lambda_i(t)$ - will be used instead of the deadline which is an absolute time. From a point of view of the electronic chip, these data correspond to some variables: they are the piecewise defined functions which respectively correspond to the instruction number, the deadline and the laxity of the running task at the given time t. One could note moreover that these parameters can change during the running time of a task if the operating system decides to update them for some reasons, this is why they are time-dependant. Finally, these variables are shown in figure 2.18 for a three-task example, where t_i in time-coordinate is the beginning time of the task T_i. The idea is now to use these data as well as possible in order to build an energy-efficient speed setpoint.

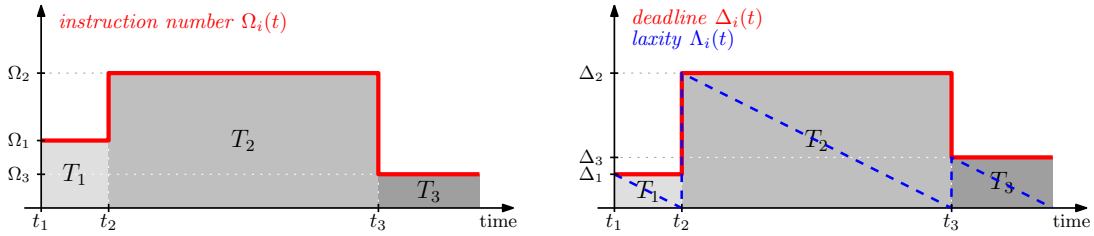


Figure 2.18: References sent by the operating system for each task T_i to compute: the instruction number Ω_i and the deadline Δ_i (or the laxity Λ_i).

Actually, the presence of deadline and time horizon to compute the tasks naturally leads to a predictive control strategy. Indeed, a predictive control consists in finding an open-loop control profile over some time horizons and applying it until the next sampling time instant. The control problem is then reconsidered using the new state variables and a new control profile is generated. At the end, this yields a closed-loop control where the stability relies in the way that the open-loop control is chosen. The most classical strategy consists in taking the open-loop control that minimizes some cost functions. The horizon can be constant, infinite or less classically, a *contractive horizon* like in the present case. This notion means that the task horizon decreases until achieving the deadline, then a new horizon has to be achieved again for a new task and so on. For further details on predictive control, one can refer to [46]. Note that the key point of the present control problem is the choice of the open-loop strategy and its computational cost. Indeed, if the predictive control is known to be a robust approach, that is also often associated to high computational cost which is not acceptable in an embedded chip. However, the strategy adopted here is called *fast predictive control* and consists in taking advantage of the structure of the dynamical system to fasten the search of the open-loop control [3]. The simplicity of the system considered here is therefore very suitable for such strategies. The rest of the current subsection explains intuitively the predictive strategy whereas its formal expression is given in subsection 2.3.2 and 2.3.3 for two different closed-loop architectures.

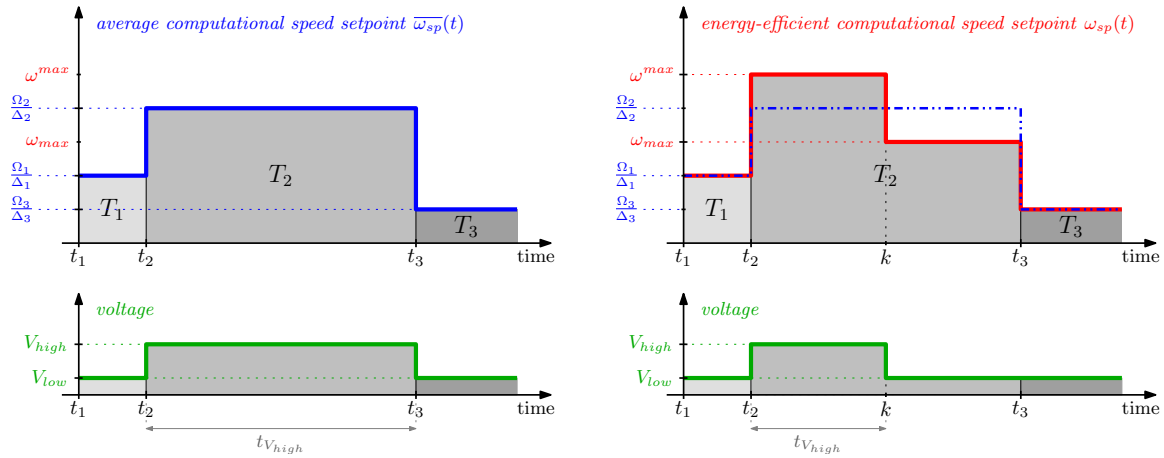
2.3.1.1 The intuitive average speed setpoint

An intuitive reference is the *average computational speed setpoint* $\overline{\omega_{sp}}(t)$ (the one previously used in section 2.2). This average speed is, for each task T_i, the ratio between the instruction number to compute and the amount of time to do it, that is $\overline{\omega_{sp}}(t) = {}^{\Omega_i(t)}/_{\Delta_i(t)}$. This intuitive setpoint is very simple to compute from the given data. Moreover, it leads to reduce the energy consumption by allowing the system to run sometimes at the low voltage level. One could see in figure 2.19(a) that some maximal possible speeds are indicated. Thus, ω^{max} *(superscript) refers to the maximum computational speed when the system is running at high voltage*. It is defined such that $\omega^{max} = \mu \cdot F_{V_{high}max} \cdot V_{high}$ from the system model in equation (2.4). Respectively, ω_{max} *(subscript) is the maximum possible speed at low voltage* and is defined as $\omega_{max} = \mu \cdot F_{V_{low}max} \cdot V_{low}$. Note that how to measure these maximum speeds will be explained in subsection 2.3.4. As a result, one can easily imagine that a task could be computed at V_{low} when its average speed setpoint is lower than ω_{max}, such as for the tasks T_1 and T_3 on the example, and will run at V_{high} otherwise, such as for T_2. Finally, this intuitive behavior reduces the energy consumption but it could be improved again by reducing even more the high voltage running time. This is really the key point in minimizing the consumption because of the (quasi)-quadratic relationship between both variables, as explained in section 1.3.

2.3.1.2 A more energy-efficient reference

A better solution consists in avoiding the running of a whole task at the penalizing high supply voltage level. In fact, a task whose average speed setpoint is higher than ω_{max} could be divided into two parts, a highly consuming and a lower consuming part, which have to be intelligently scheduled. Such an example is shown in figure 2.19(b) for the case of the task T_2. The idea is to begin the execution of the task at V_{high} (because this level is needed anyway to be able to perform the task before its deadline). However, as this first part will consume a lot because of its running at the penalizing high voltage, it has to be as short as possible by running with the maximal possible speed, that is ω^{max} at high voltage. Therefore, **we propose to apply the maximum possible speed ω^{max} when the system is running at the penalizing high voltage** in order to go as fast as possible. Consequently, after a given high voltage and maximum speed running time, the executed computational load will be advanced enough and the system could run back to the low voltage level, and so begins the second part. This lower consuming part will hence perform the end of the task at V_{low} and with a low computational speed. To sum up, the energy-efficient method performs a task by minimizing the penalizing high voltage running time. In other words, a task with an important computational load (such as T_2) will be executed at V_{high} and the maximal speed ω^{max}, from its beginning (t_2) until a certain amount of time (k). Then the task could be finished at V_{low} and a speed under ω_{max}, which will be enough to fit it with its deadline (t_3). At the end, the high voltage running time $t_{V_{high}}$ is strongly reduced - and so is the energy consumption - while guaranteeing the same computational performance than with the intuitive building. Indeed, the computational load used to treat a given task is equal with both strategies since the darken surface areas of the task T_2 are respectively the same on both drawings in figure 2.19. Nevertheless, the time k could not be *a priori* known and therefore, a predictive control law is required to dynamically calculate this switching time. That point will be detailed in the following subsections. Furthermore, the task information given by the operating system are not enough anymore to build such an energy-efficient computational speed setpoint. We also need some information about the system resources, such as the maximum speed for the different voltage levels, i.e. ω^{max} and ω_{max}. Moreover, we need to know what it has already been done, and for this reason **we propose to use a closed-loop system in order to predict the minimum high voltage running time**. Two feedbacks are hence suggested,

that are ***i)*** *the measured computational speed* and ***ii)*** *the computational speed setpoint.* Both schemes are detailed in subsections 2.3.2 and 2.3.3 respectively.



(a) Intuitive average speed setpoint building

(b) Energy-efficient speed setpoint building

Figure 2.19: Computational speed setpoint: different buildings and their impact on the energy consumption.

2.3.2 Fast predictive control using the measured speed as a feedback

In this proposal the computational speed controller firstly uses the measured speed as a feedback, as represented in figure 2.17, in order to calculate the energy-efficient speed setpoint depicted in subsection 2.3.1.2. Thus, as previously explained, the energy consumption is minimized by running the shortest possible amount of time with the penalizing high supply voltage. To do that, the controller has to dynamically calculate if the system needs to run at V_{high} - and at the maximum possible speed ω^{max} - or if the low voltage level - and a speed lower than the maximal speed at V_{low}, that is ω_{max} - will be enough to compute the task before its deadline. This principle could be formulated as an optimization problem: for each task T_i, what is the computational speed setpoint which minimizes the high voltage running time $t_{V_{high}}$ while guaranteeing that the executed instruction number is equal to the number of instructions to do, which yields

$$\min \; t_{V_{high}} \qquad \text{s.t.} \qquad \int_{\Delta_i(t)} \omega(t) \; dt = \Omega_i(t) \tag{2.16}$$

where $\int \omega(t) \; dt$ is the integral of the measured computational speed, which corresponds to the executed number of instructions for the current task. On the other hand, $\Omega_i(t)$ is the number of instructions to do and $\Delta_i(t)$ is the given amount of time to treat the task, i.e. its deadline. Note that $\Omega_i(.)$ and $\Delta_i(.)$ - and also the laxity $\Lambda_i(.)$ - are time-dependant for the reasons already explained in subsection 2.3.1. The optimization criteria in equation (2.16) allows to solve the predictive problem but is too complex to be implemented in an electronic chip with low resources, as in the present case. Nevertheless, the closed-loop solution yields an easier and faster algorithm. Indeed, one simply needs to know ***i)*** *what the processor has to do* and ***ii)*** *how much time is available to do it.* As the speed setpoint is dynamically calculated the remaining time before the end of the task is necessary, this is why the laxity $\Lambda_i(t)$ will be used instead of the deadline $\Delta_i(t)$. Eventually, the speed required to perform the task exactly on its deadline - afterwards denoted the *predicted speed* $\delta(.)$ - is calculated at each sampling instant. Then, the

discrete value of $\delta(t)$ can be easily described as the ratio between *what the processor has to do to compute the task* minus *what it has already done* - that is corresponding to *what it remains to do* - and *the remaining time before the end of the task*. In fact this principle can be mathematically expressed as

$$\delta(t_{k+1}) = \frac{\Omega_i(t_k) - \sum_{t_i}^{t_k - t_i} \omega(t_k)}{\Lambda_i(t_k)} \tag{2.17}$$

where t_k and t_{k+1} are respectively the current and the next sampling time, and t_i is the beginning of the task T_i. Finally and in order to be implementable, this equation becomes

$$\Omega(t_k) = \Omega(t_{k-1}) + T_s \cdot \omega(t_k) \tag{2.18a}$$

$$\delta(t_{k+1}) = \frac{\Omega_i(t_k) - \Omega(t_k)}{\Lambda_i(t_k)} \tag{2.18b}$$

where T_s is the sampling period of the controller and $\Omega(.)$ *is the discrete integration of the computational speed* $\omega(.)$ (using the backward difference approximation). Furthermore, a conditional instruction is added to be coherent. Indeed, in equation (2.17) the computational speed is integrated on the running time of each task and so has to be $\Omega(t)$. Thus, when a task is computed - i.e. in the last sampling time before its deadline - the variable $\Omega(t)$ is reset. More precisely, this variable is not set to zero - to prevent the case where the task is not completely executed at its deadline - but $\Omega(t)$ is adjusted with the difference between what it has already been done and what it was required to do, that is

$$\Omega(t_k) = \Omega(t_k) - \Omega_i(t_k) \qquad \text{if} \quad \Lambda_i(t_k) \leq T_s \tag{2.18c}$$

Finally, the predicted speed is very easy to calculate while the optimization problem defined by equation (2.16) was initially not implementable. This so-called *fast predictive control* law is hence a good solution to build the energy-efficient speed setpoint. A scheme of the possible implementation is proposed in figure 2.20 to show the achieved simplicity, where a simple division leads to calculate it. Moreover one could note that this division - which can be too complex in a low-resource electronic device - will be removed in section 2.5 using some tricks.

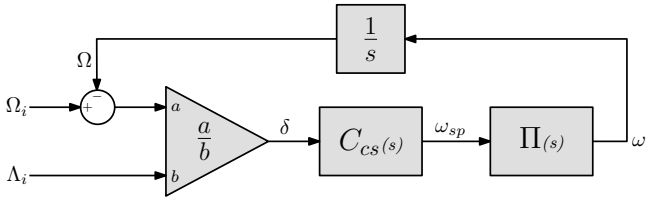


Figure 2.20: Details of the computational speed control.

On the figure, let $\Pi(s)$ be the transfer function between the measurement and the speed setpoint. This block includes the frequency and voltage level controller plus the monocore system drawn in figure 2.17. Eventually, the controller - denoted $C_{cs}(s)$ - simply calculates $\omega_{sp}(t)$ since the computational speed setpoint is easily deduced from the value of $\delta(t)$. Indeed, if the predicted speed is higher than the maximum speed at low voltage, i.e. $\delta(t) > \omega_{max}$, then the system has to run with the penalizing high supply voltage and the maximum speed has to be achieved. On the other hand, as soon as $\delta(t)$ becomes lower than ω_{max}, the system could switch to the low voltage and it will be able to finish the task without going back to V_{high} (if the instruction

number and/or the deadline do not change). Moreover, the current value of $\delta(t)$ allows to fit the task with its deadline. The relation which summarizes this behavior is

$$\omega_{sp}(t_k) = \begin{cases} \omega^{max} & \text{if} \quad \delta(t_{k+1}) > \omega_{max} \\ \delta(t_{k+1}) & \text{otherwise} \end{cases} \tag{2.19}$$

An energy-efficient computational speed setpoint is achieved thanks to this control law and one can ensure that the number of instructions to do will be correctly done because, if the system is slower than required, $\omega_{sp}(t)$ will be dynamically adjusted thanks to the measurement feedback loop. As explained in introduction, the frequency and voltage level controller has to calculate the control variables in such a way that the measured speed tracks the speed setpoint and consequently, this will be very easy because the computational speed controller already compensates for some delays in voltage and/or frequency transitions. This is developed in subsection 2.3.5.

2.3.3 Fast predictive control using the speed setpoint as a feedback

In subsection 2.3.2 we presented a computational speed controller which calculates an energy-efficient speed setpoint using the measured speed as a feedback loop. Here, **we propose to improve again the energy consumption reduction** in closing the system with the calculated computational speed setpoint $\omega_{sp}(t)$ (instead of the speed measurement), as drawn in figure 2.21. This architecture modification directly impacts the optimization criteria presented in equation (2.16) and, consequently, the corresponding algorithm previously detailed in subsection 2.3.2. The control problem becomes

$$\min \ t_{V_{high}} \qquad \text{s.t.} \quad \int_{\Delta_i(t)} \omega_{sp}(t) \ dt = \Omega_i(t) \tag{2.20}$$

where $\int \omega_{sp}(t) \ dt$ is the integral of the computational speed setpoint, which corresponds to the instruction number setpoint. Thus, this criteria tries to minimize the high voltage running time $t_{V_{high}}$ while guaranteeing that the final instruction number required to treat the current task is equal to the number of instructions to do.

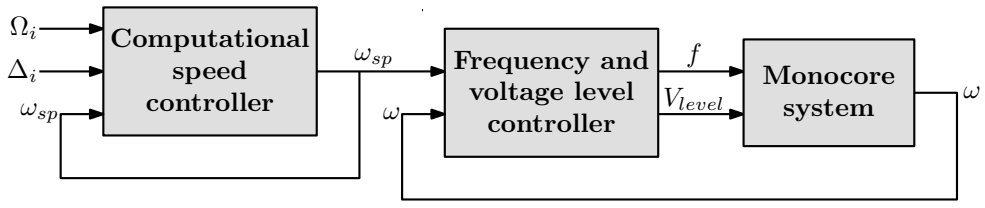


Figure 2.21: Closed-loop system: the computational speed controller (using the computational speed setpoint as a feedback).

In the same way, it is required to replace $\omega(t)$ by $\omega_{sp}(t)$ in the predicted speed defined in equation (2.17) and $\delta(t)$ thus becomes the *predicted speed setpoint* $\delta_{sp}(.)$. Then, the implementable version in equation (2.18) turns into

$$\Omega_{sp}(t_k) = \Omega_{sp}(t_{k-1}) + T_s \cdot \omega_{sp}(t_k) \tag{2.21a}$$

$$\delta_{sp}(t_{k+1}) = \frac{\Omega_i(t_k) - \Omega_{sp}(t_k)}{\Lambda_i(t_k)} \tag{2.21b}$$

$$\Omega_{sp}(t_k) = \Omega_{sp}(t_k) - \Omega_i(t_k) \qquad \text{if} \quad \Lambda_i(t_k) \leq T_s \tag{2.21c}$$

where $\Omega_{sp}(.)$ *is the discrete integration of the computational speed setpoint* $\omega_{sp}(.)$ (still using the backward difference approximation). Then, the principle remains unchanged and the reference - previously obtained in equation (2.19) - becomes

$$\omega_{sp}(t_k) = \begin{cases} \omega^{max} & \text{if} \quad \delta_{sp}(t_{k+1}) > \omega_{max} \\ \delta_{sp}(t_{k+1}) & \text{otherwise} \end{cases} \tag{2.22}$$

At the end, while the previous scheme adapts the speed setpoint with *what it has really been done*, the new proposal does not care anymore about the possible system delay between *what it has been required to do* and *what it has really been done*. This is because the measurement feedback information does not exist anymore. Therefore, the high voltage running time decreases again. Indeed, if one looks at figure 2.19(b), the optimal energy-efficient computational speed setpoint for the task T_2 implies to increase the voltage at time t_2 and decrease it at time k. However, the system speed is not instantaneously equal to ω^{max} at time t_2 because of the transition time required to increase the voltage with the Vdd-hopping (see subsection 2.1.2.1 for further details). As regards the first controller architecture, this delay impacts on $\delta(t)$ and the speed setpoint will finally drop under ω_{max} after the theoretical amount of time k. With the new proposed feedback loop, the rising voltage transition time delay does not impact on $\delta(t)$ and, as a result, the decrease of $\omega_{sp}(t)$ will be exactly at time k. On the other hand, the delay has yet to be considered and the frequency and voltage level controller results in a more complex law to compensate, as detailed next in subsection 2.3.5.

2.3.4 Measurement of the maximum computational speeds

We defined in subsection 2.3.1 the maximum computational speeds at low and high voltage, that are ω_{max} and ω^{max} respectively. These parameters are then useful in the predictive control laws - detailed in subsections 2.3.2 and 2.3.3 - to decide the running supply voltage to apply. However, their value are not *a priori* known and one needs a method to obtain them. In this section, a simple solution would be preferred.

We already explained that the maximum speeds come from the system model given in equation (2.4). This leads to $\omega_{max} = \mu \cdot F_{V_{low}max} \cdot V_{low}$ and $\omega^{max} = \mu \cdot F_{V_{high}max} \cdot V_{high}$. Nevertheless, neither the maximal frequencies nor the μ values are known, and more particularly μ cannot be measured due to space and time dispersion. This parameter belongs to the electronic circuit and could highly vary from one chip to another because of process variability (see section 1.2 for further details). Consequently, **we propose to directly measure the maximum computational speeds**. That can be done off-line - or more precisely during the initialization sequence when the chip starts - running the circuit at low voltage (respectively the high one) and with the maximal frequency just for some measurements. The value of ω_{max} and ω^{max} are eventually stored and then the system could really run and execute some real tasks. Note that this measurements could be done several times during the system running in order to adjust the maximum speed values with respect to environmental/ageing variations (due to the temperature for instance). In this section, we hence choose to simply measure the maximum speeds but a better method - which consists in estimating them - is afterwards developed in subsection 2.4.4.

2.3.5 Frequency and voltage level controller for the new setpoints

As already explained, the frequency and voltage level controller aims at calculating the frequency level $f_{level}(t)$ and the voltage level $V_{level}(t)$ in order to track a given speed setpoint $\omega_{sp}(t)$. We already detailed in section 2.2 a control law for the average one. In this section, **we**

propose to adapt the frequency and voltage level control strategy to a more energy-efficient setpoint (defined in subsection 2.3.1.2). Actually, two closed-loop architectures - which were presented in subsections 2.3.2 and 2.3.3 - lead to two different computational speed setpoints, and two frequency and voltage level control laws are so required. Both solutions are detailed in the end of this subsection.

Computational speed setpoint feedback

With the closed-loop architecture presented in subsection 2.3.3, the speed setpoint is calculated in such a way that the final instruction number required to treat the current task is equal to the number of instructions to do. Thus, the setpoint is not self-adapted to a possible problem in the speed tracking. Indeed, the frequency controller has not only to track the computational speed setpoint anymore, now it has also to compensate for an overshoot or undershoot of the system speed - because there is no measured speed feedback loop (contrary to the measured speed feedback case) - in order to ensure that what it is required to do is really done. For this reason, a complex speed setpoint tracking is required since the frequency controller needs to guarantee a null static error and a null integral of the error too. This was clearly explained in subsection 2.2.1 and eventually results to a proportional integral controller applied to the integral of the error, that is in frequency-domain

$$C_{f_1}(s) = \frac{1}{s} \cdot \frac{1}{\sigma(s)} \cdot \left(K_p + \frac{K_i}{s} \right) \tag{2.23}$$

where K_p and K_i are some tunable parameters (the proportional and the integral parameters) while the variable gain $\sigma(s)$ - which is an approximation of the electronic device - was previously defined in equation (2.9) in the previous section. The corresponding discrete-time controller was given in equation (2.8) and is called back here. An anti-windup mechanism is added. This is

$$\begin{aligned}
\varepsilon(t_k) &= \omega_{sp}(t_k) - \omega(t_k) \\
E(t_k) &= E(t_{k-1}) + T_s \cdot \varepsilon(t_k) \\
\sigma(t_k) &= \frac{\omega(t_k)}{f(t_{k-1})} \\
f_p(t_k) &= \frac{1}{\sigma(t_k)} \cdot K_p \cdot E(t_k) \\
f_i(t_k) &= f_i(t_{k-1}) + T_s \cdot \frac{1}{\sigma(t_k)} \cdot K_i \cdot E(t_k) - T_s \cdot K_a \cdot \Big(f(t_{k-1}) - f_{sat}(t_{k-1}) \Big) \\
f(t_k) &= f_p(t_k) + f_i(t_k)
\end{aligned}$$

where K_a is the anti-windup parameter.

The voltage level is then deduced from the frequency according to the hysteresis behavior represented in figure 2.22. This hysteresis is directly an improvement of the previous proposal drawn in figure 2.15 in subsection 2.2.2. Indeed, only one frequency is now available for the high voltage level V_{high} in order to minimize the penalizing high voltage running and so be coherent with the energy-efficient speed setpoint building (see subsection 2.3.1.2 for further details). Thus, the high voltage level is automatically set when the calculated frequency is higher than the maximum frequency at low voltage level, i.e. $F_{V_{low}max}$. This finally yields

$$V_{level}(t_k) = \begin{cases} V_{level\_high} & \text{if} \quad f(t_k) \geq F_{V_{low}max} \\ V_{level\_low} & \text{otherwise} \end{cases} \tag{2.24}$$

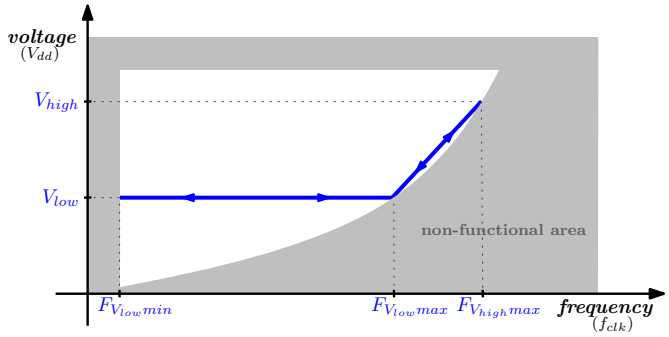


Figure 2.22: Voltage scalable device: new hysteresis function to adjust the voltage with the frequency minimizing the penalizing high voltage running.

Furthermore, as it was initially explained in section 2.2, the two control variables are calculated in two times - the frequency is calculated in such a way that the measurement tracks the setpoint and the voltage level is then chosen to reduce the energy consumption - but, at the end, both are adapted together to ensure the maximum delay over the critical path. Several strategies are possible. They are depicted in subsection 2.2.3 and the linearly varying one is represented in figure 2.22 for instance. Nevertheless, in order to simplify the algorithm we assume that the oscillator controls on its own the critical path (one could refer to subsection 2.2.3.3 for further details).

Measured computational speed feedback

As regards the closed-loop architecture introduced in subsection 2.3.2, the speed setpoint is calculated in such a way that the executed instruction number is equal to the number of instructions to do. Thus, if the measured speed does not track the setpoint as well as it would do, the computational speed controller will compensate by dynamically changing $\omega_{sp}(t)$ in consequence. For this reason, a simple speed setpoint tracking is enough and the frequency controller hence only needs to guarantee a null static error. One could refer to subsection 2.2.1 and adapt the technique to the present case to find the control law. This eventually results to an integral controller, that is in frequency-domain

$$C_{f_2}(s) = \frac{1}{\sigma(s)} \cdot \frac{K}{s} \tag{2.25}$$

where K is a tunable parameter (the integral parameter). As previously, an anti-windup mechanism is added. The corresponding discrete-time controller is

$$\begin{aligned}
\varepsilon(t_k) &= \omega_{sp}(t_k) - \omega(t_k) \\
\sigma(t_k) &= \frac{\omega(t_k)}{f(t_{k-1})} \\
f(t_k) &= f(t_{k-1}) + T_s \cdot \frac{1}{\sigma(t_k)} \cdot K \cdot \varepsilon(t_k) - T_s \cdot K_a \cdot \Big(f(t_{k-1}) - f_{sat}(t_{k-1}) \Big)
\end{aligned}$$

where K_a is the anti-windup parameter. Regarding the voltage level control, the previous principle remains unchanged and equation (2.24) can hence still be used.

2.3.6 Control algorithm

Eventually, the control strategy consists in minimizing the penalizing high voltage running time while guaranteeing good computational performance. The resulting algorithm is repre-

sented in figure 2.23. The algorithm for both feedback loops used in the control law (see subsections 2.3.2 and 2.3.3) are quite similar but the one depicted here is when using the speed measurement as a feedback.

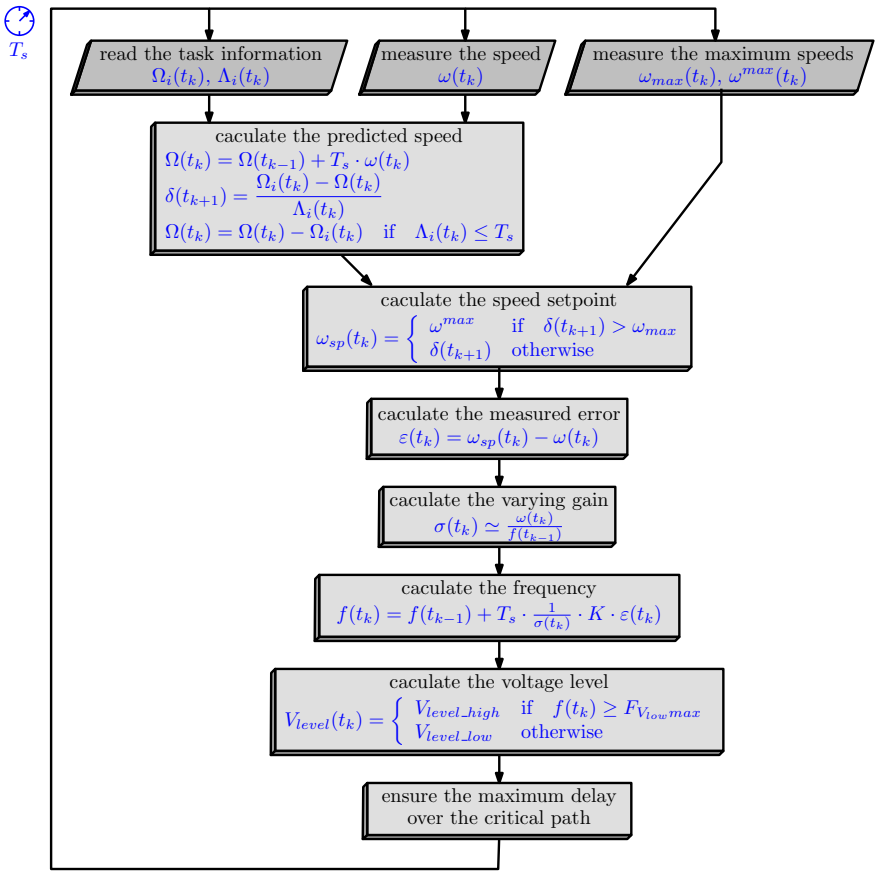


Figure 2.23: Algorithm: the computational speed controller.

2.4 Fully discrete control scheme

Two control strategies were presented in sections 2.2 and 2.3 for an electronic device with two discrete voltage levels and a continuous frequency range for each level. However, considering a continuously varying frequency is not realistic in practice and, for this reason, a discrete scheme would be preferable. As a result, **we propose to extend the previous control strategy to a fully discrete control architecture**. This means that the supply voltage and the clock frequency can only take some values: M voltage and N frequency levels are hence considered straight afterwards. Note that the definition of M and N - which was initially given in section 2.1 - is called back here:

- The input of a M-voltage level Vdd-hopping belongs to a M-value set and, with such a mechanism, the supply voltage V_m is provided when $V_{level}(t) = V_{level\_m}$. We define $m \in \{1, 2, ..., M\}$ and $V_m > V_{m+1}$, respectively $V_{level\_n} > V_{level\_n+1}$. Considering that this inner-loop is extremely fast with respect to the control loop considered here, the dynamics of the Vdd-hopping can be neglected.

- The input of a N-frequency level oscillator belongs to a N-value set and, with such a mechanism the clock frequency f_n is provided when $f_{level}(t) = f_{level\_n}$. We define $n \in$

$\{1, 2, ..., N\}$ and $f_n > f_{n+1}$, respectively $f_{level\_n} > f_{level\_n+1}$, and we choose $N \geq M$ (for the reasons explained just after in subsection 2.4.1). Eventually, switching from one level to another is considered as instantaneous.

Finally, the control architecture of this discrete scheme is presented in figure 2.24. In fact this is almost the same than the one depicted in section 2.3 where the control variables are now the *voltage level* $V_{level}(t)$ and the *frequency level* $f_{level}(t)$, and not the expected frequency $f(t)$ anymore. Moreover, we decide to base this new control strategy on the measured computational speed feedback scheme because the algorithm is less complex than the computational speed setpoint feedback one (see subsections 2.3.2 and 2.3.3 respectively for further details), more especially as regards the frequency and voltage level control laws (see subsection 2.3.5).

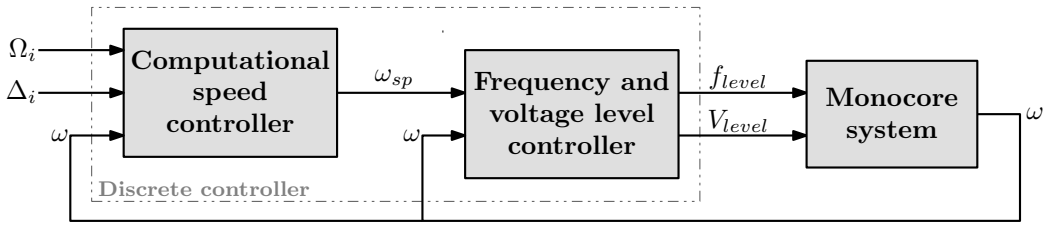


Figure 2.24: Closed-loop system: the fully discrete controller.

The computational speed setpoint building is explained in subsection 2.4.1 for a M-voltage level mechanism. Then an extension of the fast predictive control law is detailed in subsection 2.4.2. Thus, the *discrete controller* - highlighted in figure 2.24 - now allows to directly calculate the control variables. It includes the *computational speed controller* and the *frequency and voltage level controller* which were two different control blocks before and which required a control law in two steps. At the end, an extra degree of freedom is proposed in subsection 2.4.3 using the clock-gating principle and a trick to estimate the maximum speeds - required in the predictive control strategy - is depicted in subsection 2.4.4. The general control algorithm is given in subsection 2.4.5.

2.4.1 Energy-efficient setpoint for a M-voltage level mechanism

As explained in subsection 2.3.1 for a two-voltage level mechanism, the supply voltage is the penalizing parameter in an electronic device with an energy-performance tradeoff. Consequently, the lower is the voltage level of the chip, the better the energy savings are. The system has hence to run at the maximum possible computational speed for all the voltage levels - except the lowest one - in order to reduce the high voltage level running time. For this reason, **we propose to use only one possible frequency f_m per voltage level**, which is chosen as the maximal available frequency when the circuit is running at V_m. Then, *the corresponding (maximum) speed at V_m is ω_m*, defined such that $\omega_m = \mu \cdot f_m \cdot V_m$ obtained from equation (2.4) defined in section 2.1. We will see in subsection 2.4.4 how to obtain these maximum speeds in practice. Note that this speed value is implicitly maximum by construction since f_m is the maximal value in the available frequency range. For example when the system runs with the supply voltage V_2 and the clock frequency f_2, the corresponding computational speed is ω_2. The only case where several frequency levels are possible is for the lowest voltage level V_M. Indeed, the energy consumption could not be reduced anymore since no lower voltage level exists. The degree of freedom on the frequency will thus allow to fit the task with its deadline (so far as possible). Therefore, for the last voltage level V_M, the electronic device could run with one of the clock frequencies belonging to the set $\{f_M, f_{M+1}, ..., f_N\}$. This leads to the computational speed

set $\{\omega_M, \omega_{M+1}, ..., \omega_N\}$ available at low voltage. However, in order to simplify the afterwards equations, one could note that $V_x = V_M \quad \forall\, M \leq x \leq N$. An example of such a principle is presented in figure 2.25. In this example, the system can run with three voltage levels, i.e. V_1, V_2 and V_3, and four frequency levels which hence leads to four maximum speeds, i.e. ω_1, ω_2, ω_3 and ω_4. Thus, the possible cases are the voltage level V_1 and the corresponding speed ω_1, or V_2 and ω_2, or V_3 and ω_3. Moreover, an extra possible speed is ω_4, available to run at the last voltage level V_3.

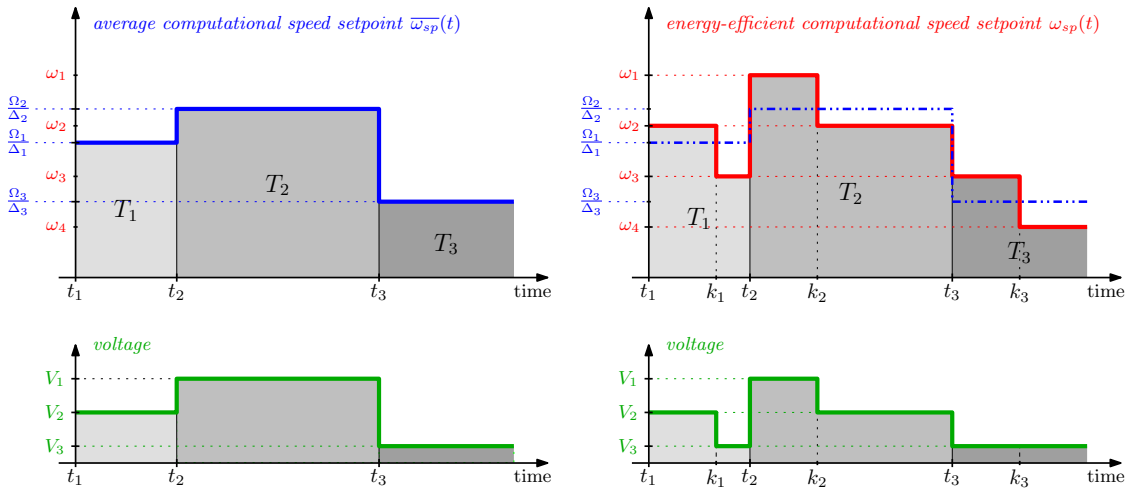


Figure 2.25: Computational speed setpoint: extension of the energy-efficient computational speed setpoint building for a M-voltage level mechanism.

Eventually, the control strategy remains the same than in the two-voltage level case (detailed in section 2.3) since, for each task, the two computational speeds which are immediate neighbors to the average speed setpoint minimize the energy consumption (see subsection 1.3.2 for further details). As a result, **we propose to calculate the two neighbor parameters in order to reduce the problem to a two-level system**. Of course, this has to be done for each task to treat. Afterwards, *ω_j and ω_{j+1} are the neighbor speeds*, with $\omega_j > \omega_{j+1}$ in such a way that the task could be executed running firstly at V_j and then at V_{j+1}. In other words, the discrete controller has finally to dynamically predict the switching time k_i - for each task T_i - to go from V_j to V_{j+1} which minimizes the penalizing V_j running time while ensuring that the task will not miss its deadline. One could refer to the energy-efficient speed setpoint building in subsection 2.3.1.2, and more particularly with figure 2.19(b), since this is exactly what we did by minimizing the high voltage running time (only two voltage existed there). On the proposed example in figure 2.25, only one voltage level is used to compute a task with the intuitive average speed building (left plot) - the voltage corresponding to the maximum speed just above the average speed, i.e. the ratio $\Omega_i(t)/\Delta_i(t)$ for each task T_i to treat - whereas the energy-efficient building requires two levels by task in order to reduce the energy consumption (right plot). These two voltage levels are those corresponding to the maximum speeds immediately neighbors to the average setpoint. Thus, for T_1 we have $\omega_2 > \Omega_1/\Delta_1 > \omega_3$ and the task will be executed with the corresponding voltage levels V_2 and V_3. The idea is to begin with the highest one, that is V_2, and after a certain amount of time k_1 the task could be finished at the lowest voltage level V_3. Respectively, the task T_2 will be computed at V_1 and then V_2. Finally, the average speed of the third task T_3 is $\omega_3 > \Omega_3/\Delta_3 > \omega_4$ and yet, the corresponding voltage is V_3 for both neighbor speeds. This task will hence be completely treated with the same voltage level but with two different frequency levels in order to fit with its deadline, that are ω_3 and ω_4.

2.4.2 Extension of the fast predictive control

As for the two-level case (detailed in subsection 2.3.2), the aim of the predictive control is to dynamically calculate the predicted speed $\delta(t)$. Let $\omega_j > {}^{\Omega_i(t)}/_{\Delta_i(t)} \geq \omega_{j+1}$ for the current task T_i to treat, and so is $\omega_j > \delta(t) \geq \omega_{j+1}$ (in a first time). One hence needs to know if the task has to be executed at the more penalizing voltage level V_j, or if V_{j+1} will be enough to perform it before its deadline. As previously, this principle could be formulated as an optimization control problem: for each task T_i, what is the computational speed setpoint which minimizes the high voltage running time while guaranteeing that the executed instruction number is equal to the number of instructions to do. In order to not compute this optimization problem for each voltage level, one needs to calculate the average speed $\overline{\omega_{sp}}(t) = {}^{\Omega_i(t)}/_{\Delta_i(t)}$ of the current task T_i and deduce the neighbor voltage levels V_j and V_{j+1}. Eventually, the problem can be formulated as

$$\min\ t_{V_j} \qquad \text{s.t.} \quad \int_{\Delta_i(t)} \omega(t)\ dt = \Omega_i(t) \quad \text{and} \quad \omega_j > {}^{\Omega_i(t)}/_{\Delta_i(t)} \geq \omega_{j+1} \tag{2.26}$$

where $\int \omega(t)\ dt$ is the integral of the measured computational speed which corresponds to the executed number of instructions for the current task. The fast predictive control problem then consists in calculating the value of $\delta(t)$ which can be easily described as the ratio between *what it remains to do* and *the remaining time before the end of the task*. This is quite similar to the previous two-level case and, consequently, the computation of $\delta(t)$ remains the same than before. The equation (2.18) becomes

$$\Omega(t_k) = \Omega(t_{k-1}) + T_s \cdot \omega(t_k) \tag{2.27a}$$

$$\delta(t_{k+1}) = \frac{\Omega_i(t_k) - \Omega(t_k)}{\Lambda_i(t_k)} \tag{2.27b}$$

$$\Omega(t_k) = \Omega(t_k) - \Omega_i(t_k) \qquad \text{if} \quad \Lambda_i(t_k) \leq T_s \tag{2.27c}$$

where T_s is the sampling period, $\Omega(.)$ *is the discrete integration of the computational speed* $\omega(.)$ and equation (2.27c) is a conditional instruction to be coherent with the end of a task. Finally, the difference between both algorithms is in calculating the control variables. Indeed, the computational speed setpoint $\omega_{sp}(t)$ is deduced from the dynamical value of $\delta(t)$ and so are the voltage and frequency levels too. Thus the device runs with the more penalizing neighbor speed ω_j during an amount of time, which means the penalizing voltage V_j and the corresponding frequency f_j. Since the running computational speed of the device is higher than the average speed - because $\omega_j > \delta(t)$ by construction - the value of $\delta(t)$ decreases until achieving the less consuming speed ω_{j+1} and the task can then be finished with the less penalizing voltage V_{j+1} and the frequency f_{j+1}. In fact the computational speed setpoint is not really required since the control variables are immediately deduced from the predicted speed $\delta(t)$, but we still notice $\omega_{sp}(t)$ anyway - for a well understanding - in the resulting algorithm which summarizes the control decisions. This yields

$$\left.\begin{array}{ll} \omega_{sp}(t_k) & = \omega_j \\ V_{level}(t_k) & = V_{level\_j} \\ f_{level}(t_k) & = f_{level\_j} \end{array}\right\} \quad \text{if} \quad \delta(t_{k+1}) > \omega_{j+1}$$

$$\left.\begin{array}{ll} \omega_{sp}(t_k) & = \omega_{j+1} \\ V_{level}(t_k) & = V_{level\_j+1} \\ f_{level}(t_k) & = f_{level\_j+1} \end{array}\right\} \quad \text{otherwise} \tag{2.28}$$

One could note that we assume here that the maximum delay over the critical path is ensured thanks to the oscillator (see subsection 2.2.3.3 for further details). Furthermore, an anti-windup mechanism is not required since the frequency level can only take a pre-defined value. Eventually, we previously defined $\delta(t)$ as $\omega_j > \delta(t) \geq \omega_{j+1}$ for a given task, but, in practice, the predicted speed can go out of these bounds. Indeed, if the measured computational speed goes faster than expected - due to the dynamics of the Vdd-hopping for example (see subsection 2.1.2.1 for more information) - $\delta(t)$ could become lower than ω_{j+1}. Nevertheless, the algorithm depicted with equation (2.28) still works: the value of the predicted speed simply becomes $\omega_{j+1} > \delta(t) \geq \omega_{j+2}$.

At the end, the computational performance is guaranteed since the speed setpoint to track is always higher or equal than required because of the measure computational speed feedback used in this fast predictive control law. Indeed, as explained in introduction of this section, we decided to base the M-level algorithm on the control closed-loop architecture with the measured speed as a feedback, because the frequency and voltage level control strategy is less complex than in the second case (see subsections 2.3.2 and 2.3.3 respectively). Above all, we chose this scheme because it allows to directly calculate the control variables from the value of $\delta(t)$. Thus, if some delays occur - due to the frequency and/or voltage transition time (especially because of the Vdd-hopping behavior which does not instantaneously change the voltage level) - the measurement will be impacted and the value of $\delta(t)$ recalculated in consequence (which is not the case with the second proposal). Therefore, the robustness is ensured thanks to the measurement feedback.

2.4.3 Clock-gating control

On top of the previous depicted strategy, **we propose to add an extra control decision based on the clock-gating principle**. Indeed, it is possible to "deactivate" the clock of the device when the task is completed, such as using the clock-gating technique (see subsection 1.3.1 for further details). In this case, the device will run with the lowest voltage V_M and a null frequency (in fact the clock is only paused but a null frequency will be used in simulation to highlight the clock-gating intervals). This behavior is useful, especially when the number of voltage and frequency levels is poor. Indeed, due to the small possible supply cases, a higher frequency level than required will be used most of the time and, consequently, the computational load of a task T_i will be executed before its deadline. Thus, it could be interesting to pause the clock until the beginning of the next task T_{i+1}. However, in order to minimize the use of the clock-gating principle, we will pause the clock only if the beginning of the next task is not too close to the end of the current one, that is

$$\begin{aligned} &\text{if} \quad \delta(t_{k+1}) \leq 0, \\ &\quad f_{level}(t_k) = \begin{cases} 0 & \text{if} \quad \Lambda_i(t_k) > \Lambda_{min} \\ f_{level}(t_{k-1}) & \text{otherwise} \end{cases} \end{aligned} \tag{2.29}$$

where Λ_{min} is a constant chosen by the designer. One could note that the condition of this algorithm is $\delta(t) \leq 0$. In fact the clock-gating principle is applied if the task is finished. That is the case when *what the processor has to do to compute the task* is equal to *what it has already done*, i.e. $\Omega_i(t) = \Omega(t)$, and when $\delta(t) = 0$ thanks to equation (2.27b). However, for such a reason or another, what the device did could be higher than what it has to do. For this reason, we prefer the inequality $\delta(t) \leq 0$.

2.4.4 Estimation of the maximum computational speeds

The control strategy - proposed in subsection 2.4.2 - is easy to implement, except for the parameters ω_m, i.e. the possible computational speeds when the device is supplied with the voltage V_m and the clock frequency f_m. Of course, these parameters could be calculated using the system model equation (2.4), that leads to $\omega_m = \mu \cdot f_m \cdot V_m$ where the constant μ belongs to the electronic circuit. However, we would like to have a controller robust to process variability (see section 1.2 for further details) which means that the value of μ, f_m and V_m are not known and could vary. For this reason, **we propose to dynamically estimate the maximum speeds**. Thus, $\tilde{\omega}_m(.)$ *is the estimation of the computational speeds* ω_m. The solution consists in measuring the speed for each couple voltage/frequency. Therefore, the speed ω_m is measured when the system is running with the supply voltage V_m and the clock frequency f_m. Moreover, **we propose to use a weighted average of the measured speed** in order to filter the (possible) fluctuations of the measurement. In such a weighted mean principle, some points contribute more than others (instead of contributing equally to the final average). The resulting algorithm which summarizes that principle is

$$\begin{aligned} &\text{if} \quad \begin{cases} V_{level}(t_{k-1}) = V_{level\_m} \\ f_{level}(t_{k-1}) = f_{level\_m} \end{cases}, \\ &\tilde{\omega}_m(t_k) = (1-\nu) \cdot \tilde{\omega}_m(t_{k-1}) + \nu \cdot \omega(t_k) \end{aligned} \tag{2.30}$$

where $0 \leq \nu \leq 1$ *is the weighted value*. Thus, the value of the estimated computational speeds $\tilde{\omega}_m(t)$ takes the measured speed $\omega(t)$ into account, but only for a small part, in order to soften some high variations that could appear during the chip running.

Nonetheless, a problem could appear during the voltage transitions. Indeed, the previous algorithm on equation (2.28) allows to dynamically calculate the predicted speed $\delta(t)$ and compare this value with the computational speeds ω_m. First of all, the algorithm has to be modified in order to now compare $\delta(t)$ with the estimation of the speeds $\tilde{\omega}_m(t)$. Then, for a given task, the system will run with a given voltage V_j and the corresponding frequency f_j when $\delta(t)$ is higher than $\tilde{\omega}_{j+1}$, but as soon as $\delta(t)$ becomes lower or equal than $\tilde{\omega}_{j+1}$ the controller will be able to change the voltage and frequency levels to V_{j+1} and f_{j+1} in order to be less consuming. However, during the level transition the estimated speed could vary - due to the fluctuations in the estimation - and could become higher than the current value of $\delta(t)$. Because of this phenomenon, the levels might switch and switch again and, therefore, a solution is required. For this reason, **we propose to dynamically calculate $\nu(t)$ and bound its value** in such a way that the variation of the estimation remains always lower than the variation of the predicted speed. At the end, the proposed algorithm is divided into three parts, as detailed with some items afterwards:

- First, let $\Delta\tilde{\omega}_m(t)$ denote the variation of the computational speed estimation, obtained from equation (2.30). This yields

$$\begin{aligned} \tilde{\omega}_m(t_k) &= \Big(1 - \nu(t_k)\Big) \cdot \tilde{\omega}_m(t_{k-1}) + \nu(t_k) \cdot \omega(t_k) \\ \Delta\tilde{\omega}_m(t_k) &= \frac{\tilde{\omega}_m(t_k) - \tilde{\omega}_m(t_{k-1})}{T_s} \\ \Leftrightarrow \Delta\tilde{\omega}_m(t_k) &= \frac{\nu(t_k)}{T_s} \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big) \end{aligned}$$

- Then, let $\Delta\delta(t)$ denote the variation of the predicted speed, calculated from equation (2.27b),

that is

$$\begin{aligned}\delta(t_{k+1}) &= \frac{\Omega_i(t_k) - \Omega(t_k)}{\Lambda_i(t_k)} \\ &= \frac{\Omega_i(t_k) - \Big(\Omega(t_{k-1}) + T_s \cdot \omega(t_k)\Big)}{\Lambda_i(t_k)} \\ &= \frac{\Omega_i(t_k) - \Omega(t_{k-1})}{\Lambda_i(t_k)} - \frac{T_s \cdot \omega(t_k)}{\Lambda_i(t_k)}\end{aligned}$$

Usually, the number of instructions does not change for a given task, which means $\Omega_i(t_k) = \Omega_i(t_{k-1}) = C_i$. Moreover, the laxity is only different from a sampling period between two measurements, i.e. $\Lambda_i(t_k) = \Lambda_i(t_{k-1}) - T_s$, and neglecting this leads to $\Lambda_i(t_k) \simeq \Lambda_i(t_{k-1})$. The first term on the right hand can hence be approximated by $\delta(t_k)$, which leads to

$$\begin{aligned}\delta(t_k) = \frac{\Omega_i(t_{k-1}) - \Omega(t_{k-1})}{\Lambda_i(t_{k-1})} &\simeq \frac{\Omega_i(t_k) - \Omega(t_{k-1})}{\Lambda_i(t_k)} \\ \Leftrightarrow \delta(t_{k+1}) &\simeq \delta(t_k) - \frac{T_s \cdot \omega(t_k)}{\Lambda_i(t_k)}\end{aligned}$$

Finally, the variation of $\delta(t)$ is given by

$$\begin{aligned}\Delta\delta(t_k) &= \frac{\delta(t_{k+1}) - \delta(t_k)}{T_s} \\ \Leftrightarrow \Delta\delta(t_k) &\simeq -\frac{\omega(t_k)}{\Lambda_i(t_k)}\end{aligned}$$

- At the end, we need that the variation of the estimation is lower than the variation of the predicted speed, that is

$$\begin{aligned}&\Delta\tilde{\omega}_m(t_k) \leq \Delta\delta(t_k) \\ &\Leftrightarrow \frac{\nu(t_k)}{T_s} \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big) \leq -\frac{\omega(t_k)}{\Lambda_i(t_k)} \\ &\Leftrightarrow 0 \leq \nu(t_k) \leq -\frac{T_s \cdot \omega(t_k)}{\Lambda_i(t_k) \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big)}\end{aligned} \tag{2.31}$$

This result is possible because ***i)*** $\nu(t_k) \geq 0$ by construction and ***ii)*** we consider that a problem could only appear during a decreasing switch of the voltage and frequency levels, that is when $\omega(t_k) - \tilde{\omega}_m(t_{k-1}) \geq 0$. Therefore, equation (2.31) allows to bound the parameter $\nu(.)$ and this bounding has to be considered in the implementation.

Furthermore, no information on the system parameters is required at all, which is very important for process variability since the voltages and the frequencies are not known and could vary. Thus, the controller just needs to measure the computational speed $\omega(t)$ and know the laxity $\Lambda_i(t)$ - provided by the operating system - in order to calculate $\nu(t)$ and estimate the value of the computational speeds $\omega_m(t)$.

2.4.5 Control algorithm

Eventually, the control strategy is the extension of the one depicted in the previous section, for a fully discrete scheme running with M voltage and N frequency levels. As before, the

control law consists in minimizing the penalizing high voltage running time while guaranteeing good computational performance. The estimation of the possible computational speeds is also taken into account here. The resulting algorithm is represented in figure 2.26.

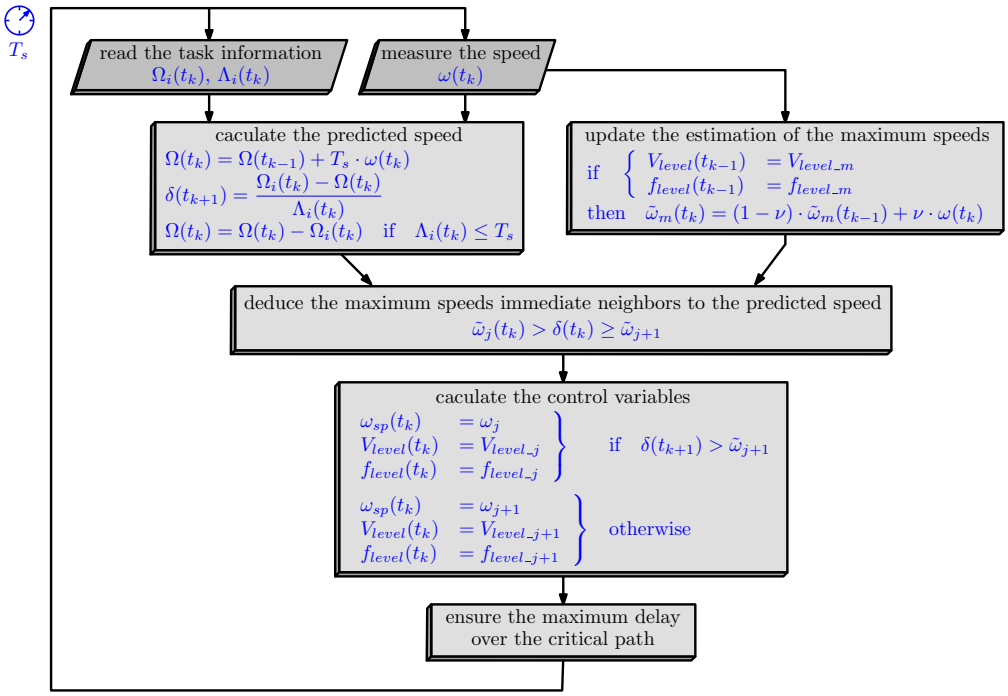


Figure 2.26: Algorithm: the fully discrete controller.

2.5 Simplification of the algorithms for a low control computational cost

In order to simplify the different control algorithms (detailed in the previous sections), **we propose to apply some simplifications to the control algorithms**. More especially, several algorithms exist to perform divisions in digital designs, but it would be better just to throw them off, in order to prevent from precision and overflow problems. For this reason, we try to not have division in our algorithms at all.

The fast predictive control law - previously introduced in the computational speed controller in section 2.3 - aims at dynamically calculating the so-called predicted speed $\delta(t)$. However, as explained before, a fast predictive control takes advantage of the structure of the system to control in order to fasten the computation of the control law (unlike classical predictive control where an optimization problem has often to be solved and, therefore, leads to a high computational cost). Nevertheless, one can remarks that a division still has to be executed to obtain $\delta(t)$ in the referring control algorithm. In fact, the value of the predicted speed is required in the computational speed control strategy to build an energy-efficient speed setpoint because then, the voltage and frequency levels are calculated in order to track this setpoint. On the other hand, the predicted speed is computed in the fully discrete control scheme also - detailed in section 2.4 - but its value is not really needed. Just to bring back the algorithm, the idea is to calculate the predictive average speed and then compare it with a given value to decide the

control variables. The expression of the predicted speed - defined in equation (2.27b) - is

$$\delta(t_{k+1}) = \frac{\Omega_i(t_k) - \Omega(t_k)}{\Lambda_i(t_k)}$$

where $\Omega_i(t)$ and $\Lambda_i(t)$ are respectively the instruction number to do and the laxity (the remaining amount of time) of the task T_i to treat, while $\Omega(t)$ is the measured number of instructions. Then, in the discrete control scheme, $\delta(t)$ is compared to the maximum speed ω_{j+1} in order to directly deduce the control variables. The control decision - defined in equation (2.28) - is

$$\left.\begin{aligned} V_{level}(t_{k+1}) &= V_{level\_j} \\ f_{level}(t_{k+1}) &= f_{level\_j} \end{aligned}\right\} \quad \text{if} \quad \delta(t_{k+1}) > \omega_{j+1}$$

$$\left.\begin{aligned} V_{level}(t_{k+1}) &= V_{level\_j+1} \\ f_{level}(t_{k+1}) &= f_{level\_j+1} \end{aligned}\right\} \quad \text{otherwise} \quad .$$

Note that the computational speed setpoint is also originally obtained, but only for information and its value is then not needed in the algorithm. As a result, the predicted speed is only needed in $\delta(t_{k+1}) > \omega_{j+1}$ and finally, a very simple trick can consist in modifying this inequality in order to not have any division anymore. To do that, one can substitute the expression of $\delta(t)$ into the inequality and change the denominator from the left side to the right side. The condition in the previous relation hence becomes

$$\text{if} \quad \Omega_i(t_k) - \Omega(t_k) > \omega_{j+1} \cdot \Lambda_i(t_k)$$

where no division occurs anymore.

Eventually, the same thing can be applied for the estimation of the maximum computational speeds. Indeed, a weighted average is presented in subsection 2.4.4 in order to filter the fluctuations of the measured speed. The corresponding relation - defined in equation (2.30) - is

$$\tilde{\omega}_m(t_k) = \big(1 - \nu(t_k)\big) \cdot \tilde{\omega}_m(t_{k-1}) + \nu(t_k) \cdot \omega(t_k)$$

where $\tilde{\omega}_m(t)$ is the estimation of the maximum speed ω_m while $\nu(t)$ is the weighted value. This parameter is dynamically computed and a certain restriction was proposed in order to not change too quickly the value of the estimation during a voltage transition. Thus, the restriction of the weighted value - defined in equation (2.31) - is

$$0 \leq \nu(t_k) \leq -\frac{T_s \cdot \omega(t_k)}{\Lambda_i(t_k) \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big)}$$

where T_s is the sampling period. A first approach consists in applying the upper bound as the value of $\nu(t_k)$ (if this value is not negative), but a better one exists. Summarizing the previous idea yields

$$\begin{aligned} \tilde{\omega}_m(t_k) &= \Big(1 - \nu(t_k)\Big) \cdot \tilde{\omega}_m(t_{k-1}) + \nu(t_k) \cdot \omega(t_k) \\ &= \tilde{\omega}_m(t_{k-1}) + \nu(t_k) \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big) \\ &= \tilde{\omega}_m(t_{k-1}) - \frac{T_s \cdot \omega(t_k)}{\Lambda_i(t_k) \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big)} \cdot \Big(\omega(t_k) - \tilde{\omega}_m(t_{k-1})\Big) \\ &= \tilde{\omega}_m(t_{k-1}) - \frac{T_s}{\Lambda_i(t_k)} \cdot \omega(t_k) \end{aligned}$$

At the end, the resulting expression still has a division to compute but this is quite simpler than previously.

2.6 Intuitive stability analysis

The notion of stability is important in control theory and Lyapunov theory immediately occurs in this field of knowledge. Consequently, basing our analysis on that is a natural way. Note that a theoretical background on Lyapunov stability is introduced in the second part of the thesis (in subsection 7.2.1). Anyway, the basic theorems are directly related to some particular functions $V(x)$ (continuous and positive definite) - denoted *Lyapunov-candidate functions* - which are defined with respect to the state x. This variable represents the internal state of the controlled system at any given time. Then, studying the time derivative of the Lyapunov-candidate function and applying some well-known methods allows to discuss on the stability of the system. Actually, the Lyapunov stability is based on a mathematical translation of an elementary physical constatation: if the total energy of the system tends to continuously decline, then this system is stable since it is going to an equilibrium state. For this reason, the Lyapunov-candidate functions are often based on some energetic functions, that are most of time quadratic functions of the state variables, such as

$$V(x) = x^T \cdot P \cdot x$$

We decide to use this classical function. As a result, we now have to find x which leads to continuously decrease $V(x)$. A solution could be

$$x(t_k) = \Omega_i(t_k) - \sum_{t_i}^{t_k - t_i} \omega(t_k) \tag{2.32}$$

where $\Omega_i(t_k)$ is the number of instructions required to treat the task T_i while $\sum_{t_i}^{t_k-t_i} \omega(t_k)$ is the sum of the measured computational speed which corresponds to the executed instruction number for the current task (since t_k is the current sampling time and t_i is the beginning of the task T_i). This expression comes from equation (2.17) in the fast predictive control law building (depicted in subsection 2.3.2) and, eventually, the value of $x(t_k)$ refers to the remaining number of instructions before the end of the task. Therefore, the Lyapunov function intuitively decreases during the running time of a task because the computational speed of the controlled electronic device can only be positive, and so is ensured the stability of a task. After that, one has to bring back the fact that the time horizon is contractive in the present study case. This means that the task horizon decreases for a given task until achieving its deadline. Then a new time horizon has to be achieved for the next task to treat, where the different variables in equation (2.32) change (since they depends on the task to treat). Finally, we can assume that the system is stable in the Lyapunov sense.

2.7 Synthesis

This chapter defines an architecture in order to control the energy-performance tradeoff of a single processing node. Actually, a voltage scalable processor makes possible to reduce the computational speed of the device and, consequently, its power consumption by reducing the supply voltage and/or the clock frequency. Two actuators respectively provide these variables while a controller decides their inputs thanks to a feedback loop. Based on this architecture, different control laws were then proposed:

- A frequency and voltage control strategy allows the measured computational speed to track an intuitive speed setpoint. In this first scheme, the setpoint is the average speed required to fit the task with its deadline. The strategy consists in calculating the (continuously

varying) frequency required for the tracking and then applying the minimum possible voltage (between two possible levels) in order to reduce the energy consumption. Of course, the maximum delay over the critical path has to be ensured and both control variables are finally restricted together.

- Based on this first work, a computational speed control strategy builds a more energy-efficient speed setpoint in order to minimize the penalizing high voltage running time, while still guaranteeing some computational performance. In fact, a task can now be executed with two voltage levels whereas it was computed with the highest one during the whole running time before. A fast predictive control law is applied to build this better reference, and the previous frequency and voltage control strategy is adapted to this new proposal.

- The computational speed control strategy is finally extended to a fully discrete scheme, where M voltage levels are now possible while a discretely varying frequency can only achieve N levels. Moreover, an estimation of the possible computational speeds (when the system runs with a given voltage and frequency levels) allows to calculate the control variables without any information on the system parameters. This proposal is hence strongly robust to process variability which occurs in sub-micrometric electronic chips.

At the end, some tricks are also proposed to reduce the control computational cost of the different strategies and an approximated stability analysis is performed to intuitively show that the controlled system is stable when applying our proposals.

Chapter

3 Global control in multicore systems

A closed-loop architecture was developed in chapter 2 in order to control the energy-performance tradeoff in an electronic chip. Only a single device had to be controlled (in voltage and frequency) in this seminal work. The goal was to minimize the energy consumption (reducing the penalizing high voltage running time) while guaranteeing some good computational performance (fitting the tasks to treat with their deadlines). In this chapter, an extension for several devices working together on the same chip is proposed. A first multicore architecture is thus defined in section 3.1, where the different devices are all power supplied with the same voltage and frequency (or a ratio of the clock). Some extensions of the previous monocore proposals are then detailed in section 3.2. They consist in scaling the voltage and the frequency of the chip in order to reduce the energy consumption of the whole circuit while guaranteeing good performance of each device. As previously, the control strategies are strongly robust to tackle variability and therefore suitable for $32\,nm$ technology or smaller implementations. Another multicore architecture is also depicted in section 3.3, where the different devices still work together but now with their own clock frequency. This scheme is only shortly depicted without developing any control laws. However, this second approach is in fact simpler than the first one and, consequently, the multicore strategies can easily be transposed. Eventually, a synthesis is performed in section 3.4. Some simulation results will be presented in chapter 4.

3.1 Several chips working together in the same power domain

We initially presented in chapter 2 a monocore system, where only one electronic device has to be controlled. In this chapter, **we propose to extend the monocore architecture to a multicore system**. The system architecture with several devices to control is shown in figure 3.1. In fact this system is not so different from the monocore one - introduced in section 2.1 and represented in figure 2.1 - since the principle remains (almost) the same: a controller dynamically calculates the frequency level $f_{level}(t)$ and the voltage level $V_{level}(t)$ to send to the *actuators*. They are a ring oscillator and a Vdd-hopping (see subsection 2.1.2 for further details), which respectively provide the clock frequency $f_{clk}(t)$ and the supply voltage $V_{dd}(t)$ to the *electronic devices* (see subsection 2.1.1 for more information on the model of a single processing node). The main difference is that there are now *X devices to control*, which means as many references $ref^X(t)$ given by the operating system (the computational load for each task to treat) and as many measured speeds $\omega^X(t)$ (the current activity) as devices. Therefore the *multicore controller* has to control the whole system but devices do not work independently since they are all supplied with the same voltage and clock frequency. This constraint has hence to be taken into account in the multicore control strategy. Furthermore, a certain dimension of freedom is allowed in triggering a device with a ratio of the clock. This principle is detailed in subsection 3.1.1. As a result, the controller has also to provide the *frequency ratios $\rho^X(t)$*.

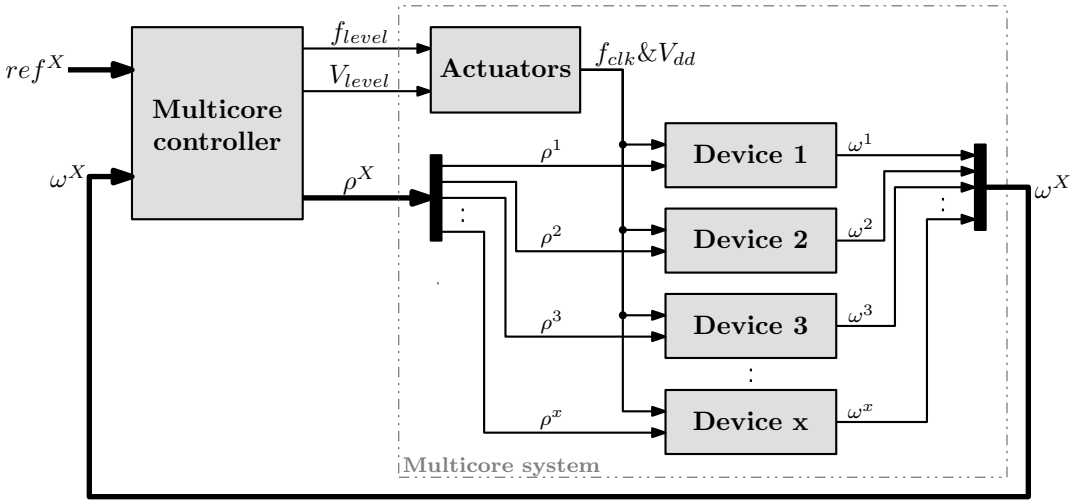


Figure 3.1: Architecture of the multicore system.

Notations:

- $\rho^X(t)$ (upper case superscript) means that there are X signals $\rho(t)$, one for each device.
- $\rho^d(t)$ (lower case superscript) denotes the signal $\rho(t)$ of the device d, where $d \in \{1, 2, \ldots, X\}$.
- $\rho^x(t)$ denotes the signal $\rho(t)$ of the last device.

This is valid for all the variables except the voltage and frequency ones which are still single signals. This is due to the multicore architecture which is composed of only one voltage/frequency actuator.

One could note that, in the ARAVIS project case - see subsection 1.4.2 - this multicore architecture defines a cluster of the SoC ARAVIS as represented in figure 1.9. Indeed, the chip is divided into different clusters, whose each one corresponds to one power domain (only one

voltage/frequency actuator) in order to independently work regardless the process variability. Then, several computational nodes have a single clock (and supply voltage) which triggers the set of processors in such a way that they all work together into a cluster. These computational nodes are the so-called electronic devices.

3.1.1 A certain degree of freedom thanks to some frequency ratios

As explained in introduction, the different electronic devices of the multicore architecture work together, with the same supply voltage and clock frequency, and have hence to be controlled in consequence. However, a certain degree of freedom is allowed in triggering a device with a ratio of the clock because in fact, in practice, it is possible to dynamically add one or two NOPs (i.e. No OPeration) between each instruction in such a way that the device runs twicetwo or three times slower. Another alternative would be to use the clock-gating principle - introduced in subsection 1.3.1 - in order to dynamically pause the clock and achieve the expected behavior. Whatever the solution, a frequency ratio is important - and more interesting from a point of view of the control - to not have all the devices running with the same computational speed.

3.1.2 The multicore controller

The controller aims at calculating the control signals which minimize the energy consumption of the chip while guaranteeing some good computational performance of all the devices. A strategy is hence required to control the energy-performance tradeoff of the whole system. Afterwards denoted the *multicore system*, it is composed of the electronic devices and the two actuators. The resulting block merging was firstly highlighted in figure 3.1 (with the doted line) and eventually is gathered in figure 3.2.

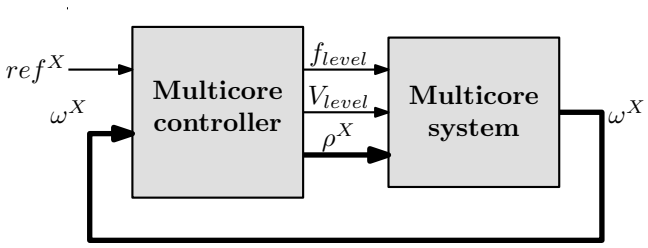


Figure 3.2: Details of the architecture: the multicore controller.

The complete system model which was given in subsection 2.1.3 for a single device remains almost the same for several processors. Actually, the measured computational speed of a device is an affine function with respect to the expected frequency $f(s)$ - defined in subsection 2.1.2.2 - and the supply voltage $V_{dd}(s)$ (in the Laplace domain). Furthermore, a device in the multicore architecture can be triggered with a ratio of the clock frequency and the measurement is hence function of this ratio too. Thus, equation (2.4) becomes

$$\omega^d(s) \simeq \mu^d \cdot \left(\rho^d \cdot f(s)\right) \cdot V_{dd}(s) \tag{3.1}$$

where μ^d belongs to the electronic device d and could be different for each device because of process variability (see section 1.2 for further details). The resulting computational speed of a given device is also function of the frequency ratio ρ^d introduced in subsection 3.1.1. Moreover, the power sources which exist in such a system are

$$P_{multicore\ system}(s) = \sum_{d=1}^{X} {P_{circuit}}^d(s) + P_{hopping}(s) \tag{3.2}$$

where $P_{circuit}(.)$ and $P_{hopping}(.)$ were previously defined in equation (2.5). Just to bring back the idea, the first source comes from the CMOS technology and depends on the circuit whereas the second are due to the voltage transitions. As a result, reducing the voltage/frequency will decrease the energy consumption - but with an increase of the delay in return (see subsection 1.3.1) - and a small number of voltage transitions is required too. The difficulty in this chapter is that all devices work together with the same supply voltage and frequency (or a ratio of the clock frequency). Consequently, the energy-performance tradeoff controller has to reduce the energy consumption of the critical device, minimizing its high voltage running time and the voltage transitions, while guaranteeing a good computational performance of the whole system, fitting the tasks with their deadline thanks to the frequency ratios. A key point also consists in well defining the criticality of a task. The control variables are the voltage level $V_{level}(t)$ and the frequency level $f_{level}(t)$ which directly control the actuators, and the frequency ratios $\rho^X(t)$ which manage the triggering mechanism of the different devices. On the other hand, the current computational speeds $\omega^X(t)$ are measured and compared to the computational load references $ref^X(t)$ to decide the control law.

In the following section, different extensions of the monocore control laws - initially depicted in chapter 2 - are detailed.

3.2 Extension of the monocore control strategies

Considering the work presented in chapter 2, **we propose to extend the monocore control strategies to the multicore architecture** where several electronic devices have to be controlled together. Actually, the proposal developed in section 2.3 could be divided into two overlapped parts (one can see the referring architecture in figure 2.17):

The computational speed controller provides an energy-efficient computational speed setpoint - using a fast predictive control law - in order to minimize the penalizing high voltage running time (and so the energy consumption) while guaranteeing the computational performance of the device.

The frequency and voltage level controller fits the measured computational speed with this setpoint adapting the frequency and the voltage level.

Finally, the whole monocore controller leads to a robust control law and will hence be adapted to the multicore case. One could remark that the frequency was continuously varying in the considering monocore architecture and so is varying here too. This means $f(t) = f_{level}(t)$, as explained in subsection 2.1.2.2, and the expected frequency $f(t)$ will hence be used afterwards instead of the frequency level $f_{level}(t)$. Furthermore, the multicore controller needs to calculate the frequency ratios - defined in section 3.1 - which allow a certain dimension of freedom by triggering a device with a ratio of the clock frequency. A third part has hence to be introduced in the control architecture:

The ratio controller calculates the ratio between the clock frequency - that is the frequency required to treat the critical task - and the frequency of the current device, for each device.

In the following subsections, two control frameworks are detailed: a first intuitive one which duplicates the monocore principle as many times as devices in subsection 3.2.1, and a second one which tries to minimize the computational cost of the controller in subsection 3.2.2. In both strategies, the frequency ratios are considered continuously varying and, at the end, these extra control variables are discretized in subsection 3.2.3.

On the other hand, another proposal was introduced in section 2.4 and also leads to a robust control law. This refers to a fully discrete scheme where only a small number of voltage and frequency levels are available to control the circuit. In this case, the control variables are the frequency level $f_{level}(t)$ and the voltage level $V_{level}(t)$. As a result, the control strategy is lightened and so is the computational cost. This is eventually adapted to the multicore controlled system in subsection 3.2.4.

3.2.1 Full duplication of the monocore control strategy

An intuitive and easy way to control several devices is to consider the different devices working alone and then decide for a global strategy. Consequently, **we propose to duplicate the monocore control strategy as many times as devices**. The resulting multicore architecture - based on the work previously done in section 2.3 when the measured speed setpoint is used as a feedback - is represented in figure 3.3 and could be divided into three steps:

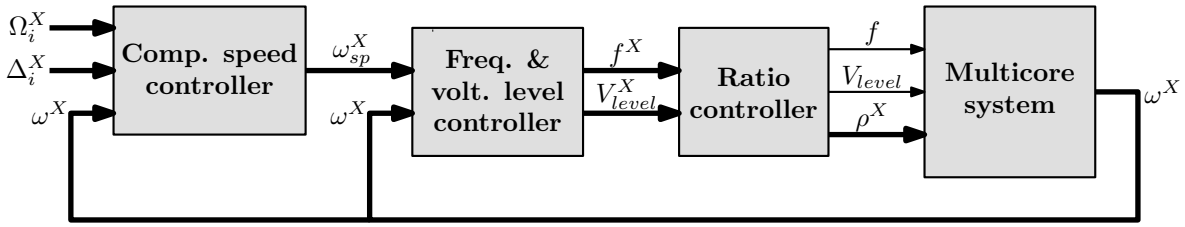


Figure 3.3: Closed-loop system: the multicore controller based on full duplication of the monocore control strategy.

1. First, the *computational speed controller* dynamically calculates the speed setpoints $\omega_{sp}^X(t)$ for all the devices (using a predictive control law). The setpoint $\omega_{sp}^d(t)$ is independently calculated for each device d, using some task information given by the operating system - the number of instructions $\Omega_i^d(t)$ and the deadline $\Delta_i^d(t)$ of the task T_i currently executed on the device - and the measured speed $\omega^d(t)$.

2. Then the *frequency and voltage level controller* independently calculates the expected frequencies $f^X(t)$ and the voltage levels $V_{level}^X(t)$ usually required to control a single device.

3. Finally the *ratio controller* compares the calculated frequencies $f^X(t)$ to deduce the device to control in priority. Thus, the critical task is the one which needs the maximal frequency to fit with its deadline. The device which has to treat this task is afterwards denoted the critical device and indexed with the letter c. The expected frequency $f(t)$ and the voltage level $V_{level}(t)$ sent to the actuators are those from the critical device, i.e. $f^c(t)$ and $V_{level}^c(t)$. Eventually, the frequency ratios $\rho^N(t)$ are obtained by doing the ratio between the frequency of the current device $f^d(t)$ and the one of the critical device $f^c(t)$.

Eventually, this intuitive strategy ensures that the tasks are correctly performed for all devices because each device is independently controlled using the monocore strategy. Indeed, the monocore strategy works for one device and we focus the frequency and the voltage level decision on the critical one, tat is the device which has to treat the task with the highest computational needs. Eventually, all the non-critical tasks will be executed with the critical voltage level and a frequency lower than or equal to the critical frequency. Moreover, a non-critical task could become the critical one when its computational needs increase and increase again.

Finally, this intuitive full duplication of the monocore principle leads to reduce the energy consumption of several devices working together while guaranteeing good computational performance for the whole system. Nevertheless, a consequence is that the control computational needs are multiplied as many times as devices and the number of variables significantly increases too. Consequently, a better solution would be to duplicate only some parts of the monocore control strategy. This is detailed in the following subsection.

3.2.2 Partial duplication for a lower control computational cost

A first intuitive multicore control architecture was presented in the previous subsection where the monocore strategy is duplicated as many times as devices. However, a better solution - still based on the work done in section 2.3 - would allow to reduce the control computational cost and, in this perspective **we propose to not intuitively duplicate all the monocore control strategy**. In fact, the frequency ratios $\rho^X(t)$ require to be calculated and some parts have necessary to be duplicated in order to obtain X signals. The aim is to repeat as least code as possible. The best solution would be to use the references $ref^X(t)$ - provided by the operating system - to deduce the ratios without duplicating any part of the monocore strategy, but these signals are not relevant enough. Indeed, the critical task could not be known only from the number of instructions and the deadline because the computational load which was already executed is necessary too. As a result, only the computational speed controller really seems to have to be repeated anyway and hence needs to be duplicated. The resulting multicore architecture is proposed in figure 3.4 where three steps are still needed:

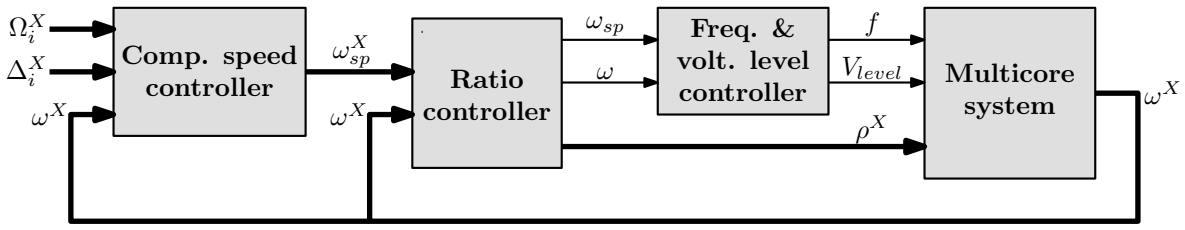


Figure 3.4: Closed-loop system: the multicore controller based on partial duplication of the monocore control strategy.

1. First, the *computational speed controller* provides the speed setpoints $\omega_{sp}^X(t)$, from which the frequency ratios $\rho^X(t)$ could be directly obtained since they provide information on the remaining computational load.

2. Then the *ratio controller* compares all the speed setpoints $\omega_{sp}^X(t)$ to deduce the device to control in priority. Thus, the critical task is the one which needs the maximal speed to fit with its deadline. The device which has to treat this task is also afterwards denoted the critical device and indexed with the letter c. The speed setpoint $\omega_{sp}(t)$ and the measured speed $\omega(t)$ sent to the *frequency and voltage level controller* are those calculated for that critical device, i.e. $\omega_{sp}^c(t)$ and $\omega^c(t)$, while the frequency ratios $\rho^X(t)$ are obtained by doing the ratio between the speed setpoint of the current device $\omega_{sp}^d(t)$ and the one of the critical device $\omega_{sp}^c(t)$.

3. Finally the *frequency and voltage level controller* calculates the frequency $f(t)$ and the voltage level $V_{level}(t)$ to send to the actuators only for the critical device.

With this partial duplication proposal, only the computational speed controller is repeated and not the frequency and voltage level controller anymore. Consequently, one could hope a reduction of the control computational cost without impacting the gain on energy savings.

Although all the devices are not independently controlled using the monocore strategy, the computational performance is yet guaranteed for each device. Indeed, with this second architecture, the monocore control strategy only ensures that the critical task will fit with its deadline, since the monocore control strategy is only applied to the critical device. The frequency ratios for the non-critical devices are then calculated from the computational load of the task of each device which is finally adjusted thanks to the computational speed controller. As a result, all the non-critical tasks are executed until their deadline anyway, or a task becomes the critical one when its computational needs become more important.

3.2.3 Discrete values of the frequency ratios

The control frameworks proposed in both previous subsections were developed with *ideal* continuously varying frequency ratios. However, as explained in section 3.1, some devices could be triggered with a ratio of the clock frequency $f_{clk}(t)$ by adding NOPs between instructions, for instance, in such a way that the device runs slower. This is why the frequency ratios could only have discrete values in practice which refer to the number of NOPs. For instance, a value of 1, $1/2$ or $1/3$ means that 0, 1 or 2 NOPs are respectively added between each instruction. Thus **we propose to calculate a discrete ratio from the continuously varying one** which is obtained thanks to the ratio controller (detailed in subsections 3.2.1 and 3.2.2). Afterwards, $\varrho^r(.)$ *is the discrete frequency ratios*. In order to implement such a behavior, one still has to calculate the continuous ratios $\rho^d(t)$ for each device d, that is $\rho^d(t) = f^d(t)/f^c(t)$ for the multicore control strategy based on full duplication and $\rho^d(t) = \omega^d_{sp}(t)/\omega^c_{sp}(t)$ for the one based on partial duplication. Then, some iterations have to be done in order to deduce the possible discrete frequency ratio $\varrho^d(t)$ immediate neighbors and just upper than the value of the continuous ratios $\rho^d(t)$. The following algorithm summarizes the principle:

$$\varrho^d = \begin{cases} 1 & \text{if} \quad 1/2 < \rho^d \leq 1 \\ 1/2 & \text{if} \quad 1/3 < \rho^d \leq 1/2 \\ 1/3 & \text{if} \quad 0 < \rho^d \leq 1/3 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

Of course, the possible values can be chosen uniformly in the space or not, as this is the case in the given algorithm. Moreover, a large number of frequency ratios is not so evident - as explained in section 1.3 for the number of voltage levels - because a small number will lead to a small degree of freedom but a low extra control computational cost, while a large number will explode the computation of the algorithm. Indeed, the control computational needs would increase with the number of possible ratios. Finally, this discrete ratio behavior would lead to a less energy-efficient system because the frequencies of the non-critical devices will be higher than required - thanks to equation (3.3) - contrary to the continuous case where these frequencies correspond exactly to the expected ones.

As explained in section 2.5, it is important to remove divisions so far as possible in order to reduce the control computational cost. As a result, **we propose to avoid the divisions needed to calculate the frequency ratios in the discrete case**. Actually, the continuous ratios $\rho^d(t)$ are calculated from the speed setpoint or the frequency (regarding the chosen duplication scheme) and then compared with the possible discrete values. This eventually results in testing the inequality $1/3 < \rho^d \leq 1/2$ for instance. In fact one can reduce the control cost simply

substituting the expression of the ratios directly into the inequality. The previous algorithm - in the partial duplication case - hence yields

$$\varrho^d = \begin{cases} 1 & \text{if} \quad 1/2 \cdot \omega_{sp}^c(t) < \omega_{sp}^d(t) \leq \omega_{sp}^c(t) \\ 1/2 & \text{if} \quad 1/3 \cdot \omega_{sp}^c(t) < \omega_{sp}^d(t) \leq 1/2 \cdot \omega_{sp}^c(t) \\ 1/3 & \text{if} \quad 0 < \omega_{sp}^d(t) \leq 1/3 \cdot \omega_{sp}^c(t) \\ 0 & \text{otherwise} \end{cases}$$

Of course, this can also be applied to the full duplication case, replacing the speed setpoints by the expected frequencies.

3.2.4 Fully discrete control scheme

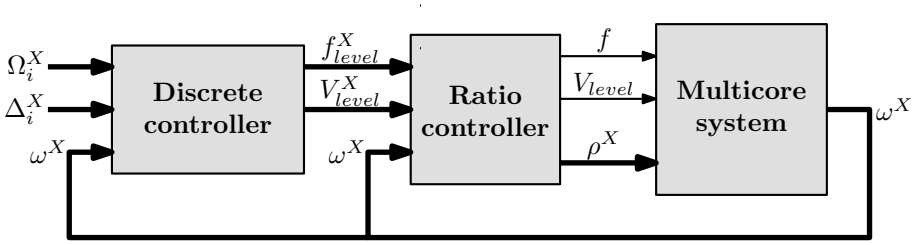
Different control strategies were presented in the previous sections to control several electronic devices with two discrete voltage levels and a continuous frequency range for each level. However, considering a continuously varying frequency is not realistic in practice and, for this reason, a discrete scheme would be preferable. Thus, as previously done in the monocore case, a fully discrete control scheme can be imagined. This means that the supply voltage and the clock frequency can only take some values: M voltage and N frequency levels are hence considered straight afterwards. Note that the definition of M and N - which was initially given in section 2.1 - is called back here just after. One could note that the control variables are now the voltage level $V_{level}(t)$ and the frequency level $f_{level}(t)$, and not the expected frequency $f(t)$ anymore.

- The input of a M-voltage level Vdd-hopping belongs to a M-value set and, with such a mechanism, the supply voltage V_m is provided when $V_{level}(t) = V_{level\_m}$. We define $m \in \{1, 2, ..., M\}$ and $V_m > V_{m+1}$, respectively $V_{level\_n} > V_{level\_n+1}$. Considering that this inner-loop is extremely fast with respect to the control loop considered here, the dynamics of the Vdd-hopping can be neglected.

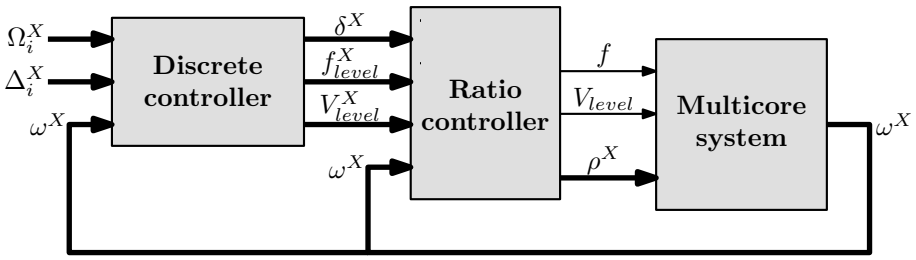
- The input of a N-frequency level oscillator belongs to a N-value set and, with such a mechanism the clock frequency f_n is provided when $f_{level}(t) = f_{level\_n}$. We define $n \in \{1, 2, ..., N\}$ and $f_n > f_{n+1}$, respectively $f_{level\_n} > f_{level\_n+1}$, and we choose $N \geq M$ (for the reasons explained in subsection 2.4.1). Eventually, switching from one level to another is considered as instantaneous.

In the previous subsections we introduced two different ways of extending a monocore control strategy to the multicore architectures. Both consist in ***i)*** calculating a speed setpoint in order to minimize the penalizing high voltage running time and then ***ii)*** calculating the control variables in such a way that the system speed tracks this setpoint. The first multicore architecture is based on a full duplication of the monocore strategy, whereas the other one applies a partial duplication in order to reduce the control computational cost (see subsections 3.2.1 and 3.2.2 respectively for further details). Actually, only the first part of the controller is repeated - in the partial proposal - and the control variables are deduced from the critical speed setpoint. However, in the monocore discrete control scheme (detailed in section 2.4) the control variables are directly obtained from the calculated speed setpoint - or more precisely from the predicted speed calculated thanks to the fast predictive control law - and no tracking is needed. In this context, the full and/or partial duplications does not have any sense since the part which is not duplicated in the second approach does not exist anymore. Anyway, a duplication remains here a solution because the monocore discrete control strategy has a low computational cost (see sections 2.4 and 2.5 for further details). As a result, **we propose to adapt the monocore**

discrete control strategy to the multicore scheme. Actually, in the full duplication the frequency ratios $\rho^X(t)$ are deduced from the frequencies $f^X(t)$ calculated for the different devices to control while the partial duplication deduces them from the speed setpoints $\omega^X_{sp}(t)$. Both approaches could be transposed to discrete ones where these ratios would be calculated from the *frequency levels* $f^X_{level}(t)$ and the *predicted speeds* $\delta^X(t)$ respectively. The control architectures for both proposals are presented in figure 3.5.



(a) Frequency ratios deduced from the frequency levels (the *discrete controller* calculates the control variables and then, a *ratio controller* deduces the critical frequency level and calculates the frequency ratios).



(b) Frequency ratios deduced from the predicted speeds (the *discrete controller* calculates the control variables and the predicted speeds and then, a *ratio controller* deduces the critical prediction and calculates the frequency ratios).

Figure 3.5: Closed-loop system: the multicore controller based on discrete scheme.

In fact the *discrete controller* algorithm is really similar to the one of the full duplication proposal but the control law is now simplified thanks to the discrete scheme. Indeed, this controller dynamically calculates the predicted speeds for all the devices, which allow to directly obtain the voltage and frequency levels. Then, the controller has to deduce the device to control in priority. The corresponding critical device is indexed with the letter c and the frequency level $f_{level}(t)$ and voltage level $V_{level}(t)$ sent to the actuators are finally those from the critical device, i.e. $f^c_{level}(t)$ and $V^c_{level}(t)$. This depends on the proposal:

Ratios deduced from the frequency levels: in this first approach the critical task is the one which has the maximal frequency level to fit with its deadline. In the previous subsections, the frequency ratios $\rho^N(t)$ are obtained by doing the ratio between the variable of the current device and the one of the critical device. That variable would have to be $f^d_{level}(t)$ in the present case. However, a ratio of the frequency levels is not possible since these variables are not proportional to the clock frequency. Indeed, the frequency level values are only some discrete numbers, only to indicate that the system has to run with the frequency f_n when the required level is f_{level_n} for instance. Consequently, the frequency ratios can only be either 1 or 0 if the computational load of the current task is completed or not.

Ratios deduced from the predicted speeds: in this second approach the critical task is

the one which has the maximal predicted speed to fit with its deadline. Eventually, the frequency ratios $\rho^N(t)$ are obtained by doing the ratio between the predicted speed of the current device $\delta^d(t)$ and the one of the critical device $\delta^c(t)$. However, the predicted speed was not really calculated in the original fully discrete control scheme (for a computational cost reason) - as explained in section 2.5 - and, consequently, using them will complexify the discrete algorithm.

Eventually, as for the full duplication scheme, the discrete control strategy ensures that the tasks are correctly performed for all devices because each device is independently controlled using the monocore strategy. Furthermore, this extension takes benefit from the robustness to process variability developed in the monocore case, thanks to the estimation of some system parameters (see subsection 2.4.4 for further details).

3.3 Several chips working with their own clock

In section 3.1, a multicore architecture was presented where all the devices to control work together since they are all supplied with the same voltage and clock frequency. Here, **we propose a new multicore architecture where each device is triggered with a different oscillator**. The new system architecture is shown in figure 3.6. The principle remains (almost) the same: a controller dynamically sends the frequency level and the voltage level to the *actuators* which respectively provide the clock frequency and the supply voltage to the *electronic devices*. The main difference is that there are now *X oscillators*, which independently provide a clock frequency to each device. As before, the *new multicore controller* has to control the whole system but devices still do not work independently since they are all supplied with the same voltage. This constraint has hence to be taken into account in the new multicore control strategy. One could note that the control variables are now the *voltage level* $V_{level}(t)$ and the *frequency levels* $f^X_{level}(t)$, one for each device. Moreover, note that the frequency ratios $\rho^X(t)$ introduced in the initial multicore architecture do not exist anymore.

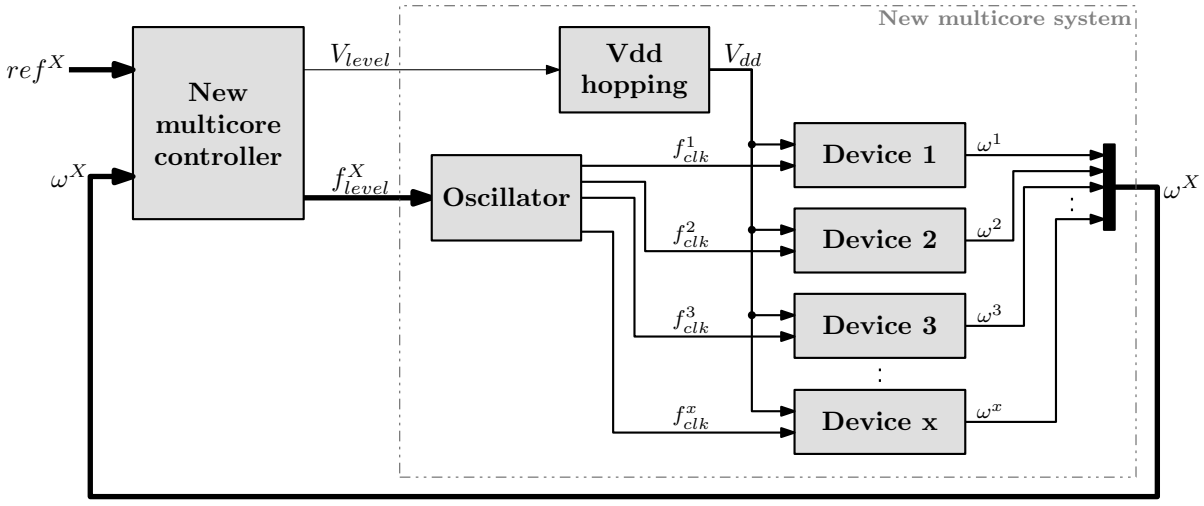


Figure 3.6: Architecture of the new multicore system.

In fact the *new multicore controller* algorithm is really similar to the previous one, but the control law is now simplified. Indeed, this controller now dynamically calculates the frequency levels $f^X_{level}(t)$ for all the devices and deduces the device to control in priority. Thus, the critical task is the one which has the maximal frequency level to fit with its deadline and the voltage

level $V_{level}(t)$ sent to the Vdd-hopping is the one from the corresponding critical device.

Eventually, the different control strategies previously developed in section 3.2 can be easily adapted to this new multicore architecture, but this will not be done here.

3.4 Synthesis

This chapter extends the monocore architecture - introduced in chapter 2 - to a multicore one, where several processing nodes, working together on a single chip, are all power supplied with the same voltage and frequency. However, a certain degree of freedom is possible thanks to some frequency ratios (which allow the different devices to work with a ratio of the clock). The monocore control strategies have hence been adapted to now control the whole system. The proposed strategies consist in focusing on the critical task to treat (the task which requires the maximal frequency/speed to fit with its deadline) and minimizing the penalizing high voltage running time of the corresponding device - to reduce the energy consumption - while guaranteeing the computational performance of the whole system. Different control schemes were hence proposed:

- An intuitive multicore strategy based on the full duplication of the monocore one allows to easily obtain a control algorithm but, as a result, this multiplies the control computational cost as many times as devices existing on the chip. Furthermore, this first strategy keeps the robustness and stability properties of the monocore proposals.

- A second strategy consists in not repeating the monocore scheme as many times as devices in order to reduce the control cost. The proposal deduces the frequency ratios using the calculated speed setpoint for each task. The controller hence does not need to be fully duplicated since the control variables are then calculated only for the critical device. Anyway, we developed a multicore control law which is still robust to process variability and stable.

- A fully discrete control scheme is finally also proposed for an architecture working with M voltage levels and N frequency levels.

At the end, another multicore architecture is depicted. This consists in controlling several chips working with the same supply voltage but with their own clock frequency. This second scheme is only presented but the resulting control laws are not detailed.

Chapter

4 Simulation results

Different architectures including one or several voltage scalable processors were presented in chapter 2 and 3. Theses components allow to make both the supply voltage and the clock frequency of the chip dynamically varying, in order to reduce the power consumption. However, the computational speed of the devices is reduced also in return and an energy-performance tradeoff has hence to be handled. Several control laws were developed for the monocore and the multicore scheme. They are recaped in section 4.1 as well as a brief presentation of the systems. The simulation test benches are also presented while some indexes are defined to evaluate the performance of the different strategies. At the end, the proposals are tested: the intuitive frequency and voltage controllers are executed in section 4.2, the more energy-efficient ones are simulated in section 4.3 and the fully discrete schemes are studied in section 4.4. The robustness to process variability was also demonstrated in these different sections. A performance analysis finally compares all of them in section 4.5, before synthesizing all the simulation results in section 4.6.

4.1 Presentations

Before performing some results, the context of the simulations is presented. This section starts with a recap of the different control strategies in subsection 4.1.1 while the monocore and multicore system parameters are given in subsection 4.1.2. The simulation test benches for both architectures are then presented in subsection 4.1.3 and, finally, some performance indexes are introduced in subsection 4.1.4.

4.1.1 Recap of the different control strategies

This subsection aims at summarizing all the control strategies developed in chapter 2 and 3 and afterwards used in simulations. Firstly, some basic existing strategies without power management (or quasi not) are introduced because they will be compared to our proposals:

System without DVFS mechanism: In a system without dynamic voltage and frequency scaling mechanism, only one voltage level and one frequency level are possible, that are the maximum ones. This is the usual running of all processors which are not voltage scalable.

System without DVS mechanism: In a system without dynamic voltage scaling mechanism, the clock frequency can vary. Thus the system speed has to track a given reference. The intuitive average speed setpoint building principle - depicted in subsection 2.3.1.1 - is used while the supply voltage is fixed to the most penalizing level. Moreover, the frequency is now controlled in order to ensure that what the system has to do is really done and this implies to use a complex law. That is a proportional integral controller applied to the integral of the error between the speed setpoint and the measured system speed (for the reasons explained in subsection 2.2.1).

These techniques do not really reduce the energy consumption since the supply voltage is the penalizing parameter in a power management scheme. For this reason, we developed several control strategies controlling both the frequency and the voltage levels. In a first hand, only a monocore system was considered in chapter 2:

Frequency and voltage level control: This controller - developed in section 2.2 - consists in scaling the frequency in such a way that the measured computational speed tracks a given speed setpoint and then adapting the voltage level to reduce the energy consumption. At the end, both control variables are adapted together to ensure the maximum delay over the critical path of the device to control and different restrictions were proposed in subsection 2.2.3. Two voltage levels and a continuously varying frequency allow such a mechanism. As in the previous case, the setpoint to track is also the average speed setpoint and a proportional integral controller has also to be applied to the integral of the error.

Computational speed control: This controller - detailed in section 2.3 - consists in minimizing the penalizing high voltage running time while guaranteeing good computational performance. This is possible in calculating a more energy-efficient speed setpoint thanks to a fast predictive control law. Two different closed-loop architectures were proposed to calculate the predicted speed and be able to obtain the control variables: ***i)*** a first one using the speed setpoint itself as the feedback and ***ii)*** a second using the measured speed instead. In the first scheme, the complex PI controller applied to the integral of the error is still required while a simple integral controller applied to the error is enough in the second case.

Fully discrete control scheme: The discrete controller - developed in section 2.4 - is an extension of the previous computational speed controller, for a fully discrete scheme running with M voltage and N frequency levels. As before, the control law consists in minimizing the penalizing high voltage running time while guaranteeing good computational performance and the control variables are now directly deduced from the predicted speed. This considerably reduces the complexity of the control law. Moreover, this proposal is highly robust to process variability (see section 1.2 for further details) since no information on the system is needed in the control law anymore.

Then, different strategies were also developed in chapter 3 for a multicore system, where several devices work together, running with the same supply voltage and clock frequency (or a ratio of the frequency). All the developed strategies consist in minimizing the penalizing high voltage running time of the critical task while guaranteeing good computational performance of all the devices:

Full duplication of the monocore strategy: This principle - introduced in subsection 3.2.1 - consists in repeating the monocore scheme as many times as devices and then deciding the control variables regarding the critical task (the task which requires the maximal frequency to fit with its deadline). This is based on the computational speed control law (depicted above). Two voltage levels and a continuously varying frequency are hence available to control the whole system.

Partial duplication: This second multicore proposal - detailed in subsection 3.2.2 - consists in not repeating all the monocore strategy in order to decrease the control computational cost. The critical task is deduced from the maximal speed setpoints, which leads to calculate the control variables only for the critical device. As previously, the referring monocore strategy is the computational speed control law.

Discrete frequency ratios: The two previous proposals consider continuously varying frequency ratios which allow to individually reduce the frequency of the devices. However, in practice, only discrete values are possible. Thus, some discrete frequency ratios are calculated - see subsection 3.2.3 - from the corresponding continuous ones.

Discrete control: The discrete controllers - developed in subsection 3.2.4 - are an extension of the full and partial duplication proposals, applied to the fully discrete control scheme (presented above). M voltage and N frequency levels are available to control the processing power of the different devices. Both approaches deduce the critical task either from the maximal calculated frequency levels or from the maximal predicted speeds.

All these control strategies will be tested in simulation in the next sections.

4.1.2 Controlled systems

In both monocore and multicore system architectures, the model of the controlled devices - introduced in subsection 2.1.1 - is $\omega(t) \simeq \alpha \cdot f_{clk}(t) + \beta$, where $\alpha = 0.08$ and $\beta = 38000$.

Then, two actuators provide the supply voltage and the clock frequency. Their model were presented in subsection 2.1.2. The first one is the Vdd-hopping and directly varies with respect to the voltage level $V_{level}(t)$. On the other hand, the oscillator model is function of the expected frequency, that is $f_{clk}(t) = \gamma \cdot f(t) \cdot V_{dd}(t)$ where $\gamma = 0.9038$. Moreover, the available frequency range - described in subsection 2.2.2 - is $0 < f_{clk}(t) < 500\,MHz$ while the different power supply schemes depicted in the previous subsection are summarized as follows:

1. In a first scheme, two voltage levels and a continuously varying frequency are possible.
   - The voltages are $V_{high} = 1.1065\,V$ and $V_{low} = 0.8\,V$.
   - On the other hand, the resulting frequency ranges are $F_{V_{low}min} < f_{clk}(t) < F_{V_{low}max}$ when $V_{dd}(t) = V_{low}$ and $F_{V_{high}min} < f_{clk}(t) < F_{V_{high}max}$ when $V_{dd}(t) = V_{high}$, where $F_{V_{low}min} = 0$, $F_{V_{low}max} = 350\,MHz$, $F_{V_{high}min} = 250\,MHz$ and $F_{V_{high}max} = 500\,MHz$.
   - The measured maximum computational speeds (used in the fast predictive control law, as explained in subsection 2.3.4) are $\omega^{max} = 40.038\,MIPS$ and $\omega_{max} = 20.282\,MIPS$ (in million of instructions per second).
   - Finally, the linear restriction to ensure the maximum delay over the critical path - introduced in subsection 2.2.3.2 - is $f(t) = a \cdot V_{dd}(t) + b$ under some conditions, where $a = 4.8940 \cdot 10^8$ and $b = -4.1517 \cdot 10^7$.
2. In the fully discrete scheme, the system could run with M voltage and N (discretely varying) frequency levels. In fact, we will only simulate 2 and 3 voltages.
   - In the first case, $V_1 = V_{high}$ and $V_2 = V_{low}$ while the possible frequencies are $F_{high} = 500\,MHz$, $F_{low} = 350\,MHz$.
   - In the second case, $V_1 = 1.1065\,V$, $V_2 = 0.9533\,V$ and $V_3 = 0.8\,V$ while the possible frequencies are $F_1 = 500\,MHz$, $F_2 = 425\,MHz$, $F_3 = 350\,MHz$ and $F_4 = 175\,MHz$.
   - Note that when the cock-gating principle - depicted in subsection 2.4.3 - is used, the resulting frequency is $F_0 = 0$ (in fact the clock is only paused but a null frequency will be used in simulation to highlight the clock-gating intervals) and applying this mechanism is possible only if $\Lambda_i(t) > \Lambda_{min}$, where $\Lambda_{min} = 5 \cdot T_s$.

The sampling period used in all the control laws - developed in chapter 2 and 3 - is $T_s = 4\,ns$. The parameters used in the different frequency control strategies are $K_p = 5 \cdot 10^7$ and $K_i = 6.25 \cdot 10^{14}$ for the controller applied to the integral of the error, and $K = 5 \cdot 10^7$ for the controller applied directly to the error. The anti-windup parameter is $K_a = 2.25 \cdot 10^7$ in both cases.

Eventually, as the energy is the key point in this part of the thesis, the consumption is calculated for the different architectures. The relation is detailed in subsections 2.1.3 and 3.1.2, where the parameter values are $K_{dyn} = 1.1435 \cdot 10^{-8}\,J$, $K_{sc} = 1.2653 \cdot 10^{-9}\,J$, $K_{leak} = 0.0633\,J$ and $K_{hopp} = 0.2\,J$ during the voltage transitions or $K_{hopp} = 0.03\,J$ during the steady-state intervals.

4.1.3 Test benches

As the different control strategies recalled in subsection 4.1.1 will be simulated in the next sections, some test benches are now detailed in order to be able to compare all the proposals. We will depict a scenario ***i)*** for the monocore scheme and then ***ii)*** for the multicore scheme (introduced in chapter 2 and 3 respectively).

Test bench for the monocore systems

Several tasks have to be executed in the monocore case. For each task the controller requires the number of instructions to treat and the available amount of time to do that (its deadline). These data are usually given by the operating system but here **we propose to pre-define a scenario for the simulation bench**. In the monocore case, this is:

***Monocore node*:** three tasks to execute. The first task starts with 4 instructions to do in $0.5\,\mu s$, then a 65 instruction task has to be executed in $2.5\,\mu s$ and the last one has to compute 10 instructions in $1\,\mu s$.

The scenario is finally represented in figure 4.1. The top plot shows the number of instructions to compute for each task, with respect to time, and the amount has to be executed at the end of a given task. The bottom plot shows the deadline and the laxity, also with respect to time. As regards the deadline, the value indicates at the beginning instant the available amount of time to treat the task. Note that the laxity - the remaining amount of time before the end of the task - could be simply built: at the beginning of the task the laxity is equal to the deadline and then, at each sampling interval the laxity is decreased of the value of the sampling time T_s in order to be null at the end.

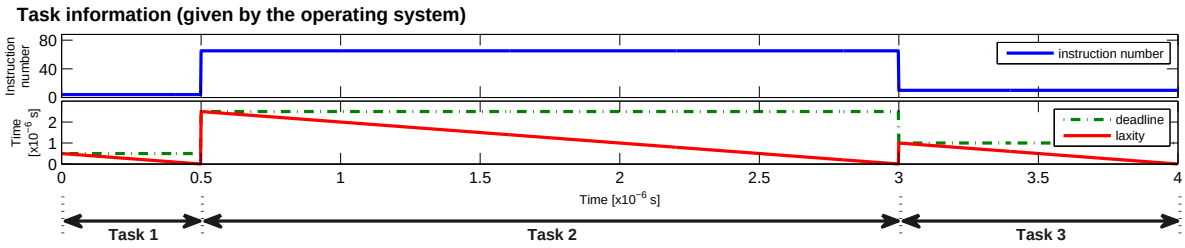


Figure 4.1: References used in the simulation test bench: the monocore control case.

Test bench for the multicore systems

Extending the bench to several electronic devices to control consists in defining references for each node. Four processing nodes have to be controlled together in the multicore case, with a specific reference for each one:

***Multicore node 1*:** three tasks to execute. The first task starts with 4 instructions to do in $0.5\,\mu s$, then a 65 instruction task has to be executed in $2.5\,\mu s$ and the last one has to compute 10 instructions in $1\,\mu s$.

***Multicore node 2*:** three tasks also. A 15 instruction task to execute in $1.25\,\mu s$, a task with 45 instructions to do in $2.25\,\mu s$ and then 5 instructions to execute in $0.5\,\mu s$.

***Multicore node 3*:** a single task of 40 instructions to do in $4\,\mu s$.

***Multicore node 4*:** three tasks again. 10 instructions to compute in $0.75\,\mu s$, a task with 20 instructions to do in $0.75\,\mu s$ and a last 40 instruction task to execute in $2.5\,\mu s$.

These data (usually provided by the operating system) are represented in figure 4.2. As previously, the top plot shows the number of instructions and the bottom plot shows the deadline and the laxity.

Note that, in the following multicore simulations, a legend will be shown only for the results of the third device, in order to lighten the plots.

4.1.4 Indexes of performance

By running one or the other test bench with the different controllers will lead to some simulation results in the next sections, where only the computational speeds - the measurement

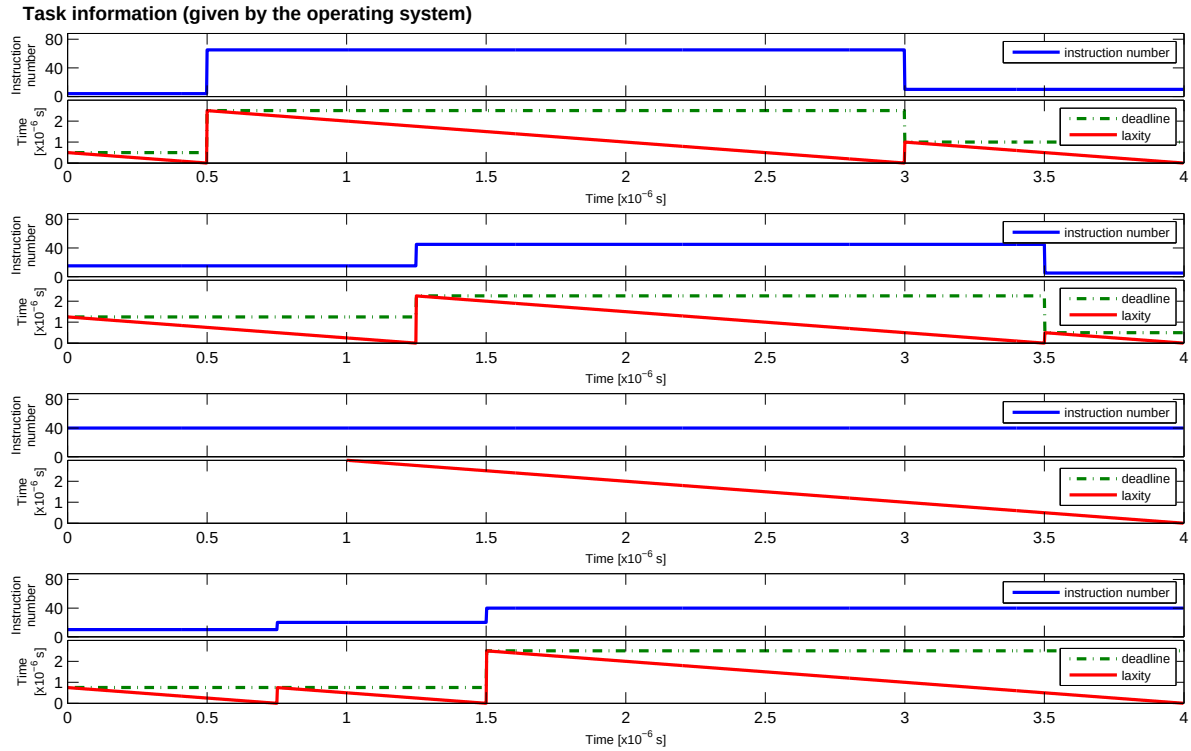


Figure 4.2: References used in the simulation test bench: the multicore control case.

$\omega(t)$, the setpoint $\omega_{sp}(t)$ or the predicted speed $\delta(t)$ - and the supply voltage $V_{dd}(t)$ are drawn. The clock frequency $f_{clk}(t)$, the expected frequency $f(t)$ or the frequency level $f_{level}(t)$, and the voltage level $V_{level}(t)$ are not plotted because they do not provide relevant information. Indeed, the frequencies are directly proportional to the speed while the voltage level can be deduced from the voltage. Note that these variables were previously defined in previous chapters. The computational speeds are given in number of instructions per second (afterwards denoted IPS) while the voltage is given in volt (denoted V). Both are represented with respect to time. Then, the results are quantified in terms of energy consumption of the system and computational cost of the control law:

Energy consumption of the system: The energy consumption is calculated in order to have an idea of the reduction achieved thanks to our proposal. The power consumption is given in equation (2.5) in the monocore case and equation (3.2) in the multicore one. They relate to two parts, that are a consumption due to the electronic components plus a ratio added due to the Vdd-hopping principle. Finally, an integration during the whole simulation running time gives the total energy consumption. Note that *this index of performance is denoted E* in the following plots of the simulation results and is given in joules (or equivalent-joules afterwards denoted eJ).

Computational cost of the control law: The control law is compared in terms of computational needs, that are the number of operations required to calculate the control variables. We base this analysis on the *Lightspeed Matlab toolbox* proposed by *T. Minka* in [49], which provides a number of operations (afterwards denoted OPs) for each instruction. The cost is different for an addition, a multiplication (twice the cost of an addition) or a division (the most consuming operation which is eight times the cost of an addition) for instance. Note that *this index of performance is denoted C* in the following plots of the simulation

results and is given in number of operations.

Finally, the proposed controllers are compared all together and with some classical systems without power management technique. All were recaped in subsection 4.1.1.

4.2 Frequency and voltage level control to track a given computational speed setpoint

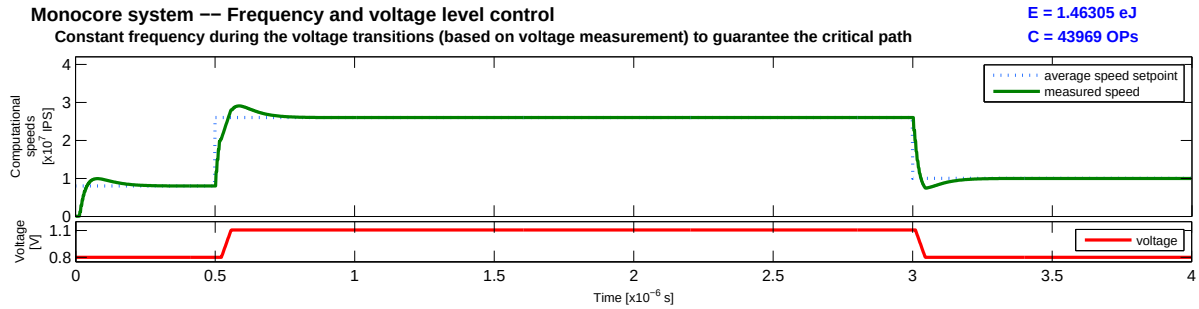
The frequency and voltage level control strategy - introduced in section 2.2 and brought back in subsection 4.1.1 - was developed for a system running with two voltage levels and a continuously varying frequency. That consists in scaling the frequency in such a way that the measured computational speed tracks the given speed setpoint - that is the intuitive average speed setpoint here - to fit the tasks to treat with their deadline. Then, the lower voltage level is applied as soon as the computational speed becomes low in order to reduce the energy consumption. At the end, both control variables are adapted together and, if necessary, a restriction is set during the voltage transitions in order to ensure the maximum delay over the critical path of the chip. Indeed, it is required to decrease the frequency before decreasing the voltage and, respectively, to increase the voltage before increasing the frequency (see subsection 1.3.2 for further details). In this section, **we propose to analyze the impact of the different restricting conditions on the energy consumption of the system and their computational cost in the control law**. The different proposals are tested and lead to the simulation results plotted in figure 4.3. The top plots show the average speed setpoint and the measured speed while the bottom plots show the supply voltage.

- The first condition is to set the frequency constant during a voltage transition. Thus, when the voltage increases from the low level to the high one - actually there is no problem when the voltage decreases - the frequency is fixed to the maximal possible frequency at low voltage during the whole transition. This can be done ***i)*** measuring the supply voltage (represented in figure 4.3(a)) or ***ii)*** simply knowing the current state of the voltage (represented in figure 4.3(b)). The system energy consumption and the control computational cost are quite close in both cases. However, the first case requires a more complex hardware solution to catch the voltage measurement while the state of the supply voltage can be easily provided by the Vdd-hopping (the voltage actuator detailed in subsection 2.1.2.1). For this reason, the second measured signal would be preferred.

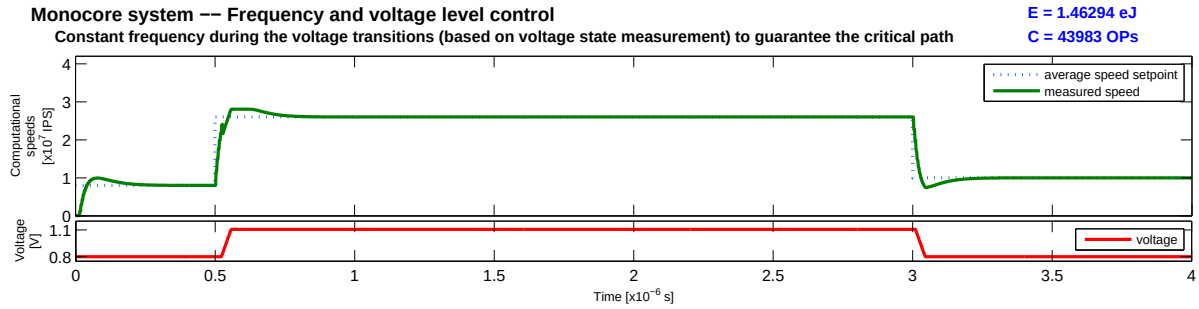
- A second condition is to make the frequency linearly varying during the voltage transitions in such a way that the system could run faster than in the previous (constant) case. Applying this behavior is represented in figure 4.3(c). In fact the resulting control strategy is more complex - and so is increased the control computational cost - while no gain is really provided on the energy consumption of the system (because the voltage transitions are very fast thanks to the Vdd-hopping mechanism). Moreover, this implies to measure the supply voltage since a linear transition varies with respect to that variable.

- The last proposal does not take care about the critical path in the control law. This problem is taken into account directly in the oscillator (the frequency actuator detailed in subsection 2.1.2.2) which provides the clock frequency and is function of the supply voltage anyway. The results without any restriction are given in figure 4.3(d). The complexity of the control algorithm is reduced (about 10 % of operations less than before) without impacting the system energy consumption. Note that this strategy will be applied afterwards (for simplification reasons).

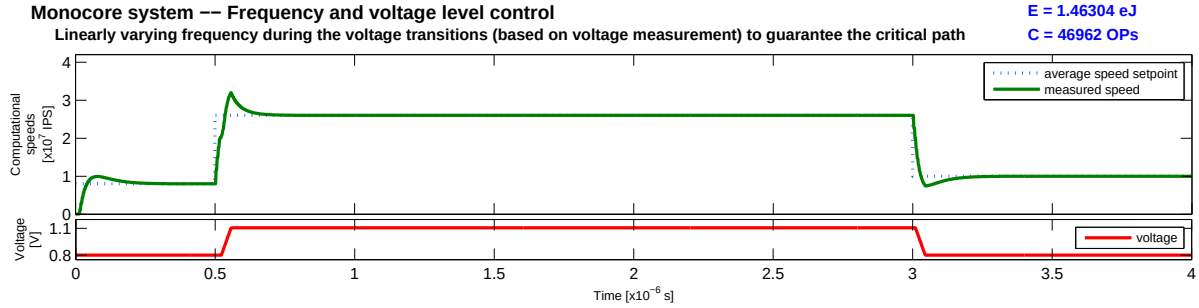
Finally, all simulation results allow the measurement to correctly track the setpoint. A reduction of the energy consumption of about 10 % is achieved compared to a system without DVS mechanism, and 55 % compared to a system without DVFS mechanism. However, when a task has an important computational load and that the high voltage level is required, the system runs



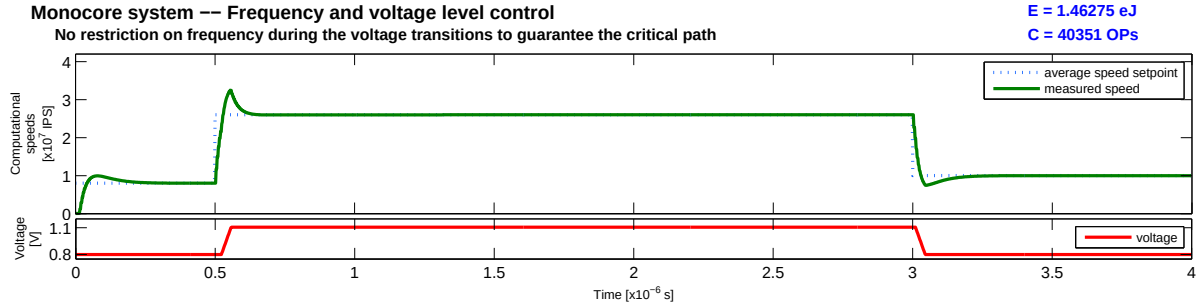
(a) Constant frequency (restriction based on the voltage measurement)



(b) Constant frequency (restriction based on the voltage state measurement)



(c) Linearly varying frequency



(d) No restriction in the control law

Figure 4.3: Simulation results of the frequency and voltage level controller: different proposals to ensure the maximum delay over the critical path during the voltage transitions.

during the whole task at high voltage and therefore consumes a lot. At the end, the system runs during more than 60 % of the simulation time at high voltage with the present test bench. A more energy-efficient speed setpoint will then reduce much more the high voltage running time. The corresponding simulation results are depicted in the next section.

4.3 Computational speed control to build a more energy-efficient setpoint

The computational speed control strategy - introduced in section 2.3 and brought back in subsection 4.1.1 - was also developed for a system running with 2 voltage levels and a continuously varying frequency. The principle consists in minimizing the penalizing high voltage running time - using a fast predictive control law - while guaranteeing good computational performance. In this section, **we propose to highlight the advantages of the predictive proposal** in both monocore and multicore controlled systems.

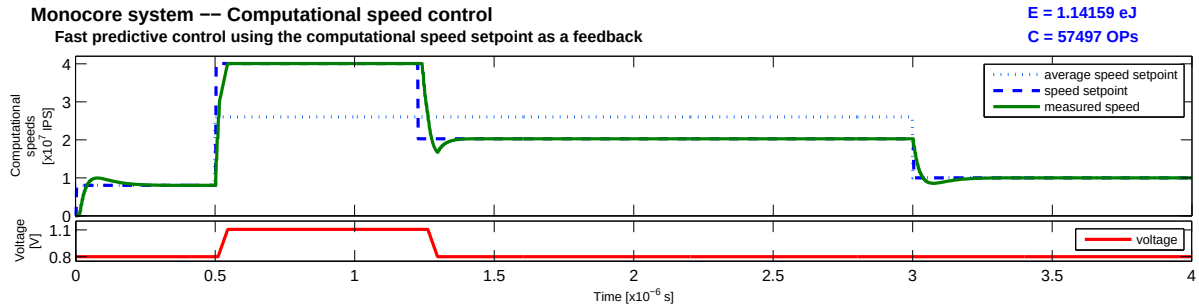
4.3.1 Fast predictive control law

In order to minimize again the energy consumption, a predictive control strategy is applied. This consists in beginning a task with an important computational load at the high voltage level and with the maximum possible speed. Thus, after a given amount of time, the system could go back to the low voltage and the task be finished with this less penalizing level. Such tasks are hence divided into two parts instead of being executed at high voltage for its whole running (at this was the case in the previous section tracking the intuitive average speed setpoint). Actually, the fast predictive control law allows to calculate an energy-efficient speed setpoint - which minimizes the penalizing high voltage running time - and then, the frequency and voltage level controller can make the speed measurement tracking this reference. Two closed-loop systems were suggested in section 2.3 to build this computational speed setpoint: either ***i)*** using the speed setpoint itself as a feedback or ***ii)*** using the measured speed as a feedback. Both architectures are tested and lead to the simulation results plotted in figure 4.4. The top plot shows the average speed setpoint (for guideline), the speed setpoint (calculated thanks to the predictive control law) and the measured speed while the bottom plot shows the supply voltage. One can verify that the penalizing voltage running time is drastically reduced compared to the results in figure 4.3(d) (about 45 % of reduction of the penalizing high voltage running time with the same test bench) and so is the energy consumption. Eventually, the main differences between the two plots are during the voltage transitions:

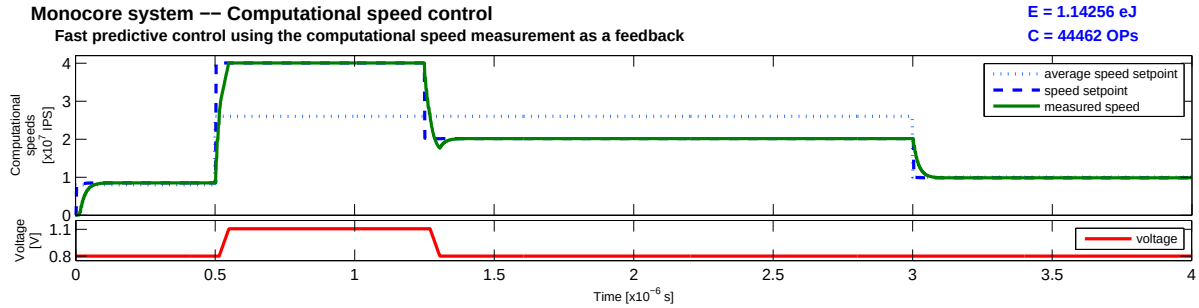
- In the measurement feedback case, the computational speed setpoint is calculated regarding what the device has really done. Thus, during the voltage transitions the setpoint is adjusted with respect to the measured error, as one can see in figure 4.4(b): the setpoint is reduced during falling transitions if the system ran faster than required (due to the Vdd-hopping dynamics) and, inversely, the setpoint is increased during rising transitions.
- In the setpoint feedback case, the computational speed setpoint does not change with respect to the speed measurement and the system speed has hence to adapt itself in order to compensate. As a result, some overshoots appear during rising transitions (respectively undershoots appear during falling transitions), as one can see in figure 4.4(a).

Anyway, in both cases the system runs during more than 80 % of the simulation time at low voltage with the present test bench. A reduction of the energy consumption of about 35 % is achieved compared to a system tracking the average speed setpoint (depicted in previous section), 30 % compared to a system without DVS mechanism and 65 % compared to a system without

DVFS mechanism. These results are quite interesting. Moreover, while the energy consumption is very similar for both architectures, the control computational cost varies and the number of operations needed in the measurement feedback case is about 20 % less than in the setpoint one. This is due to the frequency control law which is more complex in the second case, as explained in subsection 2.3.5. The measurement feedback is hence preferable and, consequently, will be used afterwards.



(a) Computational speed setpoint used as a feedback

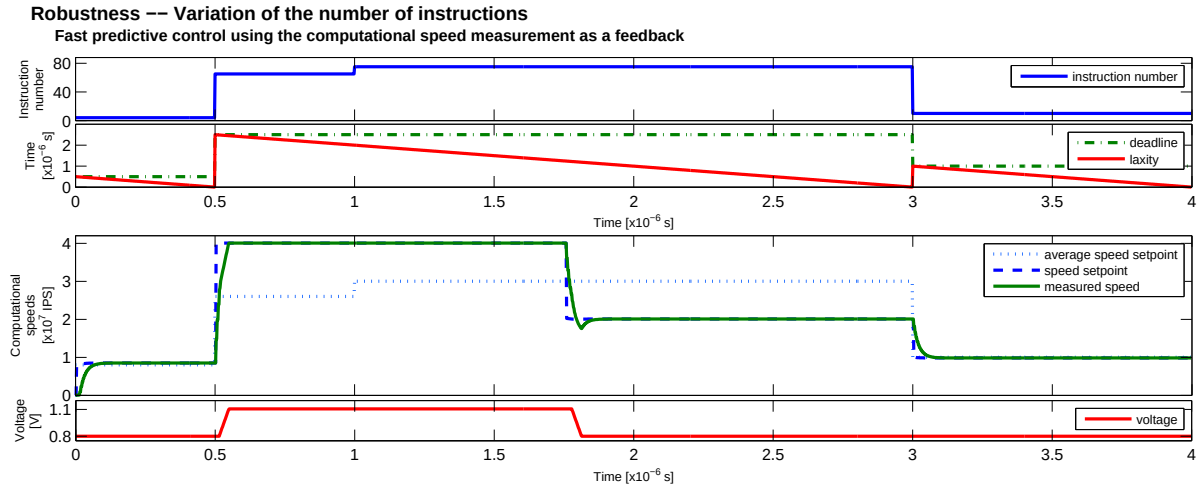


(b) Computational speed measurement used as a feedback

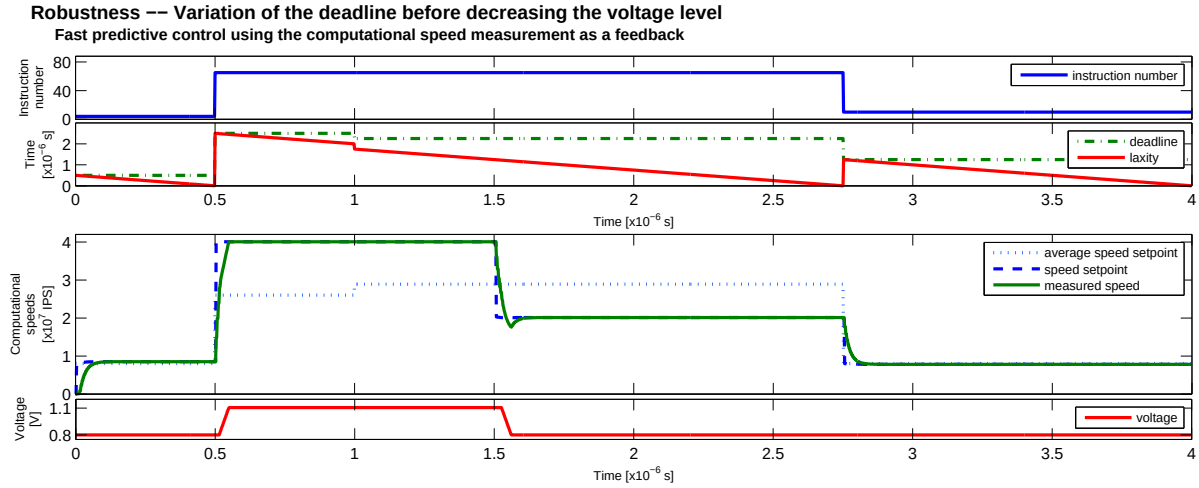
Figure 4.4: Simulation results of the computational speed controller: interest of the fast predictive control law.

4.3.2 Adaptation to a variation of the task information

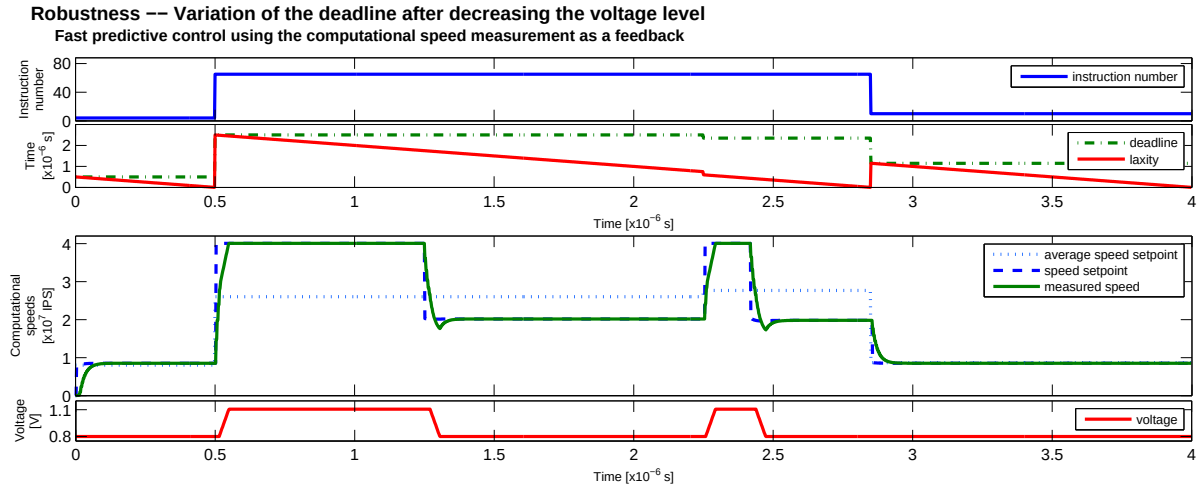
As previously explained, both architectures adapt either the speed setpoint or the system speed to compensate for a possible error in the tracking during the voltage transitions. For this reason, the control strategy has to adapt itself to some variations of the reference given for each task by the operating system. This is important when the exact number of instructions to compute is not a priori known (when some loops exist in the code of the application for instance) or the deadline. Such behaviors are depicted in figure 4.5 in the case where the speed measurement is used as a feedback (this also works in the other case). The references are represented on the simulation results in order to see when the variation happens. One could remark that the controller decides to increase the high voltage running time when the modification occurs before decreasing the voltage level, in order to be able ***i)*** to execute the extra computations or ***ii)*** to fit the task with its new deadline, such as in figures 4.5(a) and (b) respectively. On the other hand, when the modification occurs after decreasing the voltage level, the system needs to run at high voltage again, such as in figure 4.5(c). Anyway, the tasks are correctly executed in all cases. Note that if the modification occurs too late to correctly fit the task with its deadline, the task will be performed anyway overlapping on the next one.



(a) Modification of the instruction number: 10 extra instructions to treat at time $1\,\mu s$



(b) Modification of the deadline: at time $1\,\mu s$, it finally remains $1.75\,\mu s$ to treat the task instead of $2\,\mu s$ as initially planned



(c) Modification of the deadline: at time $2.25\,\mu s$, it finally remains $0.6\,\mu s$ to treat the task instead of $0.75\,\mu s$ as initially planned

Figure 4.5: Simulation results of the computational speed controller: robustness to a variation of the reference of the second task during its running time.

4.3.3 Duplication of the monocore strategy

As explained in chapter 3, we suggested to duplicate the monocore control strategy in order to control four devices working together. The first idea was to ***i)*** fully duplicate the architecture as many times as devices and, at the end, decide the control variables regarding the critical task (the task which requires the maximal frequency to fit with its deadline). The second approach suggests to ***ii)*** not repeat all the monocore strategy in order to decrease the computational cost of the controller. Eventually, some ratios allow a certain dimension of freedom in both cases, in triggering the devices with different frequencies. The simulation results for both control strategies (with *ideal* continuous frequency ratios) are shown in figure 4.6. The four top plots - one for each controlled device - show the average speed setpoint (for guideline), the speed setpoint, the measured speed and the critical speed (for guideline too). Note that the critical speed is the maximal speed between all devices, and one could check for instance that this maximal speed is the one of the critical device (highlighted by a gray area on plots). The bottom plot shows the supply voltage which is the same for the whole set of devices since they all work together. One can remark that the system speed perfectly tracks the speed setpoint, thanks to some continuously varying frequency ratios which allow a large possible frequency range. Finally, the system runs during about 80 % of the simulation time at low voltage. A reduction of the energy consumption of about 35 % is achieved compared to a system without DVS mechanism and 75 % compared to a system without DVFS mechanism. The differences between the two control strategies are during the voltage transitions and come notably from the choice of the critical device:

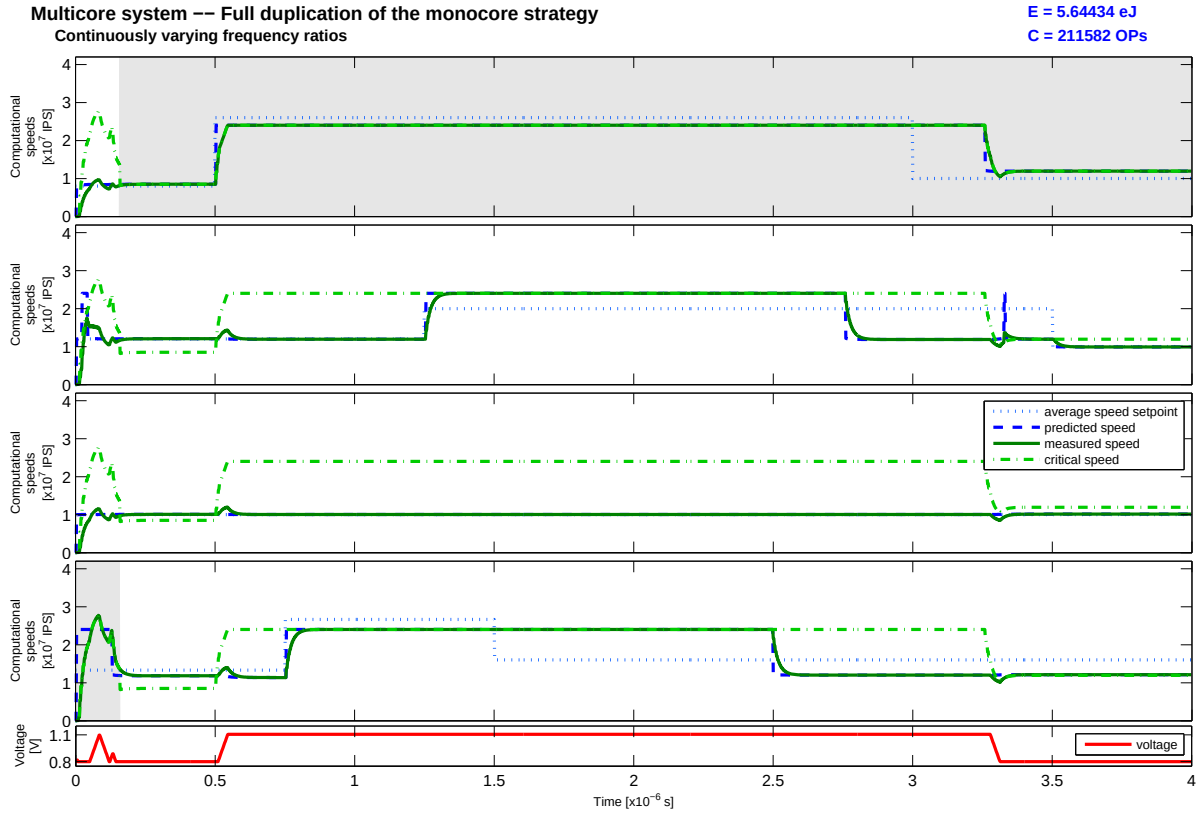
- In the multicore control strategy based on full duplication (subsection 3.2.1), one could see in figure 4.6(a) that the measured speed is continuously varying for all the devices. This is because the frequency ratios are obtained from the frequencies (independently calculated for each device).

- In the strategy based on partial duplication (subsection 3.2.2), one could see in figure 4.6(b) a discontinuity in the measured speeds as soon as the critical device changes. Indeed, the frequency ratios are obtained from the speed setpoints which are in fact some switching variables due to their construction (see section 2.3 for further details). The speed setpoint value of a device could suddenly change and so are the ratios. Nevertheless, the critical frequency - and so the critical speed - remains continuously varying.

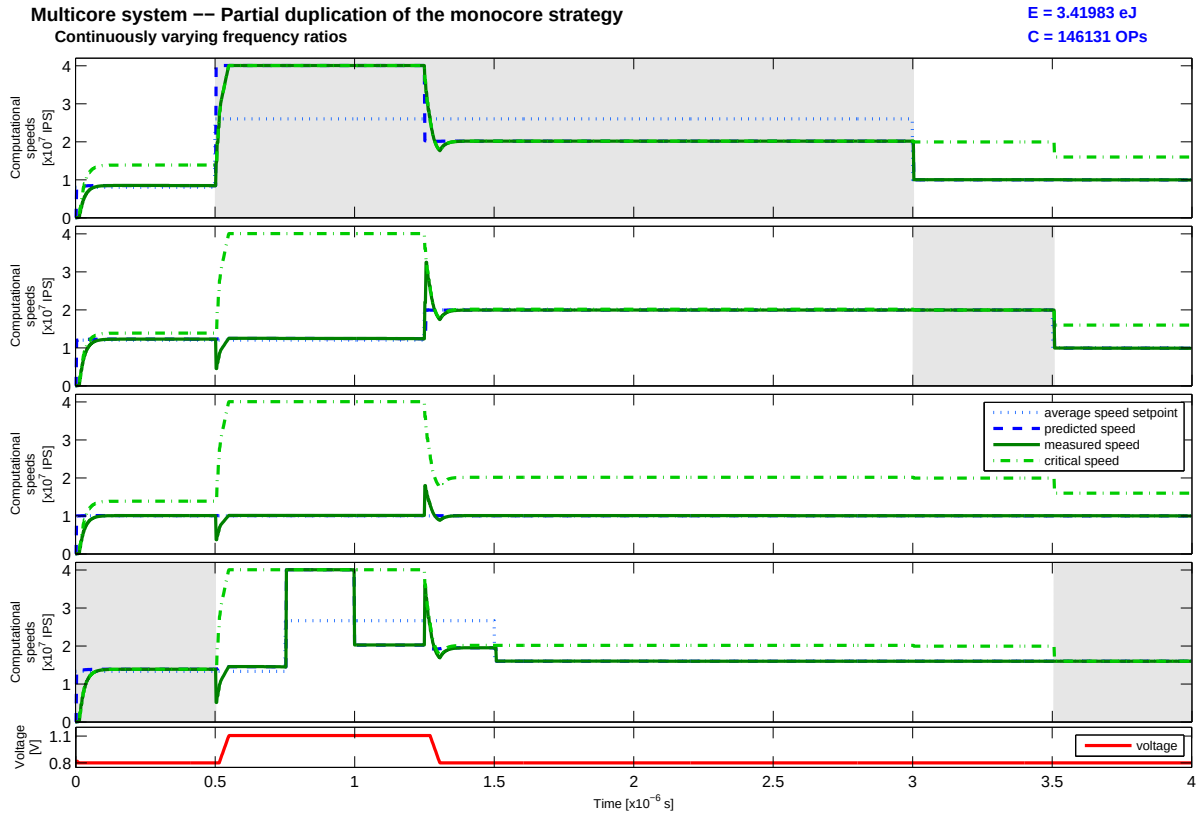
At the end, the computational needs are considerably reduced for the second scheme whereas the energy consumption is very similar in both cases. A drop of the number of operations of about 35 % is achieved. For this reason, it would be the strategy to use in practice.

4.3.4 Discrete values of the frequency ratios

The frequency ratios of the control strategy with partial duplication are then discretized (see subsection 3.2.3). The results are drawn in figure 4.7. One could immediately remark that the results are quite similar to the previous ones (where the ratios were continuously varying). The main difference is that the measured speed does not track the speed setpoint in the discrete case as well as in the continuous case (due to the small number of possible discrete values). Nevertheless, the amount of computations to do is correctly computed at the end of the task because the speed is at least higher than the speed setpoint - by construction - and so is the computational load. Furthermore, this discrete scheme is interesting since it leads to reduce the control computational cost (about 10 % less) without impacting too much the energy consumption. One could also note that decreasing the number of possible discrete values leads to

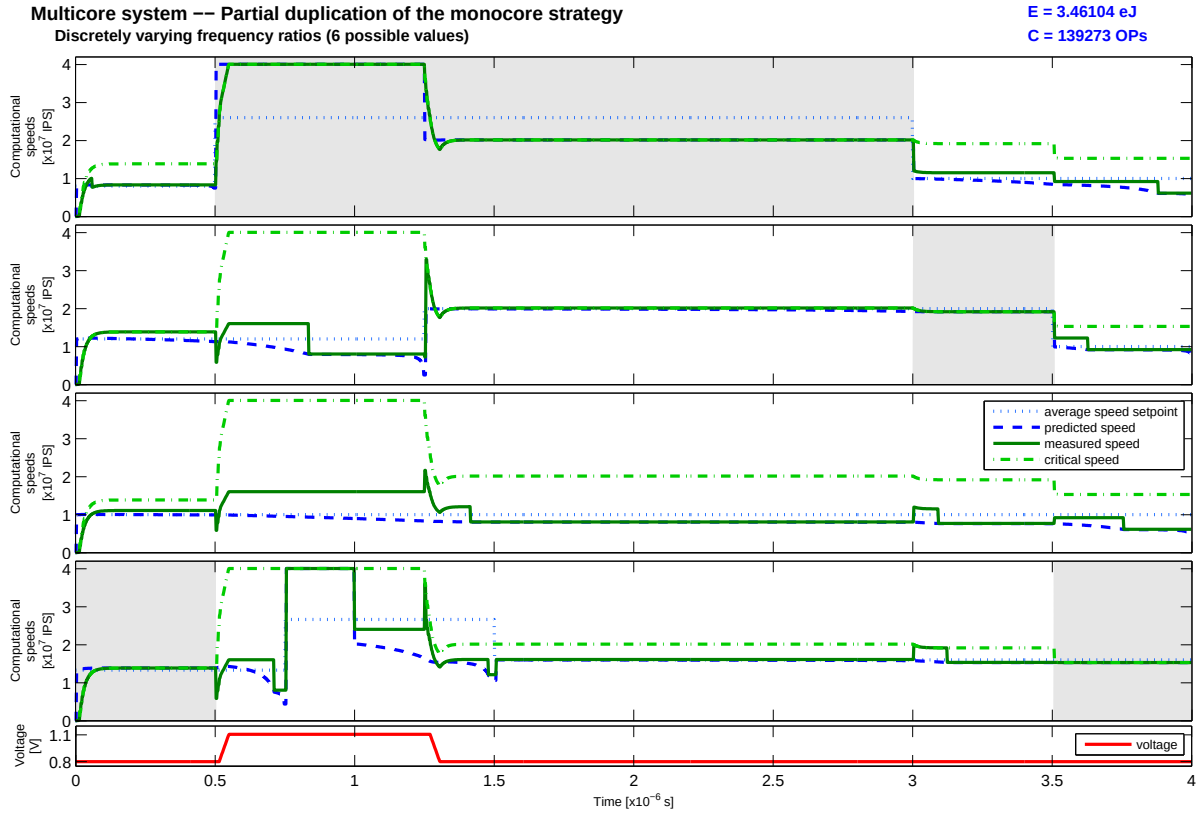


(a) Full duplication of the monocore strategy

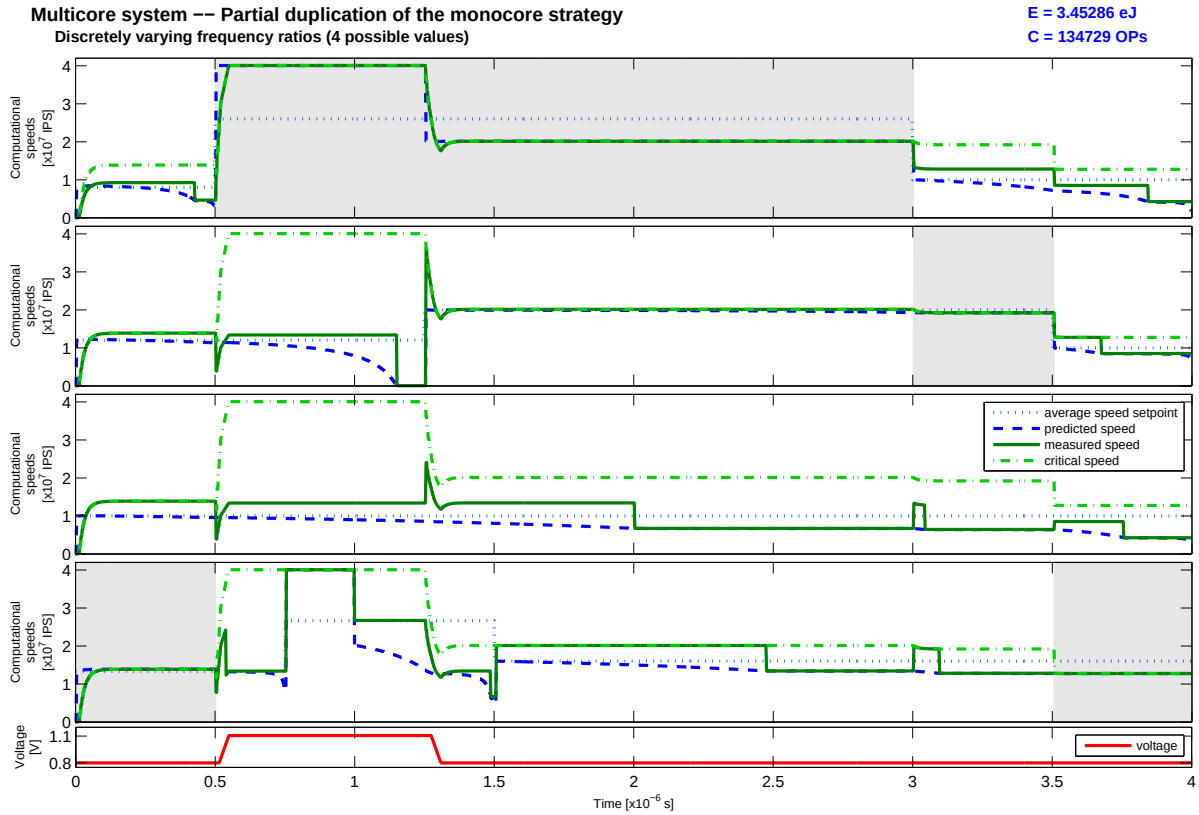


(b) Partial duplication of the monocore strategy

Figure 4.6: Simulation results of the multicore controller based on a duplication of the monocore strategy: continuously varying frequency ratios.



(a) 6 possible discrete frequency ratio values (1, $^4/_5$, $^3/_5$, $^2/_5$, $^1/_5$ and 0)



(b) 4 possible discrete frequency ratio values (1, $^2/_3$, $^1/_3$ and 0)

Figure 4.7: Simulation results of the multicore controller based on partial duplication of the monocore strategy: discretely varying frequency ratios.

reduce the computational cost (because the extra code to calculate the discrete frequency ratios is reduced) but increases the energy consumption in return (because the difference between the speed setpoint and the possible system speed of the systems increases). This is highlighted comparing the results in figures 4.7(a) and (b), where six and four possible discrete frequency ratio values are possible respectively. This discrete scheme would be the strategy to use (the continuous one could not be implemented in practice anyway).

Eventually, a last simulation is performed. In this scheme, the frequency ratio behavior runs as an on-off mechanism: the ratio value can only be equal to 1 or 0, that is either the device runs with the clock frequency or the device is stopped (using the clock-gating technique depicted in subsection 1.3.1). Results are shown in figure 4.8 where one can see a very chaotic behavior due to the drastically reduced possibilities. Nevertheless, the system is still working anyway.

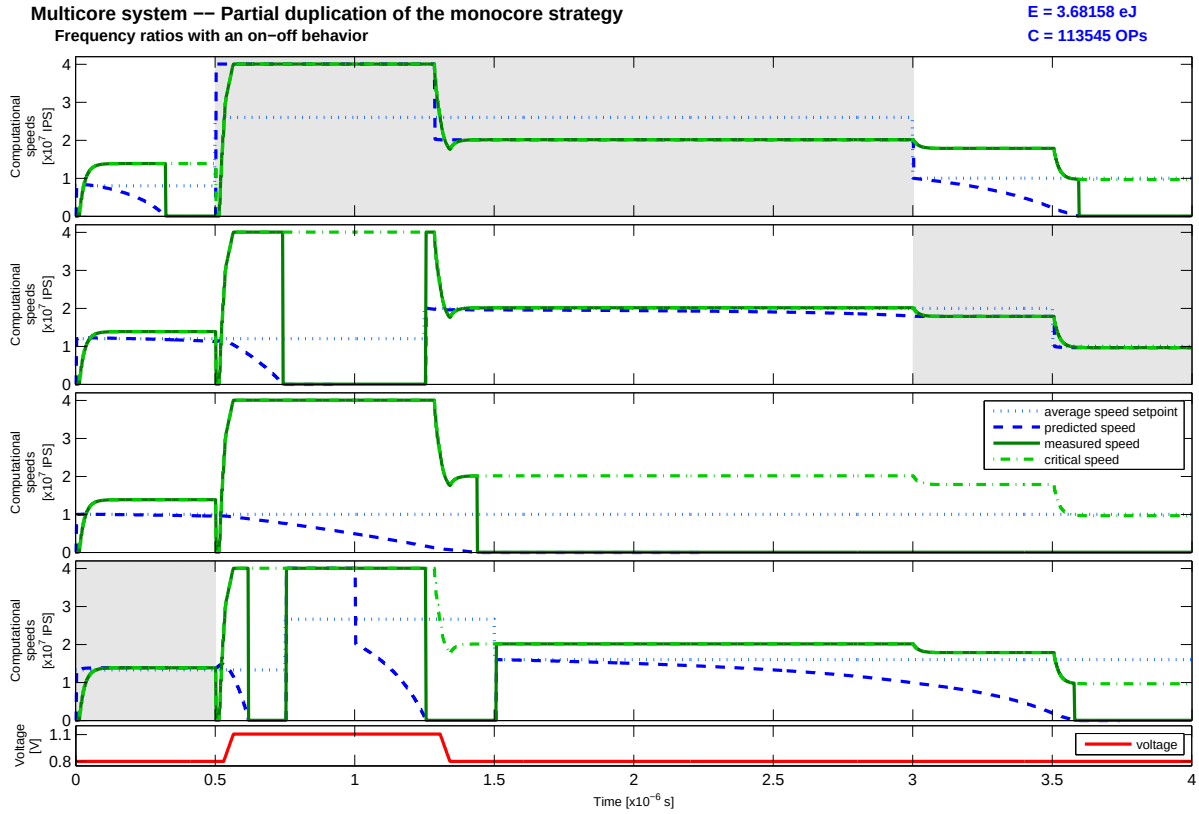


Figure 4.8: Simulation results of the multicore controller based on partial duplication of the monocore strategy: on-off frequency ratios.

4.4 Fully discrete control scheme

The fully discrete scheme control strategy - introduced in section 2.4 and brought back in subsection 4.1.1 - was developed for a system running with M voltage levels and N frequency levels. Actually, this strategy is an extension of the computational speed control one (illustrated in the previous section). The principle hence consists in minimizing the penalizing high voltage running time while guaranteeing some good computational performance. In this section, **we propose to show that the energy consumption can be strongly reduced even if the number of voltage/frequency levels is very small**. This will be applied to both monocore and multicore controlled systems.

4.4.1 Results with small numbers of voltage and frequency levels

As in the previous case, a fast predictive control law allows to minimize the penalizing high voltage running time. In this fully discrete scheme the control variables are immediately deduced from the predicted speed and this is not required anymore to explicitly calculate a speed setpoint. Eventually, different values of voltage and frequency levels are tested in this subsection:

***Scheme 1*:** 2 voltage levels and 2 frequency levels,

***Scheme 2*:** 2 voltage levels and 2 frequency levels using the clock-gating principle,

***Scheme 3*:** 2 voltage levels and 3 frequency levels,

***Scheme 4*:** 2 voltage levels and 3 frequency levels using the clock-gating principle,

***Scheme 5*:** 3 voltage levels and 3 frequency levels,

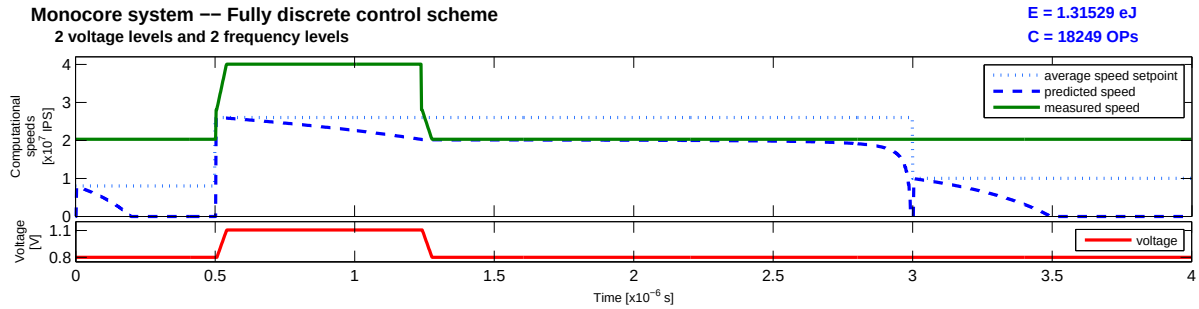
***Scheme 6*:** 3 voltage levels and 3 frequency levels using the clock-gating principle.

The different schemes are then tested and lead to the simulation results plotted in figures 4.9 and 4.10. The top plots show the average speed setpoint (for guideline), the predicted speed (for guideline) and the measured computational speed while the bottom plots show the supply voltage. Note that the predicted speed is plotted instead of the speed setpoint (as it was the case in section 4.3). This variable is dynamically calculated and represents the remaining number of instructions to treat the current task. Moreover, one can see that this variable decreases (almost) all the time because the levels are always higher (or equal) than required - by construction - due to the limited number of frequency values.

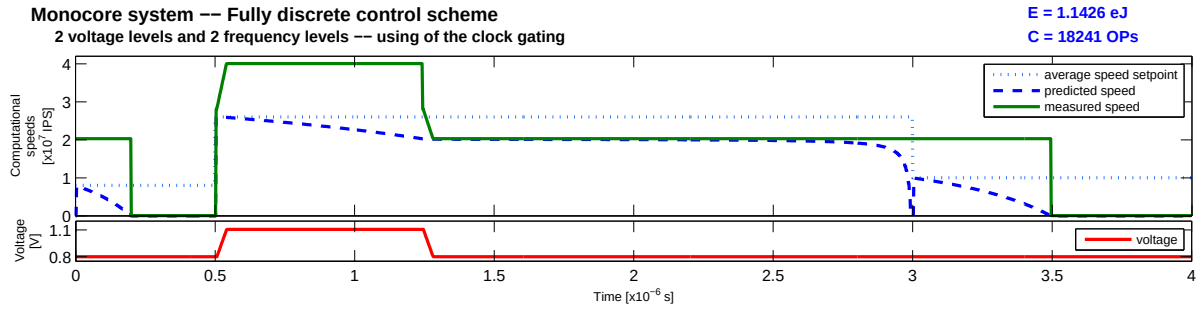
Looking at the first simulation results, one could immediately note that with two possible voltage levels, the system runs during almost 80 % of the simulation time at low voltage. This is shown in figure 4.9. A reduction of the energy consumption of about 30 % is achieved compared to a system without DVS mechanism and 65 % compared to a system without DVFS mechanism. Furthermore, the discrete schemes require a lower computational cost than in the continuously varying frequency case - previously represented in figure 4.4(b) - because, for almost the same energy consumption reduction, the control computational cost is divided by more than two. In fact, only the first discrete scheme - with 2 voltage and 2 frequency levels - is a little bit more consuming since the frequency cannot be lower than the maximal possible one at low voltage. However, the other schemes allow to achieve a consumption as close as using a continuous frequency, which is quite amazing. On the other hand, with three voltage levels the system does not need to go to the highest level to treat the three tasks of the proposed test bench, as shown in figure 4.10, but it runs a larger time (during about 60 % of the simulation time) at the second voltage level to compensate. This leads to a reduction of the energy consumption of about 10 % anyway whereas the paying tradeoff is an increase of the control computational cost (about 10 % more) due to the extra voltage level to control. Eventually, applying or not a clock-gating mechanism and the impact of the number of levels is analyzed as follows:

Clock-gating principle: Comparing figures 4.9(a) and (b) (or figures 4.9(c) and (d) for instance) shows that the energy consumption can be reduced again when using a clock-gating mechanism (detailed in subsection 2.4.3). The principle consists in pausing the clock of the device when a task is performed before its deadline, such as for the task 1 and 3 on the simulation results. This is not applied when the task almost fits with its deadline since it is preferable to wait with the current frequency level until the beginning of the next task instead of pausing the frequency.

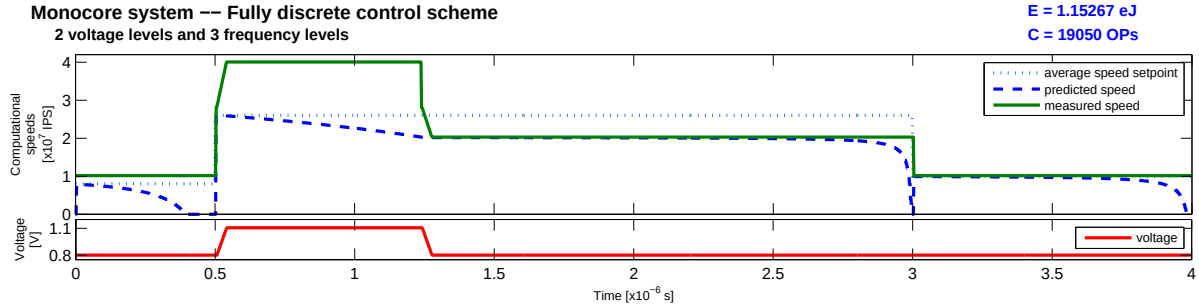
Number of voltage/frequency levels: Comparing figures 4.9(b) and (d) for the number of frequency levels, or figure 4.9(d) and figure 4.10 for the voltage levels, shows that in both cases, the control computational cost increases with respect to the number of levels - due to the extra levels to manage - without reaching a better energy consumption saving.



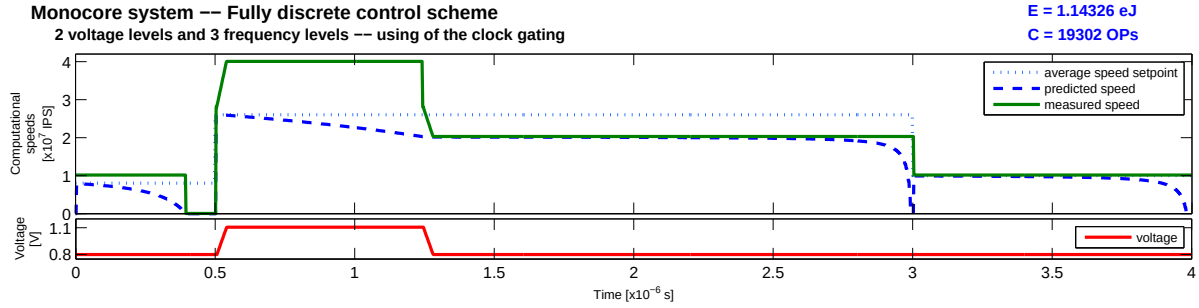
(a) 2 voltage levels and 2 frequency levels



(b) 2 voltage levels and 2 frequency levels using the clock-gating principle



(c) 2 voltage levels and 3 frequency levels



(d) 2 voltage levels and 3 frequency levels using the clock-gating principle

Figure 4.9: Simulation results of the multicore controller based on discrete scheme: 2 voltage levels and a very small number of frequency levels.

The number of levels is important anyway, but it is important to notice that in practice, designing a circuit with several voltage levels is more complex and less area-efficient than adding some possible frequency levels in the oscillator. For this reason, one would prefer to have a small number of voltage levels - two seem to be enough - and choose the number of frequency levels regarding the expected performance.

To summarize, the energy consumption is quite similar between the continuously varying frequency architecture and the fully discrete scheme where only some small numbers of voltage and frequency levels are possible. However, the control computational cost is highly decreased in the second case since the frequency and voltage levels are directly deduced from the predictive control law (without requiring to calculate a speed setpoint and applying a setpoint tracking). Eventually, the gain on the control computational cost is reduced when the number of voltage levels increases. For these reasons, the fully discrete scheme is preferable with only two voltage levels and few frequency levels (three or four seem to be enough).

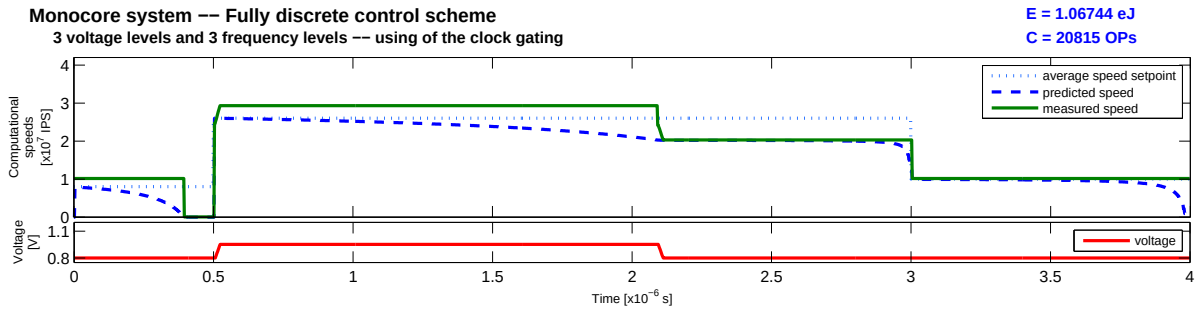
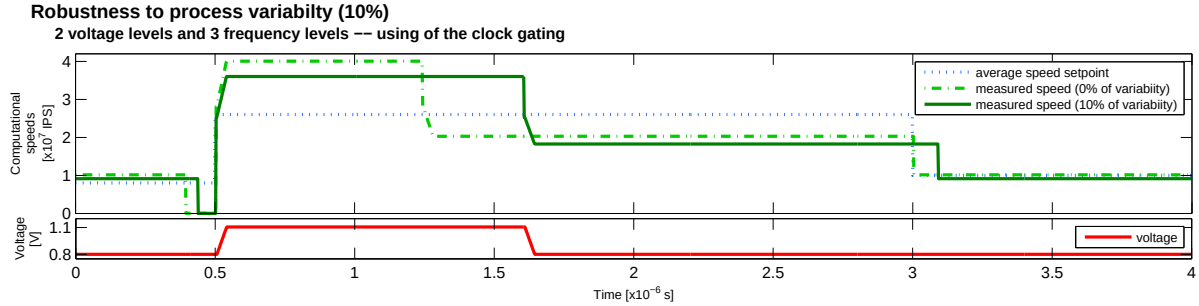


Figure 4.10: Simulation results of the multicore controller based on discrete scheme: 3 voltage levels, 3 frequency levels and the clock-gating principle.

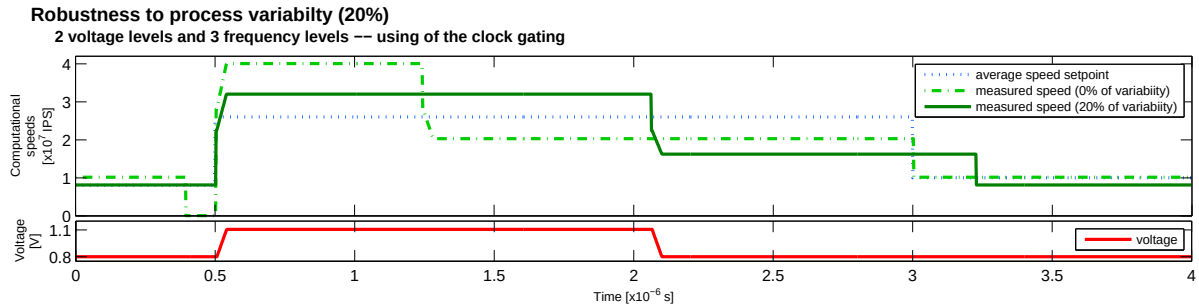
4.4.2 Robustness to process variability

The full discrete control scheme proposed in section 2.4 is highly robust to process variability (see section 1.2 for further details). Just to recall the idea, this issue introduces an uncertainty about how a manufactured system will perform: although a circuit or chip is designed to run at a nominal clock frequency, the implementation may vary far from this expected performance. Actually, the process variability phenomenon can be modeled as an unknown gain in the equation of the chip, that is $\omega = \big(1-\kappa\big)\cdot\big(\alpha\cdot f_{clk}+\beta\big)$ (see section 2.1.1 for further details). Note that $0 \leq \kappa \leq 1$ by construction: $\kappa = 0$ refers to a chip without any process variability while the chip does not work at all when $\kappa = 1$. Finally, *κ refers to the percentage of process variability*. For this reason, when $\kappa > 0$ the measured system speed is reduced and so are the maximum possible speeds at the different voltage levels (required in the control law). At the end, this is implemented in simulation in order to highlight the process variability robustness of our proposal. Indeed, the simulation results in figure 4.11 show how the system is robust to process variability - for different values of κ - since it is still working regardless the chip performance. This is possible because the proposed control strategy does not need any information on the system. The estimation of the maximum speeds also allows this robustness (note that a constant weighted value is applied here). Of course, in order to compensate a reduced computational capacity induced by the process variability, the system will run a larger amount of time at the penalizing supply voltage. Moreover, the robustness is limited by the maximum possible activity of the device. Thus, if the chip is too bad to compute the task while running at the maximum possible speed (the chip runs with the highest voltage and highest frequency) the controller would not be able to do anything

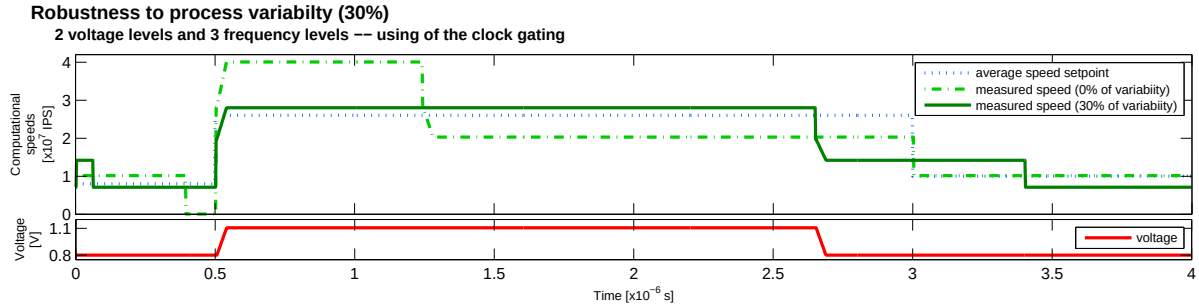
to solve this failure. In this case, the deadlines cannot be ensured anymore and the amount of remaining instructions hence encroaches on the next task. This is the case in figure 4.11(d) for instance. A solution to still use this chip - with high process variability - is to reduce its activity, and this has to be detected by the operating system.



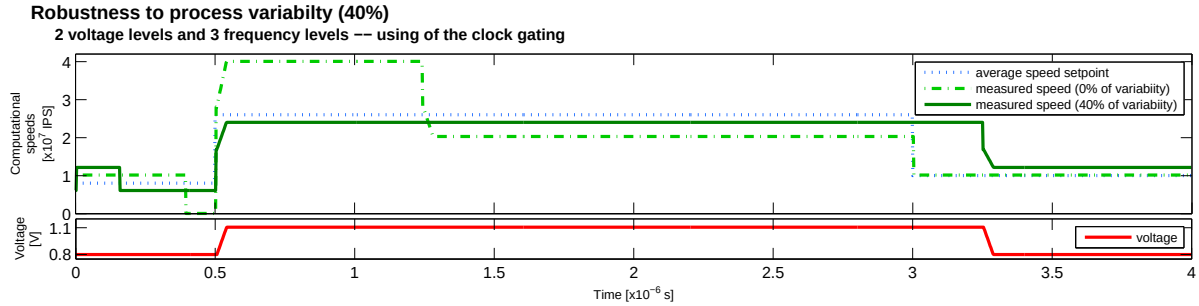
(a) 10 % of process variability



(b) 20 % of process variability



(c) 30 % of process variability



(d) 40 % of process variability

Figure 4.11: Simulation results of the multicore controller based on discrete scheme: robustness to process variability.

4.4.3 Extension to four computational nodes to control together

The monocore discrete control scheme is then extended to the multicore architecture (as presented in subsection 3.2.4). This consists in duplicating the monocore strategy and control the frequency ratios either ***i)*** *f*rom the calculated frequency levels or ***ii)*** *f*rom the predicted speeds. The discrete scheme used for simulations - initially defined in subsection 4.4.1 - is the forth one, where 2 voltage levels and 3 frequency levels are possible while using the clock-gating principle.

First results are shown in figure 4.12 for the case where the ratios are obtained from the calculated frequency levels. In this approach, the frequency ratios can only be 0 or 1 by construction. One can see that the computational load is correctly executed. Two cases are depicted: in figure 4.12(a), the weighted value ν used to estimate the maximum speeds - needed in the control law - is constant ($\nu = 0.1$) but some oscillations occur sometimes. This is because the control decision directly varies with respect to this estimation and we hence proposed a restriction on ν to avoid this problem (see subsection 2.4.4 for further details). However, in figure 4.12(b) the proposed restriction is applied and the weighted value is thus dynamically calculated. Of course, this removes the oscillations and, moreover, decreases the control computational cost (about 10 % of operations less). A stable simulation results with constant weighted value is possible but this requires to decrease the value until $\nu = 0.0001$, which means that the estimation of the maximum speeds varies very slowly. The varying technique is preferred.

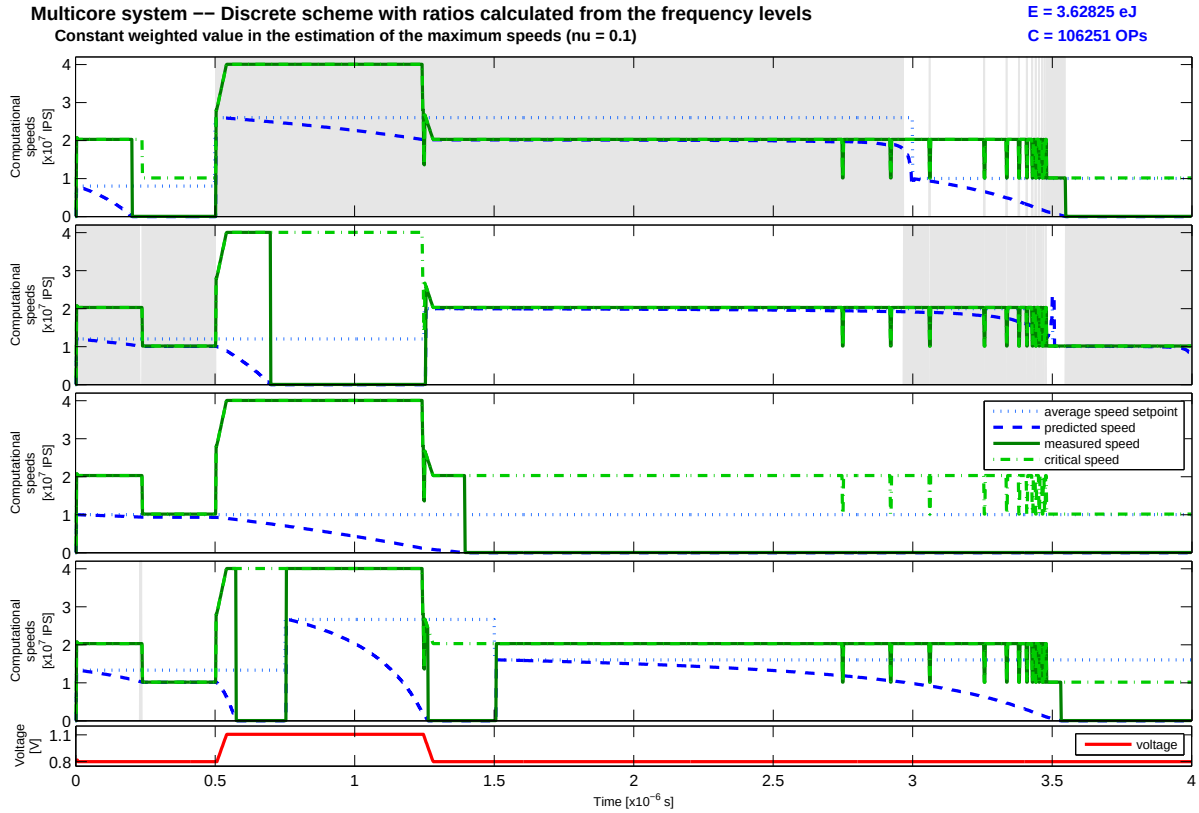
The second approach leads to the simulation results in figure 4.13. In this case, the frequency ratios are obtained from the predicted speeds (which was not really calculated before, as explained in subsection 3.2.4). This inevitably increases the cost of the control law. Two schemes are illustrated, the first one where some continuous frequency ratios are calculated, and a second which applies the on-off principle (previously depicted in subsection 4.3.4). This is represented in figures 4.13(a) and (b) respectively. Finally, comparing both discrete strategies with this on-off mechanism shows that 30 % of operations more are necessary.

Eventually, the discrete scheme based on the frequency levels, where only some small values are available for the frequency level, allows a reduction of about 20 % of the control computational cost (compared to the continuous case based on partial duplication, depicted in subsection 4.3.4) while the energy consumption of the system is almost the same.

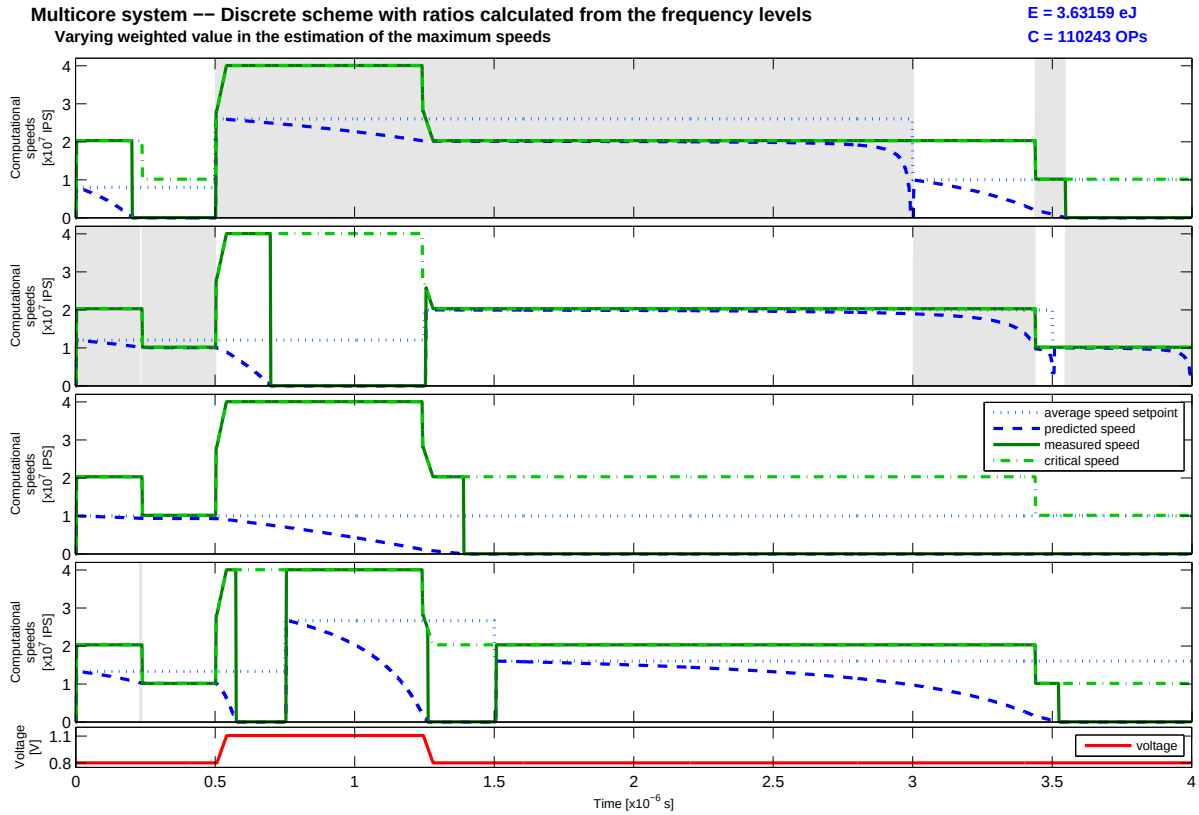
4.5 Performance analysis

In order to evaluate the different algorithms, we summarize the system energy consumption (in equivalent joules eJ) and the control computational cost (in number of operations OPs) obtained in all cases. The monocore scheme is listed in table 4.1 where the minimum values are finally highlighted. By analyzing the results, one could see that a DVFS mechanism allows to reduce the energy consumption but different strategies can be adopted. Thus, controlling the processing power task by task - as this is the case with the first intuitive voltage and frequency control - is not enough and the penalizing voltage has hence to be reduced inside a task. This is done using a predictive control law (into the computational speed strategy) and allows to save 30 % of energy (whatever the feedback loop) compared to a system without DVS mechanism, without increasing the computational cost too much. Then, discretizing the possible frequency levels eventually allows to decrease again the system energy and especially the control cost (more than 50 % of operations less). Note that these results are given for the best discrete strategy implementation whereas different ones exist, as explained just after.

Actually, comparing the control computational cost of some different implementations of the

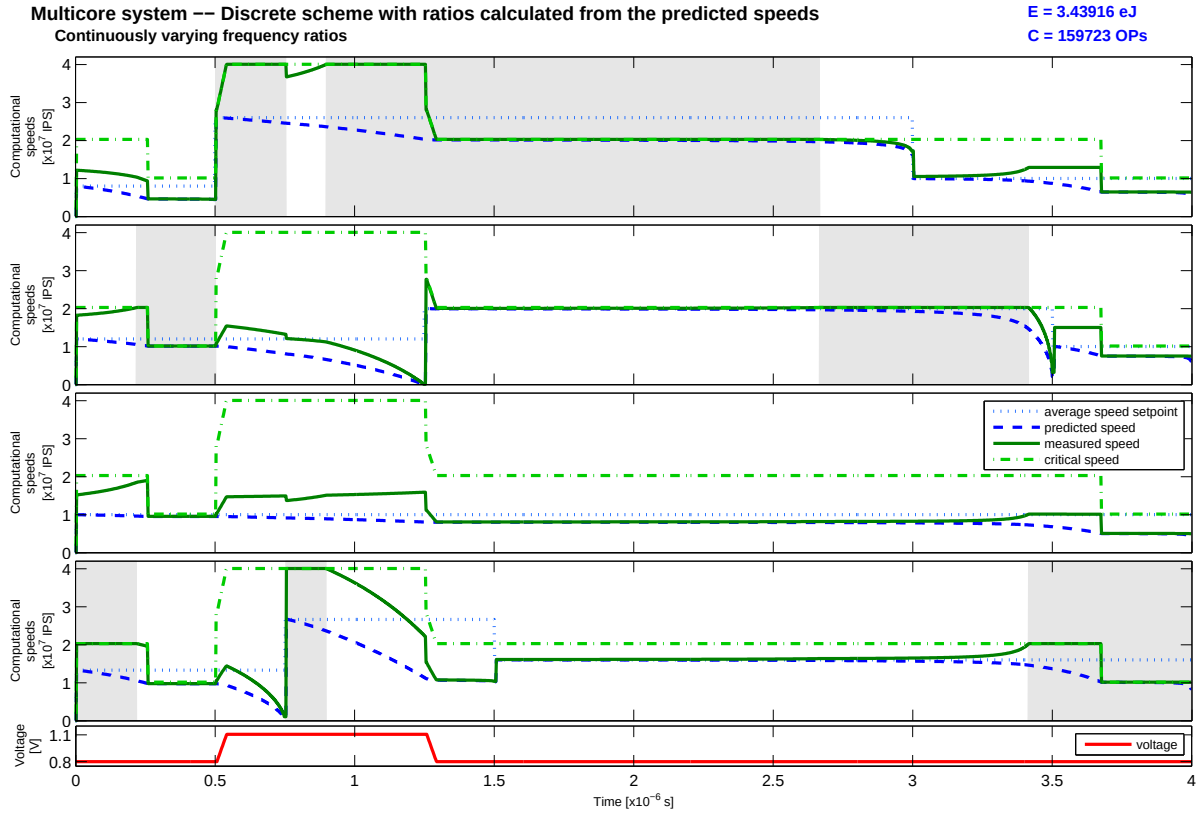


(a) Constant weighted value in the estimation of the maximum speeds ($\nu = 0.01$)

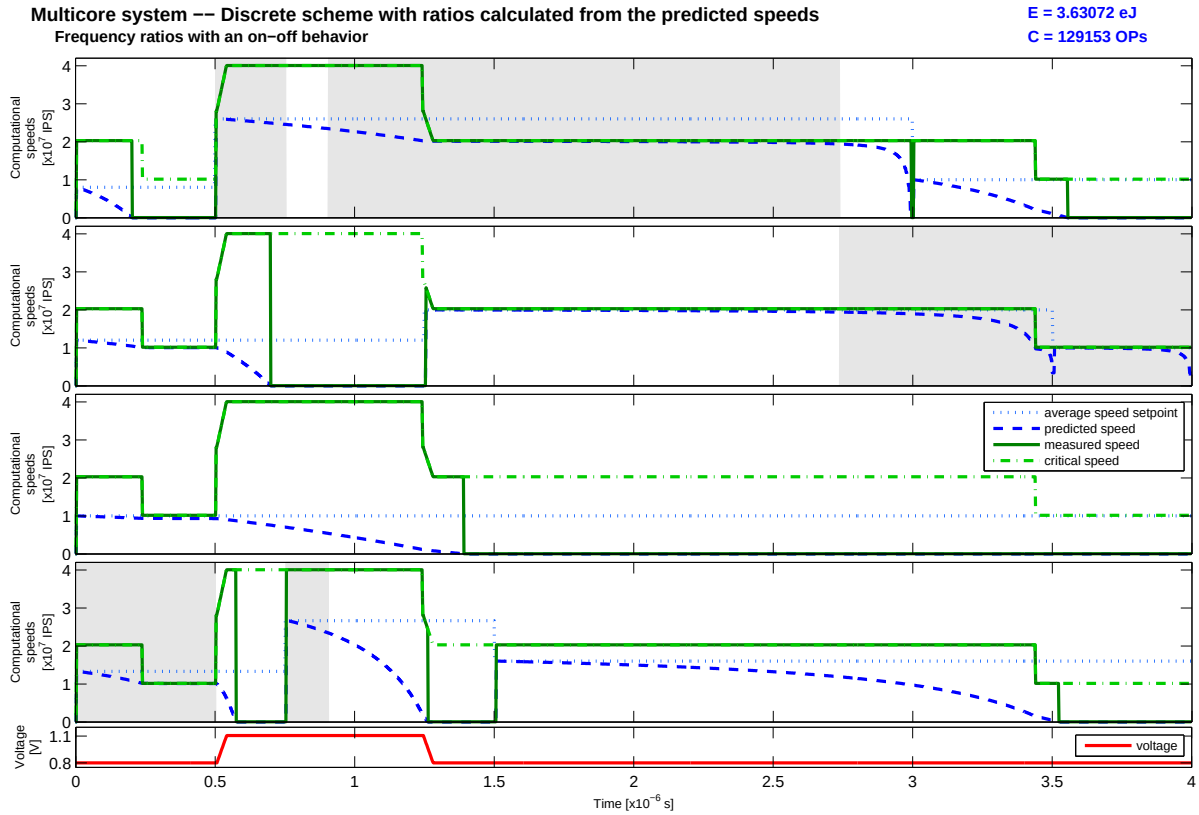


(b) Varying weighted value in the estimation of the maximum speeds

Figure 4.12: Simulation results of the multicore controller based on discrete scheme: frequency ratios calculated from the calculated frequency levels.



(a) Continuously varying frequency ratios



(b) On-off mechanism of the frequency ratios

Figure 4.13: Simulation results of the multicore controller based on discrete scheme: frequency ratios calculated from the predicted speeds.

Table 4.1: Performance analysis: comparison of the different strategies to control the energy-performance tradeoff in a monocore system.

		System energy value	System energy ratios DVFS	System energy ratios DVS	Control cost value
Without DVFS		3.20 eJ	100 %	–	0 OPs
Without DVS		1.59 eJ	49.7 %	100 %	37962 OPs
F.&V. control	Fix1	1.46 eJ	45.7 %	91.9 %	43969 OPs
	Fix2	1.46 eJ	45.7 %	91.9 %	43983 OPs
	Varying	1.46 eJ	45.7 %	91.9 %	46962 OPs
	No restriction	**1.46 eJ**	45.7 %	91.9 %	**40351 OPs**
C.S. control	Setpoint	**1.14 eJ**	35.7 %	71.8 %	57497 OPs
	Measurement	1.14 eJ	35.7 %	71.8 %	**44462 OPs**
Disc. control	2V, 2F	1.32 eJ	41.1 %	82.6 %	15912 OPs
	2V, 2F & cg	1.14 eJ	35.7 %	71.8 %	**16510 OPs**
	2V, 3F	1.15 eJ	36.0 %	72.4 %	16713 OPs
	2V, 3F & cg	1.14 eJ	35.7 %	71.8 %	17049 OPs
	3V, 3F	1.08 eJ	33.7 %	67.7 %	17622 OPs
	3V, 3F & cg	**1.07 eJ**	33.4 %	67.1 %	17959 OPs

Notes:

- *Without DVFS: system without dynamic voltage and frequency scaling mechanism.*
- *Without DVS: system without dynamic voltage scaling mechanism.*
- *F.&V. control: Frequency and voltage level control strategy (see section 2.2). Different restrictions to ensure the maximum delay over the critical path are presented (see subsection 2.2.3), that are either fixing the frequency during the voltage transitions (based on voltage and voltage state measurement), or linearly varying the frequency, or not applying any restriction at all.*
- *C.S. control: Computational speed control strategy (see section 2.3) using either the computational speed setpoint or the speed measurement as a feedback.*
- *Disc. control: Fully discrete control scheme (see section 2.4) with different number of voltage and frequency levels. The strategies can use the clock-gating principle or not.*

fully discrete control scheme is depicted in table 4.2. Thus, the initial proposal - introduced in section 2.4 - consists in calculating an energy-efficient speed setpoint and then deducing the frequency and voltage levels to apply to the actuators. This refers to the constant weighted value before simplification. A first improvement is to not calculate the speed setpoint anymore - which results in not doing a computationally penalizing division - since this is not really required (as explained in section 2.5), and leads to a reduction of about 30 % of the control cost. On the other hand, the dynamical estimation of the maximum speeds - required in the control law - requires some extra operations which considerably increase the cost (10 % more comparing the constant and varying weighted value, both before simplification). Fortunately, another simplification - also introduced in section 2.5 - allows to finally decrease the control cost of 45 % compared with the original proposal.

As regards the multicore case, the results are listed in table 4.3. Firstly, the computational speed control strategy is duplicated in order to drive four electronic devices together. Both proposals consist in ***i)*** repeating the whole monocore strategy as many times as devices to calculate the frequency ratios and deduce the critical task (which is the task to control in

Table 4.2: Performance analysis: comparison of the different control computational cost of the monocore fully discrete strategies regarding the implemented weighted value ν in the estimation of the maximum speeds.

	Constant ν $\nu = 0.1$ (before)	Constant ν $\nu = 0.1$ (after)	Varying ν (before)	Varying ν (after)
2V, 2F	28852 OPs	19879 OPs	31289 OPs	15912 OPs
2V, 2F & c.g.	28440 OPs	19467 OPs	28049 OPs	16510 OPs
2V, 3F	29653 OPs	20680 OPs	32090 OPs	16713 OPs
2V, 3F & c.g.	29849 OPs	20876 OPs	31894 OPs	17049 OPs
3V, 3F	31530 OPs	22557 OPs	36543 OPs	17622 OPs
3V, 3F & c.g.	31732 OPs	22759 OPs	36367 OPs	17959 OPs

Notes:

- *Constant ν (before): Constant weighted value before simplification (see subsection 2.4.4).*
- *Constant ν (after): Constant weighted value after simplification (see subsection 2.4.4).*
- *Varying ν (before): Varying weighted value before simplification (see subsection 2.4.4).*
- *Varying ν (after): Varying weighted value after simplification (see subsection 2.4.4).*

priority) and then, ***i)*** duplicating only a small part of the monocore scheme and calculate the control variables only for the critical task. In both cases, a reduction of about 35 % of the energy consumption is achieved (with a continuously varying frequency ratio behavior). One could verify that the multicore control cost is four times the monocore one plus an extra cost because of the frequency ratios in the full duplication case, while the partial duplication scheme allows to achieve a reduced cost (less than four time the monocore cost). However, the cost is still important since the part which is still duplicated requires a lot of operations. Nevertheless, the multicore discrete schemes yield a high reduction of the control computational cost, especially for the strategy which calculates the ratios from the calculated frequency levels. Note that the energy consumption is increased a little bit in this case because of the only possible on-off mechanism.

4.6 Synthesis

In this chapter, the different schemes previously proposed to control the energy-performance tradeoff in an electronic chip were tested in simulation. Firstly, the different monocore and multicore control strategies developed in chapter 2 and 3 respectively are recalled. Then, the system parameters and the testbeds are introduced. The results are eventually detailed while a performance analysis is also proposed.

- The intuitive frequency and voltage level control strategy is applied to a single processing node. In order to ensure the maximum delay over the critical path, different voltage/frequency restrictions are simulated and the computational speed measurement eventually tracks the average speed setpoint in all cases.

- The computational speed control strategy is then simulated for a monocore and multicore systems. In both schemes the energy consumption is highly reduced (thanks to the predictive control law which allows to minimize the penalizing high voltage running time). Furthermore, the proposal adapts itself to some variations of the task information. Actually, they are provided by the operating system for each task to treat but can be adjusted on line when the OS realizes that the given values were not exact.

Table 4.3: Performance analysis: comparison of the different strategies to control the energy-performance tradeoff in a multicore system.

		System energy value	System energy ratios DVFS	System energy ratios DVS	Control cost value
Without DVFS		12.80 eJ	100 %	–	0 OPs
Without DVS		5.14 eJ	40.1 %	100 %	185420 OPs
Full duplication	Cont.	**3.41 eJ**	26.7 %	66.4 %	223000 OPs
	Disc. (6 values)	3.76 eJ	29.4 %	73.2 %	213001 OPs
	Disc. (4 values)	4.03 eJ	31.5 %	78.4 %	208333 OPs
	On-off	4.73 eJ	37.0 %	92.1 %	**190852 OPs**
Partial duplication	Cont.	**3.42 eJ**	26.7 %	66.4 %	146131 OPs
	Disc. (6 values)	3.46 eJ	27.0 %	67.2 %	139273 OPs
	Disc. (4 values)	3.45 eJ	27.0 %	67.1 %	134729 OPs
	On-off	3.68 eJ	28.8 %	71.5 %	**113545 OPs**
Disc. control (f_{level})	$\nu = 0.0001$	3.63 eJ	28.4 %	70.7 %	105422 OPs
	varying ν	**3.63 eJ**	28.4 %	70.7 %	**92735 OPs**
Disc. control (δ) (varying ν)	Cont.	**3.44 eJ**	26.9 %	66.9 %	159723 OPs
	On-off	3.63 eJ	28.4 %	70.6 %	**129153 OPs**

Notes:

- *Full duplication: Multicore control strategy based on full duplication of the monocore one (see section 3.2).*
- *Partial duplication: Multicore control strategy based on partial duplication of the monocore one (see section 3.2).*
- *Both full and partial duplication strategies either calculate some continuously varying frequency ratios, or discretely varying ones (with 6 or 4 possible values), or apply an on-off mechanism to calculate these ratios.*
- *Disc. control (f_{level}): Fully discrete multicore control scheme with ratios deduced from the frequency levels (see section 3.2), after simplification (see subsection 2.4.4). This strategy can only apply an on-off mechanism to calculate the frequency ratios. Moreover, either a constant or varying weighted value is applied.*
- *Disc. control (δ): Fully discrete multicore control scheme with ratios deduced from the predicted speeds (see section 3.2), after simplification (see subsection 2.4.4). This strategy either calculates some continuously varying or discretely varying (on-off principle) frequency ratios.*
- *Both discrete control strategies use 2 voltage and 3 frequency levels and the clock-gating principle is possible.*

- Finally, the fully discrete control scheme is tested (also in the monocore and multicore cases) with different possible numbers of voltage/frequency levels. This proposal allows to reduce the control computational cost. Moreover, the strategy is strongly robust to process variability because no information is needed in the control law.

To sum up, the simulation results demonstrate that a fast predictive control technique allows to minimize the energy consumption while guaranteeing some computational performance. The different control strategies give an important reduction of the energy consumption in comparison with a system without DVFS or DVS mechanism. Furthermore, the proposals lead to a low control computational cost, more especially for the fully discrete proposals. At the end, the control strategies are highly robust in the case of high dispersion phenomena like the one arising in 45 nm and smaller technologies. This is notably the case within the ARAVIS project (see subsection 1.4.2 for further details).

Part II

Asynchronous control scheme for closed-loop systems

Chapter

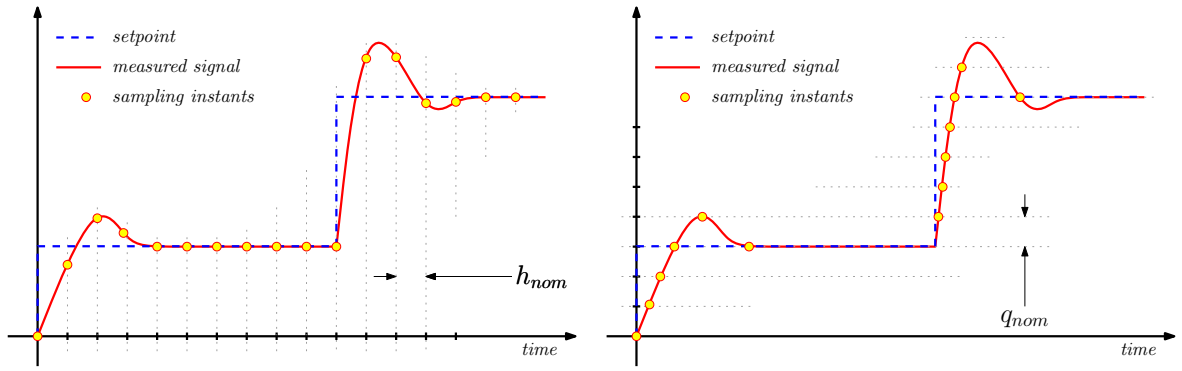
5

Context and motivations

The event-based (or asynchronous) controller, contrary to a classical time-triggered scheme where the control signal is computed at each sampling time, calculates the new control signal only when the measured signal "sufficiently" changes. This behavior seems to be a good solution for control computational cost savings and, for this reason, it is an opportunity for embedded low-power systems with low resources (for instance) where a significant power consumption reduction - by decreasing the samplings and consequently the CPU utilization - would be very appreciated. Many other reasons are motivating the event-triggered systems, in particular because more and more asynchronous systems are encountered. However, this original approach requires to develop new theoretical tools - because nothing really exists in the literature in this sense - and this second part of the thesis contributes on this field. Thus, this chapter starts introducing the classical time-triggered and the event-based sampling approaches in section 5.1. Then, the asynchronous needs existing in the different scientific communities are summarized in section 5.2. Some known problems to provide new theoretical methods are also depicted. One could note that this work was done within the TATIE project context - see subsection 5.2.3 - which aims at giving some architectural solutions for implementing new event-driven algorithms in numerical systems designed in an asynchronous technology. The goal is to demonstrate the superiority of such a framework too. However, the proposals are formulated in such a way that they can be easily transposed to general systems. Incidentally, a highly nonlinear system will be used at the end - in chapter 8 - for some experimental results, in order to show how is adaptable and advantageous the asynchronous scheme.

5.1 Time-based vs. event-based sampling

The classical so-called discrete-time framework of controlled systems consists in sampling the system uniformly in time, with a constant sampling period h_{nom}, and computing and updating the control law every time instants $t_k = k \cdot h_{nom}$, where $k \in \mathbb{N}$. This principle, denoted the time-triggered case (or the synchronous case in sense that all the signal measurements are synchronous), is depicted in figure 5.1(a). This field has been widely investigated [16] even in the case of sampling jitter - a temporal perturbation on the sampling instant - or measure loss that can be seen like some asynchronicities (see [45, 19] for such examples). However, some works addressed more recently event-based sampling where the sampling intervals are event-driven. This is, for instance, when the output changes and crosses a given level $q_j = j \cdot q_{nom}$, where $j \in \mathbb{Z}$. This principle, denoted the event-triggered case (or the asynchronous case in comparison with the first approach), is shown in figure 5.1(b).



(a) Time-triggered behavior

(b) Event-triggered behavior

Figure 5.1: Time-based vs. event-based sampling: the system is uniformly sampled ***i)*** in time with a constant sampling period h_{nom} in the time-triggered case and ***ii)*** in amplitude with a constant detection level q_{nom} in the event-driven case.

Extending the analogy to the calculation of Riemann and Lebesgues integral (the first one summing the heights at each instant whereas the second sums the instants at all heights), the notion of Lebesgues sampling was introduced to denote this sampling scheme: the measurements are taken only when variables cross some specific levels by opposition to the Riemann sampling where the measurements are taken at specific instants. Thus, one could remark the different sampling instants in both cases (highlighted with some bullets in figure 5.1). The idea is to soften the computational load of the controller by reducing the number of samples with the event-based behavior. Indeed, in fact this is not required to calculate (again) the control signal during the steady-state intervals since the system achieved a stable state (else that is not a steady-state interval). On the other hand, it could be interesting to calculate more often the control signal during transients, when the system is moving far from an equilibrium state. Actually, the potential interest of such a control scheme was introduced for the automotive domain in [35, 33].

In the event-triggered sampling scheme, the term *sampling interval* denotes a time interval between two consecutive level crossings of the measure, that is two successive sampling instants. The sampling intervals are hence not equidistant in time anymore, as one can see by analyzing the time intervals between two successive instants in figure 5.1(b). Actually, it varies with respect to time and, for this reason, will be denoted $h(t)$ afterwards.

5.2 Event-driven sampling as an opportunity for embedded systems

Although periodicity simplifies the design and analysis, it results in a conservative usage of resources since the control law is computed and updated at the same rate regardless it is really required or not. As a result, a resource-aware implementation of the control law, using event-based sampling, could be a promising solution for embedded chips. For this reason, the different scientific communities are implicated to develop new event-driven strategies. This is presented in subsection 5.2.1. However, directly extending some results of Riemann sampled systems to Lebesgues sampled systems seems not to be a good strategy since the time will inevitably appear in the new relations. The conversion has hence to be done carefully. The main issues are briefly introduced in subsection 5.2.2. Then, a study case is proposed in subsection 5.2.3 where a French project is depicted. It aims at providing some new architectural solutions for designing numerical systems in an asynchronous technology.

5.2.1 Asynchronous needs in the different communities

Many reasons are motivating the event-triggered systems and in particular because more and more asynchronous systems or systems with asynchronous needs are encountered. Actually, the demand of low-power electronic components in all embedded and miniaturized applications encourages companies to develop asynchronous versions of the existing time-triggered components, where a significant power consumption reduction can be achieved by decreasing the samplings and consequently the CPU utilization: about four times less power than its synchronous counterpart for the 80C51 microcontroller of Philips Semiconductors in [66] for example. Moreover, the absence of synchronization in the asynchronous circuits considerably reduces the noises and the electromagnetic emissions by improving the time repartition of the events [65, 64]. As a result, various publications occur on this subject in the signal processing community (see for example [1] and the references therein).

Note that the sensors and the actuators based on level crossing events also exist, rendering a complete asynchronous control loop now possible. As a result, event-based notion is taking more and more importance in the control community. Indeed, typical event-detection mechanisms are functions on the variation of the state (or at least the output) of the system [10, 58, 57, 56]. Although the event-triggered control is well-motivated and allows to relax the periodicity for computations of the control law, only few works report theoretical results about the stability, convergence and performance of event-triggered control systems. In [11, 12] for instance, it is proved that such an approach reduces the number of sampling instants for the same final performance. It is also shown in [43] that controlling a Lebesgues sampled system or a continuous-time system with quantized measurements and a constant control law over sampling periods are equivalent problems. Recent works deal with the problem of scheduling the control task for continuous-time linear systems [34, 42, 25] and discrete-time linear system [26] where stability and some robustness proprieties are exploited. Furthermore, in [26] an Model Predictive Control schema is used where the event-triggered policies are used for relaxing the computationally demanding algorithms. In [34], a formal analysis makes the tradeoff between a reduction in resource utilization and the control performance for a particular event-driven scheme. In the same idea, an alternative approach consists in enforcing events when required from a stability point of view, that is related to the variation of a Lyapunov function for instance, like in [68]. Convergence and stability in the nonlinear case is studied in [63, 9]. Some important contributions also come from the real-time control community. Indeed, the real-time synchronous control tasks are often considered as hard tasks in term of time synchronization, requiring some strong

real time constraints. The main consequence is oversized computers entailing additional costs not very compatible with a large production as for embedded systems. Therefore, efforts are carried on the co-design between the controller and the task scheduler in order to soften the time constraint due to the synchronous framework. The adopted approach in this field is often either to change dynamically the sampling period related to the load [59, 60] or to use an event-driven control where the events are generated with a mix of level crossings and a maximal sampling period [10, 58]. Eventually, an event-triggered paradigm calls for resources whenever they are indeed necessary. At the end, an alternative asynchronous scheme is the so-called self-triggered control, initially introduced in [69] and then developed in [71, 8, 72, 47]. This latter principle consists in augmenting the system model using the sampling interval as a new state variable of the system. As a result, the controller triggers itself.

5.2.2 Difficulties to untie some well-established paradigms

Most of the asynchronous approaches (introduced in subsection 5.2.1) are implicitly synchronous in the sense that the time is used to determine if the control must be updated. In fact, the synchronous paradigm is so well established that it is difficult to untie that. For instance, the maximal sampling period in [10] was added for stability reasons, in order to fulfill the condition of Nyquist-Shannon sampling theorem. Thus, a new control signal is performed when the amount of time elapsed since the last sample exceeds a certain limit. However, this safety limit condition is no more consistent thanks to the level detection and, intuitively, it can be removed to reduce again the CPU cost. Anyway, this safety limit is uselessly applied in several papers, such as in [57, 47]. Nevertheless, a first approach could be to analyze some event-based control strategies without any safety limit. Then, it could also be interesting to find some theoretical tools to prove that event-driven controllers allow to ensure some good performance, even if the system is not sampled during a long time. Eventually, a fully asynchronous control framework is expected since that would be adequate for many systems. For instance, biological systems are most of the time reacting to events, even if biological clocks are existing. A new framework is required for such systems. Several researches are thus developed in this way. This is for instance the case of the TATIE project which is next presented.

5.2.3 Study case: The TATIE project

The TATIE project (French acronym used for "asynchronous technology and non-uniform signal processing) proposes to overcome the asynchronous needs (depicted in subsection 5.2.1). Above all, it comes from the mix between different partners of different knowledge domains:

The CIS[1]group , expert in integrated system design. The main research topics are to study and develop some new methods for analog and digital complex systems. The asynchronous technology is hence an essential point of research to design some robust and flexible integrated systems with a low energy consumption and low electromagnetic emissions. A solution to build "intelligent sensors" and "smart objects" is to control the whole conception of analog/digital systems using a non-uniform sampling scheme. In this context, the CIS research group develops from few years some analog-to-digital converters [6, 5, 2] which are not based on the classical toolchains used in signal processing and/or control theory. The adopted strategy consists in using an event-based sampling, driven by the measured signal itself, instead of the conventional time-triggered case which does not take into account what kind of information is present into the signal. Thus, with the proposals, the signal is not only represented as some samples taken at a given predefined time anymore, but the signal dynamics now allows to detect when an event occurs and so is taken

the corresponding temporal information. Therefore, all the signal processing chain (signal conditioning, periodical sampling and synchronous numerical processing) is contradicted. This non-uniform sampling is completely adapted to the asynchronous systems because they are both controlled by events whose time instants are a priori unknown.

The LJK[2]laboratory , expert in numerical analysis. A non-uniformly sampling study refers to some mathematical problems which are not treated in the classical numerical signal processing (because the synchronicity leads to a lot of simplifications). A partner with no bearing on the historic of such synchronous analysis was hence important in order to not make these reductions. Moreover, a non-uniform sampling is quite usual in applicative mathematics, when discretizing some differential equations or doing some interpolations and numerical resolution for instance [18]. A collaboration between the CIS group and the LJK laboratory already conducts to some asynchronous filters algorithms and besides, they are implemented into the SPASS (signal processing for asynchronous systems) Matlab toolbox [17].

The NeCS[3]project-team , expert in control theory. This team has a large experience in developing dynamical control loops for complex systems with high constraints. One of the main objectives is to develop some control theories for embedded and wireless systems, more especially low-power and distributed (sensor network) systems. The conception of such controlled systems is done integrating some constraints on communication, computational and energy limited resources. NeCS team is researching for innovating solutions for these systems, using non-uniform sampling [22, 43, 23] or differential modulations [37, 40, 31].

Eventually, the TATIE project aims at ***i)*** developing some new filtering methods for numerical systems designed in an asynchronous technology, ***ii)*** developing some new algorithms and determining some architectural solutions to implement them and ***iii)*** studying the optimal observability problem for non-uniformly sampled signals, that is an optimal Kalman filter. The results will be used to demonstrate the superiority of the asynchronous scheme - both in term of energy consumption and electromagnetic emissions - where a gain of once or twice is expected. The proposed strategies could be integrated in several embedded applications, such as personal digital assistant devices, mobile phones, autonomous sensors, etc... Nevertheless, the development of new techniques for analog-to-digital conversion based on a non-uniform sampling leads to completely redesign the numerical analysis toolchains. This asynchronous paradigm also leads to redefine the methods usually adopted in signal processing and discrete control techniques. In this way, the TATIE project proposes to develop a theoretical framework for numerical signal processing in order to be able to build some asynchronous embedded systems. Regarding particularly the control theory aspect, some results on level-crossing detection techniques are expected with some new theoretical tools. These aspects will be analyzed in chapter 6 and 7 respectively. Furthermore, the advantages on using an asynchronous scheme have to be highlighted in order to convince the industrials, more especially in term of energy consumption and computational needs. This is why some experimental results will be eventually depicted in chapter 8.

[1]The CIS (concurrent integrated systems) group comes from TIMA (laboratory in techniques of informatics and microelectronics for integrated systems architecture: `http://tima.imag.fr/`

[2]Laboratoire Jean Kuntzmann: `http://ljk.imag.fr/`

[3]The NeCS (networked controlled systems) project-team is bi-located at INRIA (the French national institute for research in computer science and control: `http://www.inria.fr/`) and GIPSA-lab (laboratory of Grenoble in Image, Parole, Signal processing and Control: `http://www.gipsa-lab.inpg.fr/`)

Chapter

6

Event-based PID controllers using level-crossing detection

The proportional integral derivative (PID) controller is a generic architecture widely used in industrial control systems since it is easily tunable. Thus, a large number of applications is available and we hence initially choose this controller to show the advantages on using an event-triggered strategy. We base our approach on the theory widely existing for the classical time-triggered case, and also on the simple event-based *PID* controller introduced by *Karl-Erik Årzén* in [10]. Different event-driven strategies are developed in order to reduce the computational cost of the controller - by reducing the number of samples - while still guaranteeing some good performance of the controlled systems.

This chapter starts bringing back the conventional time-based control principle in section 6.1. Then, the event-triggered architectures are detailed in section 6.2. In a first time, we focus our study more particularly on the integral part of the controller and, for this reason, only some event-based proportional integral controllers are detailed. The original Årzén's event-based PI controller is introduced and, based on this work, we then present new strategies. We eventually propose an extension to event-triggered PID controllers. Then, a recap of the different proposals is done in section 6.3 and stability and robustness analysis is performed in section 6.4. Some criteria are given in section 6.5 in order to compare the performance of the different control strategies and some simulation results are finally presented in section 6.6, with a very simple first-order system and a more complex cruise control mechanism. At the end, section 6.7 synthesizes the whole chapter.

6.1 The conventional time-based approach

The classical time-triggered control architecture - running in discrete-time - is represented in figure 6.1. An input signal $e(t)$ is firstly discretized in time by an analog-to-digital (AD) converter. It becomes $e(t_k)$. Then, the control logic calculates the control signal $u(t_k)$ and a digital-to-analog (DA) converter finally provides the output signal $u(t)$ to the system to control.

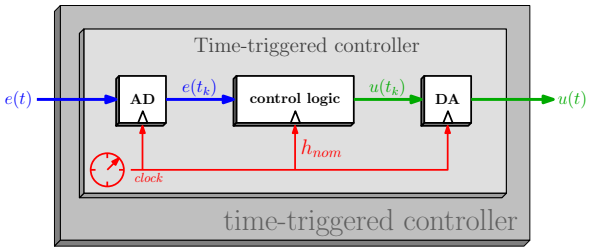


Figure 6.1: Architecture of the time-based controller.

Several strategies exist and could be used to control a system. Here, **we propose to use a proportional integral derivative control**.

6.1.1 Time-based PID control

The textbook PID controller equation (in frequency domain) is given as follows

$$U(s) = K_p \cdot \left(E(s) + \frac{1}{T_i \cdot s} \cdot E(s) + T_d \cdot s \cdot E(s) \right)$$

where s is the *Laplace's variable*, $U(s)$ is the control signal, $E(s)$ is the error between an expected setpoint $Y_{sp}(s)$ and a measured signal $Y(s)$ and K_p, T_i and T_d are some tunable control parameters. Actually, this equation can be divided into a proportional, an integral and a derivative part, denoted $U_p(s)$, $U_i(s)$ and $U_d(s)$ respectively. The proportional part allows to determine the reaction to the current error, the integral one is used to determine the reaction based on the sum of recent errors and the derivative one determines the reaction based on the rate at which the error has been changing. Finally, the weighted sum of these three actions is used to dynamically adjust the system output in term of responsiveness, overshoots (of a given reference) and oscillations. This textbook equation can then be modified in order to improve the performance of the controller, as proposed in [14]. The most classical approach is to add a low-pass filter in the derivative term in order to avoid problems with high frequency measurement noise:

$$\begin{aligned} U_p(s) &= K_p \cdot E(s) \\ U_i(s) &= \frac{K_p}{T_i \cdot s} \cdot E(s) \\ U_d(s) &= \frac{K_p \cdot T_d \cdot s}{1 + T_d \cdot s/N} \cdot E(s) \end{aligned}$$

where N is the low-pass filter gain.

A discrete-time PID controller is finally obtained discretizing the previous equations. A common way - and more especially the one used in the Årzén's algorithm (detailed in subsection 6.2.1) on which are based our proposals - is to straightforward the proportional part while using the forward and backward difference approximation for the integral part and the derivative

part respectively. The discrete-time proportional part $u_p(t_k)$ is easily obtained replacing the continuous variables with their sampled versions. On the other hand, the backward approximation to calculate $u_d(t_k)$ requires the previous values of the derivative part and the measured signal, i.e. $u_d(t_{k-1})$ and $y(t_{k-1})$ respectively. As regards the integral part, $u_i(t_{k+1})$ is pre-calculated at time t_k. Finally, the resulting discrete time-triggered PID equations are:

$$\begin{aligned} e(t_k) &= y_{sp}(t_k) - y(t_k) \\ u_p(t_k) &= K_p \cdot e(t_k) \\ u_d(t_k) &= \frac{T_d}{T_d + N \cdot h_{nom}} \cdot u_d(t_{k-1}) + \frac{K_p \cdot T_d \cdot N}{T_d + N \cdot h_{nom}} \cdot \Big(e(t_k) - e(t_{k-1})\Big) \\ u(t_k) &= u_p(t_k) + u_i(t_k) + u_d(t_k) \\ u_i(t_{k+1}) &= u_i(t_k) + K_i \cdot h_{nom} \cdot e(t_k) \end{aligned} \tag{6.1}$$

where t_k is the instant time of the current sample, t_{k-1} is that of the previous sample and t_{k+1} is that of the next one. One could note that in this time-triggered case $t_k = t_{k-1} + h_{nom}$, $t_{k+1} = t_k + h_{nom}$ and so on, with a *constant sampling period* h_{nom}. This is because the system is uniformly sampled in time in the classical time-triggered scheme. We also define $K_i = {}^{K_p}/_{T_i}$ to simplify further utilizations.

6.1.2 Time-based PI control

As already explained in introduction of this chapter, in the next section we will only deal - in a first time - with some event-based PI controllers. Therefore, the derivative part will not be taken into account anymore to control the system. In this case, the control signal of the PID algorithm - given in equation (6.1) - becomes $u(t_k) = u_p(t_k) + u_i(t_k)$ and the controller is called a PI controller. Eventually, the resulting time-based PI algorithm is described in figure 6.2.

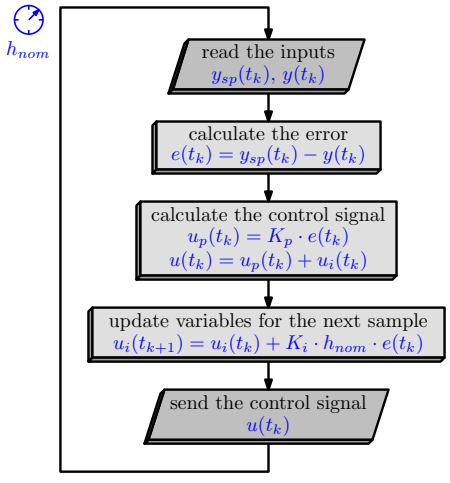


Figure 6.2: Algorithm: the time-triggered PI controller.

6.2 New event-based strategies

Karl-Erik Årzén was the first to propose an event-based PID controller in [10] and we base our analysis on that. The basic setup - the control strategy will be more detailed in subsection 6.2.1 - consists in two parts: ***i)*** *a time-triggered event detector* used for the level-crossing detection of the input signal $e(t)$ and ***ii)*** *an event-triggered controller* which calculates

the control signal $u(t)$. This architecture is drawn in figure 6.3. The first part runs with the constant sampling period h_{nom} - that is the same than for the corresponding conventional time-triggered controller - whereas the second part is driven by some requests sent by the event detector. These requests are provided when a new control signal has to be calculated and the length of the *varying sampling intervals* $h(\cdot)$ for the control part is the amount of time between two successive requests.

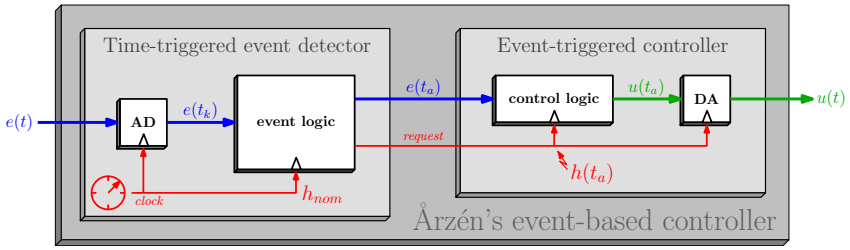


Figure 6.3: Architecture of the event-based controller proposed by Årzén.

Let t_a denote the beginning time of the current control sample, that is the last time a request was sent by the event detector because the input signal crossed a level $q_j = j \cdot q_{nom}$. Respectively, let t_{a+1} denote the next sampling time and so on. The sampling intervals $h(\cdot)$ are then function of these instants. Let $h(t_a)$ denote the sampling interval used to calculate the current control signal, that is the amount of time between the current sample and the last one, i.e. $h(t_a) = t_a - t_{a-1}$. Respectively, let $h(t_{a+1})$ denote the next sampling interval, i.e. $h(t_{a+1}) = t_{a+1} - t_a$, and so on. The different notations are represented in figure 6.4.

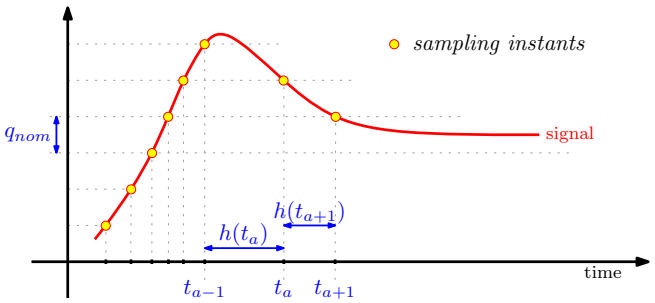


Figure 6.4: Event-driven control scheme: representation of the time instants and the sampling intervals.

In fact, before each sampling instants the value of the current instant time t_a increases while a event does not occur. In other words, $t_a = t_k$ is dynamically updated until a new control signal is computed and so is achieved the final value of t_a. As a result, $h(t_a)$ is a multiple of h_{nom}, by construction, since $t_k = k \cdot h_{nom}$ and $h(t_a) = t_a - t_{a-1}$.

The input signal involved in the event detection mechanism can be of different types. Different strategies are depicted in [57] for instance. The solution used in the original Årzén's event-based controller is introduced in subsection 6.2.1 and several proposals are then developed in the next subsections. Finally, an extension for PID controllers is presented in subsection 6.2.6.

6.2.1 Årzén's event-based PI control

The Årzén's event-based controller aims at tracking a given setpoint $y_{sp}(\cdot)$. For this reason, the level-crossing detection is based on the error between the reference and the measurement,

i.e. $e(t_k) = y_{sp}(t_k) - y(t_k)$. Eventually, the Årzén's setup updates the control signal either ***i)*** when the relative measurement crosses a given level, that is when the absolute value of the difference between the measured error of the last sampling and that of the current instant time crosses the limit q_{nom}, i.e. $abs\big(e(t_a) - e(t_{a-1})\big) > q_{nom}$, or ***ii)*** if the maximal sampling period is achieved, i.e. $h(t_a) \geq h_{max}$. This second condition is added in order to ensure the stability by fulfilling the Nyquist-Shannon sampling condition. Finally, the algorithm of the resulting Årzén's event-based PI controller is shown in figure 6.5.

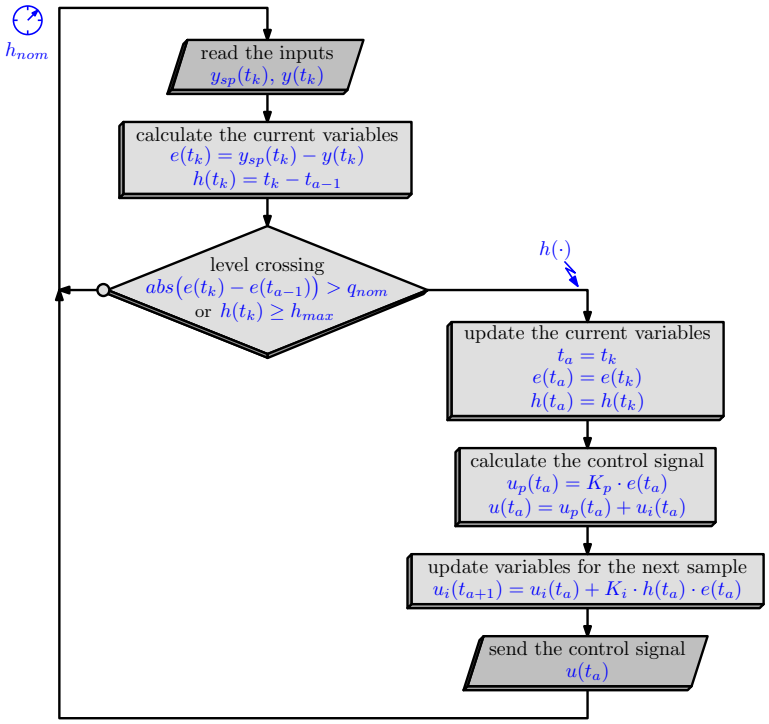


Figure 6.5: Algorithm: the Årzén's event-based PI controller.

6.2.2 Discretization improvement for the Arzen's PI control

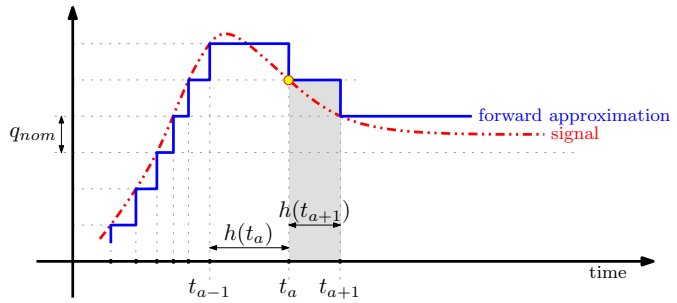
Before really developing some new event-based strategies, **we propose to improve the initial Årzén's proposal** (just detailed above), **more particularly regarding the discretization of the integral part**. Indeed, the forward difference approximation was originally applied. This means that the integral part is pre-calculated at the current time for the next sample, i.e. $u_i(t_{a+1}) = u_i(t_a) + K_i \cdot h(t_a) \cdot e(t_a)$. However, one could note a misunderstanding in this equation, looking more precisely the building of the discretization. In fact the forward difference approximation needs the current value of the signal $e(t_a)$ and the next sampling interval $h(t_{a+1})$ - and not the current one $h(t_a)$ as in the Årzén's mix up - in order to calculate the next integral part. One can see that on the example of figure 6.6(a). As a result, the forward equation is

$$u_i(t_{a+1}) = u_i(t_a) + K_i \cdot h(t_{a+1}) \cdot e(t_a) \tag{6.2}$$

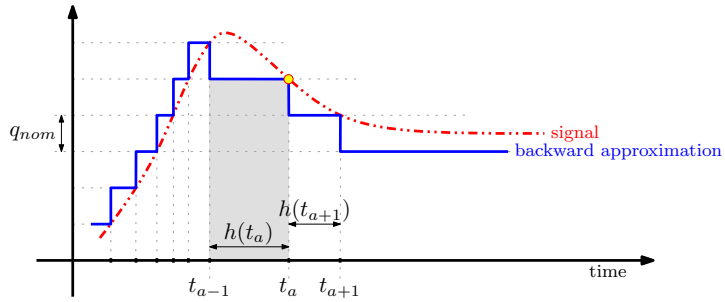
On the first hand, the forward method is a good choice for time-triggered controllers because the sampling intervals h_{nom} are constant. On the other one, the sampling intervals vary for event-triggered controllers and, consequently, the next value $h(t_{a+1})$ is not *a priori* known. Nevertheless, if one still wants to use the forward approximation, a solution could be

to post-calculate the current integral part by shifting the instant times in the equation, i.e. $u_i(t_a) = u_i(t_{a-1}) + K_i \cdot h(t_a) \cdot e(t_{a-1})$. Calculating the integral part with a more recent value of the error seems to be a better solution anyway. As a result, **we propose to apply the backward difference approximation**. This leads to calculate the current integral part during the current time t_a with the current sampling period $h(t_a)$ and the current error $e(t_a)$, as shown in figure 6.6(b). Finally the backward equation for the integral part becomes

$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot h(t_a) \cdot e(t_a) \tag{6.3}$$



(a) Forward difference approximation



(b) Backward difference approximation

Figure 6.6: Discretization: comparison between the forward and the backward difference approximations.

The resulting event-based PI controller algorithm is represented in figure 6.7.

In the following subsections, **we propose to base our work on the improved algorithm**. This event-based PI controller is then improved by changing either the event conditions (modifying the way to send a request in order to calculate a new control signal) or the control equations (adapting the conventional control strategy to the event-driven framework).

6.2.3 Event-based PI control without safety limit condition

In this subsection, **we propose to remove the safety limit condition $h(t_a) \geq h_{max}$ introduced by *Årzén* for stability reason** (as explained in subsection 6.2.1) in order to improve and simplify the event-based setup. In this case, a new control signal is computed only when the relative measurement crosses the detection level. This intuitive simplification yields the length of the sampling intervals to become very large if the measured signal does not change for a large amount of time, i.e. as large as the steady-state interval is. In other words, each time the setpoint changes after a long steady-state interval the controller will correct the system output too much - because of the huge sampling interval used to calculate the integral part - with the effect that important overshoots will occur.

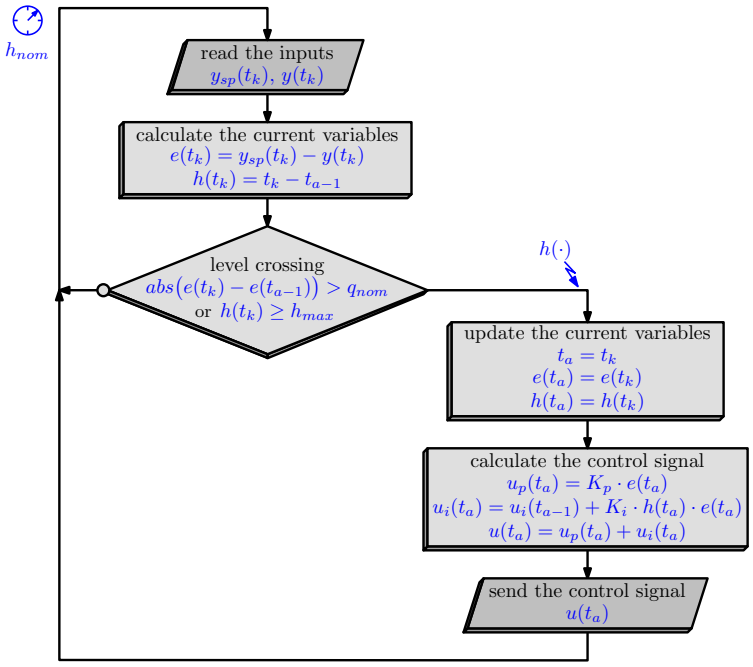


Figure 6.7: Algorithm: the Årzén's event-based PI controller with improved discretization.

Actually, the integral part of the event-triggered PI controller, i.e. $u_i(t_a) = u_i(t_{a-1}) + K_i \cdot h(t_a) \cdot e(t_a)$ from equation (6.3), is responsible of this problem. This is because the value of the sampling interval $h(\cdot)$ becomes huge due to the absence of event. Moreover, the steady-state interval often terminates when the setpoint changes, that is when the error $e(\cdot)$ drastically becomes high. As a result, the calculation of the integral part needs the product $h(\cdot)e(\cdot)$, afterwards called the *integral gain* $he(\cdot)$, and yet, this product explodes at the end of the steady-state interval. This results in over-controlling the next transient. In order to avoid this issue, the steady-state intervals are now analyzed in more detail. In fact, they can be divided into two parts, as drawn in figure 6.8, that are ***i)*** the time interval where the signal is really in a steady state and ***ii)*** the time interval required to detect a new level crossing. The first part starts the last time a control signal was computed, that is at the sampling instant t_{a-1}, and finishes just before the setpoint changing. Then, while the error becomes higher than the level detection q_{nom}, the second part starts. A request is sent and a new control signal is finally calculated at time t_a. However, the exact time of the level crossing could be between two successive time-triggered sampling instants t_{k-1} and t_k (because the event detector is time driven). The worst case occurs when the setpoint changes at time $t_{k-1} + \varepsilon$, where ε is very small. In this case, the request is only sent at the second sampling instant t_k and, eventually, the measured error $e(t_a)$ was large during almost all the sampling period h_{nom}. The exact time could not be known but the integral gain $he(\cdot)$ can be upper-bounded anyway during the detection time of a transient, by the product $h_{nom} \cdot e(t_a)$. Consequently, the first part runs only from the instant time t_{a-1} to $t_a - h_{nom}$, which is equal to the time interval $h(t_a) - h_{nom}$. During this period, the error remains very small - the error is lower than the detection level q_{nom} else the steady state is not achieved - and so is the product $he(\cdot)$. Thus, the integral gain is also upper-bounded here by the product $\big(h(t_a) - h_{nom}\big) \cdot q_{nom}$. To summarize, a steady-state interval can be divided into ***i)*** a first part where the sampling interval increases a lot but the error remains small and ***ii)*** a second part where the error becomes very large but only during a few instant. Therefore, the product $he(\cdot)$ does not explode since $h(\cdot)$ and $e(\cdot)$ compensate themselves each other. In fact, it

was over-estimated in previous works and **we propose to include a more precise value of the integral gain in our proposals**. The integral part now becomes

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he(t_a) \\ \text{where} \quad & he(t_a) \leq \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a) \end{aligned} \tag{6.4}$$

One could note that this inequality, which was initially built for the steady-state intervals, is finally correct for the whole running. Indeed, equation (6.4) becomes equation (6.3) during a transient, since $h(t_a) = h_{nom}$ in this case. Based on this assumption, **we propose several algorithms without safety limit condition**. They are then detailed.

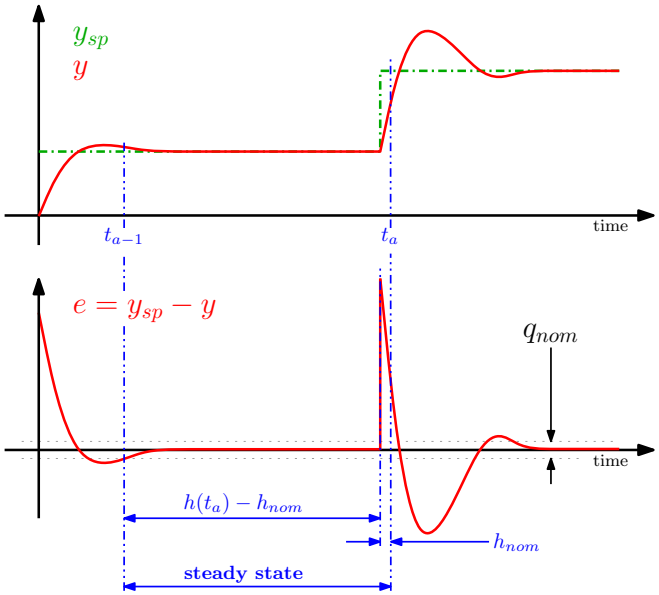


Figure 6.8: Event-driven control scheme: decomposition of a steady-state interval.

Regarding the level-crossing mechanism, Årzén suggests to update the control signal when the error changes enough from its last value (see subsection 6.2.1 for further details). However, this has to be modified for the new proposals and, therefore, **we propose to change the event detection mechanism, replacing the relative measurement by the absolute one**. A new control signal is calculated as soon as the current measured error crosses the detection level, i.e. $abs\big(e(t_a)\big) > q_{nom}$. With this method, the number of samples inevitably increases during the transients but, at least, the error between the system and the setpoint is sure to be lower than q_{nom} during the steady-state intervals. This was not the case before. In fact, this is required for event-based controllers without safety limit condition in order to ensure that a steady state is really achieved before deciding to not update the control signal anymore while the setpoint does not change, or a perturbation does not occur. As a result, the measured signal has to track the expected setpoint during the steady-state intervals, which means a small error (and not a small relative error because the relative error could be small, i.e. $abs\big(e(t_a) - e(t_{a-1})\big) > q_{nom}$, while the system is far from the setpoint).

Six different strategies are finally proposed: the first one where nothing else is done than removing the safety limit condition, and the other one where the integral part is modified in order to reduce its impact after a long steady-state interval. The proposed approaches are somehow similar to the anti-windup mechanism used in control theory, where the error induced by the saturation has to be compensated. Finally, the procedure to implement such a controller

is (almost) the same for all the different proposals. The resulting algorithm is represented in figure 6.9, where the integral gain $he(\cdot)$ depends on the chosen strategy (depicted just below).

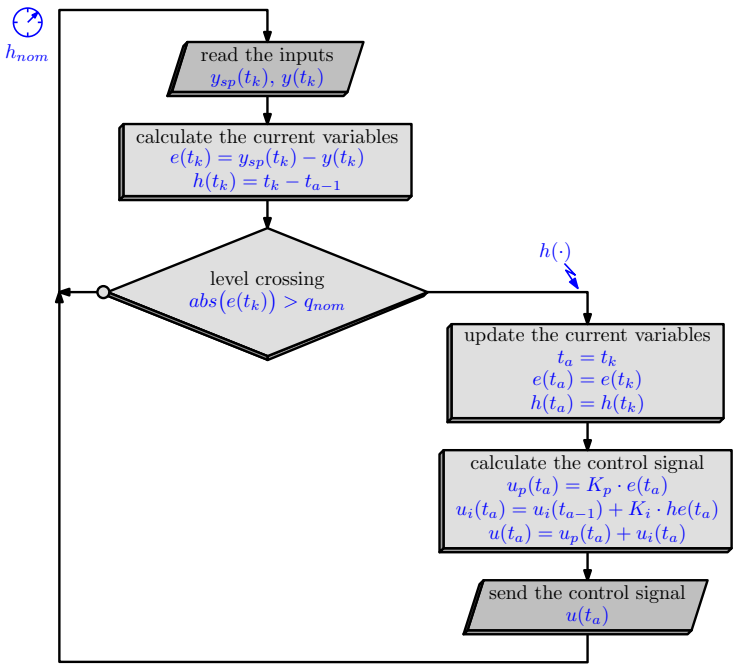


Figure 6.9: Algorithm: the event-based PI controllers without safety limit condition.

6.2.3.1 Algorithm 1: only without safety limit condition

This algorithm corresponds to the Årzén's one where the safety limit condition $h(t_a) \geq h_{max}$ is removed without doing anything else. For this first algorithm without safety limit condition, the integral part remains the one expressed by equation (6.3), that is

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he(t_a) \\ \text{where} \quad he(t_a) &= h(t_a) \cdot e(t_a) \end{aligned} \tag{6.5}$$

As previously explained, important overshoots are expected after a large steady-state interval.

6.2.3.2 Algorithm 2: saturation of the integral gain

This second algorithm consists in reducing the product $he(\cdot)$ after a long steady-state interval, in order to reduce the overshoots. Thus, the integral gain is bounded according the principle depicted in introduction. Note that the integral part of this algorithm comes from equation (6.4). This yields

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he_{sat}(t_a) \\ \text{where} \quad he_{sat}(t_a) &= \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a) \end{aligned} \tag{6.6}$$

6.2.3.3 Algorithm 3: exponential forgetting factor of the sampling interval

Another method consists in reducing the impact of the sampling interval which largely increases over the steady-state time. Thus, a forgetting factor of the sampling interval is added. An

exponential function is chosen to decrease its impact as the elapsed steady-state time increases, that is

$$h_{exp}(t_a) = h(t_a) \cdot exp\Big(\alpha \cdot \big(h_{nom} - h(t_a)\big)\Big) \tag{6.7}$$

where the parameter α allows a degree of freedom to increase or decrease the exponential sampling interval. This yields an *exponential sampling interval* $h_{exp}(\cdot)$ in the integral part of the controller, as follows

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he_{exp}(t_a) \\ \text{where} \quad & he_{exp}(t_a) = h_{exp}(t_a) \cdot e(t_a) \end{aligned} \tag{6.8}$$

This function leads to have ***i)*** a nominal sampling interval during the transients - when $h(t_a) = h_{nom}$ the exponential sampling interval is $h_{exp}(t_a) = h_{nom}$ - and ***ii)*** an exponential decreasing sampling interval during the steady-state intervals.

6.2.3.4 Algorithm 4: hybrid strategy

This algorithm is finally a mix between the saturation of the integral gain and the exponential forgetting factor of the sampling interval. In fact, in the first algorithm the product $he(\cdot)$ increases with respect to the sampling interval $h(\cdot)$ and the measured error $e(\cdot)$, as drawn in figure 6.10(a). The second algorithm minimizes the impact of the product $he(\cdot)$ on the integral term - bounding the integral gain - but it still increases with respect to both $h(\cdot)$ and $e(\cdot)$, as shown in figure 6.10(b). Finally, the third algorithm adds an exponential forgetting factor of the sampling interval in such a way that the integral gain decreases. However, the product $he(\cdot)$ is higher when the sampling interval is small, as represented in figure 6.10(c). The idea here is to have a small impact of the sampling interval all the time. For this reason, the exponential forgetting factor - given in equation (6.7) - is used within the algorithm with saturation. Thus, the product $he(\cdot)$ is now upper-bounded by $\big(h_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$ and the integral part becomes

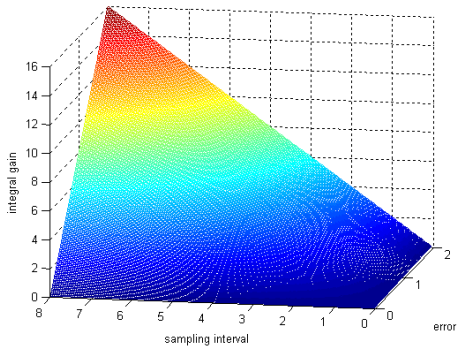
$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he_{hybrid}(t_a) \\ \text{where} \quad & he_{hybrid}(t_a) = \big(h_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a) \end{aligned} \tag{6.9}$$

The evolution of the integral gain of this hybrid algorithm is plotted in figure 6.10(d).

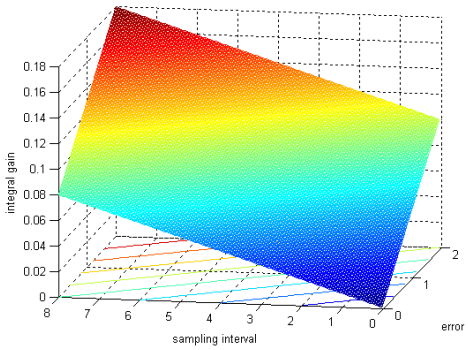
The previous algorithms reduce the impact of the integral gain in the integral part of the controller. Nevertheless, the exponential function $h_{exp}(t_a)$ used in algorithms 3 and 4 could be a problem when implementing it on a low resource system. A look-up table with pre-calculated values of the function can easily replace the online calculation and, therefore, highly reduce the computational cost. Otherwise, **we propose some algorithms with low-cost implementation**, that are algorithms 5 and 6. They are really close to the original ones in term of responsiveness, but yield in a less costly solution (replacing the exponential function by a simple linear piecewise defined function).

6.2.3.5 Algorithm 5: exponential forgetting factor of the sampling interval with low-cost implementation

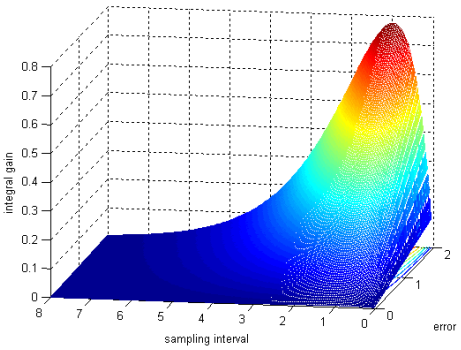
The exponential forgetting factor $h_{exp}(t_a)$ of the sampling interval represented in figure 6.10(c) firstly increases before decreasing, and finally is quasi-null. Based on that, we define the *low-cost*



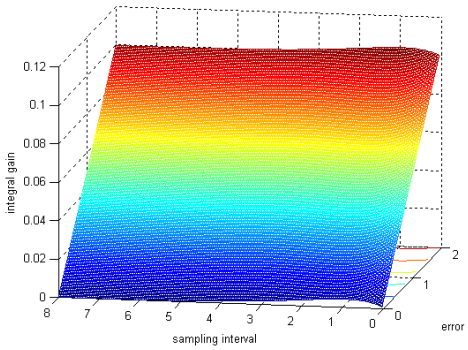
(a) Algorithm 1: he



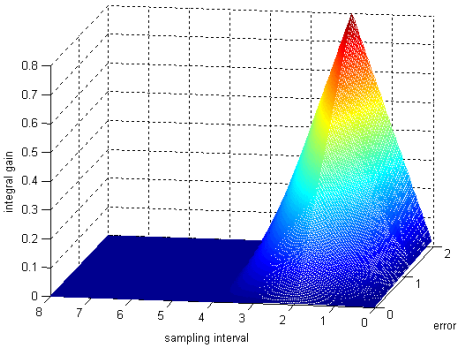
(b) Algorithm 2: he_{sat}



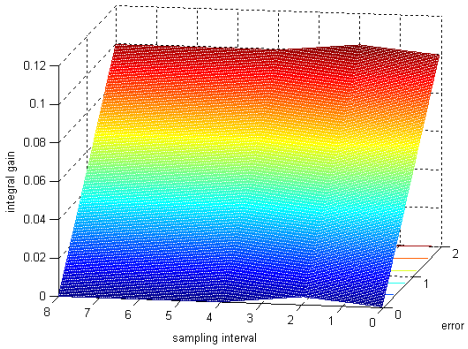
(c) Algorithm 3: he_{exp}



(d) Algorithm 4: he_{hybrid}



(e) Algorithm 5: he^i_{exp}



(f) Algorithm 6: he^i_{hybrid}

Figure 6.10: Event-driven control without safety limit condition: dynamics of the integral gain ($h_{nom} = 0.05s$, $q_{nom} = 0.01$, $\alpha = 1$ and $h^i_{max} = 40 \cdot h_{nom}$).

exponential sampling interval $h^i_{exp}(\cdot)$. It is depicted such as

$$h^i_{exp}(t_a) = \begin{cases} \dfrac{h(t_a)}{5 \cdot \alpha} & \text{if} \quad h(t_a) \leq h^i_{max} \\ \dfrac{2 \cdot h^i_{max} - h(t_a)}{5 \cdot \alpha} & \text{if} \quad h^i_{max} < h(t_a) \leq 2 \cdot h^i_{max} \\ 0 & \text{otherwise} \end{cases} \tag{6.10}$$

where h^i_{max} is chosen to be as close as possible of the original exponential function. Then the low-cost function replaces the original exponential function in the integral part, which yields

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he^i_{exp}(t_a) \\ \text{where} \quad he^i_{exp}(t_a) &= h^i_{exp}(t_a) \cdot e(t_a) \end{aligned} \tag{6.11}$$

At the end, the evolution of the integral gain in figure 6.10(e) is as close as possible to the original one in figure 6.10(c).

6.2.3.6 Algorithm 6: hybrid strategy with low-cost implementation

This last strategy is based on the original hybrid algorithm using the low-cost exponential forgetting factor given in equation (6.10), which leads

$$\begin{aligned} u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he^i_{hybrid}(t_a) \\ \text{where} \quad he^i_{hybrid}(t_a) &= \left(h^i_{exp}(t_a) - h_{nom}\right) \cdot q_{nom} + h_{nom} \cdot e(t_a) \end{aligned} \tag{6.12}$$

The integral gain of this low-cost hybrid algorithm is depicted in figure 6.10(f).

6.2.4 Event-based PI control with minimum sampling condition

An improvement could be done on the time-triggered event detector introduced by Årzén - presented in subsection 6.2.1 and depicted in figure 6.3 - which is currently a discrete-time system. Indeed, an event could only be detected at the time instants $t = k \cdot h_{nom}$ and, as a result, several levels could miss if they appear between two sampling instants. For this reason, **we propose to use a continuous-time event detector instead**. In fact, this is closer than the real case since a sensor based on level crossing will send a request as soon as a level is achieved. This new architecture - afterwards denoting the *asynchronous event-based controller* - is represented in figure 6.11.

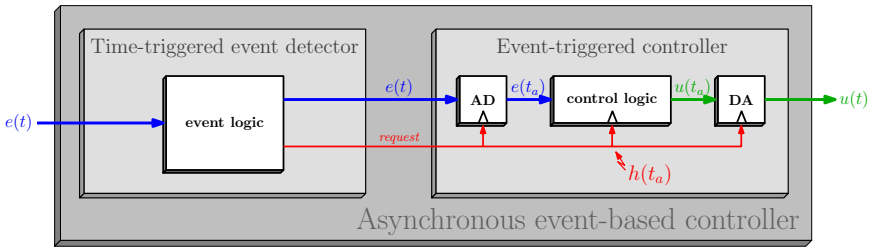


Figure 6.11: Architecture of the asynchronous event-based controller.

The event-based algorithms without safety limit condition - previously detailed in subsection 6.2.3 - remain available with this architecture. However, whereas the number of samples was important during transients (because a new control signal is computed when the measured

error is higher than the detection limit), this problem is amplified with the asynchronous framework since requests are now (quasi)-continuously sent during the whole transient, that is when $abs\big(e(t_a)\big) > q_{nom}$. To avoid that, **we propose to add a minimum sampling interval condition** to lighten the transients: a new control signal is performed only if a given amount of time was elapsed since the last sample, i.e. $h(t_a) \geq h_{min}$. This condition is shown in figure 6.12.

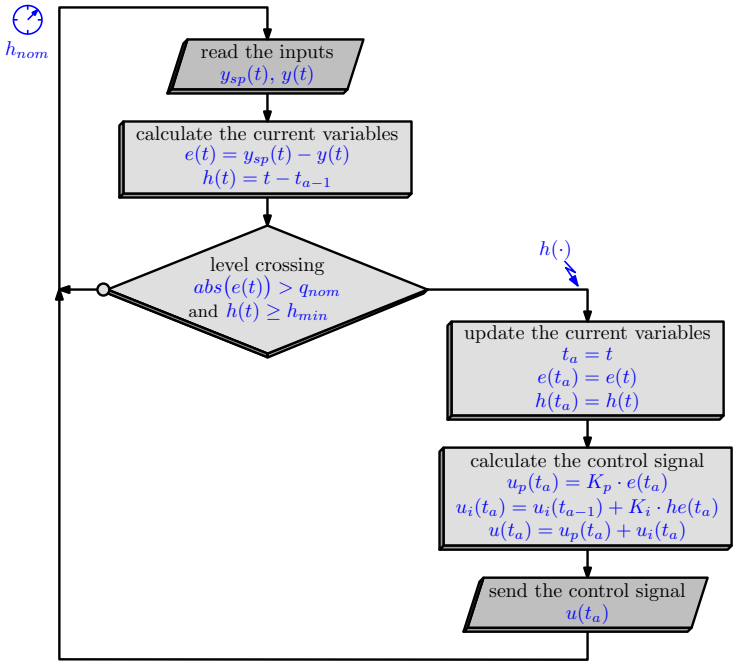


Figure 6.12: Algorithm: the event-based PI controllers with minimum sampling condition.

The minimum sampling interval could be chosen as the discrete sampling period h_{nom} (corresponding to the conventional time-triggered controller) or not, but it does have to satisfy the Nyquist-Shannon sampling condition. Finally, the choice $h_{min} = h_{nom}$ leads to ***i)*** a discrete-time event detector when the dynamics is important and to ***ii)*** a continuous-time event detector when the dynamics is slow (quasi-steady state). Thus, when an event occurs after a steady-state configuration, a new control signal is instantaneously computed. Whatever that may be the h_{min} value, an important reduction of the computational cost would be achieved.

6.2.5 Event-based PI control with extra samples

The event-based scheme can be improved again, adding a few number of samples after a transient. The idea here is to decrease even more the error during the steady-state intervals. Currently, one could ensure that the error is lower than the limit q_{nom} but cannot know how much lower. Moreover, one could not know if the measured signal is going closer or moving away from the setpoint. Therefore, **we propose to add some extra samples after a transient** (while an event-based controller would do not do anything because the condition $abs\big(e(t_a)\big) > q_{nom}$ is wrong). Thus, an extra event is sent to the controller if nothing appends after the last time a control signal was calculated plus a given sampling interval h_{extra}. Then, this is repeated while the error is higher than a minimum level q_{min}. At the end, one only needs to define the expected margin of error and some extra samples will be added to achieve that. Note that the lower q_{min} is chosen the higher the number of extra samples will be. The principle is represented in figure 6.13.

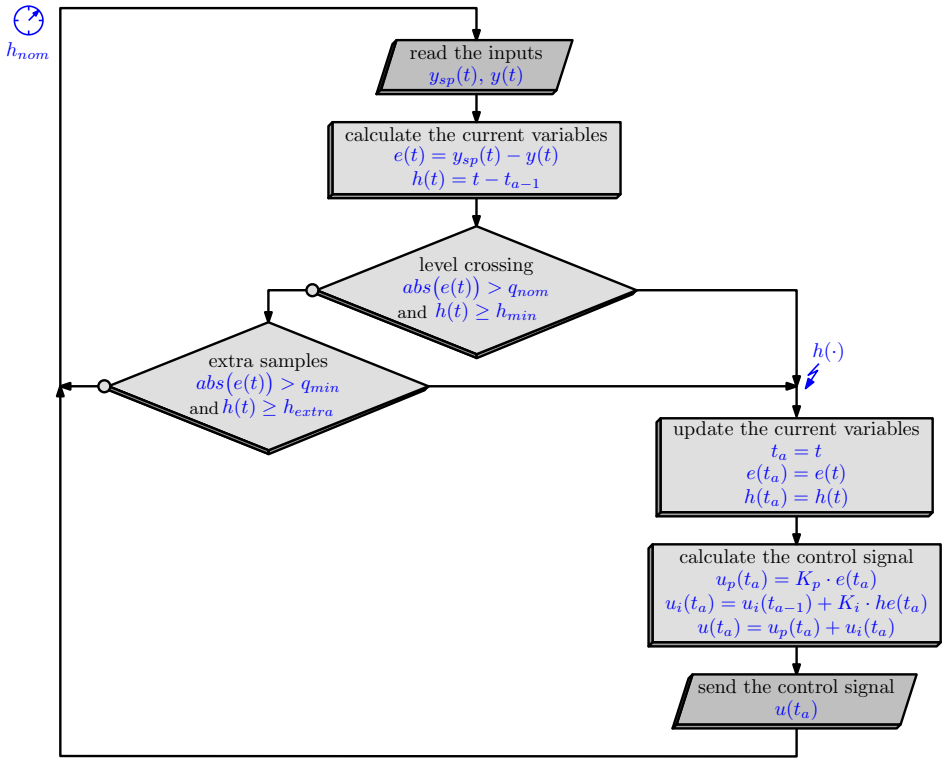


Figure 6.13: Algorithm: the event-based PI controllers with extra samples.

6.2.6 Extension to event-based PID controllers

In the current section, we previously detailed several event-based PI controllers. More especially, we modified the conventional integral part of the Årzén's controller. Nevertheless, **we propose to apply the proposals to a more general proportional integral derivative control strategy**. Indeed, only the integral part is impacted by the modifications and the derivative part remains the same than for the time-triggered case (presented in section 6.1). The discrete derivative part comes from the equation (6.1), that is:

$$u_d(t_a) = \frac{T_d}{T_d + N \cdot h(t_a)} \cdot u_d(t_{a-1}) + \frac{K_p \cdot T_d \cdot N}{T_d + N \cdot h(t_a)} \cdot \big(e(t_a) - e(t_{a-1})\big)$$

and the control signal eventually becomes the one depicted in figure 6.14.

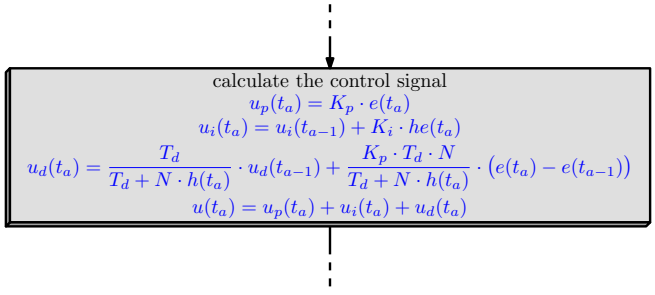


Figure 6.14: Algorithm: extension to event-based PID controllers.

6.3 Recap of the different level-crossing strategies

This section aims at summarizing the different asynchronous control strategies proposed above. The basic setup - depicted in introduction of section 6.2 - remains the same. Only the integral part expression and the level-crossing detection condition are different. In the following, **we propose to recap the different strategies based on level crossing**. One could then refer to this list to test the different strategies. This will be notably useful for the simulation/experimental results (in section 6.6 and chapter 8 respectively).

Time-based PI control strategy

The time-triggered controller is sampled at every periodic discrete-time instants $t_k = k \cdot h_{nom}$ and its integral part is

$$u_i(t_{k+1}) = u_i(t_k) + K_i \cdot h_{nom} \cdot e(t_k) \quad \text{(Forward approximation)}$$
$$u_i(t_k) = u_i(t_{k-1}) + K_i \cdot h_{nom} \cdot e(t_k) \quad \text{(Backward approximation)}$$

Årzén's event-based PI control strategy

The integral part of the Årzén's controller is an extension of the time-based case, which becomes

$$u_i(t_{a+1}) = u_i(t_a) + K_i \cdot h(t_{a+1}) \cdot e(t_a) \quad \text{(Initial)}$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot h(t_a) \cdot e(t_a) \quad \text{(Improved, Absolute)}$$

This controller is event-triggered with some not-equidistant sampling periods, such that $h(t_a) = t_a - t_{a-1}$ or $h(t_{a+1}) = t_{a+1} - t_a$, where t_{a-1}, t_a and t_{a+1} are three consecutive sampling instants. The proposed sampling mechanism enforces some events when

$$abs\big(e(t_a) - e(t_{a-1})\big) > q_{nom} \quad \text{or} \quad h(t_a) \geq h_{max} \quad \text{(Initial, Improved)}$$
$$abs\big(e(t_a)\big) > q_{nom} \quad \text{or} \quad h(t_a) \geq h_{max} \quad \text{(Absolute)}$$

Note that the initial strategy is the one presented in [10], the improved strategy consists in using the backward approximation instead of the forward one, and the absolute strategy uses the absolute error in the event detection condition instead of the initial relative error.

Event-based PI control strategy without safety limit condition

Several controllers without safety limit condition were proposed (six algorithms):

- ***Algo 1:*** *a*lgorithm only without safety limit condition,
- ***Algo 2:*** *s*aturation of the integral gain,
- ***Algo 3:*** *e*xponential forgetting factor of the sampling interval,
- ***Algo 4:*** *h*ybrid strategy,
- ***Algo 5:*** *e*xponential forgetting factor of the sampling interval with low-cost implementation,
- ***Algo 6:*** *h*ybrid strategy with low-cost implementation,

The integral part $he(\cdot)$ is different for each algorithm, that is

$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he(t_a) \qquad \text{(Algo 1)}$$
$$\text{where} \quad he(t_a) = h(t_a) \cdot e(t_a)$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he_{sat}(t_a) \qquad \text{(Algo 2)}$$
$$\text{where} \quad he_{sat}(t_a) = \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he_{exp}(t_a) \qquad \text{(Algo 3)}$$
$$\text{where} \quad he_{exp}(t_a) = h_{exp}(t_a) \cdot e(t_a)$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he_{hybrid}(t_a) \qquad \text{(Algo 4)}$$
$$\text{where} \quad he_{hybrid}(t_a) = \big(h_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he^i_{exp}(t_a) \qquad \text{(Algo 5)}$$
$$\text{where} \quad he^i_{exp}(t_a) = h^i_{exp}(t_a) \cdot e(t_a)$$
$$u_i(t_a) = u_i(t_{a-1}) + K_i \cdot he^i_{hybrid}(t_a) \qquad \text{(Algo 6)}$$
$$\text{where} \quad he^i_{hybrid}(t_a) = \big(h^i_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$$

Note that the exponential function $h_{exp}(t_a)$ used in algorithm 3 and 4 is defined such as

$$h_{exp}(t_a) = h(t_a) \cdot exp\Big(\alpha \cdot \big(h_{nom} - h(t_a)\big)\Big)$$

whereas its low-cost implementation equivalent is

$$h^i_{exp}(t_a) = \begin{cases} \dfrac{h(t_a)}{5 \cdot \alpha} & \text{if} \quad h(t_a) \leq h^i_{max} \\ \dfrac{2 \cdot h^i_{max} - h(t_a)}{5 \cdot \alpha} & \text{if} \quad h^i_{max} < h(t_a) \leq 2 \cdot h^i_{max} \\ 0 & \text{otherwise} \end{cases}$$

Finally, the event detection condition is the same for all the algorithms. This is

$$abs\big(e(t_a)\big) > q_{nom}$$

which is in fact the same than the absolute Årzén's controller condition (but without the safety limit condition $h(t_a) \geq h_{max}$).

Event-based PI control strategy with minimum sampling condition

The integral part of the controllers with minimum sampling condition is the same than the controllers without safety limit condition (six algorithms). Only the event detection condition changes. Thus, a minimum sampling condition $h(t_a) \geq h_{min}$ is added, which eventually yields

$$abs\big(e(t_a)\big) > q_{nom} \quad \text{and} \quad h(t_a) \geq h_{min}$$

Event-based PI control strategy with extra samples

The integral part of the controllers with extra samples is also the same than the controllers without safety limit condition (six algorithms). The event detection condition is the last one plus an extra decision, which consists in enforcing a new event - after a given amount of time h_{extra} - as long as $abs\big(e(t_a)\big) > q_{min}$. In this case some extra events are hence generated. The final condition is

$$\Big(abs\big(e(t_a)\big) > q_{nom} \quad \text{and} \quad h(t_a) \geq h_{min}\Big) \quad \text{or} \quad \Big(abs\big(e(t_a)\big) > q_{min} \quad \text{and} \quad h(t_a) \geq h_{extra}\Big)$$

6.4 Intuitive stability and robustness analysis

The notions of stability is important in control theory but, as explained in subsection 5.2.2, it is difficult to prove for asynchronous controlled systems using a level-crossing mechanism (since non zero states cannot be distinguished from zero). Removing the safety limit condition was intuitively done. We simply assume that the condition of Nyquist-Shannon sampling theorem is no more consistent thanks to the level detection. Note that we then develop some theoretical tools to prove that event-driven controllers allow to decrease the computational cost even if the system is not sampled during a long amount of time. This is done in chapter 7.

As regards the robustness, we assume that the system will track a given reference, even in case of perturbation, because as soon as the error increases (or decreases) an event will be enforced and so is updated the control signal. The same thing is verified for an error in modeling the system to control: an event occurs when the error between the measured signal and the setpoint to track is higher than the detection level q_{nom}.

6.5 Indexes of performance

Several criteria could be used to compare the performance of each controller. An intuitive one is the number of samples required to control the system during the whole simulation time. This index - afterwards denoted *calls* - is interesting to show the gain on control computational needs using an event-based technique. However, some other indexes can be useful, like those providing information on the quality of the system response (see for instance [57]). Eventually, **we propose to use three performance indexes to analyze the performance of our proposals**, that are

- The integral absolute error:

$$IAE = \int_0^\infty \left|e(t)\right| dt$$

  This index shows how far is the system response compared with a given setpoint. The smallest value will highlight the strategy which best fits the system with the reference.

- The integrated absolute difference between the system response of the time-based strategy and that of the event-based ones:

$$IAEP = \int_0^\infty \left|y_{time-based}(t) - y_{event-based}(t)\right| dt$$

  This index allows to compare the system response of the event-based controllers with the time-triggered one.

- The integral absolute difference between the IAE of the time-based strategy and the IAE of the event-based ones:

$$IAD = \int_0^\infty \left|IAE_{time-based}(t) - IAE_{event-based}(t)\right| dt$$

  This index calculates the error between the system response and the setpoint for each strategy, in order to finally compare them for both the time-triggered and the event-based ones.

6.6 Simulation results

In this section, **we propose to present some simulation results** - done with *Matlab/Simulink* - in order to highlight the advantages of an event-based approach (detailed in section 6.2) compared to the classical time-based strategy (called back in section 6.1). In subsection 6.6.1, a simple first-order system is controlled with some event-based PI devices, while a more complex cruise control system is used in subsection 6.6.2 for the PID case. A performance analysis is eventually performed in both cases (using the criteria previously enumerated in section 6.5).

6.6.1 Application to a first-order system

A first-order system can be described as follows

$$H(s) = \frac{G}{1 + \tau \cdot s}$$

where $G = 1$ and $\tau = 1$ in this very simple case. This system will be controlled with different controllers: firstly with the conventional time-triggered PI controller, then with the Årzén's event-based PI controller and finally with our proposals, that are the event-based PI controllers without safety limit condition, the event-based PI controllers with minimum sampling condition and the event-based PI controller with extra samples.

The parameter's values of these controllers are obtained by pole placement of the closed-loop system in the time-triggered case. The event-based controllers are then designed with the same values, with a view to be as close as possible of the time-triggered closed-loop shaping. At the end, $K_p = 1.83$, $T_i = 0.457$ and the nominal sampling interval is $h_{nom} = 0.05\,s$. The system is simulated for $20\,s$ and the test bench consists in two steps: the setpoint is changed from 0 to 1 at time $1\,s$ and changed again at time $10\,s$ to achieve an amplitude of 2.

Conventional strategy

The simulation results for the conventional time-triggered PI controller - introduced in subsection 6.1.2 - are represented in figure 6.15. The top plot shows the setpoint and the measured signal whereas the bottom plot shows the sampling intervals. As one can see, the value is constant in the classical case, that is corresponding to h_{nom}. Eventually, the total number of samples is also indicated in the figure.

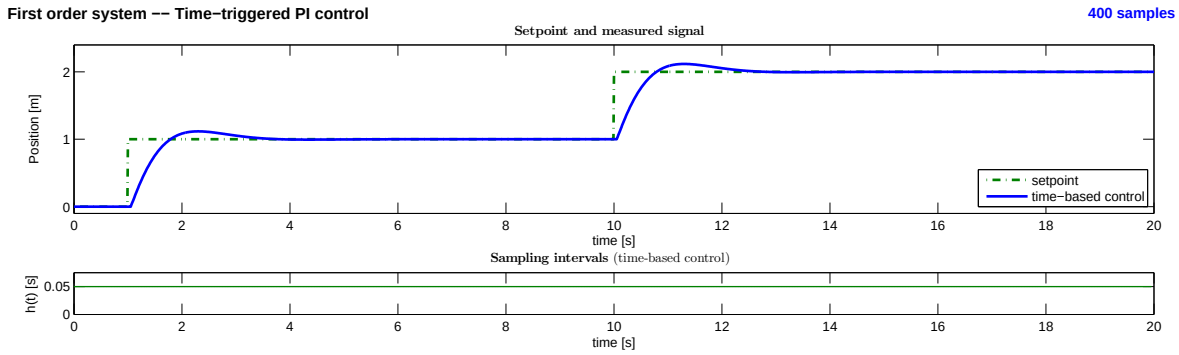
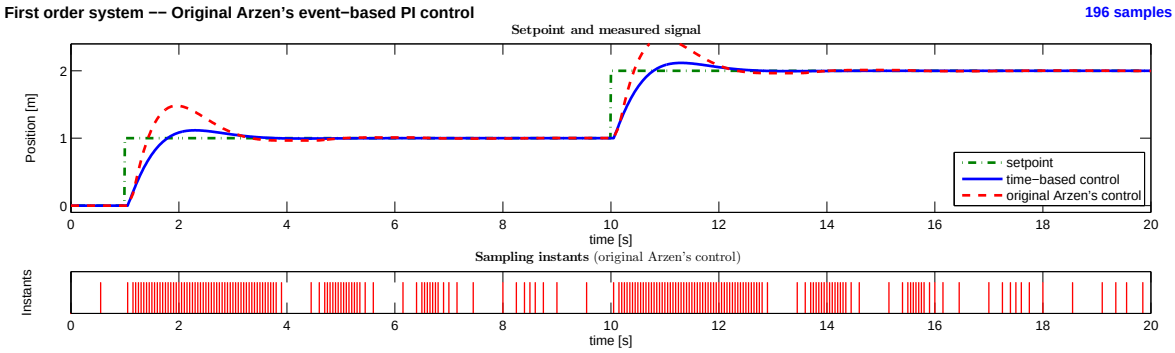


Figure 6.15: Simulation results: the conventional time-triggered PI controller.

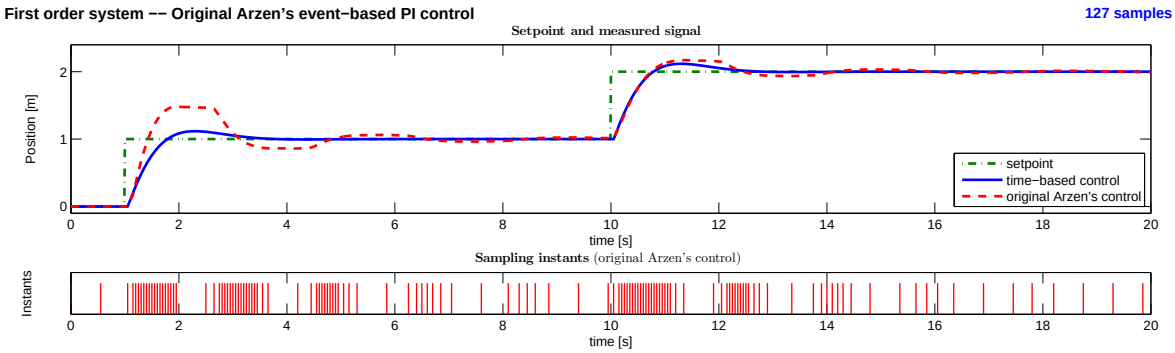
Then, several simulation results follow in order to compare the event-based PI control approaches to this classical time-based strategy.

Original Årzén's strategy

The event-based controller proposed by Årzén - and depicted in subsection 6.2.1 - consists in calculating a new control signal either when the relative measured error crosses a given level, i.e. $abs\big(e(t_a) - e(t_{a-1})\big) > q_{nom}$, or when the sampling interval becomes larger than the maximal safety limit, i.e. $h(t_a) \geq h_{max}$. This principle is simulated with $h_{max} = 10 \cdot h_{nom} = 0.5\,s$, for different values of the event detection level. Thus, figure 6.16(a) shows that the Årzén's controller allows to obtain a system response as quick as the time-triggered one by calculating a control signal twice less when $q_{nom} = 0.001$. Note that the bottom plot now refers to the sampling instants: an event is drawn each time the control signal is updated. This representation will be preferred in the following results. Furthermore, whereas the event detection level is increased, the results become deteriorated and the system oscillates, as represented in figure 6.16(b) when $q_{nom} = 0.01$. This issue is due to the discretization choice.



(a) Event detection with $q_{nom} = 0.001$



(b) Event detection with $q_{nom} = 0.01$

Figure 6.16: Simulation results: the time-triggered PI controller vs. the Årzén's one.

Discretization improvement

The previous oscillations come from the discretization of the integral part. Actually, a misunderstanding was done in the Årzén's algorithm (see subsection 6.2.2 for further details). However, a solution consists in using the backward difference approximation instead of the forward one. Thus, figure 6.17 compares this improvement with the original Årzén's event-based PI controller. The gain is immediate since the results obtained with $q_{nom} = 0.01$ are better than the original Årzén's controller when $q_{nom} = 0.001$. Moreover, whereas some perturbations could appear before, the steady-state intervals are now only triggered by a periodic sampling period due to the safety limit condition $h(t_a) \geq h_{max}$, that is the expected behavior.

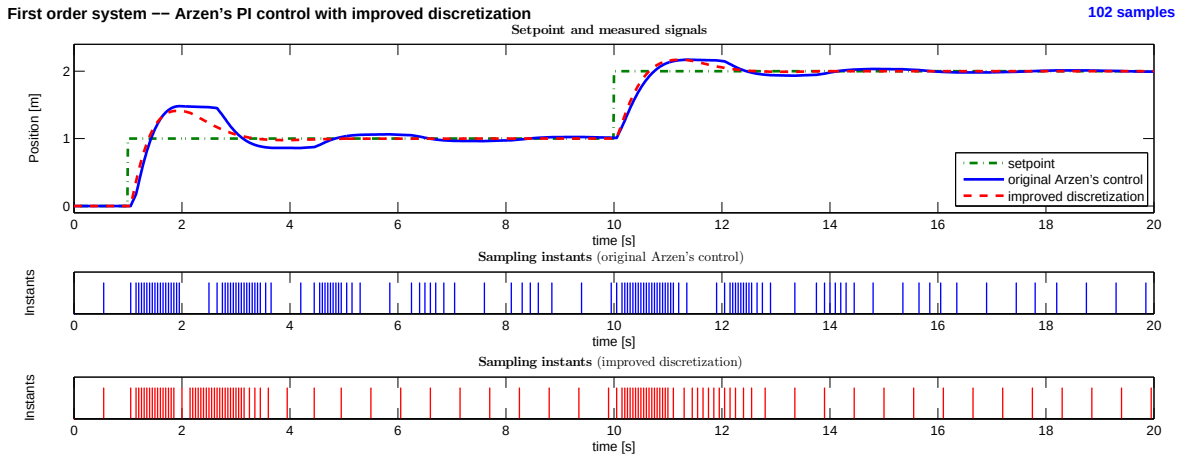


Figure 6.17: Simulation results: the original Årzén's PI controller vs. the Årzén's controller with improved discretization (with $q_{nom} = 0.01$).

Afterwards, the event-based PI architectures are based on the backward difference approximation - denoted the improved Årzén's PI controller - and the simulations are performed with a level detection equal to $q_{nom} = 0.01$.

Changing the level-crossing detection mechanism

Before removing the safety limit condition (in order to avoid periodic sampling during the steady-state intervals, such as one can see in figure 6.17 during the simulation time 4 and 10 s), we change the level-crossing detection mechanism, replacing the relative measured error by the absolute one. A new control signal is now calculated as soon as the absolute error crosses the detection level, i.e. $abs\big(e(t_a)\big) > q_{nom}$, or when the maximal sampling interval is achieved, i.e. $h(t_a) \geq h_{max}$. This inevitably increases the number of samples during the transients. Nevertheless, the measured error is now sure to be very small - lower than q_{nom} - during the steady-state intervals. The simulation results in figure 6.18 show that the system responses are quite similar with both level-crossing detection techniques. The only difference is for the final number of samples which increases as expected (about 40 % of samples more).

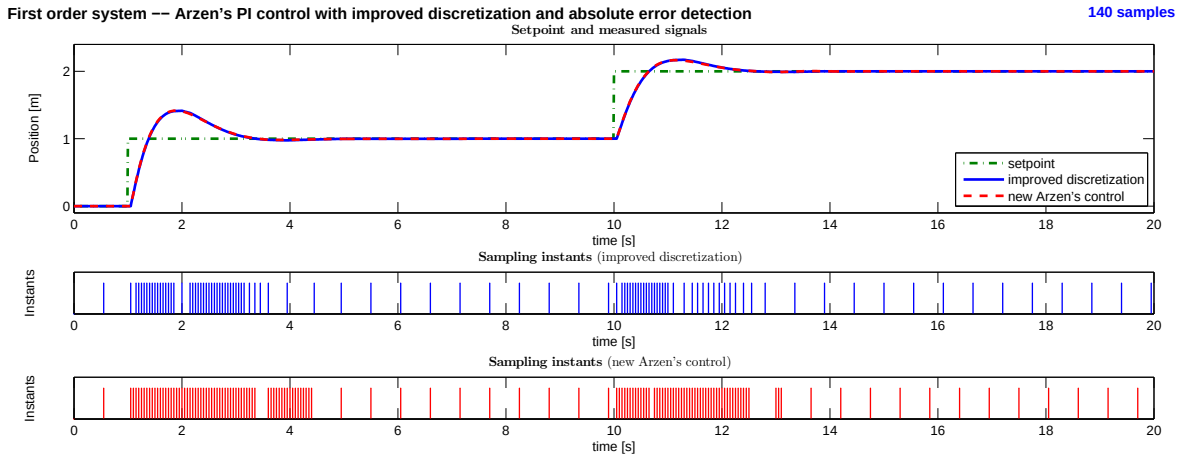


Figure 6.18: Simulation results: the improved Årzén's PI controller with relative error detection vs. the improved Årzén's one with absolute error detection.

Afterwards, the absolute error is used for level-crossing detection. Thus, the Årzén's controller with improved discretization and absolute error detection is denoted the new Årzén's controller.

Removing the safety limit condition

Thanks to the level detection, the maximal sampling period h_{max} - initially introduced by Årzén for stability reasons in order to fulfill the condition of Nyquist-Shannon sampling theorem - is no more consistent and, consequently, it can be removed. Several algorithms running without this safety limit condition were proposed (see subsection 6.2.3 for further details):

1. *Algorithm only without safety limit condition:*
   By removing only the safety limit condition without doing anything else, important overshoots should appear after a long steady-state interval because of the integral gain $he(t_a)$ which explodes. This issue clearly occurs in simulations, as one can see in figure 6.19(a).

2. *Algorithm with saturation of the integral gain:*
   In order to reduce the impact of the integral gain we proposed to bound its value. Indeed, in fact during the steady-state intervals only $h(t_a)$ or $e(t_a)$ increases a lot in the product $he(t_a)$ and the integral gain can hence be divided into two parts, where only one or the other becomes large. Using this principle, the integral gain becomes $he_{sat}(t_a) = \big(h(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$ and the overshoots disappear, as one can see in figure 6.19(b).

3. *Algorithm with an exponential forgetting factor of the sampling interval:*
   An exponential forgetting factor is then used to reduce the impact of the sampling interval $h(t_a)$ after a long steady-state interval. The integral gain becomes $he_{exp}(t_a) = h_{exp}(t_a) \cdot e(t_a)$, where $h_{exp}(t_a) = h(t_a) \cdot exp\Big(\alpha \cdot \big(h_{nom} - h(t_a)\big)\Big)$. The simulation results are drawn in figure 6.19(c) for $\alpha = 10$.

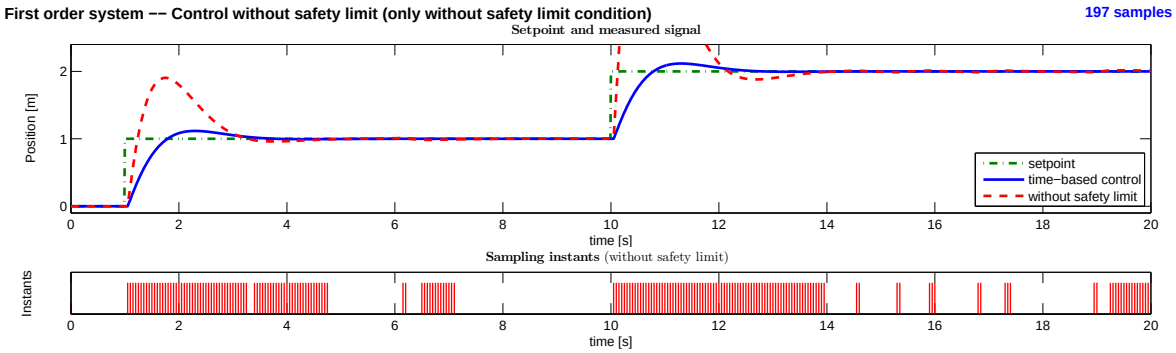
4. *Algorithm using a hybrid strategy:*
   Finally, a mix between the saturation of the integral gain and the exponential forgetting factor of the sampling interval leads to use an exponentially decreasing sampling interval into a bounded integral gain, that is $he_{hybrid}(t_a) = \big(h_{exp}(t_a) - h_{nom}\big) \cdot q_{nom} + h_{nom} \cdot e(t_a)$. Results are represented in figure 6.19(d).

The simulation results of the proposals are quite interesting. Indeed, the responses are really similar to the conventional time-triggered one, both in term of transients and overshoots (except for the first algorithm). Two algorithms can be highlighted:

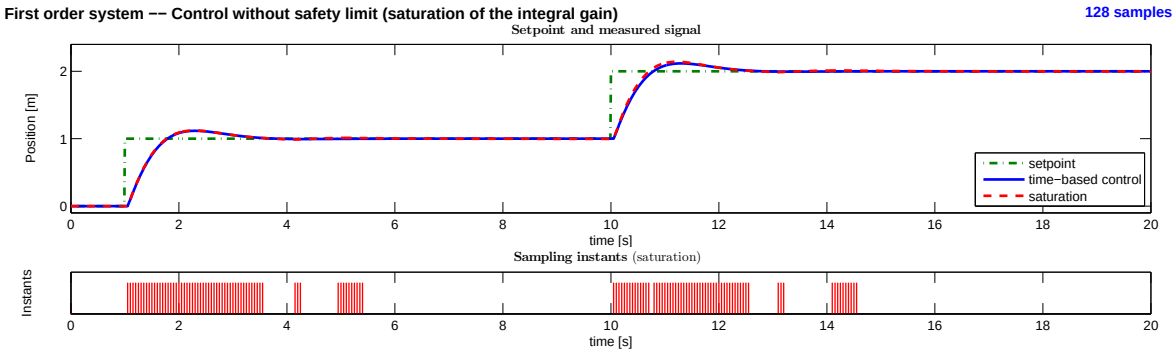
a) The hybrid algorithm is the best one. It leads to a control without performance degradation, by calculating a control signal about 75 % less often than with the time-triggered controller. Moreover, if we compare the results with the improved Årzén's controller (with absolute error for the level-crossing detection in order to be coherent), depicted in figure 6.18, the gain is about 20 % and the performance improvements are very important. However, the main problem of the hybrid strategy is the computational complexity of the algorithm in practice because of the exponential function.

b) On the other hand, the algorithm with saturation of the integral gain $he(\cdot)$ is quite simple and gives similar results. This could be a good alternative for an implementation with high resource constraints.

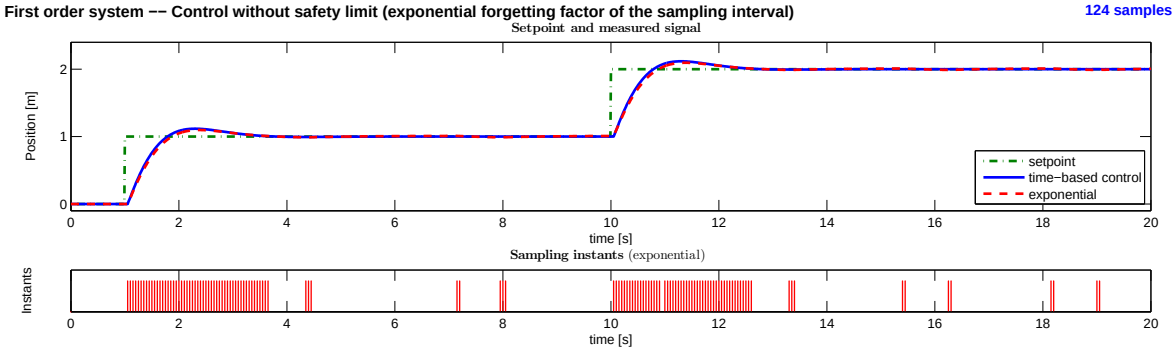
Otherwise, algorithms using a low-cost exponential function can be used, as algorithms 5 and 6 also proposed in subsection 6.2.3.



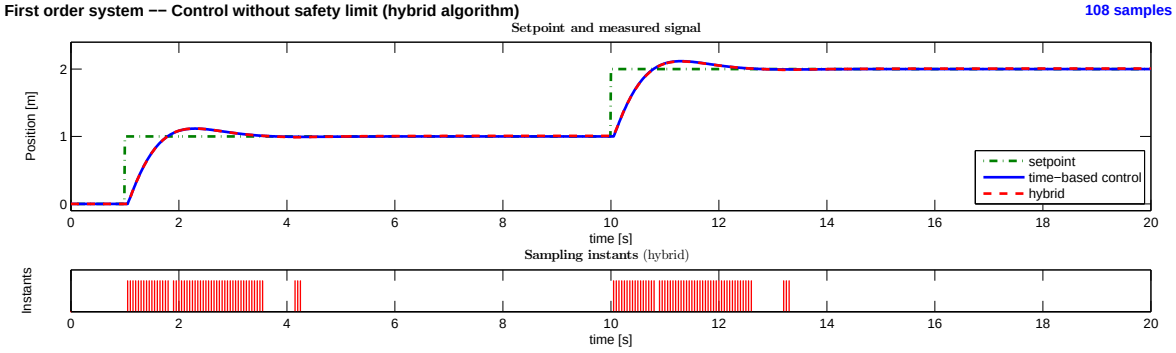
(a) Algorithm 1: only without safety limit condition



(b) Algorithm 2: with saturation of the product he



(c) Algorithm 3: with an exponential forgetting factor of h



(d) Algorithm 4: hybrid strategy

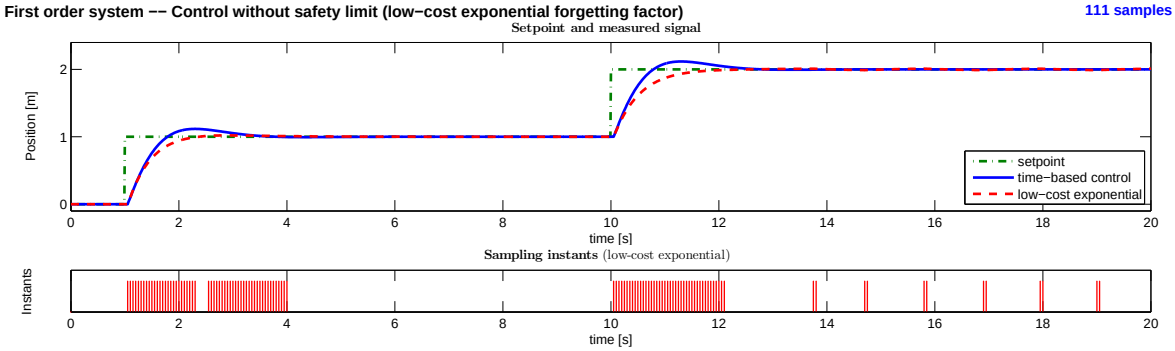
Figure 6.19: Simulation results: the time-triggered PI controller vs. the event-based PI controllers without safety limit condition.

5. *Algorithm with an exponential forgetting factor of the sampling interval with low-cost implementation:*
   A low-cost exponential forgetting factor $he^i_{exp}(t_a)$ is finally used to simplify the original one. The integral gain becomes $he^i_{exp}(t_a) = h^i_{exp}(t_a) \cdot e(t_a)$. The simulation results are plotted in figure 6.20(a) for $h^i_{max} = 12 \cdot h_{nom}$.

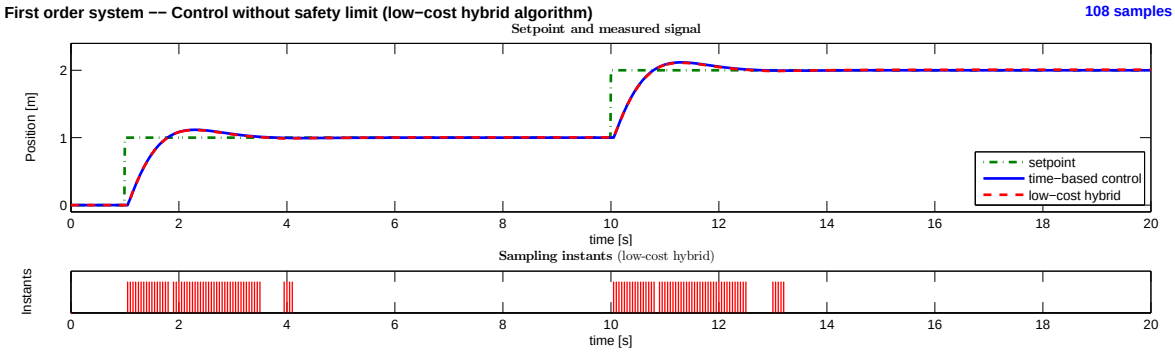
6. *Algorithm using a hybrid strategy with low-cost implementation:*
   The hybrid strategy is also simplified, using the low-cost exponential function. This yields $he^i_{hybrid}(t_a) = \left(h^i_{exp}(t_a) - h_{nom}\right) \cdot q_{nom} + h_{nom} \cdot e(t_a)$ and results are shown in figure 6.20(b).

As one can see, the simulation results for both low-cost proposals are quite similar to the original ones and hence could be a good alternative for systems with low computational resources.

Afterwards, the hybrid strategy is applied.



(a) Algorithm 5: with a low-cost exponential forgetting factor of h



(b) Algorithm 6: with a low-cost exponential forgetting factor of h

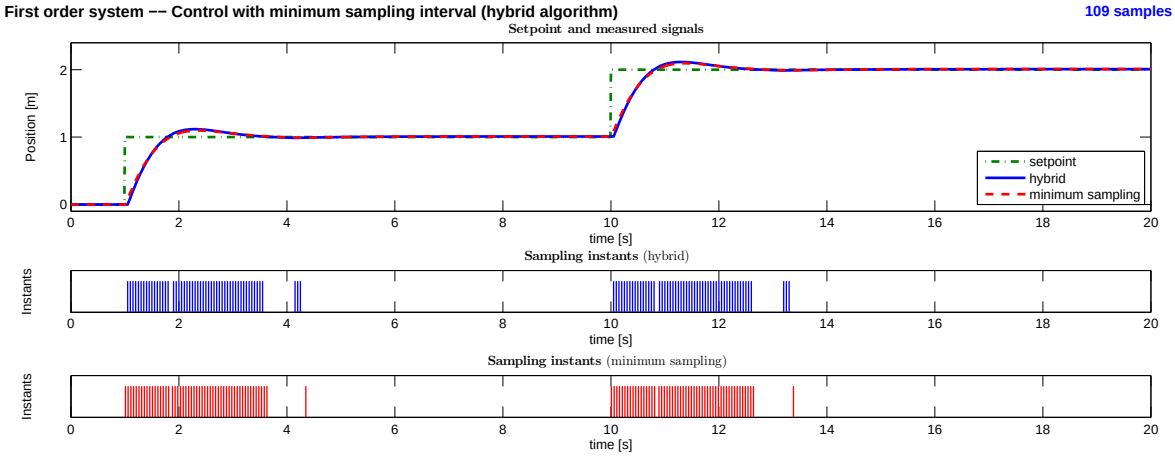
Figure 6.20: Simulation results: the time-triggered PI controller vs. the event-based PI controllers without safety limit condition and with low-cost implementation.

Adding a minimum sampling interval

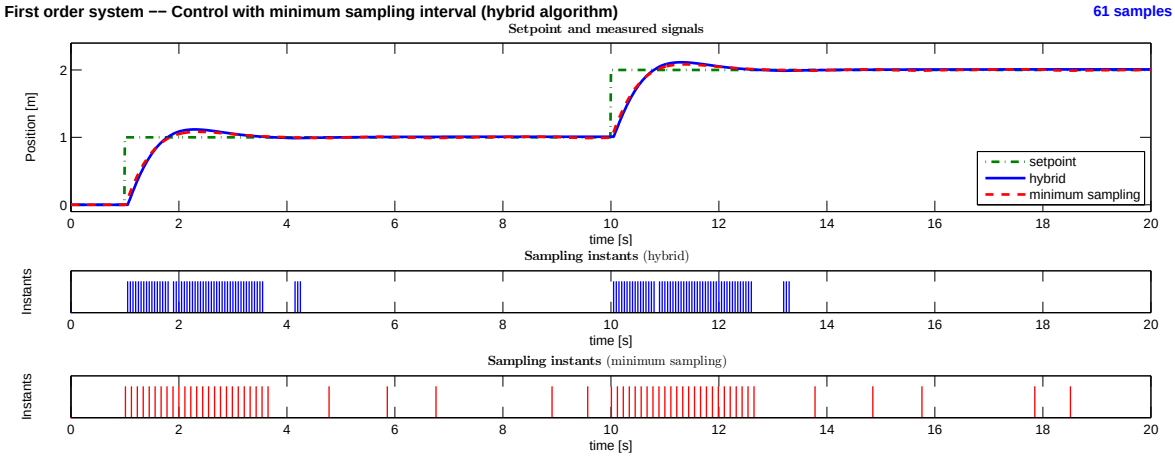
We also change the event detector architecture in subsection 6.2.4: the event detector is now time triggered with a continuous-time framework in order to immediately detect a level crossing (while the system had to wait for the next sampling step $t_k = k \cdot h_{nom}$ in the discrete-time case). Furthermore, although the algorithms without safety limit condition give acceptable results, the number of samples can be reduced again and more especially during the transients. A minimum sampling interval condition is proposed in such a way that a new control signal is calculated when the absolute error crosses the detection level q_{nom} - as previously - but only if

a given amount of time was elapsed since the last sample, that is when $h(t_a) \geq h_{min}$. Choosing this minimum sampling interval equal to the discrete sampling period $h_{nom} = 0.05\,s$ leads to a (quasi)-similar behavior than before, but with a more reactive event detection. This scheme is shown in figure 6.21(a). However, increasing the minimum sampling interval decreases the number of samples. This is represented in figure 6.21(b) for $h_{min} = 0.1\,s$, where doubling the minimum sampling interval almost reduces by two the final number of samples. Comparing these results with some existing methods, the controller with minimum sampling condition updates the control signal almost 90 % less than the classical time-triggered PI controller for the same achieved performance and about 40 % less than the improved Årzén's one.

Afterwards, the value of $h_{min} = 0.1\,s$ is used in simulations.



(a) Minimum sampling with $h_{min} = h_{nom} = 0.05\,s$



(b) Minimum sampling with $h_{min} = h_{nom} = 0.1\,s$

Figure 6.21: Simulation results: the hybrid event-based PI controller vs. the event-based PI controller with minimum sampling interval.

Adding some extra samples

Having reach a high reduction of the number of samples in the previous simulation results (adding a minimum sampling interval), we finally proposed - in subsection 6.2.5 - to improve the error margin during the steady-state intervals. The idea is to enforce events some few

times more after a transient in order to reduce the error again. Thus, figure 6.22 depicts this principle by adding two extra samples in a first time: some requests are sent to the controller - with the constant sampling period $h_{extra} = 0.5\,s$ - just after a transient. One can remark that some unexpected samples (which could appear during the steady-state intervals in the control strategies without extra samples, such as during the simulation time 4 and $10\,s$) are not persistent anymore when some samples are added. Indeed, in fact the extra samples allow to reduce the error, more than the detection level q_{nom}, and an event will hence occur later. Nevertheless, by adding to much samples can overloaded the system, as when adding ten extra samples (also plotted in figure 6.22), and one has to take care of that.

Based on this intuitive idea, an extension can be done adding some extra samples while a minimum error q_{min} is not achieved. This is denoted n extra samples in figure 6.22, where the mechanism is applied for $q_{min} = {}^{q_{nom}}/_{10} = 0.001$. One could note that, in fact, only one extra sample is enough to achieve the expected error and to have a real steady-state interval.

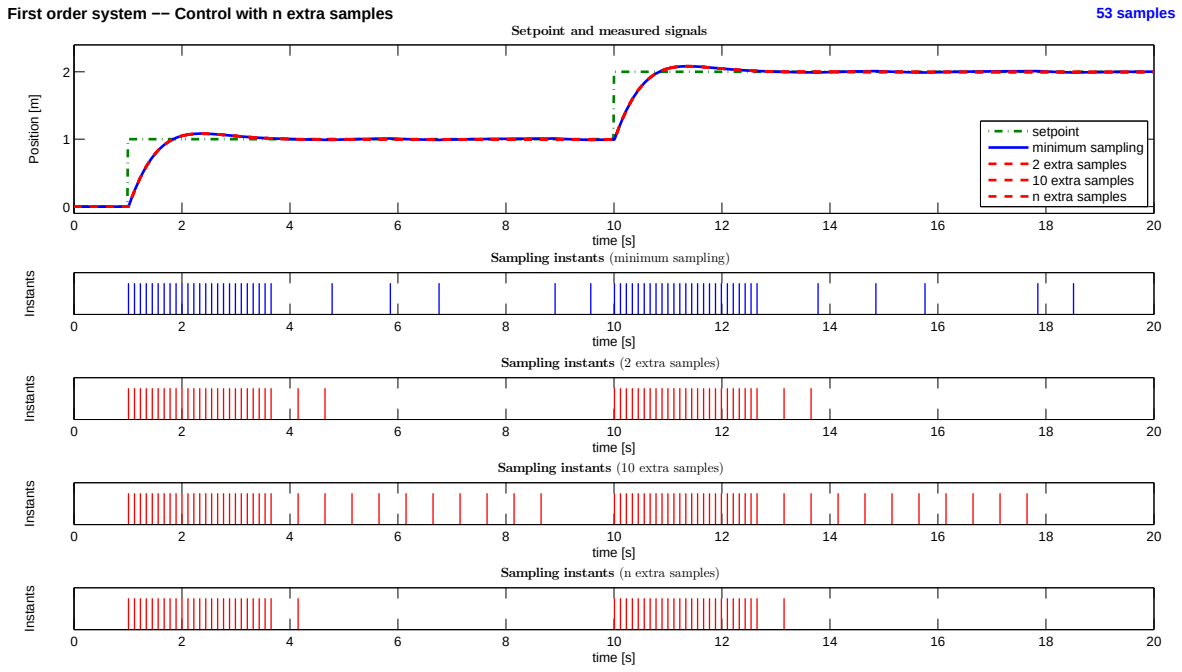


Figure 6.22: Simulation results: the hybrid event-based PI controller with minimum sampling interval vs. the one with 2, 10 ad n extra samples.

Performance analysis

The number of samples required to control the system during the whole simulation time - the number of calls - is an interesting criteria to compare the different strategies. This index was previously given in each figure. However, some other indexes of performance could be useful like those providing information on the quality of the system response. Such ones are depicted in section 6.5. Finally, running the different control strategies with the previous simulation test bench gives the values in table 6.1. Note that the minimum values are highlighted. The results show that the controllers without safety limit condition are better than the Årzén's ones, whatever the initial, improved or absolute strategy. The number of calls is quite similar while the IAE, IAEP and IAD indexes are better (except for the first algorithm: only without safety limit condition).

Table 6.1: Performance analysis: comparison of the different event-based PI strategies to control a first-order system.

		Calls		IAE	IAEP	IAD
		value	ratio			
Time-based		400	100 %	0.99	0	0
Årzén	Initial	127	31.5 %	1.96	1.21	13.31
	Improved	**102**	25.25 %	1.19	**0.60**	3.40
	Absolute	140	34.75 %	**1.18**	0.60	**3.27**
Without safety limit	Algo 1	198	49.25 %	4.34	4.28	35.88
	Algo 2	128	31.75 %	**1.03**	0.11	0.29
	Algo 3	124	30.75 %	1.08	0.18	0.75
	Algo 4	**108**	26.75 %	1.07	**0.09**	0.57
	Algo 5	111	27.5 %	1.06	0.54	0.66
	Algo 6	108	26.75 %	1.04	0.09	**0.21**
With minimum interval	Algo 1	191	47.5 %	2.54	2.62	18.99
	Algo 2	126	31.25 %	**0.93**	**0.15**	0.56
	Algo 3	116	28.75 %	0.98	0.22	**0.20**
	Algo 4	**109**	27 %	0.98	0.19	0.33
	Algo 5	116	18.75 %	0.99	0.56	0.61
	Algo 6	112	27.75 %	0.95	0.17	0.49
With 2 extra samples	Algo 1	163	40.5 %	5.92	6.08	49.57
	Algo 2	112	27.75 %	**0.92**	0.16	0.67
	Algo 3	113	28 %	0.96	0.22	**0.18**
	Algo 4	111	27.5 %	0.98	0.18	0.31
	Algo 5	114	28.25 %	0.97	0.54	0.54
	Algo 6	**107**	26.5 %	0.94	**0.15**	0.59
With 10 extra samples	Algo 1	177	44 %	3.78	3.91	30.17
	Algo 2	230	57.25 %	0.99	0.21	**0.34**
	Algo 3	129	32 %	0.93	0.18	0.36
	Algo 4	127	31.5 %	0.93	**0.13**	0.55
	Algo 5	131	32.5 %	0.96	0.52	0.53
	Algo 6	**123**	30.5 %	**0.93**	0.16	0.70
With extra samples	Algo 1	171	42.5 %	5.63	5.80	47.10
	Algo 2	**111**	27.5 %	0.95	0.18	0.59
	Algo 3	113	28 %	0.96	0.22	**0.18**
	Algo 4	161	40 %	0.93	**0.13**	0.56
	Algo 5	126	31.25 %	0.96	0.52	0.53
	Algo 6	115	28.5 %	**0.92**	0.14	0.72

Notes:

- *The sampling period used by the time-triggered controller and the event detector is* $h_{nom} = 0.05\,s$.
- *The detection level is* $q_{nom} = 0.01$.
- *The maximal sampling period used by the original Årzén's controller is* $h_{max} = 0.5\,s$. *The improved strategy uses the backward difference approximation for the integral part instead of the original forward one. The absolute strategy uses the absolute error for the detection level instead of the original relative one.*
- *The parameters used in the controllers without safety limit are* $\alpha = 10$ *and* $h^i_{max} = 12 \cdot h_{nom}$.
- *The minimum sampling interval is* $h_{min} = 0.05\,s$.
- *The extra sampling interval is* $h_{extra} = 5 \cdot h_{min}$ *while the minimum detection level is* $q_{min} = 0.001$.

Table 6.2: Performance analysis: comparison of the different event-based PI strategies to control a first-order system when the minimal sampling interval is doubled, that is $h_{min} = 2 \cdot h_{nom}$.

		Calls		IAE	IAEP	IAD
		value	ratio			
Time-based		401	100 %	0.99	0	0
Årzén	Initial	127	31.5 %	1.96	1.21	13.31
	Improved	**102**	25.25 %	1.19	**0.60**	3.40
	Absolute	140	34.75 %	**1.18**	0.60	**3.27**
Without safety limit	Algo 1	198	49.25 %	4.34	4.28	35.88
	Algo 2	128	31.75 %	**1.03**	0.11	0.29
	Algo 3	124	30.75 %	1.08	0.18	0.75
	Algo 4	**108**	26.75 %	1.07	**0.09**	0.57
	Algo 5	111	27.5 %	1.06	0.54	0.66
	Algo 6	108	26.75 %	1.04	0.09	**0.21**
With minimum interval	Algo 1	78	19.25 %	6.46	6.64	54.43
	Algo 2	70	17.25 %	0.88	0.19	1.21
	Algo 3	**58**	14.25 %	0.97	0.34	**0.26**
	Algo 4	61	15 %	0.90	0.20	1.12
	Algo 5	73	18 %	1.36	1.03	6.13
	Algo 6	53	13 %	**0.84**	**0.17**	1.61
With 2 extra samples	Algo 1	76	18.75 %	5.41	5.58	44.83
	Algo 2	114	28.25 %	0.99	0.25	0.52
	Algo 3	55	13.5 %	0.93	0.32	**0.50**
	Algo 4	55	13.5 %	**0.87**	**0.18**	1.42
	Algo 5	77	19 %	1.36	1.03	6.19
	Algo 6	**53**	13 %	0.87	0.21	1.50
With 10 extra samples	Algo 1	78	19.25 %	1.46	1.33	8.89
	Algo 2	114	28.25 %	0.99	0.25	0.52
	Algo 3	**71**	17.5 %	0.93	0.32	**0.51**
	Algo 4	**71**	17.5 %	**0.89**	**0.20**	1.26
	Algo 5	82	20.25 %	1.31	0.97	5.85
	Algo 6	80	19.75 %	0.92	0.23	1.13
With extra samples	Algo 1	77	19 %	4.88	5.04	40.03
	Algo 2	112	27.75 %	1.04	0.30	0.81
	Algo 3	64	15.75 %	0.92	0.30	**0.64**
	Algo 4	**53**	13 %	**0.83**	**0.15**	1.67
	Algo 5	105	26 %	1.63	1.40	8.06
	Algo 6	53	13 %	0.87	0.21	1.50

Notes:

- *See notes in* ***table 6.1****.*
- *The minimum sampling interval is* $h_{min} = 0.1\,s$.

Similar results are also obtained using the minimum sampling interval when the value of h_{min} is equal to h_{nom}. In this case, the behavior is almost the same but we hope a decrease of the number of calls when h_{min} increases, without degrading the system performance. The resulting values are presented in table 6.2. As expected, the number of calls drastically decreases while the performance indexes remains quite good. As regards the controllers with extra samples, the idea is to enforce some events just after the transients in order to improve the system response. In both cases, in tables 6.1 and 6.2, the expected scheme is not always achieved since the performance indexes increase sometimes. Nevertheless, one could note that this technique allows to improve the steady-state intervals by reducing the number of samples. This was depicted in the last simulation results (adding some extra samples).

Robustness to a disturbance

Finally, the event-based approach still works when a disturbance occurs. Thus, one could see the resulting behavior in figure 6.23 where the hybrid strategy is tested. A load disturbance is introduced at time $12\,s$ with an amplitude of 0.1 and the controller reacts in consequence, as well as the conventional PI controller. This is also true for the other proposals (not represented here).

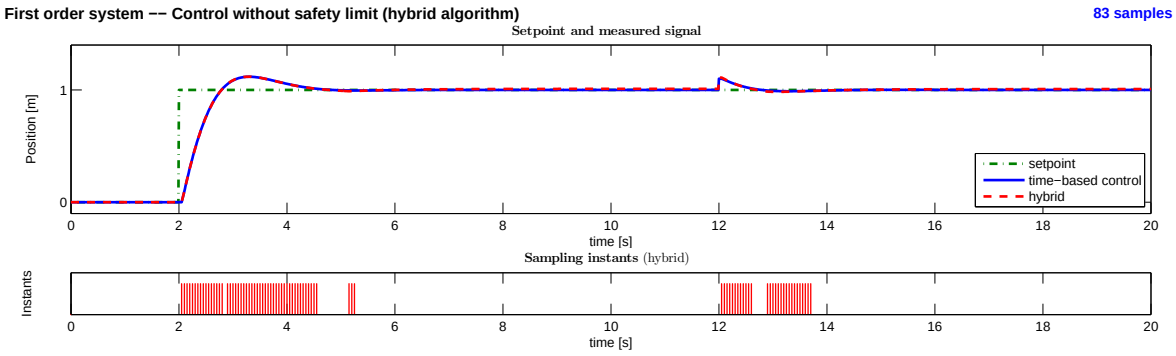


Figure 6.23: Simulation results: robustness of the hybrid event-based PI controllers to a disturbance.

6.6.2 Application to a cruise control mechanism

As explained in section 6.2, an event-based technique is a good solution to reduce the control computational needs. We studied a very simple first-order system in subsection 6.6.1 to illustrate the interest of the event-based framework. However, more complex systems could be controlled too. Such a system is the cruise control mechanism for instance, a control system that regulates the speed of a vehicle. The expected speed of the car is constant most of time and a new control signal is only required when the setpoint changes or when the load (i.e. the slope of the road) varies. The basic operation of a cruise controller - depicted in [15] - is to sense the speed of the car, compare this speed to a given reference, and then accelerate or decelerate the car as a result. The equation of motion of the vehicle is

$$m \cdot \dot{\nu} = F - F_d$$

where ν is the velocity and $m = 1000\,kg$ is the mass of the vehicle (this mass could vary with the number of passengers or while towing a tailer). The driving force F is generated by the engine, whose torque is proportional to a signal $0 \leq u \leq 1$ that controls the throttle position

and depends on the engine velocity too

$$F = \alpha_n \cdot u \cdot T_m \cdot \left(1 - \beta \cdot \left(\frac{\alpha_n \cdot \nu}{\omega_m} - 1\right)^2\right)$$

where the maximal torque $T_m = 190\,Nm$ is obtained at engine speed $\omega_m = 420\,rad.s^{-1}$ and $\beta = 0.4$. A physical interpretation of α_n, which depends on the gear ratio n, is the inverse of the effective wheel radius. On the other hand, the disturbance force has three major components, respectively due to the gravity F_g, the rolling friction F_r and the aerodynamic drag F_a

$$\begin{aligned} F_d &= F_g + F_r + F_a \\ F_g &= m \cdot g \cdot sin(\theta) \\ F_r &= m \cdot g \cdot C_r \cdot sgn(\nu) \\ F_a &= 1/2 \cdot \rho \cdot C_d \cdot A \cdot \nu^2 \end{aligned}$$

where $g = 9.8\,m.s^{-2}$ is the gravitational constant, $sgn(\nu)$ is the sign of ν or zero if $\nu = 0$, $C_r = 0.01$ and $C_d = 0.32$ are the rolling friction and the shape-dependent aerodynamic drag coefficients respectively, $\rho = 1.3\,kg.s^{-3}$ is the density of air and $A = 2.4\,m^2$ is the frontal area of the vehicle. At the end, θ is the slope of the road, that is the disturbance.

This cruise control mechanism is controlled with a PID control strategy. An anti-windup mechanism is also added in order to consider the saturation of the control signal u. The integral part hence consists in the classical integral term plus a *reset* based on the saturation of the actuator (in order to prevent windup when the actuator is saturated):

$$u_i(t_a) = \underbrace{u_i(t_{a-1}) + \frac{K_p}{T_i} \cdot he(t_a)}_{\text{integral term}} - \underbrace{\frac{1}{T_a} \cdot h(t_a) \cdot \Big(u(t_{a-1}) - u_{sat}(t_{a-1})\Big)}_{\text{anti-windup term}}$$

where $he(t_a)$ depends on the event-based algorithm (see subsection 6.2.3), T_a is a tunable parameter for the anti-windup mechanism and $u_{sat}(\cdot)$ is the saturated value of the control signal. The values used for simulations are $K_p = 0.33$, $T_i = 20$, $T_a = 5$, $T_d = 1.4$ and $N = 10$. As regards the test bench, a $200s$ simulation runtime is expected: at time 0, the setpoint is $25\,m/s$ ($90\,km/h$), then at time $10\,s$ it is changed to $30.6\,m/s$ ($110\,km/h$) and changed again to $36.1\,m/s$ ($130\,km/h$) at time $100\,s$. Furthermore, the gear ratio is chosen with respect to the speed range, that is $n = 5$ and so is $\alpha_n = 10$. Eventually, no disturbance is applied, i.e. $\theta = 0$.

Time-based strategy

The results of the conventional time-triggered case are shown in figure 6.24, where the nominal sampling period is $h_{nom} = 0.2\,s$. The top plot shows the speed setpoint and the measured signal, the bottom plot shows the sampling intervals. The resulting value is constant in this time-driven case, that is corresponding to h_{nom}.

Årzén's strategy

The same system is then controlled with the Årzén's event-based controller, with improved discretization and absolute error detection (see subsections 6.2.2 and 6.2.3 for further details). The simulation results are represented in figure 6.25. The detection level is $q_{nom} = 0.2$, which means that a new control signal is calculated only when the system output crosses this level.

Finally, the system response is as quick as the time-triggered one but a control signal is computed about twice less (with this proposed benchmark). Nevertheless, the control signal is still calculated during the steady-state intervals because of the maximal sampling interval condition. One can thus see the periodic sampling due to $h_{max} = 1\,s$.

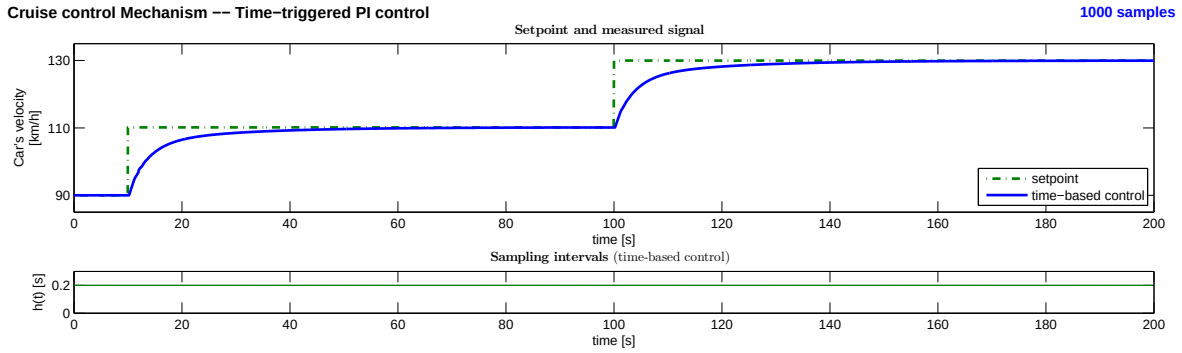


Figure 6.24: Simulation results: the conventional time-triggered PID controller.

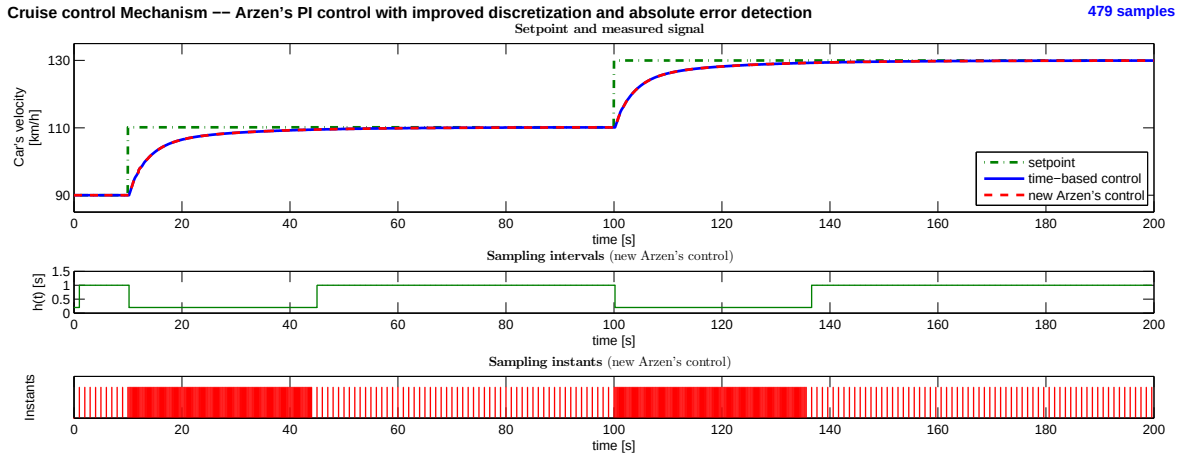
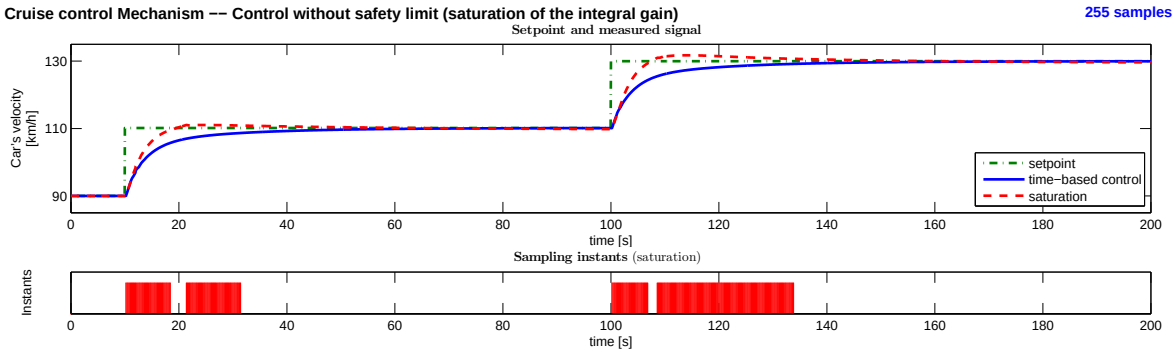


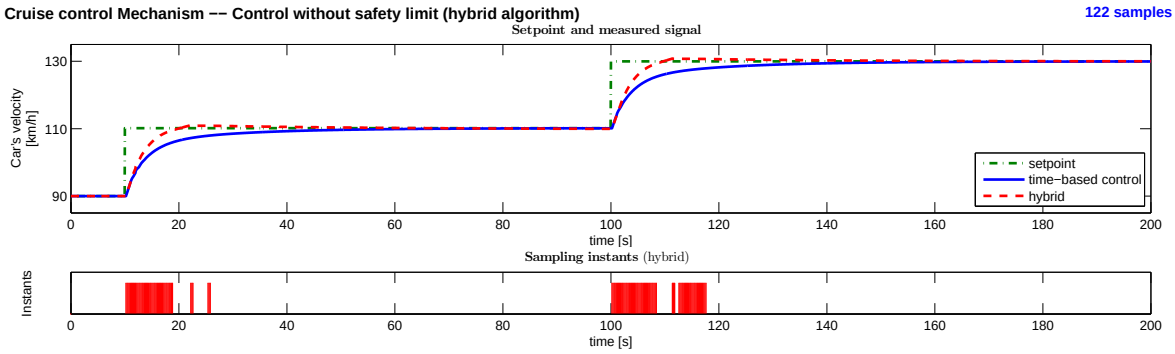
Figure 6.25: Simulation results: the new Årzén's event-based controller.

Removing the safety limit condition

Several algorithms without this safety limit condition were developed (see subsection 6.2.3 for further details). The simulation results for the saturation and the hybrid algorithms, for instance, are represented in figure 6.26. In the first case, a saturation of the integral gain $he(t_a)$ is performed in the integral part of the controller. In the second case, this saturation is used and an exponential forgetting factor of the sampling interval $h(t_a)$ is also added in order to reduce its impact after a long steady-state interval. The gain is important since only 12 % of samples are required with the hybrid algorithm to obtain a better response. The performance indexes are given in table 6.3. A smaller IAE index indicates that the system response of our proposal is closer to the setpoint than the one of the time-triggered case. As a result, the IAEP and IAD indexes are more important. These indexes compare the performance of the time-based and event-based strategies.



(a) Algorithm with saturation of the product *he*



(b) Hybrid algorithm

Figure 6.26: Simulation results: the event-based PID controllers without safety limit condition.

Table 6.3: Performance analysis: comparison of the different event-based PID strategies to control a cruise control mechanism (when $m = 1000\,kg$).

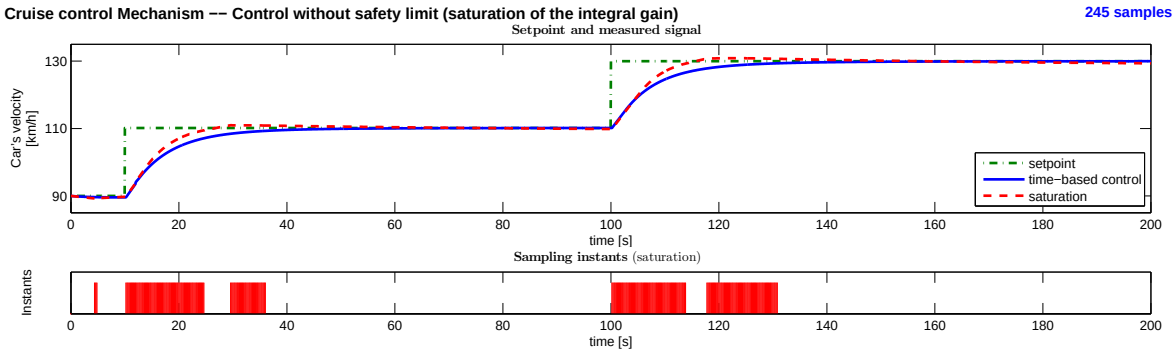
		Calls		IAE	IAEP	IAD
		value	ratio			
Time-based		1000	100 %	80.25	0	0
Årzén	Absolute	479	47.8 %	80.39	**0.14**	**11.53**
Without safety limit	Saturation	255	25.4 %	54.83	65.20	3716.88
	Hybrid	**122**	12.1 %	**47.80**	55.70	4239.09

Notes:

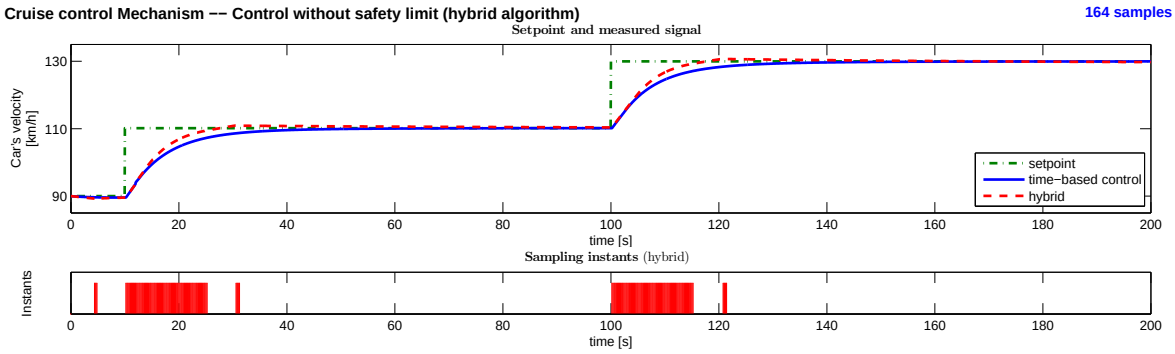
- *The nominal sampling period is* $h_{nom} = 0.2\,s$.
- *The detection level is* $q_{nom} = 0.1$.
- *The maximal sampling period used by the Årzén's controller is* $h_{max} = 1\,s$.
- *The parameter used in the controllers without safety limit is* $\alpha = 10$.

Robustness: variation of the mass of the vehicle

As explained before, the mass of the vehicle can vary. Nevertheless, the system is still working when changing the mass to $m = 2000\,kg$ (when the car is towing a tailer for example), that is increasing the mass with a factor of 2, as one can see in figure 6.27.
Moreover, the performance analysis for the loaded vehicle is the same than the vehicle only, as depicted in table 6.4.



(a) Algorithm with saturation of the product *he*



(b) Hybrid algorithm

Figure 6.27: Simulation results: event-based PID control and robustness to system error (when $m = 2000\,kg$).

Table 6.4: Performance analysis: comparison of the different event-based PID strategies to control a cruise control mechanism (when $m = 2000\,kg$).

		Calls		IAE	IAEP	IAD
		value	ratio			
Time-based		1000	100 %	93.62	0	0
Årzén	Absolute	429	42.8 %	93.74	**0.24**	**9.87**
Without safety limit	Saturation	245	24.4 %	81.48	45.00	2146.48
	Hybrid	**164**	16.3 %	**80.58**	42.38	1819.80

Notes:

- *See notes in table 6.3.*

6.7 Synthesis

This chapter presents some new event-based PID control algorithms. Such a scheme, contrary to the classical time-triggered one which calculates the control signal at each sampling time, updates the control signal only when the measurement *sufficiently* changes. Our proposals are based on the simple setup proposed by K.E. Årzén in [10]. In the original work, a safety maximum period is added forcing the control to be recomputed even if the measured signal remains unchanged. The main contribution is to avoid this re-computation. To compensate, a forgetting factor is imagined in order to reduce the sampling period impact in the integral part of the PID algorithm. This approach is somehow similar to the anti-windup mechanism,

where the error induced by the saturation has to be compensated. Then, based on this idea, six controllers without safety limit condition are proposed:

- **Algorithm only without safety limit condition**
- **Algorithm with saturation of the integral gain**, where the product $h(\cdot)e(\cdot)$ is bounded in order to reduce its impact in the integral part.
- **Algorithm with an exponential forgetting factor of the sampling interval**, in order to reduce the impact of the sampling interval $h(t_a)$ after a long steady-state interval.
- **Algorithm using a hybrid strategy**, which consists in a mix between the saturation of the integral gain and the exponential forgetting factor of the sampling interval.
- **Algorithm with an exponential forgetting factor of the sampling interval with low-cost implementation**, which uses a low-cost exponential forgetting factor to simplify the original proposal.
- **Algorithm using a hybrid strategy with low-cost implementation**, where the low-cost exponential function is also used.

These proposals are finally compared, both with the conventional time-triggered controller and the Årzén's event-based controller. Besides a noticeable reduction of the mean control computation cost, the performance of the closed-loop system is also improved in simulation.

A second contribution is a low computational cost scheme thanks to a minimum sampling interval condition. This was added to lighten the transients: a new control signal is performed only if a given amount of time was elapsed since the last sample. Eventually, we also suggested to reduce even more the error margin during the steady-state intervals by adding some extra samples just after the transients. At the end, all our proposals are simulated with a simple first-order system and then, with a more complex cruise control mechanism. The control computation cost is thus decreased again, while good closed-loop performance is obtained.

Chapter

7 State-feedback controllers based on Lyapunov sampling

The state-feedback control is another architecture widely used to control systems, since it simply consists in multiplying the system output with a certain gain and setting the resulting product as the new system input. Consequently, a large number of applications is available. We hence choose such a control architecture to show the advantages on using an event-driven strategy. Moreover, some theoretical tools can be (more or less) easily used to prove the stability of the system, using Lyapunov theory, which was the penalizing point in the previous chapter. An interesting strategy is to base the event detection on a Lyapunov sampling mechanism, as introduced by *Manel Velasco et al.* in [68]. Thus, the control signal is updated when a Lyapunov-candidate function "sufficiently" changes. Different event-based strategies are developed in this chapter, still with the aim at reducing the computational cost of the controller - by reducing the number of samples - while guaranteeing some good system performance. The theoretical background on state-feedback control and Lyapunov stability is brought back in sections 7.1 and 7.2 respectively. The latter one also recalls the existing Lyapunov sampling mechanism imagined by Velasco et al., and finally details a less-conservative sampling scheme proposal. Then, a recap of the different strategies is done in section 7.3. Some performance indexes are given in section 7.4 in order to compare the performance of asynchronous control strategies and some simulation results are eventually presented in section 7.5 with a double integrator system. At the end, a synthesis is performed in section 7.6.

7.1 Theoretical background on state-feedback control

The state-space representation is a mathematical model of a physical system as a set of input, output and state variables related by first-order differential equations. Moreover, to abstract from the number of these variables, the vector/matrix form is usually preferred (for the linear case). The state-space representation provides a convenient and compact way to model and analyze systems with multiple inputs and outputs. The internal state variables represent the entire state of the system at any given time. The most general state-space representation of a continuous-time and linear system (with p inputs, q outputs and n state variables) is written as follows

$$\begin{aligned} \dot{x}(t) &= A(t) \cdot x(t) + B(t) \cdot u(t) \\ y(t) &= C(t) \cdot x(t) + D(t) \cdot u(t) \end{aligned}$$

where $x(t) \in \mathbb{R}^n$ is called the *state vector*, $y(t) \in \mathbb{R}^q$ is called the *output vector* and $u(t) \in \mathbb{R}^p$ is called the *input (or control) vector*. Let $\dot{x}(t)$ be the derivative of the state vector. On the other hand, $A(\cdot)$, $B(\cdot)$, $C(\cdot)$ and $D(\cdot)$ are the *state*, *input*, *output* and *feedthrough (or feedforward)* matrix respectively. They are defined such that $\dim[A(\cdot)] = n \times n$, $\dim[B(\cdot)] = n \times p$, $\dim[C(\cdot)] = q \times n$ and $\dim[D(\cdot)] = q \times p$. In this general formulation, all the matrices and their elements depend on time. However, in the common linear time-invariant systems, the matrices are time invariant. Furthermore, in cases where the system model does not have a direct feedthrough, D is the zero matrix. That will be the case afterwards for a simplification reason and the previous representation hence becomes

$$\begin{aligned} \dot{x}(t) &= A \cdot x(t) + B \cdot u(t) \\ y(t) &= C \cdot x(t) \end{aligned} \tag{7.1}$$

Controllability and observability are main issues in a system analysis before deciding the best control strategy to apply, or whether it is even possible to control or stabilize the system.

Controllability is related to the possibility in forcing the system into a particular state, by using an appropriate control signal. If a state is not controllable, then no signal will ever be able to control the state. Note that if a state is not controllable but its dynamics are stable, then the state is ***stabilizable***.

Observability is related to the possibility of observing, through output measurements, the state of a system. If a state is not observable, the controller will never be able to determine the behavior of an unobservable state and hence cannot use it to stabilize the system. However, similar to the stabilizability condition above, if a state cannot be observed it might still be ***detectable***.

From a geometrical point of view, every states of the system must be controllable and observable to ensure a good behavior in the closed-loop system, else the *bad* states and their corresponding dynamics will remain untouched. If such a state is not stable, its dynamics will be present in the closed-loop system which therefore will be unstable.

Actually, the transfer function of a continuous time-invariant linear state-space model can be derived in the following way. First, taking the Laplace transform of equation (7.1) yields

$$\begin{aligned} X(s) &= \big(s \cdot I - A\big)^{-1} \cdot B \cdot U(s) \\ Y(s) &= C \cdot X(s) \end{aligned}$$

Note that the general principles presented in this section was established with the help of [61] and some articles from the free encyclopedia Wikipedia (`http://fr.wikipedia.org/`)

and then, because the transfer function $G(s)$ of a system is defined as the ratio between its output and its input, substituting $X(s)$ in the previous output equation leads to

$$G(s) = \frac{Y(s)}{U(s)} = C \cdot \big(s \cdot I - A\big)^{-1} \cdot B$$

Clearly $G(s)$ must have $q \times p$ dimensionality. This is why the state-space representation can easily be the preferred choice for systems with multiple inputs and/or outputs. Moreover, it is important to understand that converting a state-space realization to a transfer-function form may lose some internal information about the system, and may provide a description of a system which is stable, when the state-space realization is unstable at some points.

7.1.1 Feedback control for linear systems

A feedback control consists in multiplying the output (or state) signal of the system by a certain gain, and setting this product as the new system input. This results in a closed-loop system where the derivative of the state - in equation (7.1) - directly varies with respect to the state itself. The system hence becomes autonomous and does not depend on exogenous variables anymore. Two feedbacks are usually applied in control theory: state and output feedback which are now detailed.

7.1.1.1 State feedback

This method consists in closing the loop using the state variables of the system, which results in $u\big(t, x(t)\big) = -K \cdot x(t)$. Note that the presence of a negative sign is the common notation but its absence has no impact on the end results. We assume here that all the state variables are accessible and so is fully observable the system. By substituting that in equation (7.1) yields

$$\begin{aligned} \dot{x}(t) &= \big(A - B \cdot K\big) \cdot x(t) \\ y(t) &= C \cdot x(t) \end{aligned} \tag{7.2}$$

The advantage of such a feedback is that the eigenvalues of the closed-loop state matrix $\mathbf{A} = A - B \cdot K$ can be controlled by setting K appropriately through eigen-decomposition of $A - B \cdot K$. This assumes that the open-loop system is controllable or that the unstable eigenvalues of $\mathbf{A}$ can be made stable through appropriate choice of K.

In addition to feedback, a setpoint input - afterwards denoted $r(t)$ - can be added when the system has to track a given reference. In this case, the input signal is $u\big(t, x(t), r(t)\big) = -K \cdot x(t) + K_r \cdot r(t)$ and the previous state-space representation becomes

$$\begin{aligned} \dot{x}(t) &= \big(A - B \cdot K\big) \cdot x(t) + B \cdot K_r \cdot r(t) \\ y(t) &= C \cdot x(t) \end{aligned} \tag{7.3}$$

7.1.1.2 Output feedback

Contrary to the previous case where we assumed that all the state variables are accessible, in the output-feedback method the system can be not fully observable (see above for a definition). Indeed, such a method consists in closing the loop with the output of the system, that is $u\big(t, y(t)\big) = -K \cdot y(t)$. By doing that, the system equation (7.1) yields

$$\begin{aligned} \dot{x}(t) &= \big(A - B \cdot K \cdot C\big) \cdot x(t) \\ y(t) &= C \cdot x(t) \end{aligned} \tag{7.4}$$

and the eigenvalues of the new closed-loop state matrix $\mathbf{A} = A - B \cdot K \cdot C$ can be controlled by setting K appropriately.

7.1.2 Generalization for nonlinear systems

The more general form of a continuous state-space model can be written as two functions to include the nonlinear behaviors in some systems, that are

$$\begin{aligned} \dot{x}(t) &= \phi\big(t, x(t), u(t)\big) \\ y(t) &= \varphi\big(t, x(t), u(t)\big) \end{aligned} \tag{7.5}$$

The first is the state equation and the latter is the output equation. If the functions $\phi(\cdot,\cdot,\cdot)$ and $\varphi(\cdot,\cdot,\cdot)$ are a linear combination of states and inputs then the equations can be written in matrix notation (as depicted above).

Regarding a state/output feedback control, the same scheme than previously remains unchanged. Thus, the general expression of a continuous state-feedback control is

$$u\big(t\big) = \psi\big(x(t)\big) \tag{7.6}$$

where $\psi(\cdot)$ is the input (or control) function.

7.1.3 From discrete-time to event-driven controllers

In this subsection, the general nonlinear continuous-time system - given in equation (7.5) - is driven by a discrete-time controller (which is the only possible way to implement a control law in practice). The state-feedback control - defined in equation (7.6) in the continuous case - hence yields

$$\forall t \in \big[t_k, t_{k+1}\big) \qquad u(t) = \psi\big(x(t_k)\big) = \psi(x_k)$$

where the control signal is updated using only some samples of the state at some periodic discrete instants $t_0, t_1, \ldots, t_k, t_{k+1}, \ldots$. Let t_k denote the current sampling instant and t_{k+1} the next one. Finally, the closed-loop system becomes

$$\dot{x}(t) = \phi\big(t, x(t), h(x_k)\big)$$

This depicts the discrete-time state-feedback control principle.

This principle can be easily extended to an event-driven scheme, where an event could only occur at some discret instants

$$t_0, t_1, \ldots, t_a, \ldots, t_k, t_{k+1}, \ldots, t_{a+1}, \ldots \tag{7.7}$$

Let

$$\forall t \in \big[t_a, t_{a+1}\big) \qquad u(t) = \psi\big(x(t_a)\big) = \psi(x_a) \tag{7.8}$$

be the control updates. Let t_a denote the beginning time of the current control sample, that is the last time an event occurs, and t_{a+1} denotes the next time where a control signal will be calculated. The *sampling intervals* $h(\cdot)$ are then function of these time instants and for instance $h(t_a) = t_a - t_{a-1}$ is the sampling period of the current sample. Eventually, the closed-loop system is

$$\dot{x}(t) = \phi\big(t, x(t), k(x_a)\big) \tag{7.9}$$

This depicts the *event-triggered state-feedback control principle*.

7.2 Lyapunov sampling for event-driven controllers

The notion of stability is important in control theory where the behavior of dynamical systems is studied. Lyapunov theory occurs in this field of knowledge. Consequently, developing some event-triggered controllers based on that is a natural way when stability proofs are expected (as explained in section 6.4). A short recall on this theory is presented in subsection 7.2.1. Such an event-based solution - initially introduced in [68] - is then presented in subsection 7.2.2 and, based on this work, we develop some new proposals in subsection 7.2.3. A trick to reduce the computational cost of the event-detection mechanism is finally explained in subsection 7.2.4.

7.2.1 Stability and Lyapunov theory

The basic Lyapunov theorems for autonomous systems are directly related to some particular functions, called *Lyapunov-candidate functions*. These theorems are a powerful tool to prove the stability of a given equilibrium point. Conceptually and in simple terms, if all solutions of the dynamical system that start *close enough* to an equilibrium point x_e remain near x_e forever, then this equilibrium point is Lyapunov stable. More strongly, if x_e is Lyapunov stable and all solutions that start close enough to x_e converge to x_e, then the equilibrium point is asymptotically stable. Moreover, the notion of exponential stability ensures that solutions not only converge, but converge faster than, or at least as fast as a particular known rate. Note that, without loss of generality, one may assume that the equilibrium is at the origin. Finally, the formal Lyapunov theory is also called back.

Lyapunov-candidate function

Let $V : \mathbb{R}^n \rightarrow \mathbb{R}$ be a continuous scalar function. V is a *local Lyapunov-candidate function* if this is a locally positive-definite function, that is

$$\begin{aligned} &V(0) = 0 \\ &V(x) > 0 \quad \forall x \in \mathcal{B} - \{0\} \end{aligned}$$

with $\mathcal{B}$ being a neighborhood region around $x = 0$.

Respectively, V is a *global Lyapunov-candidate function* if

$$\begin{aligned} &V(0) = 0 \\ &V(x) > 0 \quad \forall x \in \mathbb{R}^n - \{0\} \end{aligned}$$

Equilibrium point of a system

Let $g : \mathbb{R}^n \rightarrow \mathbb{R}^n$ be an arbitrary autonomous dynamical system with the equilibrium point y^*, that is

$$\dot{y} = g(y), \qquad g(y^*) = 0$$

If there always exists a coordinate transformation $x = y - y^*$, such as

$$\begin{aligned} &\dot{x} = g(x + y^*) = f(x) \\ &f(0) = 0 \end{aligned}$$

then the new system $f(x)$ has an equilibrium point at the origin.

Note that the general principles presented in subsection 7.2.1 was established with the help of [61] and some articles from the free encyclopedia Wikipedia (`http://fr.wikipedia.org/`)

Basic Lyapunov theorems for autonomous systems

Let the origin be an equilibrium of the autonomous system defined as

$$\dot{x} = f(x) \tag{7.10}$$

and let

$$\dot{V}(x) = \frac{\partial V}{\partial x} \cdot \frac{dx}{dt} = \nabla V \cdot \dot{x} = \nabla V \cdot f(x)$$

be the time derivative of the Lyapunov-candidate function V.

***Stable equilibrium*:** If the Lyapunov-candidate function V is locally positive definite and the time derivative of the Lyapunov-candidate function is locally negative semi-definite, that is

$$\begin{aligned} &V(0) = 0 \\ &V(x) > 0 \quad \forall x \in \mathcal{B} - \{0\} \\ &\dot{V}(x) \leq 0 \quad \forall x \in \mathcal{B} - \{0\} \end{aligned}$$

for some neighborhood $\mathcal{B}$ of the origin, then the equilibrium is stable.

***Locally asymptotically stable equilibrium*:** If the Lyapunov-candidate function V is locally positive definite and the time derivative of the Lyapunov-candidate function is locally negative definite, that is

$$\begin{aligned} &V(0) = 0 \\ &V(x) > 0 \quad \forall x \in \mathcal{B} - \{0\} \\ &\dot{V}(x) < 0 \quad \forall x \in \mathcal{B} \setminus \{0\} \end{aligned}$$

for some neighborhood $\mathcal{B}$ of the origin, then the equilibrium is locally asymptotically stable.

***Globally asymptotically stable equilibrium*:** If the Lyapunov-candidate function V is globally positive definite, radially unbounded and the time derivative of the Lyapunov-candidate function is globally negative definite, that is

$$\begin{aligned} &V(0) = 0 \\ &V(x) > 0 \quad \forall x \in \mathbb{R}^n - \{0\} \\ &\| x \| \to \infty \Rightarrow V(x) \to \infty \\ &\dot{V}(x) < 0 \quad \forall x \in \mathbb{R}^n - \{0\} \end{aligned}$$

then the equilibrium is globally asymptotically stable.

One must be aware that the basic Lyapunov theorems can only be applied to autonomous systems, that are some systems without exogenous input, as defined in equation (7.10). This is why a state-feedback control is usually common to lead a more general system back to this restrictive case. Note that feedback controls were introduced in section 7.1. Furthermore, these theorems are a sufficient but not necessary tool to prove the stability of an equilibrium. There is no general method to construct or find a Lyapunov-candidate function which satisfies a given stability criterium. The inability to find a Lyapunov function is inconclusive with respect to stability, which means that not finding a Lyapunov function does not mean that the system

is unstable. Nevertheless, the Lyapunov stability is based on a mathematical translation of an elementary physical constatation: if the total energy of the system tends to continuously decline, then this system is stable since it is going to an equilibrium state. For this reason, the Lyapunov-candidate functions are often based on some energetic functions, that are most of time quadratic functions of the state variables, such as

$$V(x) = x^T \cdot P \cdot x \tag{7.11}$$

where $P = P^T > 0$ is a symmetric and positive-definite matrix. In this case, a linear autonomous continuous-time system $\dot{x} = A \cdot x$ is globally asymptotically stable when satisfying

$$A^T \cdot P + P \cdot A < 0 \tag{7.12}$$

This relation is equivalent to say that all real parts of the eigenvalues of the matrix A are negative. The stability of the system can finally easily be proved finding the relevant Lyapunov function defined in equation (7.11).

7.2.2 Event detection based on Lyapunov functions

Manel Velasco et al. investigated in [68] an event condition for asynchronous controllers based on Lyapunov functions. Considering that constant values of a Lyapunov function define some contour curves that form closed regions around the equilibrium point, the proposed sampling mechanism enforces job executions each time the system trajectory reaches a given contour curve. For instance, figure 7.1 illustrates such a sampling mechanism for a two-state system. The discretization of a given Lyapunov function in the energy space domain - that is the (x_1, x_2) plane in this case - defines a set of ellipses of constant energy. Control jobs are only activated each time the trajectory intersects a contour curve from outside to inside. The system trajectory can then move between them without requiring control actions. Therefore, by construction, the generated samples are stable in the Lyapunov sense - see subsection 7.2.1 for further details - since the system energy decreases event after event. This triggering mechanism - called *Lyapunov sampling* - is more detailed in subsection 7.2.2.1.

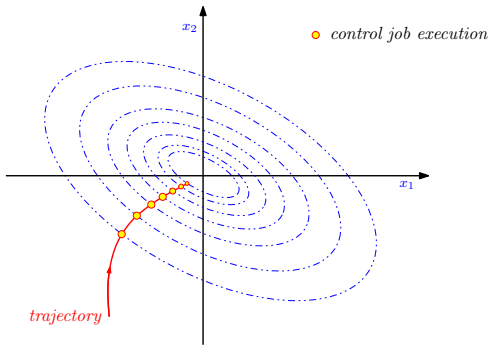


Figure 7.1: Lyapunov sampling mechanism: principle for a two-state system.

However, in order to ensure that the system trajectory will tend to the equilibrium point as time tends to infinity, it must be ensured that the sequence of samples is infinite. Although the generated sequence of samples is stable in the Lyapunov sense - decreasing the energy at each sample - the stability of the continuous dynamics is not guaranteed. That is, from the sequence of samples, it cannot be ensured that the system trajectory will tend to zero as time progresses, because the sequence of samples can be finite. Ensuring an infinite sequence of samples hence implies that all sampling intervals are bounded. This is explained in subsection 7.2.2.2. In

addition, more efficient techniques for the design of the controller are also proposed in [68] when the sampling intervals can be predicted, but this aspect will not be treated here. Eventually, a small improvement is proposed in subsection 7.2.2.3.

7.2.2.1 Lyapunov sampling

Let V be a Lyapunov function (see subsection 7.2.1 for further details). Evaluating the closed-loop system - equation (7.9) - in the discrete sampling instants - defined in equation (7.7) - the Lyapunov sampling triggering mechanism is enforced when

$$V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big) \tag{7.13}$$

where t_{a-1} and t_a are two consecutive Lyapunov sampling instants and $h(t_a) = t_a - t_{a-1}$ is the sampling interval during which the control signal is constant, i.e. equal to $u(t) = \psi\big(x(t_a)\big)$, as explained in subsection 7.1.3. On the other hand, the *energy gain factor* η is a tunable parameter used for event detection: for some small values of η, large sampling periods are expected, whereas large values will reduce the amount of time between two events. In fact, small values of η mean that the next sampling instant is set when the Lyapunov function has decreased more significantly with respect to the current value. Moreover, by construction, the sampling scheme is stable in the discrete Lyapunov sense if η is restricted to $0 < \eta < 1$. Note that no restriction is set on the controller law $\psi(\cdot)$ - previously defined in subsection 7.1.3 - to establish this Lyapunov sampling mechanism.

Actually, the depicted principle is not so far from the one detailed in the previous chapter for event-based PID control strategies using a level-crossing detection as sampling mechanism (one could refer to section 6.2 for more information). Indeed, as previously the basic setup consists in two parts: ***i)*** *a time-triggered event detector* which runs with the constant sampling period h_{nom} and sends some requests when a new control signal is required and ***ii)*** *an event-triggered controller* which calculates the control signal with varying sampling intervals $h(\cdot)$ defined by two successive requests. This architecture is represented in figure 7.2 (note that AD and DA are respectively analog-to-digital and digital-to-analog converters). The difference is that the *event logic* - in the time-triggered event detector - now leads to calculate a given Lyapunov function, in order to decide when to send a request to the control part. The input signal is hence the states $x(t)$ instead of the error $e(t)$ which was previously needed for the level detection. Moreover, the *control logic* - in the event-triggered controller - is now a state-feedback control law.

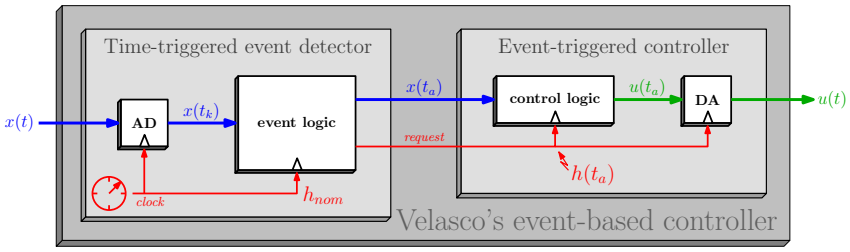


Figure 7.2: Architecture of the event-based controller proposed by Velasco et al.

One has to remember that the idea in this chapter is to analyze the stability of event-based architectures in order to validate our previous work where intuitive event-based techniques were developed. Thus, the controller applies here a state-feedback law which can easily be transposed to the PID control case done in chapter 6.

Eventually, the resulting algorithm is represented in figure 7.3.

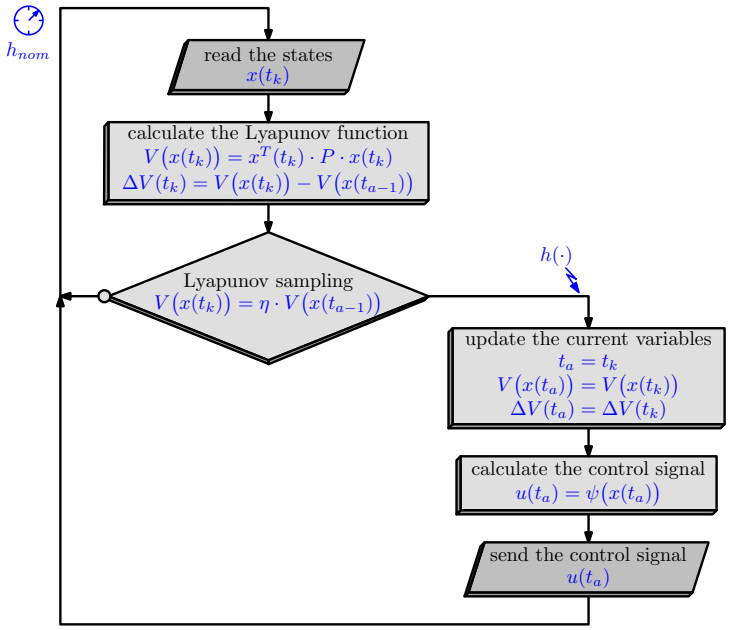
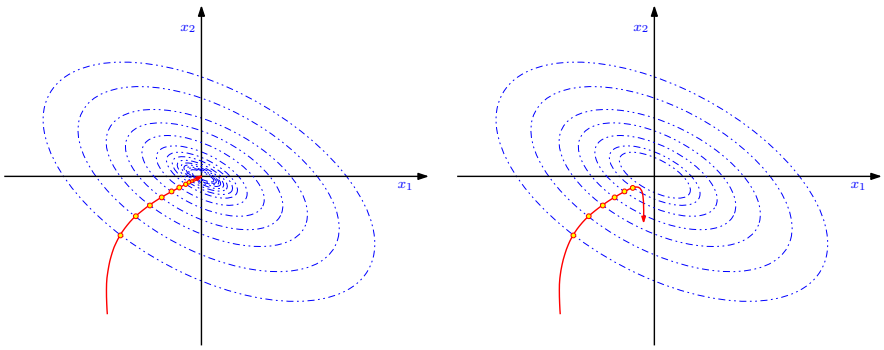


Figure 7.3: Algorithm: the event-based state-feedback controller using the Lyapunov sampling mechanism.

7.2.2.2 Stable Lyapunov sampling

Although the Lyapunov sampling ensures stable sampling sequences in the Lyapunov sense - detailed in the previous subsection - nothing is ensured about the stability of the continuous-time dynamics. Indeed, if the system energy increases before the next sampling instant, the condition from equation (7.13) will not be validated and, therefore, an event will not occur. Two Lyapunov sampling triggering cases are possible. If η is correctly chosen, the system trajectory will cross contour curves again and again until achieving the equilibrium point, as shown in figure 7.4(a). On the other hand, with a bad η value, the system energy could increase before achieving the next ellipse. In this case, a new event will never occur and the system becomes unstable, as drawn in figure 7.4(b). As a result, the sampling mechanism has to generate an infinite sequence of samples in order to ensure the stability of the continuous-time dynamics.



(a) Stable Lyapunov sampling (b) Unstable Lyapunov sampling

Figure 7.4: Lyapunov sampling mechanism: representation of a stable and an unstable behavior.

As a result, one has to determine when the energy decrease, produced by the system trajectory up to the point it starts, is gaining energy again. Placing a contour curve passing for that

point would ensure a new sample. Therefore, for any current state, one has to determine a limit on the energy decrease produced by the system trajectory, that still ensures the occurrence of the next sample. Intuitively, this strongly relates to the energy gain factor η. As a result, the authors propose to derive further restrictions for this parameter in such a way that the generated sequence of samples is infinite. The idea is summarized in the following.

Let $x(t, x_0)$ be the solution of the closed-loop system - equation (7.9) - when $u(t) = \psi(x_0)$, where x_0 is a given initial condition. The minimum achieved energy without changing the control signal for any initial condition is

$$V^*(x_0) = \min_t V\big(x(t, x_0)\big) \quad \forall t \geq 0$$

That is assumed that this minimum exists. Taking into account the initial energy $V(x_0)$ and the minimum achieved energy $V^*(x_0)$ for any initial condition, the minimum distance between them is

$$\min_{x_0}\Big(V(x_0) - V^*(x_0)\Big) = \min_{x_0}\left(1 - \frac{V^*(x_0)}{V(x_0)}\right)$$

which is always a positive quantity by construction. This minimum will occur when

$$\hat{\eta}(x_0) = \frac{V^*(x_0)}{V(x_0)} \tag{7.14}$$

is maximum. Let

$$\eta^* = \max_{x_0} \hat{\eta}(x_0)$$

be the value that minimizes the minimum distance. Note that by construction $0 \leq \hat{\eta} \leq 1$ and so is the *minimum energy gain factor* η^*. Finally, for any x_0, if $\eta^* < \eta < 1$ then the generated sequence is an infinite sequence. This condition presents the stable Lyapunov sampling mechanism, which is based on restricting η in the Lyapunov sampling condition in equation (7.13), in such a way that the space discretization given by the set of contour curves ensures some infinite samples.

Computing η^* is not trivial because this is a non-convex problem. Therefore, the corresponding algorithm needs to be executed off-line for each system to control. Indeed, $\hat{\eta}(x_0)$ has to be computed for any initial condition x_0 in order to be able to find the maximal one and hence deduce η^*. The resulting algorithm is computationally heavy and will probably take a long time, even if some simplifications can be found for linear systems, as done in [68]. It is represented in figure 7.5 as a dark block to highlight its off-line running. Anyway, an easier and more dynamical solution is proposed in subsection 7.2.3 to avoid this huge computation.

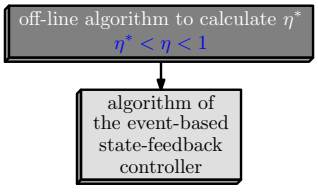


Figure 7.5: Algorithm: the computationally heavy off-line block required in the stable Lyapunov sampling mechanism.

7.2.2.3 Event-detection improvement

As explained in subsection 7.2.2.1, the Lyapunov sampling mechanism - initially depicted by Velasco et al. in [68] - allows to activate some control jobs only when the trajectory intersects a contour curve from outside to inside. Then, a restriction is added in subsection 7.2.2.2 to ensure that contour curves are crossed again and again and so is decreased the system energy. However, if the next curve could not be achieved for some reasons (if a perturbation occurs for instance) the system trajectory will diverge anyway - going from inside to outside - without (almost) any chance to cross again that contour curve. For this reason, **we propose to enforce a job execution when the system energy increases**, that is when

$$\begin{aligned} &\Delta V(t_a) > 0 \\ &\text{with} \quad \Delta V(t_a) = V\big(x(t_a)\big) - V\big(x(t_{a-1})\big) \end{aligned} \tag{7.15}$$

This leads to add a safety condition in the event-detection scheme. The resulting algorithm improvement is given in figure 7.6. This can be applied on both the Lyapunov and the stable Lyapunov sampling mechanisms.

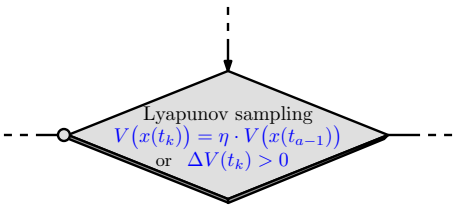


Figure 7.6: Algorithm: the extra condition when using the event-detection improvement (required in Lyapunov and the stable Lyapunov sampling mechanism).

7.2.3 A less-conservative stable Lyapunov sampling

The idea here is to soften the sampling scheme introduced by Velasco et al. Actually, the stable Lyapunov sampling mechanism is based on restricting η in the Lyapunov sampling condition $V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big)$ - as explained in subsection 7.2.2.2 - in such a way that the energy decreases again and again at each sample. A given algorithm thus allows to find the minimum energy gain factor η^* in order to ensure an infinite sequence, that is when

$$\eta^* < \eta < 1 \tag{7.16}$$

This minimum energy gain factor completely depends on the system to control and the chosen Lyapunov-candidate function. Moreover, its value is applied to the whole running time. This is highly conservative. Indeed, a system with a small value will lead to large sampling intervals whereas another system with an important η^* will reduce the amount of time between two events. This value can be large insomuch - very close to 1 - as some events will occur (quasi)-continuously. For this reason, **we propose to relax the constraint on the energy gain factor** η given in equation (7.16). Furthermore, the value which lower-bounds η is the maximal value calculated among all initial conditions (see subsection 7.2.2.2 for further details). This is why it can be assumed that the system is stable in the Lyapunov sense for a large number of values of η smaller than η^*. One has just to keep in mind that the system could become unstable. As a result, **we also propose to make the energy gain factor dynamically varying**, in order to reduce again the number of samples when the system is stable and quickly react when it becomes unstable. This parameter - afterwards denoted the *varying energy gain*

factor $\eta(t)$ - is updated in the event-driven control logic. Then,it is substituted in the Lyapunov event-detection mechanism, which becomes

$$V\big(x(t_a)\big) = \eta(t_{a-1}) \cdot V\big(x(t_{a-1})\big)$$

where $\eta(\cdot)$ is still restricted to $0 < \eta(\cdot) < 1$, by construction, to ensure Lyapunov stability. Several algorithms - which are more and more relaxing - are eventually proposed in the following subsections.

7.2.3.1 Relaxation 1: slowly decrease/drastically increase

The first guess is to slowly decrease η when the energy decreases - as the system is stable the constraint on the energy gain factor can be relaxed in order to update less often the control signal - and changing it back to η^* as soon as the energy begins to increase in order to stabilize the system again. The resulting algorithm is

$$\eta(t_a) = \begin{cases} \eta^* + \varepsilon & \text{if} \quad \Delta V(t_a) > 0 \\ (1-\nu) \cdot \eta(t_{a-1}) & \text{otherwise} \end{cases}$$

where $\varepsilon \in \mathbb{R}^+$ guarantees the left inequality in equation (7.16), while $0 < \nu < 1$ leads to decrease the value of the energy gain factor. Note that $\Delta V(t_a)$ was defined in equation (7.15). Of course, one has to take care about decreasing too much $\eta(\cdot)$ since $0 < \eta(t) < 1$ is still required by construction. A saturation function is needed.

Nevertheless, this first relaxation still runs with respect to η^*, that can only be calculated with the off-line algorithm introduced in subsection 7.2.2.2. Indeed, as already explained in introduction, the stable Lyapunov sampling mechanism requires to find the minimum energy gain factor η^*, which is obtained calculating $\hat{\eta}(x_0)$ for any initial condition x_0. This last parameter is defined in equation (7.14) as the ratio between the minimum achieved energy without changing the control signal and the initial energy and yet, the minimum achieved energy requires to compute the Lyapunov function from the initial condition until it is gaining energy again. This is a non-convex problem which requires to ***i)*** know the model of the system to control in order to be able to calculate $x(t, x_0)$ and $V\big(x(t, x_0)\big)$, ***ii)*** calculate the control signal $u(t) = \psi(x_0)$ given by equation (7.8), ***iii)*** close the system and run it until the Lyapunov function increases. Finally, this has to be done for all possible initial conditions x_0. The resulting algorithm hence requires an important off-line computation while a softer solution would be preferred. For this reason, **we propose a fully on-line Lyapunov sampling mechanism with low computational cost** in order to have a behavior similar than before but with a lightened algorithm. Consequently, the next relaxations can be applied to control a system without requiring to execute the off-line algorithm before.

7.2.3.2 Relaxation 2: improvement for an on-line running

A simple modification of the previous relaxation is imagined to not use the off-line algorithm anymore. Thus, η^* is replaced by the upper-bound of the energy gain factor defined in equation (7.16). This yields

$$\eta(t_a) = \begin{cases} 1 - \varepsilon & \text{if} \quad \Delta V(t_a) > 0 \\ (1-\nu) \cdot \eta(t_{a-1}) & \text{otherwise} \end{cases}$$

where $0 < \nu < 1$ and $\varepsilon \in \mathbb{R}^+$ were already defined. In fact, when $\eta(t) = 1 - \varepsilon$ the system almost runs with the time-triggered behavior, which means sampling at each periodic discrete instant.

However, by doing that the system trajectory tends to the equilibrium point - if the control law is well-established - and the energy decreases. Consequently, the energy gain factor will decrease and so begins the asynchronous mechanism.

7.2.3.3 Relaxation 3: slowly decrease/slowly increase

Based on the two previous relaxations, another idea is to slowly increase the energy gain factor when the energy increases - as we slowly decrease it when the energy decreases - hoping that the system could be stabilized before achieving the value of η^*. Indeed, as explained in introduction, we assume that the system is stable in the Lyapunov sense for several values of η smaller than η^*. This leads to

$$\eta(t_a) = \begin{cases} \big(1+\overline{\nu}\big)\cdot\eta(t_{a-1}) & \text{if} \quad \Delta V(t_a) > 0 \\ \big(1-\underline{\nu}\big)\cdot\eta(t_{a-1}) & \text{otherwise} \end{cases}$$

where $0 < \overline{\nu} < 1$ and $0 < \underline{\nu} < 1$ could eventually be equal. Note that, as previously, a saturation function is still required to ensure $0 < \eta(t) < 1$.

7.2.3.4 Relaxation 4: a more formal variation

The three first relaxations were intuitively built and a more formal expression would be preferred. Actually, the energy gain factor $\eta(t)$ varies and, as a result, can be written as a state of the controller, i.e. $\dot{\eta}(t)$. Moreover, the variable is function of the variation of the Lyapunov function: when the energy decreases the energy gain factor can be reduced whereas it has to be increased when the energy grows. Therefore, the derivative of the energy, i.e. $\dot{V}\big(x(t)\big)$, can be used in the algorithm. This results in

$$\dot{\eta}(t) = \upsilon \cdot \dot{V}\big(x(t)\big) \cdot \eta(t)$$

where $\upsilon \in \mathbb{R}^+$ is a tunable parameter. Note that it could be computationally complex to calculate the analytical expression of $\dot{V}(\cdot)$ if the number of system state variables is important. For this reason, an alternative is proposed. One can see in the previous proposals that, in fact, the sign of $\eta(t)$ - and therefore its variation - is function of $\Delta V(\cdot)$ and yet, this variable is already calculated in the Lyapunov sampling mechanism improvement (introduced in subsection 7.2.2.3). Consequently, that can be easily used in the algorithm, which eventually yields

$$\dot{\eta}(t) = \upsilon \cdot \Delta V(t) \cdot \eta(t)$$

and discretizing this equation (using the backward difference approximation for the same reasons than explained in subsection 6.2.2) finally leads to

$$\eta(t_a) = \Big(1 + \upsilon \cdot h(t_a) \cdot \Delta V(t_a)\Big) \cdot \eta(t_{a-1})$$

The interest of this expression is that the energy gain factor varies with respect to $\Delta V(\cdot)$ but also with the sampling interval $h(\cdot)$. That is, if the system becomes unstable after a large steady-state interval, the energy gain factor will largely react in consequence. Thus, starting with a given initial condition will lead to achieve the expected behavior in dynamically adjusting the energy gain factor.

Eventually, the resulting on-line algorithm is depicted in figure 7.7. The energy gain factor updating, i.e. $\eta(t_a) = \chi\big(\eta(t_{a-1})\big)$, depends on the relaxation among the above proposals. Note that the off-line algorithm - presented in subsection 7.2.2.2 and shown in figure 7.5 - still has to be executed in the case of the first relaxation. However, relaxation 2 is an alternative to avoid that and, as a result, one can say that our proposals are on-line executed.

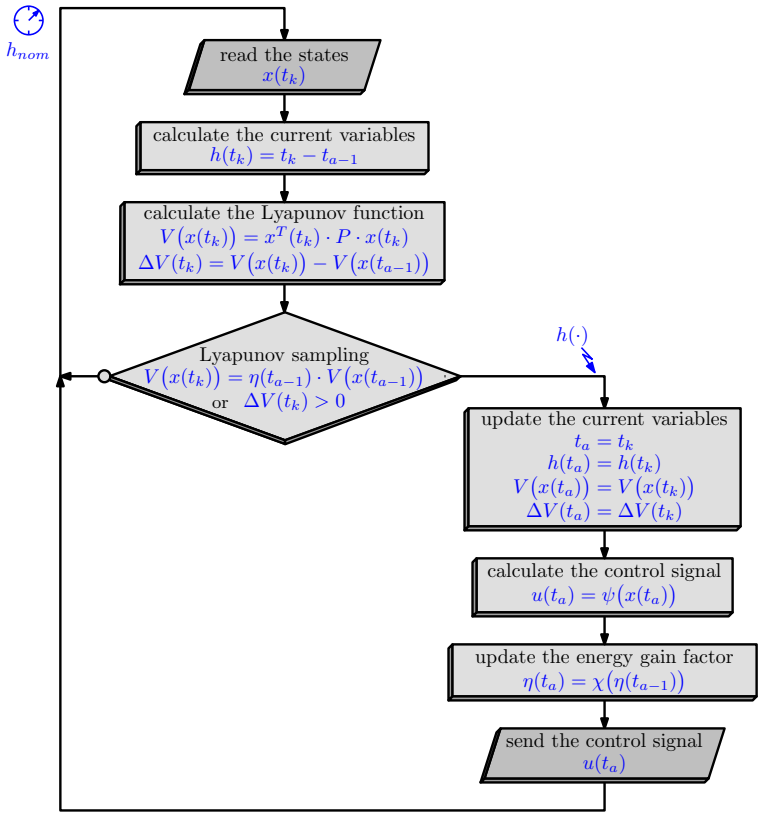


Figure 7.7: Algorithm: the event-based state-feedback controller using a less-conservative stable Lyapunov sampling mechanism.

7.2.4 Minimum sampling interval condition

The less-conservative Lyapunov sampling mechanisms depicted in the previous subsection are interesting but we would like to reduce even more the number of control updates. Consequently, **we propose to extend the minimum sampling interval condition to the Lyapunov sampling scheme**. The principle is detailed in subsection 6.2.4. It was initially developed for some PID control strategies using an event detection based on level crossing. The idea is to calculate the control signal only if a given amount of time was elapsed since the last sampling instant, that is when

$$h(t_a) \geq h_{min}$$

This setup can be directly applied in the present case. Moreover, instead to verify both event conditions in the same time, i.e. the Lyapunov sampling condition plus the minimum sampling one, **we propose to decompose the event detection in two steps** in order to perform the Lyapunov function only if the previous condition was verified. The gain in using such an approach is double. Indeed, the final number of samples will be reduced and, moreover, the computation of $V(t)$ will only be done when the condition is satisfied. Finally, this can highly decrease the computational cost of the event detection, more especially if the Lyapunov function is complex or the number of internal states is important. The resulting algorithm is represented in figure 7.8.

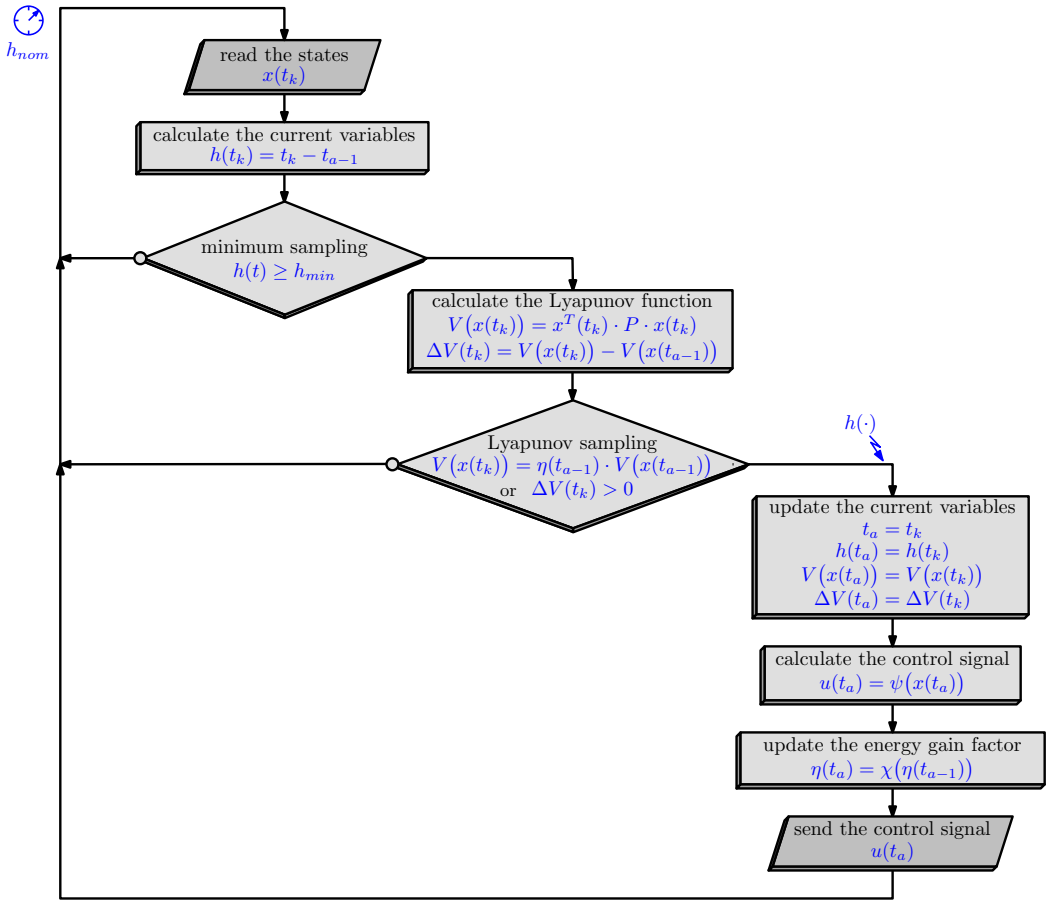


Figure 7.8: Algorithm: the event-based state-feedback controller using a less-conservative stable Lyapunov sampling mechanism with minimum sampling condition.

7.3 Recap of the different Lyapunov sampling mechanisms

This section aims at summarizing the different asynchronous control strategies based on a Lyapunov sampling mechanism. The basic setup - depicted in introduction of subsection 7.2.2 - remains the same, only the event-detection condition varies. Therefore, **we propose to recap all the strategies based on Lyapunov sampling**. This list will then be useful for the simulation/experimental results (in section 7.5 and chapter 8 respectively).

Classical time-triggered control strategy

The time-triggered state-feedback controller is sampled at every periodic discrete-time instants $t_k = k \cdot h_{nom}$. The control updates are given by

$$u\big(t_k, x(t_k)\big) = -K \cdot x(t_k)$$

This will not be impacted in the asynchronous strategies, only the instant times change in the previous relation, from t_k to t_a.

Lyapunov sampling mechanism

The asynchronous controllers using such an event-driven mechanism are sampled with some not-equidistant sampling periods $h(t_a) = t_a - t_{a-1}$, where t_{a-1} and t_a are two consecutive sampling

instants. Some events are enforced when

$$V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big) \qquad \text{(Initial)}$$
$$V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big) \quad \text{or} \quad \Delta V(t_a) > 0 \qquad \text{(Improved)}$$

where $\Delta V(t_a) = V\big(x(t_a)\big) - V\big(x(t_{a-1})\big)$ and η defines the detection level. Note that the initial strategy is the one presented by Velasco et al. in [68], while the improved strategy consists in enforcing a job execution also when the system energy increases in order to avoid unstable behavior.

The **stable Lyapunov sampling mechanism** consists in calculating η^* in order to restrict the energy gain factor $\eta^* < \eta < 1$. This is done in a computationally heavy off-line algorithm.

Less-conservative stable Lyapunov sampling

Several relaxations of the original stable Lyapunov sampling were proposed, making η dynamically varying (four relaxations):

- ***Relax 1:*** *s*lowly decrease/drastically increase $\eta(t)$,
- ***Relax 2:*** *i*mprovement for an on-line running,
- ***Relax 3:*** *s*lowly decrease/slowly increase $\eta(t)$,
- ***Relax 4:*** *a* more formal variation of $\eta(t)$.

and the way to make $\eta(t)$ varying is different for each relaxation, that is

$$\eta(t_a) = \begin{cases} \eta^* + \varepsilon & \text{if} \quad \Delta V(t_a) > 0 \\ \big(1-\nu\big) \cdot \eta(t_{a-1}) & \text{otherwise} \end{cases} \qquad \text{(Relax 1)}$$

$$\eta(t_a) = \begin{cases} 1 - \varepsilon & \text{if} \quad \Delta V(t_a) > 0 \\ \big(1-\nu\big) \cdot \eta(t_{a-1}) & \text{otherwise} \end{cases} \qquad \text{(Relax 2)}$$

$$\eta(t_a) = \begin{cases} \big(1+\overline{\nu}\big) \cdot \eta(t_{a-1}) & \text{if} \quad \Delta V(t_a) > 0 \\ \big(1-\underline{\nu}\big) \cdot \eta(t_{a-1}) & \text{otherwise} \end{cases} \qquad \text{(Relax 3)}$$

$$\eta(t_a) = \Big(1 + \upsilon \cdot h(t_a) \cdot \Delta V(t_a)\Big) \cdot \eta(t_{a-1}) \qquad \text{(Relax 4)}$$

The event-detection condition is the same for all these algorithms. This is

$$V\big(x(t_a)\big) = \eta(t_{a-1}) \cdot V\big(x(t_{a-1})\big) \quad \text{or} \quad \Delta V(t_a) > 0$$

Minimum sampling interval condition

The minimum sampling interval condition consists in adding an extra condition to reduce the number of samples again. The resulting event-detection condition is

$$\Big(V\big(x(t_a)\big) = \eta(t_{a-1}) \cdot V\big(x(t_{a-1})\big) \quad \text{or} \quad \Delta V(t_a) > 0\Big) \quad \text{and} \quad h(t_a) \geq h_{min}$$

where the first part is only verified when $h(t_a) \geq h_{min}$ is satisfied, in order to reduce the computational cost in calculating the Lyapunov function (see subsection 7.2.4 for further details).

7.4 Performance analysis

In this section, the performance indexes introduced in section 6.5 are recalled since they will be applied again to compare the different proposals.

- The number of *calls*. This index refers to the number of samples required to control the system during the whole simulation time.
- The integral absolute error:

$$IAE = \int_0^\infty |e(t)|dt$$

  This index shows how far is the system response compared with a given setpoint. The smallest value will highlight the strategy which best fits the system with the reference.

- The integrated absolute difference between the system response of the time-based strategy and that of the event-based ones:

$$IAEP = \int_0^\infty |y_{time-based}(t) - y_{event-based}(t)|dt$$

  This index allows to compare the system response of the event-based controllers with the time-triggered one.

- The integral absolute difference between the IAE of the time-based strategy and the IAE of the event-based ones:

$$IAD = \int_0^\infty |IAE_{time-based}(t) - IAE_{event-based}(t)|dt$$

  This index calculates the error between the system response and the setpoint for each strategy, in order to finally compare them for both the time-triggered and the event-based ones.

7.5 Simulation results: application to a double integrator system

In this section, **we propose to use the same system for simulation results than the one depicted in the original work of Velasco et al**. This is the double integrator system whose state-space representation is $\dot{x} = A \cdot x + B \cdot u$, where

$$A = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \quad \text{and} \quad B = \begin{bmatrix} 0 \\ 1 \end{bmatrix}.$$

The initial condition is

$$x_0 = \begin{bmatrix} 0 \\ -3 \end{bmatrix}.$$

The quadratic Lyapunov function is $V(x) = x^T \cdot P \cdot x$, where

$$P = \begin{bmatrix} 1.1455 & 0.1 \\ 0.1 & 0.0545 \end{bmatrix},$$

and the control updates are given by the linear state feedback $u(x) = -K \cdot x$, where

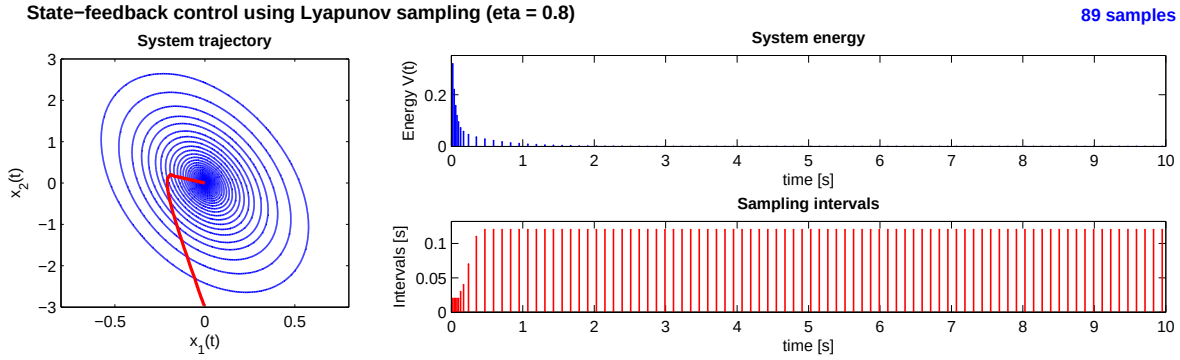
$$K = \begin{bmatrix} 10 & 11 \end{bmatrix}.$$

One could easily verify that the closed-loop system $\dot{x} = \big(A - B \cdot K\big) \cdot x = \mathbf{A} \cdot x$ satisfy the condition $\mathbf{A}^T \cdot P + P \cdot \mathbf{A} < 0$ defined in subsection 7.2.1. This means that the system is globally asymptotically stable.

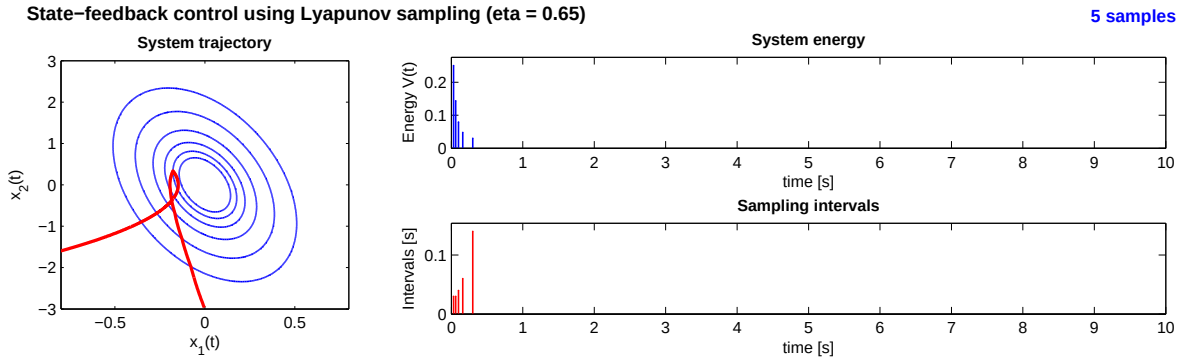
The bench used to test the different asynchronous control strategies is a 10 s simulation done with *Matlab/Simulink*. The results follow.

Lyapunov sampling

The first event-based state-feedback control strategy presented in the previous section is the one introduced by Velasco et al. The principle was summarized in subsection 7.2.2.1. The sampling mechanism is based on a Lyapunov function and consists in enforcing a control job when the system trajectory crosses a given level, i.e. $V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big)$. This Lyapunov sampling mechanism is simulated for two different values of the energy gain factor η. The results are shown in figure 7.9. The left plot shows the system trajectory in the energy space domain - that is the (x_1, x_2) plane - while the right side represents the system energy - the Lyapunov function $V(t)$ - in the top plot and the evolution of the sampling intervals with respect to time in the bottom one. The total number of samples is also indicated.



(a) Stable Lyapunov sampling with $\eta = 0.8$



(b) Unstable Lyapunov sampling with $\eta = 0.65$

Figure 7.9: Simulation results of the original asynchronous state-feedback controller based on Lyapunov sampling for two different values of the energy gain factor.

- In the first case $\eta = 0.8$. The system is stable in the Lyapunov sense - the energy function decreases - and the dynamics of the continuous system is stable too - the system trajectory tends to the origin - as depicted in figure 7.9(a). Moreover, the performance is quite similar to the classical approach, with a real reduction of the number of samples. Actually, less than 10 % of sampling instants are relevant compared with the classical scheme (with the periodic sampling period $h_{nom} = 0.01\,s$). Only 89 samples allow to control the double integrator system with the Lyapunov sampling (in the present simulation text bench). However, one can remark that this event-triggered mechanism seems to tend to a periodic sampling when the trajectory is very close to the equilibrium. This effect will disappear with our less-conservative proposal.

- When decreasing too much the energy gain factor, as for instance when $\eta = 0.65$, the system becomes unstable, as one can see in figure 7.9(b). Indeed, the system diverges after a certain amount of time because the next level is never achieved (due to a bad value of the energy gain factor). Thus, the triggering condition $V\big(x(t_a)\big) = \eta \cdot V\big(x(t_{a-1})\big)$ is never satisfied and, as a result, the control signal is not updated anymore.

A zoom on the sampling instants of the Lyapunov function is performed in figure 7.10 for the two previous value of η, in order to highlight the problem when this parameter is badly chosen.

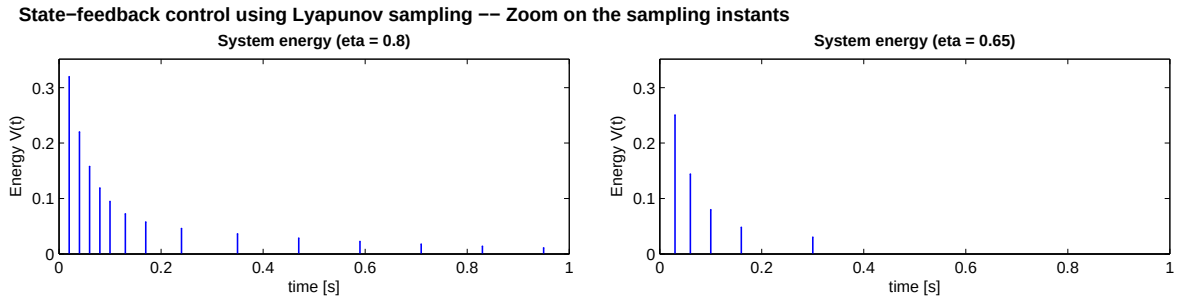


Figure 7.10: Simulation results: zoom on the control instants of a stable and an unstable Lyapunov sampling mechanism.

Fortunately, the stable Lyapunov sampling mechanism allows to avoid such a problem in restricting the energy gain factor. Nevertheless, a simple solution consists in calculating the control signal as soon as the system energy increases, i.e. when $\Delta V(t_a) > 0$ as defined in subsection 7.2.2.3. This leads to be reactive when an unstable behavior occurs. The corresponding simulation results, applying this improvement, are shown in figure 7.11. The sampling intervals vary more chaotically, but the system trajectory finally goes to the origin. Furthermore, the number of samples is reduced compared to the stable case, since the value of η is smaller (and so are larger the sampling intervals). This solution is interesting for some systems which have to track a given setpoint anyway, since the Lyapunov function will inevitably increase at each setpoint variation. For this reason, the event detection improvement will be applied in the next strategies.

Computation of the minimum energy gain factor for a stable Lyapunov sampling

As previously explained, a stable Lyapunov sampling mechanism is required to ensure that the system energy always decreases, in order to achieve the next level. This issue is related to η which has to be conditioned in such a way that the generated sequence of samples is infinite and, consequently, a complex algorithm is required to calculate the minimum energy gain factor. As

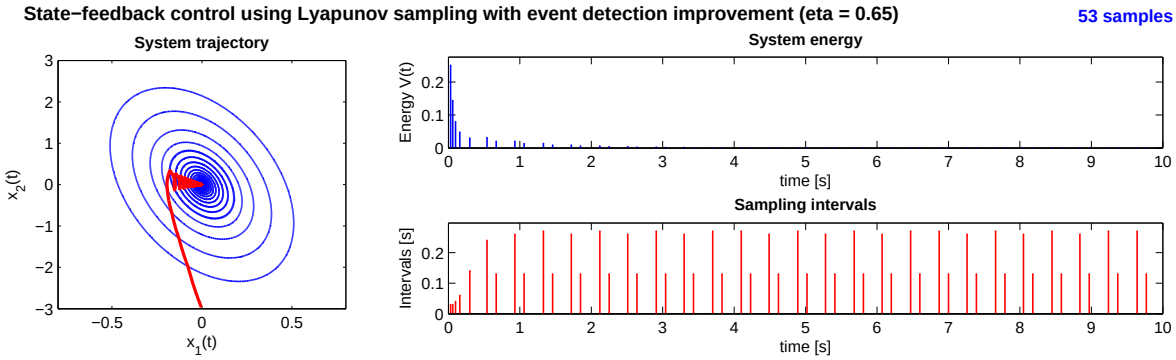


Figure 7.11: Simulation results of the asynchronous state-feedback controller based on Lyapunov sampling with event-detection improvement (with $\eta = 0.65$).

detailed in subsection 7.2.2.2, it consists in calculating the minimum achieved energy without changing the control signal for any initial condition x_0, and so obtain a certain value $\hat{\eta}(x_0)$ for each x_0. Some simplifications are possible for linear systems and, as a result, the energy gain factors $\hat{\eta}(\cdot)$ have only to be computed on the interval $\theta \in [0, \pi)$. This is demonstrated in [68]. Eventually, η^* results in the maximum energy gain factor. Applying this principle to our study case yields $\eta^* = 0.7818$ - as highlighted in figure 7.12 - and finally, restricting η such that $\eta^* < \eta < 1$ ensures that the generated sequence is infinite. One could verify, and besides, that both stable and unstable behaviors illustrated in figure 7.9 verify this condition since $0.65 < \eta^* < 0.8$ (the system is unstable using the first value and stable in the second case).

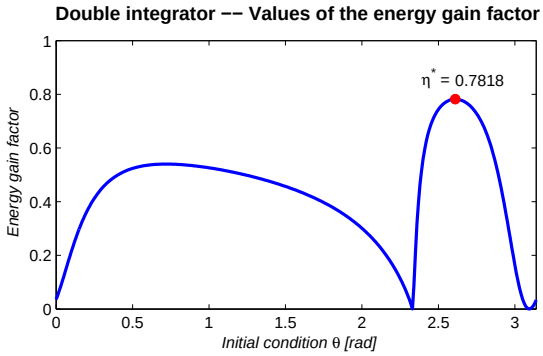


Figure 7.12: Simulation results: possible values of $\hat{\eta}(\cdot)$ calculated with the off-line algorithm (needed in the stable Lyapunov sampling mechanism) for the double integrator's control case in order to obtain η^*.

The so-proposed algorithm allows to calculate η^* but has to be executed off line since it takes a long amount of time to have a result (from few seconds to dozens of seconds). Actually, this solution is computationally heavy and, moreover, it could be more difficult to obtain the minimum energy gain factor in the case of a nonlinear system. For this reason, we developed some new strategies which are less conservative but still yield the system to tend to its equilibrium.

Less-conservative stable Lyapunov sampling

These strategies are based on the idea that the constraint on the energy gain factor η can be relaxed, making it dynamically varying, because the system is in fact stable in the Lyapunov sense for a large number of values of η smaller than η^*. Indeed, one just needs to look at

figure 7.12 to see that the next level would be crossed in a large number of cases where $\eta < \eta^*$. For this reason, several relaxations of the original constraint on the energy gain factor were proposed in subsection 7.2.3. The corresponding simulation results can be seen in figure 7.13, where the evolution of $\eta(t)$ is represented on the bottom right plot.

1. *Slowly decrease/drastically increase $\eta(t)$:*
   The first proposal simply consists in decreasing η more than suggested in $\eta^* < \eta < 1$. However, as soon as the system energy increases, the energy gain factor goes back to η^*. This intuitive guess leads to run with a value less than η^* during about 90 % of the simulation time, as one can see in figure 7.13(a). Furthermore, this reduces again the number of samples (almost 50 % less than with the original stable Lyapunov sampling) with still good performance.

2. *Improvement for an on-line running:*
   The previous relaxation is then modified to not use η^* anymore in the algorithm, in order to have a fully on-line algorithm. This parameter is replaced by $1 - \varepsilon$, which is the upper-bound of the energy gain factor by construction. The results are plotted in figure 7.13(b). These are quite close to the previous ones. The sampling intervals increase until the system becomes unstable in both cases. The main difference is that, in the on-line running, the number of samples is more important after the energy increases due to the fact that $\eta(t)$ starts from a larger value (because $\eta^* < 1 - \varepsilon$).

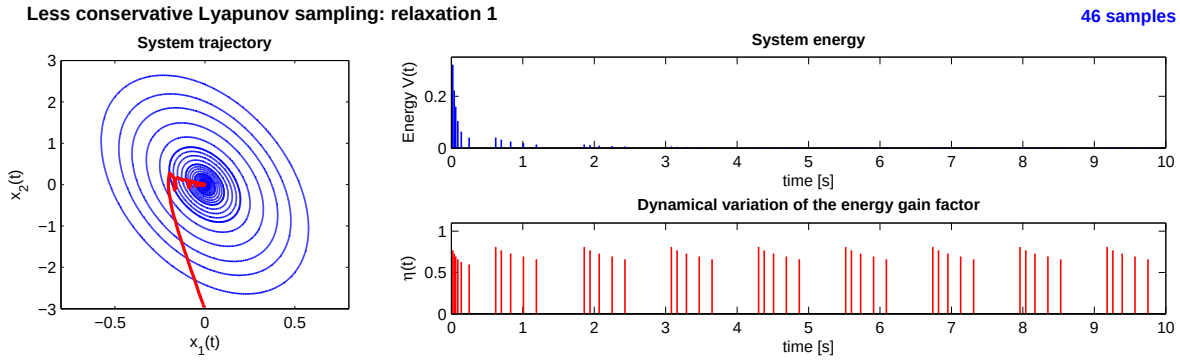
3. *Slowly decrease/slowly increase $\eta(t)$:*
   In the two previous proposals, the energy gain factor is slowly decreased when the system is stable, and drastically changed back to a value which ensures an infinite sequence when the system becomes unstable. Another solution is to slowly increase η when the system diverges. This is represented in figure 7.13(c), where one could see a lighter variation of the sampling intervals.

4. *A more formal variation of $\eta(t)$:*
   Eventually, we present a more formal expression using a state variable in the control algorithm. The energy gain factor thus now evolves with respect to the variation of the system energy - the Lyapunov function - and the current sampling interval. The simulation results of this last relaxation are drawn in figure 7.13(d), where the variation of the energy gain factor is toned down.

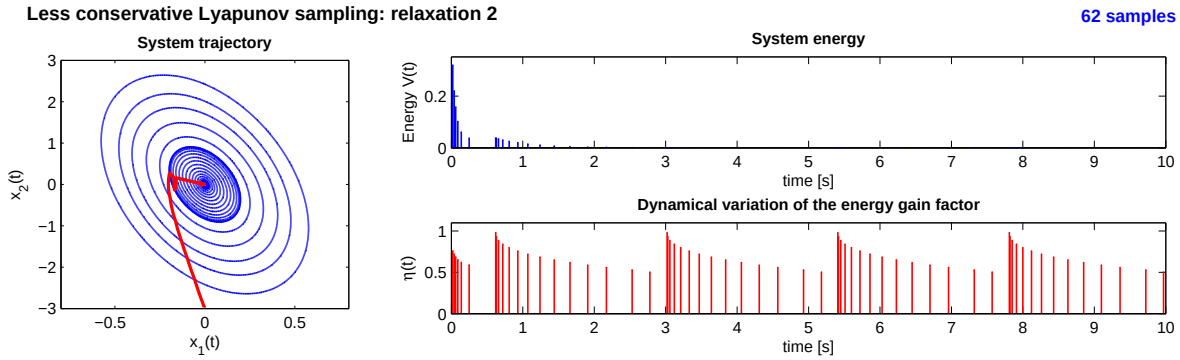
The value of the different parameters needed in the above algorithms are $\nu = \overline{\nu} = \underline{\nu} = 0.05$, $\varepsilon = 0.02$ and $\upsilon = 6.2$.

Performance analysis

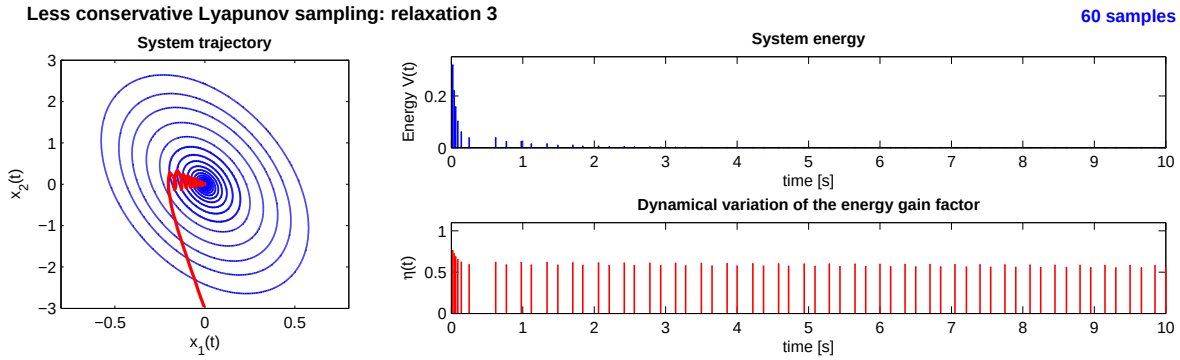
One has to keep in mind that, even if the system trajectory is not as "direct" as the classical time-triggered or the original stable Lyapunov sampling ones, our proposals take advantages on both techniques: ***i)*** the number of samples is highly reduced - 94 % of samples less than with the classical way and 50 % less than the Lyapunov sampling results - and ***ii)*** our strategies are executed in real-time and do not require a computationally heavy off-line algorithm. Furthermore, the last formal relaxation eventually allows a system response very close to the existing techniques, as depicted in figure 7.14, where it is compared with the classical time-based strategy and the original Lyapunov sampling mechanism. Eventually, the indexes of performance - called back in section 7.3 - are summarized in table 7.1 for all the strategies. One could remark that the high number of samples achieved with the original Lyapunov sampling mechanism pay with good performance indexes. However, the less-conservative proposals are also good and all



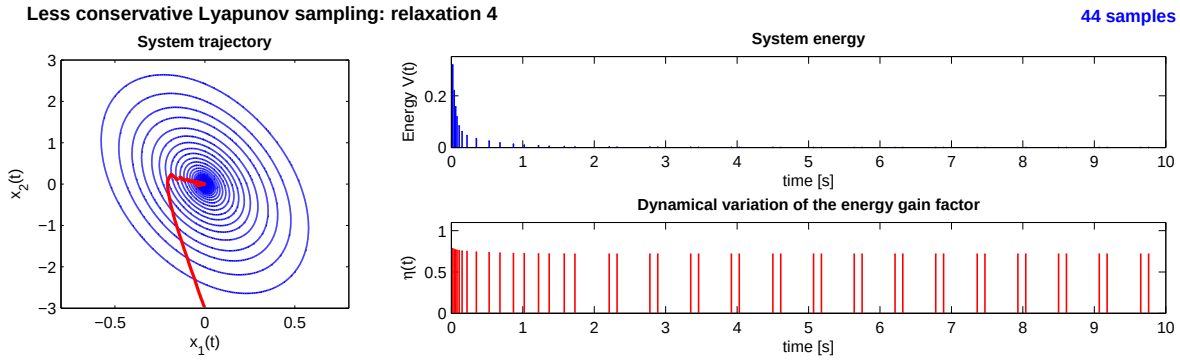
(a) Relaxation 1: slowly decrease/drastically increase $\eta(t)$



(b) Relaxation 2: improvement for an on-line running



(c) Relaxation 3: slowly decrease/slowly increase $\eta(t)$



(d) Relaxation 4: a more formal variation of $\eta(t)$

Figure 7.13: Simulation results of the asynchronous state-feedback controller using a less-conservative Lyapunov sampling mechanism: interest of the relaxations on the initial constraint $\eta^* < \eta < 1$.

the resulting values are low too (except for relaxation 1) while the number of control updates is quite small.

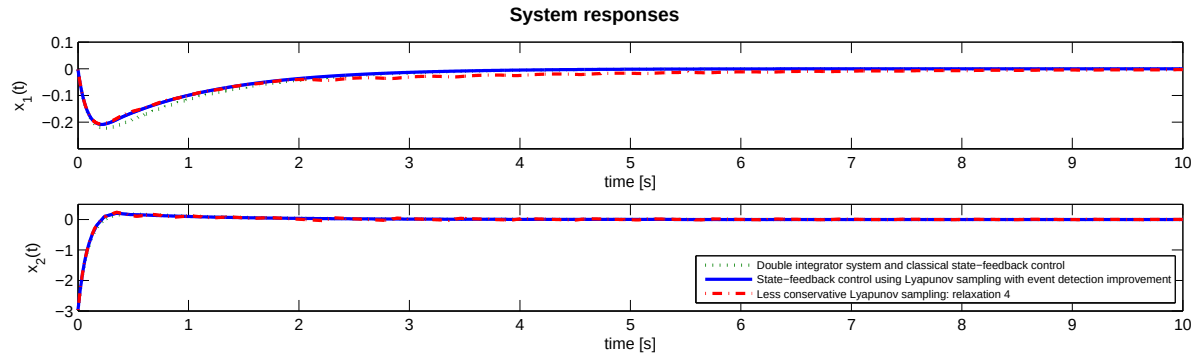


Figure 7.14: Simulation results: comparison of the system response of the classical sampling, the stable Lyapunov sampling and the less-conservative one (with relaxation 4).

Table 7.1: Performance analysis: comparison of the different asynchronous state-feedback strategies to control a double integrator system (looking at $y = x_1 + x_2$).

		Calls	IAE	IAEP	IAD
Time-based		1000	0.28	0	0
Lyapunov sampling	$\eta = 0.8$	89	**0.25**	**0.035**	**0.31**
	$\eta = 0.65$ (improved)	**53**	0.46	0.28	1.3
Less-conservative sampling	Relax 1	46	0.42	0.22	0.98
	Relax 2	62	**0.31**	**0.12**	**0.25**
	Relax 3	60	0.48	0.28	1.6
	Relax 4	**44**	0.37	0.16	0.5

Notes:

- *The nominal sampling period is* $h_{nom} = 0.01\,s$.
- *The value to guarantee that* $\eta < 1$ *is* $\varepsilon = 0.02$. *The tunable parameters in relaxation 1, 2 and 3 are* $\nu = \overline{\nu} = \underline{\nu} = 0.05$ *and the one in relaxation 4 is* $\upsilon = 6.2$.

7.6 Synthesis

This chapter firstly recalls the Lyapunov sampling mechanism initially developed in [68]. The depicted setup consists in calculating a new control signal only when the system trajectory reaches a given contour curve. Using some Lyapunov functions to decide the event detection is quite interesting since it allows to easily prove the stability of the system. However, a stable running is based on restricting the so-called energy gain factor η in the Lyapunov sampling condition, which defines the length of the level detection. The restriction is $\eta^* < \eta < 1$, where η^* is the minimum possible value to ensure a stable behavior, and yet, calculating this parameter requires to execute a computationally heavy off-line algorithm. As a result, the initial principle can only be applied to simple (linear) systems where some simplifications exist. We hence decided to develop some less-conservative methods. The main contributions are to relax the constraints on the energy gain factor and make this parameter varying. Four algorithms were thus suggested:

- **Slowly decrease/drastically increase** $\eta(t)$, where the energy gain factor is decreased more than suggested in the initial proposal and, as soon as the system energy increases, the energy gain factor goes back to η^*.
- **Improvement for an on-line running**, where the previous relaxation is modified in order to not use η^* anymore. This parameter is replaced by $1 - \varepsilon$ which is the upper-bound of the energy gain factor by construction.
- **Slowly decrease/slowly increase** $\eta(t)$ where the energy gain factor decreases when the system is stable, and increases when the system becomes unstable.
- **A more formal variation of** $\eta(t)$ where the energy gain factor becomes a state variable in the control algorithm.

The three latter proposals allow a fully on-line running, while the performance remains unchanged for a minimum of samples (when controlling a double integrator in simulation). Another contribution consists in extending the minimum sampling interval condition originally introduced for PID controllers to lighten the transients. At the end, the Lyapunov theory allows to prove that an asynchronous scheme can decrease the computational cost while ensuring the stability of the system, even if the system is not sampled during a long amount of time. Consequently, this also brings a stability proof for event-based PID controllers without safety limit conditions - detailed in chapter 6 - where no stability analysis was given.

Chapter

8 Experimental results

Two asynchronous control schemes were previously introduced in chapter 6 and 7, respectively based on a PID and a state-feedback strategy. Both principles are interesting for computational cost savings, since a significant power consumption reduction can be achieved by decreasing the number of samples and, consequently, the CPU utilization of the system. Furthermore, both setups showed in simulation that the system can be controlled as well (even better) as with a conventional time-triggered controller, with a control signal which is updated more than 80 % less often. The advantages of such an event-based control was demonstrated in simulation and some experimental results are now expected. Thus, a linear system is firstly tested in this chapter, controlling the velocity and the position of an electric motor. Then, an event-driven control is also applied on a hardly nonlinear system, with the inverted pendulum study case. The different systems are detailed in section 8.1. Some results are presented in sections 8.2 and 8.3 respectively. Eventually, all the experiments are synthesized in section 8.4.

8.1 Presentation of the system

A *pendulum* is, by definition, a weight suspended from a pivot which can freely swing. Respectively, an *inverted pendulum* is a pendulum whose mass is above its pivot point. As a result, whereas a normal pendulum is naturally stable, an inverted pendulum is inherently unstable and has to be actively balanced in order to remain upright and resistant to a disturbance. Two strategies exist to achieve the expected behavior: either applying a torque to the pivot point or moving the pivot point as part of a closed-loop feedback system. The second case is the aim of the present study. This problem - illustrated in figure 8.1 - involves ***i)*** *a cart* which is able to horizontally move and ***ii)*** *a pendulum* placed on the cart such that its two arms can freely move (in the same plane that the cart). The only way to balance the inverted pendulum then consists in applying an external control force to the system. This is done thanks to a DC (direct current) servo-motor which provides the control force to the cart through a belt drive system. A digital controller eventually allows to stabilize the pendulum in its inverted position, simply acting on the motor.

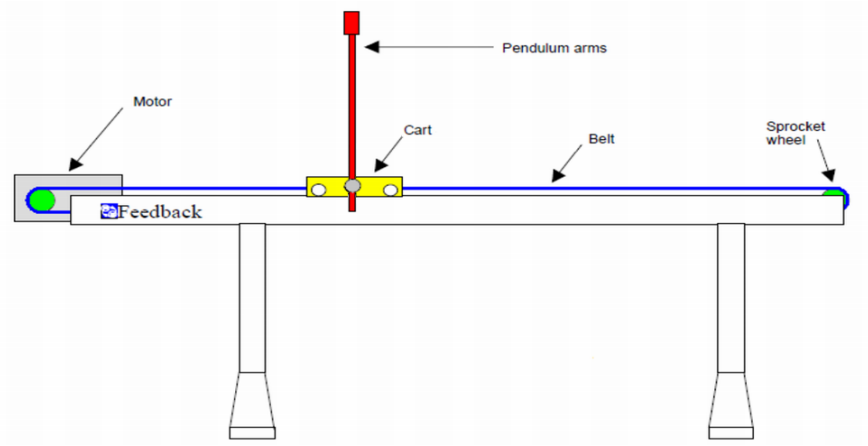


Figure 8.1: Representation of the inverted pendulum used for experimental results.

The present problem is actually difficult to control. Indeed, if the pendulum starts far from its upright position, it will begin to fall and the cart will start to move in the opposite direction. On the other hand, when the cart moves, it makes the arms becoming off center. Consequently, the cart and the pendulum's arms are coupled while only the cart is actuable. Nevertheless, it is possible to control this sub-actuated system. As explained before, it is pulled by a belt connected to an electric motor. A potentiometer measures the cart position, from its rotation, while another one measures the angle of the pendulum. Finally, the measurements are the position of the cart $p(t)$ and the angle of the pendulum $\theta(t)$. The velocity $\dot{p}(t)$ and the angular velocity $\dot{\theta}(t)$ can also be deduced, deriving the two first variables.

The goal of the control law is to move the cart to a given position without causing the pendulum to tip over. This was already done using a classical time-triggered method. See for instance [50, 62, 13] and the references therein. However, **we propose to control the inverted pendulum with an event-based strategy**. This is a challenge since this system is completely unstable and requires to update the control law very often (the pendulum will fall down as soon as the control is released). Some experiments are done with different event-based strategies, that are some proportional integral derivative and state-feedback controllers

Note that the general presentation of the inverted pendulum was established with helping from [62] and some articles from the free encyclopedia Wikipedia (`http://fr.wikipedia.org/`)

(developed in chapter 6 and 7 respectively). The different experimental results are eventually compared to the conventional time-triggered scheme in order to highlight the advantages of the asynchronous scheme. We firstly focus on some linear systems. The velocity and the position of the cart are thus controlled alone (without the pendulum's arms), in order they track a given reference. These systems are detailed in subsection 8.1.1 and the experimental results are presented in section 8.2. The whole nonlinear inverted pendulum system is then introduced in subsection 8.1.2. The corresponding results are depicted in section 8.3. Eventually, both cases are driven thanks to a *Matlab/Simulink* interface, as explained in subsection 8.1.3.

This work was done in collaboration with the *Dynamic and Control Systems* research group, which is part of the University of Puebla in Mexico (Facultad de Ciencias de la Electrónica y Facultad de Ciencias Físico-Matemáticas - Benemérita Universidad Autónoma de Puebla).

8.1.1 The electric motor

As explained in introduction, the electric motor is directly actuated by the controller. Some preliminary tests are performed on it. Two different controls are possible: the position and the velocity of the cart respectively.

Position of the cart

The position of the cart is directly measured and the resulting model is a well-known second-order system, such as

$$H_p(s) = \frac{P(s)}{U(s)} = \frac{1}{s} \cdot \frac{G_p}{1 + \tau_p \cdot s}$$

where $H_p(s)$ is the transfer function between the position $P(s)$ and the input signal $U(s)$ (the voltage which power supplies the electric motor), τ_p is the time constant (which depends upon the load drive) and G_p is the steady-state gain. An identification rapidly gives some values for the different parameters, which are $G_p = 4.3218$ and $\tau_p = 0.06\,s$.

On the other hand, it could be interesting to find a state-space representation of the system in order to be able to apply a state-feedback control next. The classical representation is

$$\begin{aligned} \dot{x}(t) &= A \cdot x(t) + B \cdot u(t) \\ y(t) &= C \cdot x(t) \end{aligned}$$

where $x(t) = \begin{bmatrix} p(t) & \dot{p}(t) \end{bmatrix}^T$, $y(t)$ and $u(t)$ are the state, output and input (or control) vectors respectively. In the present case, the output is the position and the control signal is the voltage which power supplies the electric motor. A simple identification leads

$$A = \begin{bmatrix} -16.67 & 0 \\ 1 & 0 \end{bmatrix}, \quad B = \begin{bmatrix} 72.03 \\ 0 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 1 & 1 \end{bmatrix}.$$

The state-feedback strategy then consists in multiplying the states x of the system by a certain gain and setting this product as the new system input. This will be done in subsection 8.2.2.

Velocity of the electric motor

The transfer function $H_v(s)$ between the velocity of the motor $V(s)$ and the input signal $U(s)$ is deduced from the position measurement, since this is the derivative. Indeed, $v(t) = \dot{p}(t)$

or $V(s) = s \cdot P(s)$. This results in a first-order equation as follows

$$H_v(s) = \frac{V(s)}{U(s)} = \frac{G_v}{1 + \tau_v \cdot s}$$

where τ_v is the time constant and G_v is the steady-state gain. Unfortunately, the derivative cannot be applied directly because of the encoding mechanism. Actually, the position is measured thanks to a potentiometer, and more particularly a shaft encoder. The value of this encoder increases (respectively decreases) with respect to the shaft angular position of the motor. However the encoder cannot infinitely increase and, after achieving a given value, it goes back to 0. This principle finally results in a sawtooth-waveform signal which causes problem to obtain the derivative. For this reason, we modified this signal (using a *Matlab* function) in order to remove the overflows from the encoder. Now, having a continuous function, one can calculate the derivative and an identification eventually gives $G_v = 12.25$ and $\tau_v = 0.06$. This system will then be controlled with a PID controller in subsection 8.2.1.

Afterwards, one will note some inherent perturbations which periodically occur in the different experimental results in section 8.2. They are caused by this principle each time an overflow is enforced. Nevertheless, this will not be the case in the global system running, thanks to the belt, since the possible position from one part to the other of the track cannot overflow the maximal value of the encoder.

8.1.2 The inverted pendulum

The inverted pendulum system is more complex and testing some asynchronous approaches on it will clearly demonstrate the interest of such a scheme. Indeed, as already explained in introduction, the inverted pendulum is naturally unstable. This means that it requires to continuously update the control law else its arms fall down. The idea here is to adapt the classical state-feedback control law to the asynchronous scheme. For this reason, the complete modeling of the system is not treated. We simply base our analysis on a previous work which was done on the present inverted pendulum study case. One could refer to [50] for further details. Eventually, a linearization of the system close to its equilibrium point - used to calculate the control parameters - leads to the given state-space representation

$$\begin{aligned} \dot{x}(t) &= A \cdot x(t) + B \cdot u(t) \\ y(t) &= C \cdot x(t) \end{aligned}$$

where $x(t) = \begin{bmatrix} p(t) & \theta(t) & \dot{p}(t) & \dot{\theta}(t) \end{bmatrix}^T$.

A simple identification yields

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0.28 & 0 & -1.32 \cdot 10^{-4} \\ 0 & 16.49 & 0 & -0.0079 \end{bmatrix}, \quad B = \begin{bmatrix} 0 \\ 0 \\ 0.76 \\ 1.24 \end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} 1 & 1 & 1 & 1 \end{bmatrix}.$$

Actually, controlling the inverted pendulum results in two parts: a first part is the balancing of the pendulum, from its idle position until it is upright. This balancing can be done manually or in using a special strategy, as done in [13]. Once the equilibrium point is achieved, the controller switches the strategy, in order the inverted pendulum now stabilizes its arms, thanks to a state-feedback control law. Note that, afterwards, only the second part will be treated using an event-driven scheme.

8.1.3 The Matlab/Simulink interface

The inverted pendulum used in the next experiments is the one proposed by FeedBack Incorporated[1]. The interest of this system is that the control part is done in the *Matlab/Simulink* environment. The resulting model is depicted in figure 8.2.

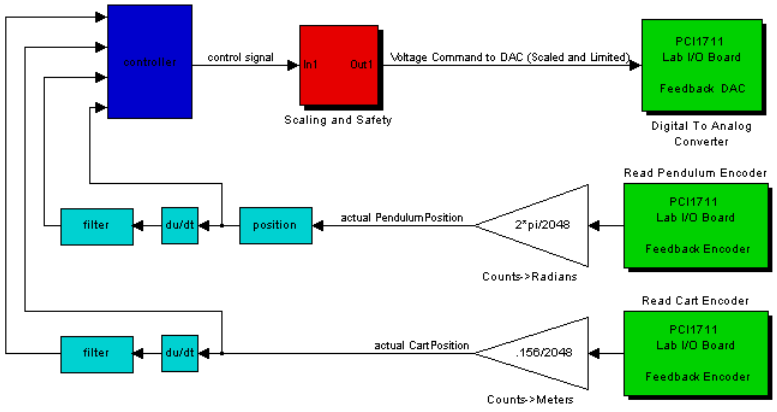


Figure 8.2: *Simulink* model used in experiments to control the inverted pendulum.

The measured signals of the position of the cart and the angle of the pendulum come from some analog-to-digital encoders (in the bottom right in the figure). Some derivative blocks give the velocity and the angle velocity respectively. All these measurements are finally monitored by a "controller" (in the top left), which calculates the control signal to send to the electric motor through a digital-to-analog converter (top right). Note that a "scaling and safety function" is placed between the controller and the actuator in order to warm about an over-supply of the motor. Indeed, the control signal can only varies from -2.5 to 2.5 ($-10\,V$ to $10\,V$ respectively, after amplification) and a saturation block is hence required. Furthermore, the "position block" is added in order to obtain the derivative of the position of the cart, as explained in subsection 8.1.1. Some filters are also applied to the derivative signals. Eventually, a *s-function* is developed into the controller block, which contains the algorithm to implement.

8.2 First results in controlling the velocity and the position of an electric motor

As explained before, **we propose to make some first tests only on the electric motor** in a first time, **controlling the velocity and the position of the cart**. Different experiments are done, using some event-based PID control techniques in subsection 8.2.1, and some state-feedback control methods in subsection 8.2.2. In both cases, the robustness of the asynchronous scheme is studied, applying some perturbations to the system.

8.2.1 PID control strategy using level-crossing detection

The PI/PID strategies (see chapter 6 for further details) consist in controlling the system in function of the error $e(\cdot)$ between a given setpoint and a measured signal. The control signal

[1] FeedBack Incorporated: `http://www.feedbackinc.com/`

is then the sum of a proportional $u_p(\cdot)$, an integral $u_i(\cdot)$ and a derivative $u_d(\cdot)$ parts, that are

$$\begin{aligned} u_p(t_a) &= K_p \cdot e(t_a) \\ u_i(t_a) &= u_i(t_{a-1}) + K_i \cdot he(t_a) \\ u_d(t_a) &= \frac{T_d}{T_d + N \cdot h(t_a)} \cdot u_d(t_{a-1}) + \frac{K_p \cdot T_d \cdot N}{T_d + N \cdot h(t_a)} \cdot \big(e(t_a) - e(t_{a-1})\big) \end{aligned}$$

where t_a is the beginning of the current sampling time and t_{a-1} the one of the last sample, which lead to a sampling period $h(t_a) = t_a - t_{a-1}$. The control signal $u(t_a) = u_p(t_a) + u_i(t_a) + u_d(t_a)$ is enforced regarding some event detection conditions, where the integral part - and more particularly the integral gain $he(t_a)$ - depends on the event-based control strategy. The different event-based algorithms are recaped in section 6.3. Furthermore, K_p, K_i, T_d and N are some tunable parameters which are calculated with respect to the system to control.

On the other hand, an anti-windup mechanism is added in the integral part in order to prevent windup when the actuator is saturated (since the control signal can only varies from -2.5 to 2.5, as explained in subsection 8.1.3). This extra term is

$$u_i(t_a) = \ldots - K_a \cdot h(t_a) \cdot \Big(u(t_{a-1}) - u_{sat}(t_{a-1})\Big)$$

where $u_{sat}(\cdot)$ is the saturated value of the control signal and K_a is another tunable parameter.

Eventually, the system has to track a given reference in the following experiments. In both cases (the velocity and the position), the setpoint starts with an amplitude of 10 during $5\,s$, before going back to 0 for $5\,s$ more.

8.2.1.1 Velocity of the electric motor

The control parameters required in the different PI strategies to control the velocity of the electric motor are $K_p = 0.01$, $K_i = 0.5$ and $K_a = 20$. The sampling intervals are $h_{nom} = 0.01\,s$ and $h_{max} = 0.1\,s$, while the level detection is $q_{nom} = 0.2$.

The experimental results for the conventional approach - detailed in subsection 6.1.2 - are represented in figure 8.3. The top plot shows the setpoint and the measured signal, whereas the bottom plot shows the sampling intervals which are, of course, all equal to h_{nom} in this time-triggered case. Note that the number of samples required to perform the test bench and some performance indexes (see section 6.5 for further details) are also indicated:

- the IAE index gives information on the setpoint tracking,
- the IAEP index compares the time-based and event-based system responses,
- the IAD index compares the time-based and event-based IAEs.

Eventually, one could remark a small perturbation at $2\,s$, which is due to the encoding mechanism (depicted in subsection 8.1.1). That will appear in all the experimental results. A certain robustness to this "error" (inherent to the system itself) is expected, before deliberately introducing some perturbations at the end of this subsection.

Respectively, the results for the Årzén's controller (with discretization improvement and absolute error, depicted in subsections 6.2.1 and 6.2.2) are represented in figure 8.4. In this case, the control updates are event-driven when the measured signal crosses a given level q_{nom}, that is why a lot of samples occur during the transients. Moreover, A safety limit condition is also introduced in order an event is enforced when the sampling interval achieves the maximal

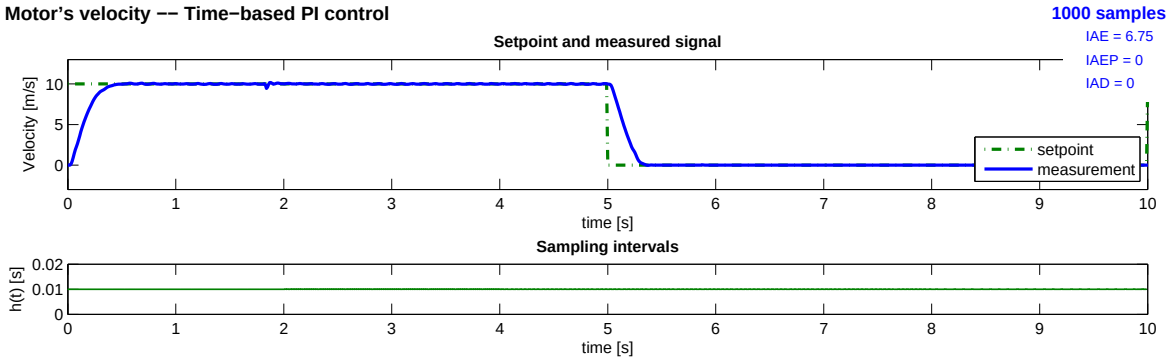


Figure 8.3: Experimental results: control of the velocity of the motor with the conventional time-triggered PI controller.

value h_{max}. This behavior occurs during the steady-state intervals, this is why some samples hit every 0.1 s. This principle allows to considerably reduce the number of samples (about 85 % less than in the conventional case) with similar final performance (the IAE index is even better than in the time-based case while the IAEP is very close). Note that an extra plot referring to the sampling instants is also shown in figure 8.4, where an event is drawn each time the control signal is updated. This representation will be preferred in the following experimental results.

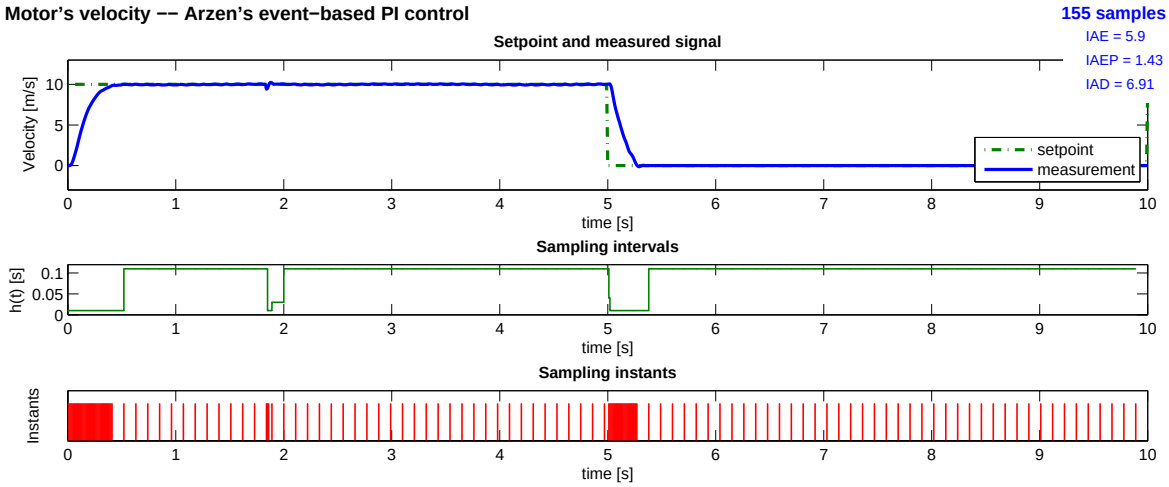
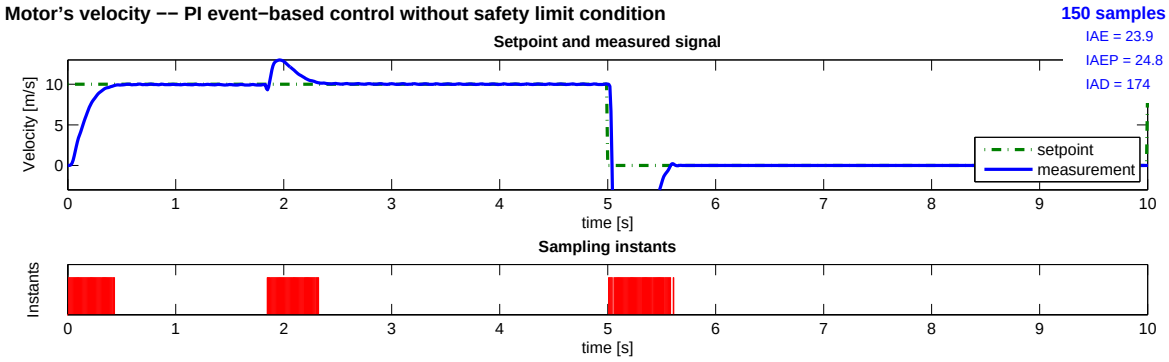


Figure 8.4: Experimental results: control of the velocity with the Årzén's event-based PI controller.

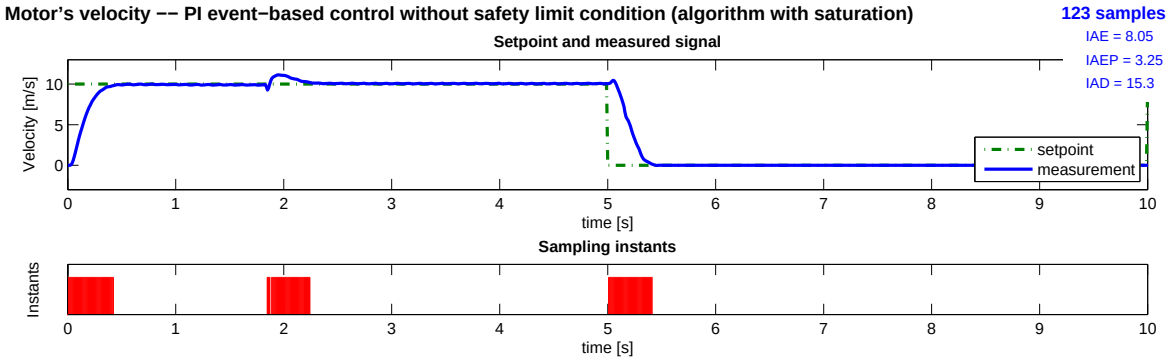
Our first proposal consists in removing the safety limit condition to decrease again the number of samples. Several algorithms were developed - in subsection 6.2.3 - and the results are shown in figure 8.5.

- The first algorithm only consists in removing the condition and leads to important overshoots, as one can see in figure 8.5(a) when the setpoint changes at 5 s or during the perturbation occurring near 1.8 s.

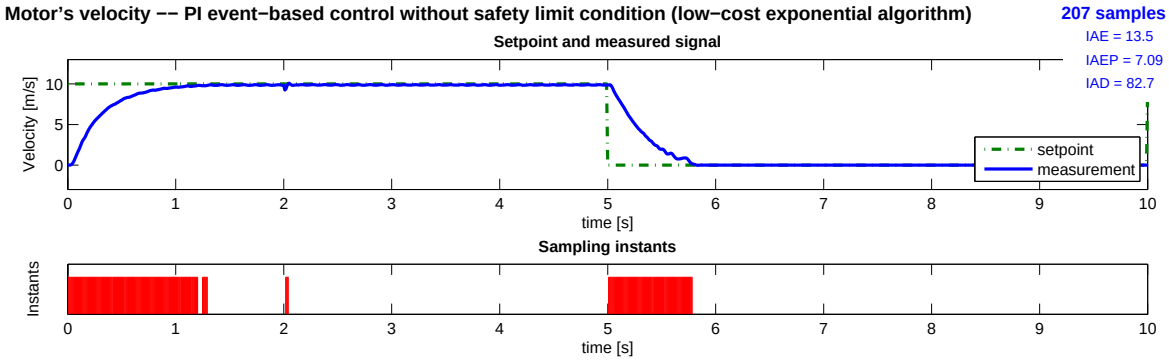
- The second algorithm applies a saturation of the product $h(\cdot)e(\cdot)$ in order to minimize its impact on the integral part of the controller after a large steady-state interval. This is done in figure 8.5(b) and one can see that the previous overshoots are reduced.



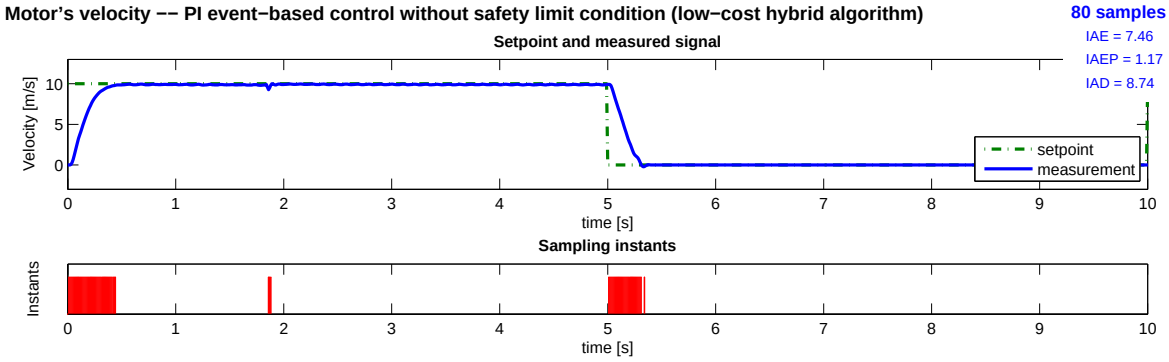
(a) Algorithm only without safety limit condition



(b) Algorithm with saturation of the integral gain



(c) Algorithm with an exponential forgetting factor of the sampling interval (with low-cost implementation)



(d) Hybrid algorithm (with low-cost implementation)

Figure 8.5: Experimental results: control of the velocity with some event-based PI controllers without safety limit condition.

- An exponential forgetting factor of the sampling interval is then introduced in a third algorithm, in order to reduce the impact of this parameter in the integral part (because the sampling interval increases a lot during a steady state). However, as an exponential function is complex to implement, we also developed a low-cost alternative. The results are represented in figure 8.5(c). The overshoots completely disappear but the system response time highly increases (and so the IAE index).

- Finally we proposed an hybrid algorithm - a mix between the saturation of the integral gain and the low-cost exponential forgetting factor - whose results are depicted in figure 8.5(d). This solution is the best one since the system response is quite good (the IAE index is close to the time-triggered case). Furthermore, a reduction of the number of samples of 92 % is achieved compared with the classical method, and about 50 % compared with the Årzén's proposal (with a better IAEP value).

These first results for a simple first-order system are very encouraging and the advantages of an asynchronous scheme is highly demonstrated.

8.2.1.2 Position of the cart

The control parameters used to control the position of the cart are $K_p = 0.2$, $K_i = 0.8$, $K_a = 100$, $T_d = 0.001$ and $N = 1$. The different sampling intervals are $h_{nom} = 0.01\,s$, $h_{max} = 0.1\,s$, $h_{min} = 0.02\,s$ and $h_{extra} = 0.1\,s$. The level detection is $q_{nom} = 0.2$.

As previously observed in the velocity case, the experimental results obtained with the Årzén's controller (still with discretization improvement and absolute error) show that the number of samples is reduced - 75 % less thanks to the level-crossing detection - with similar final performance. This is depicted in figure 8.6. One could refer to the IAE index - it is equal to 8.73 in the time-triggered case (not represented) - which is very close with the event-based approach using the same control parameters. Moreover, a perturbation still occurs at $1.8\,s$, because of the encoding mechanism, and the system correctly react to compensate.

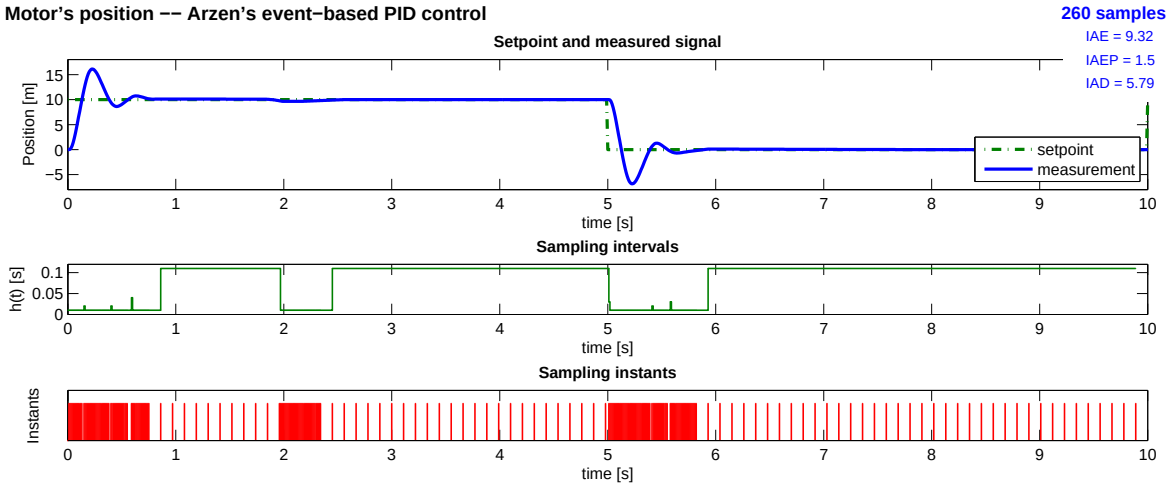


Figure 8.6: Experimental results: control of the position of the cart with the Årzén's event-based PID controller.

As already noticed too, the Årzén's technique enforces some events after a safety amount of time even if the measurement does not cross a detection level (initially for a stability reason) and yet, we demonstrated in subsection 6.2.3 that this condition is not necessary actually. Different

algorithms without this safety limit condition were developed and, for instance, the simulation results obtained with the hybrid one (with low-cost implementation) are depicted in figure 8.7. This yields an IAE index a bit more important but the number of samples if strongly reduced, that is more than 40 % of samples less than Årzén.

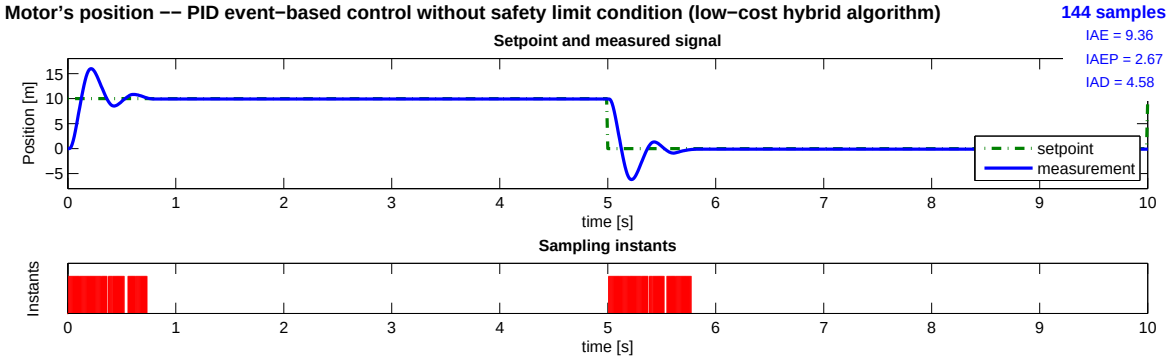


Figure 8.7: Experimental results: control of the position with the event-based PID controller without safety limit condition (hybrid algorithm).

We also proposed to improve the algorithms without safety limit condition, adding ***i)*** a minimum sampling condition and/or ***ii)*** some extra samples (depicted in subsections 6.2.4 and 6.2.5 respectively). In the first scheme, the controller updates the control signal only if a given amount of time is elapsed from the last sample, that is when $h(t_a) \geq h_{min}$. The results are shown in figure 8.8 (still with the hybrid algorithm). Eventually, the minimum sampling condition clearly allows to reduce again the number of samples (15 % of samples less than in our initial proposal). In the second case, some extra samples are enforced (h_{extra} periodically) after a transient in order to decrease the steady-state error. The results are shown in figure 8.9. However, the number of samples increases since some samples are added after each transient (two in the present case).

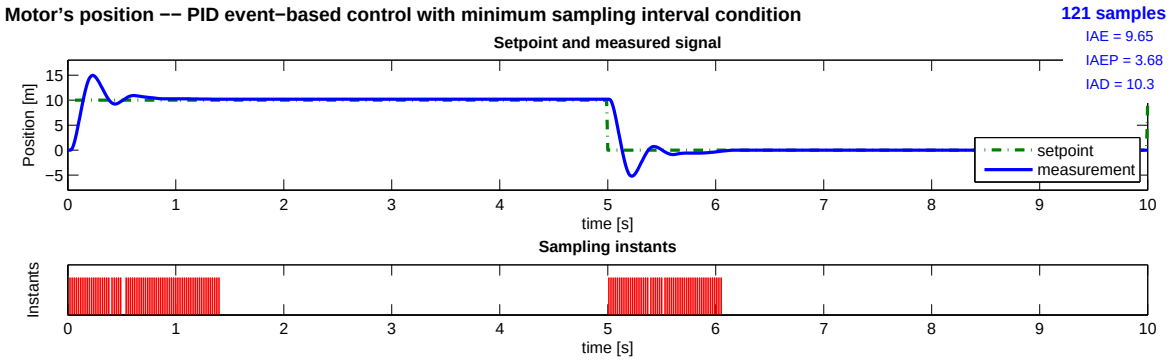


Figure 8.8: Experimental results: control of the position with a minimum sampling interval (hybrid algorithm).

8.2.1.3 Perturbations and robustness

Finally, the system is submitted to some perturbations - slowing down the drive-shaft of the motor - in order to see if an event-based scheme correctly reacts when the system does not work as well as in theory. The test bench is extended to a 20 s-experiment with two steps and the chosen algorithm is still the hybrid one. For both results in figure 8.10, when controlling the

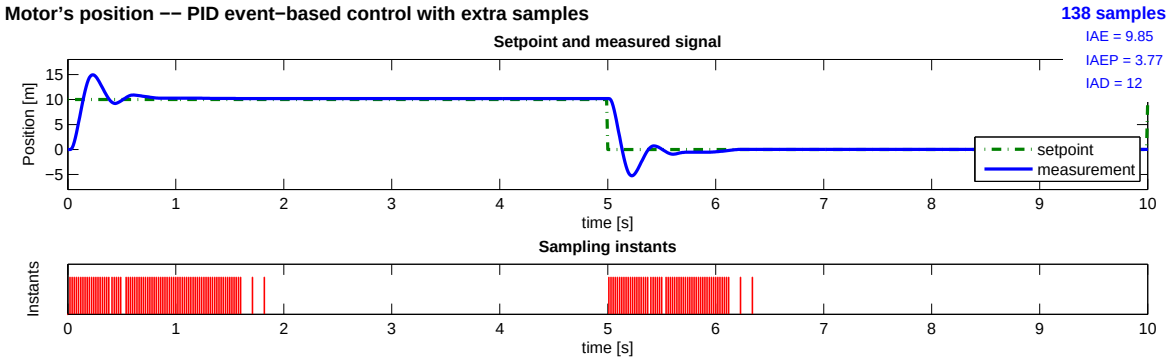
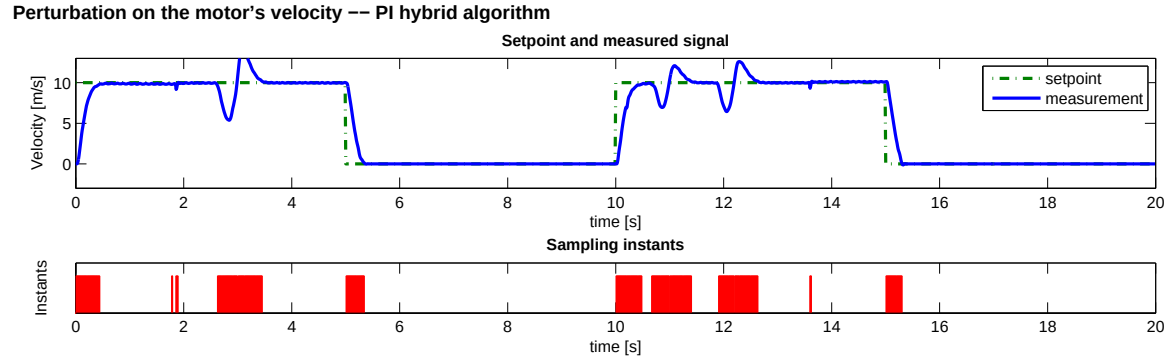
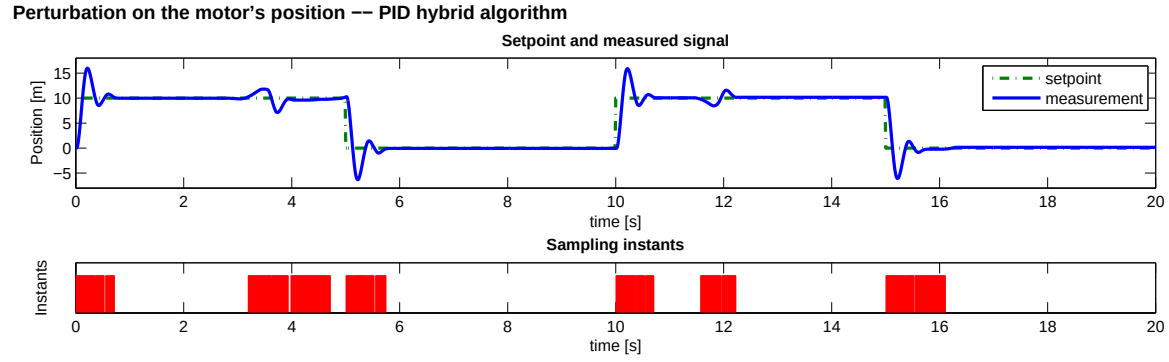


Figure 8.9: Experimental results: control of the position adding 2 extra samples (hybrid algorithm).

velocity and the position, the event-based control is robust since the measurement still tracks the setpoint. Of course the number of samples increases to dynamically take into account the perturbations. On the first plot, a perturbation is applied at about $3\,s$ and two successive ones then occur at 11 and $12\,s$, but the asynchronous PI control allows the velocity to track back the reference as soon as the perturbation disappears. Respectively, the PID strategy also correctly controls the position in spite of two perturbations at 3 and $12\,s$. At the end, the robustness of an event-based scheme is hence demonstrated in practice.



(a) Test on the velocity



(b) Test on the position

Figure 8.10: Experimental results: robustness to some perturbations on the system (slowing down the drive-shaft of the motor).

8.2.2 State-feedback control strategy using Lyapunov sampling mechanism

Let

$$\dot{x} = A \cdot x + B \cdot u$$
$$y = C \cdot x$$

be the system to control. As explained in subsection 7.1.1, a state-feedback strategy consists in applying the control law $u = -K \cdot x$ to the system input, where K is the state-feedback gain. The resulting closed-loop system becomes the autonomous system $\dot{x} = \mathbf{A} \cdot x$, where $\mathbf{A} = A - B \cdot K$. Moreover, in the Lyapunov sampling scheme, the control law is constant over sampling intervals, and an event is enforced with respect to a Lyapunov function (see chapter 7 for further details). Here, the classical energetic quadratic function is used, that is

$$V = x^T \cdot P \cdot x$$

where $P = P^T > 0$ is a symmetric positive-definite matrix which has to satisfy $\mathbf{A}^T \cdot P + P \cdot \mathbf{A} < 0$ - or alternatively $\mathbf{A}^T \cdot P + P \cdot \mathbf{A} = Q$, where Q is positive definite also - in order to ensure that the system is stable. Fortunately, the *Matlab* function $lyap(\mathbf{A}, Q)$ allows to resolve such an equation, providing both parameters P and K for a given system.

On the other hand, the system will have to track a given reference r in the next experimental tests. For this reason, an extra term is added in the control law. This yields $u = -K \cdot x + K_r \cdot r$, where K_r is a tunable gain which is calculated such as

$$K_r = -\Big(C \cdot \big(A - B \cdot K\big)^{-1} \cdot B\Big)^{-1}$$

in order the system correctly tracks the reference. Furthermore, an integral state is also applied, defined such that $\dot{z} = r - y$, in order to guarantee a null steady-state error. Eventually, the control law becomes

$$u = -K \cdot x + K_r \cdot r + K_z \cdot z$$

where K_z is a new tunable parameter. Finally, the closed-loop system thus yields

$$\begin{bmatrix} \dot{x} \\ \dot{z} \end{bmatrix} = \begin{bmatrix} A - B \cdot K & B \cdot K_z \\ -C & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ z \end{bmatrix} + \begin{bmatrix} B \cdot K_r \\ 1 \end{bmatrix} \cdot r$$

This augmented state will be used to ensure a null static error but we do not want to apply it to the Lyapunov function. Indeed, only the internal state will be used to decide when updating the control signal and, for this reason, the state z does not complexify the event detection.

As regards the experimental bench, the setpoint to track is the same than in subsection 8.2.1: the reference starts with an amplitude of 10 during $5\,s$ before going back to 0 for $5\,s$ more.

8.2.2.1 Control of the position

The principle explained in introduction allows to calculate the control parameters which will be used in the state-feedback strategies. They result in

$$P = \begin{bmatrix} 2.9406 & 0 \\ 0 & 0.2747 \end{bmatrix}, \quad K = \begin{bmatrix} 0 & 0.25 \end{bmatrix}, \quad K_r = 0.25 \quad \text{and} \quad K_z = 1$$

The different sampling intervals are $h_{nom} = 0.01\,s$ and $h_{min} = 0.02\,s$. Eventually, some other parameters required in the different asynchronous strategies are $\nu = \overline{\nu} = \underline{\nu} = 0.1$, $\varepsilon = 0.02$ and $\upsilon = 1$ (see the recap in section 7.3 for further details).

The experimental results for the classical approach - detailed in subsection 7.1.1 - are represented in figure 8.11. The top plot shows the setpoint and the measured signal, whereas the bottom plot shows the system energy, that is the Lyapunov function. Note that, as for the PID controllers, the number of samples required to perform the test bench and some performance indexes are also indicated.

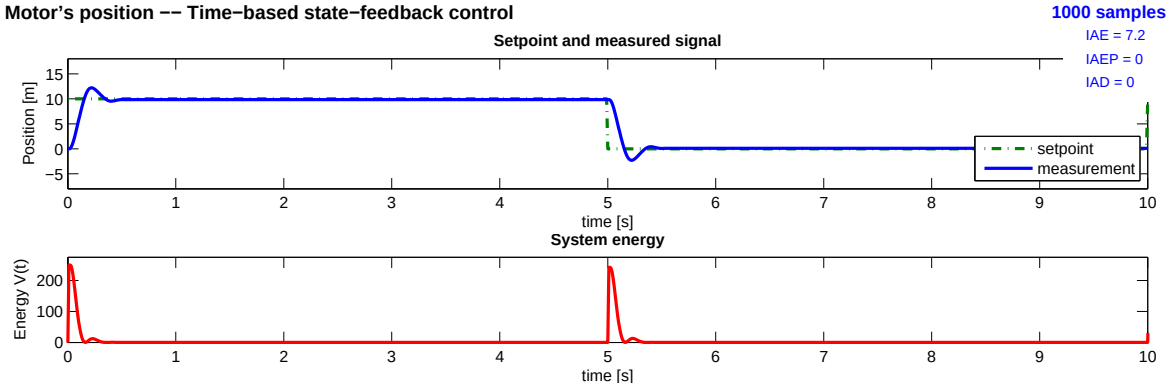


Figure 8.11: Experimental results: control of the position of the cart with a conventional (time-triggered) state-feedback controller.

The Lyapunov sampling mechanism - introduced by Velasco et al. and detailed in subsection 7.2.2 - enforces a control update when the current value of the Lyapunov function becomes equal to a ratio of its value at the previous sample. The energy gain factor η defines this ratio. This simple scheme ensures that the system is stable in the Lyapunov sense (since the energy decreases event after event) but the system dynamics could become unstable if the next level is never crossed. The stable Lyapunov sampling promises an infinite sequence of samples in restricting the value of the energy gain factor in such a way that $\eta^* < \eta < 1$. This results in an off-line algorithm used to calculate η^*, which is $\eta^* = 1$ in the present study, as highlighted in figure 8.12. The previous restricting condition cannot be satisfied and so is the stability. Consequently, the original Lyapunov sampling mechanism cannot be directly applied to control the position of the cart and an improvement is hence required.

1. In a first time, a simple modification was proposed in subsection 7.2.2.3, in order to enforce a new event as soon as the system becomes unstable (that is when the Lyapunov function increases). Applying this small improvement now allows to run the system. The simulation results are presented in figure 8.13 with $\eta = 0.1$. In fact, using this small value allows to considerably reduce the number of samples, but the system becomes unstable sometimes. Anyway, the modification leads to stabilize it back.

2. Another modification is also proposed in subsection 7.2.3 in order to not have to calculate the minimum possible value η^* (and so run the computationally heavy off-line algorithm). Some relaxations of the condition $\eta^* < \eta < 1$ were proposed and the energy gain factor is now varying. It dynamically changes to increase the sampling intervals when the system is stable, and decrease them when the system diverges. The results of these less-conservative strategies are drawn in figure 8.14. An extra plot represents the dynamics of the variable $\eta(t)$. Before detailing the different proposals, one could already note that an important

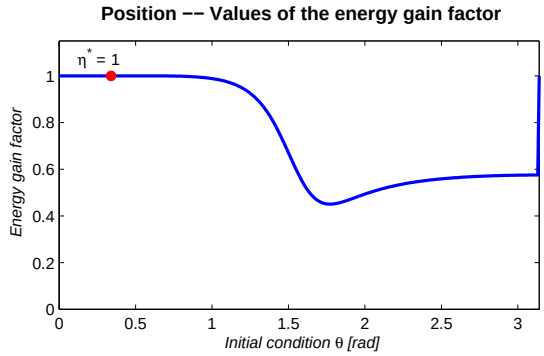


Figure 8.12: Simulation results: possible values of $\hat{\eta}(\cdot)$ calculated with the off-line algorithm (needed in the stable Lyapunov sampling mechanism) for the position's control case in order to obtain η^*.

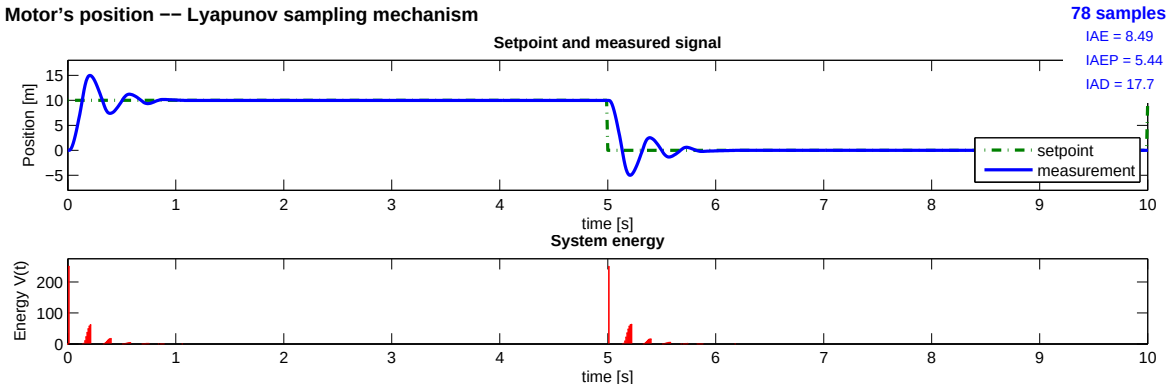
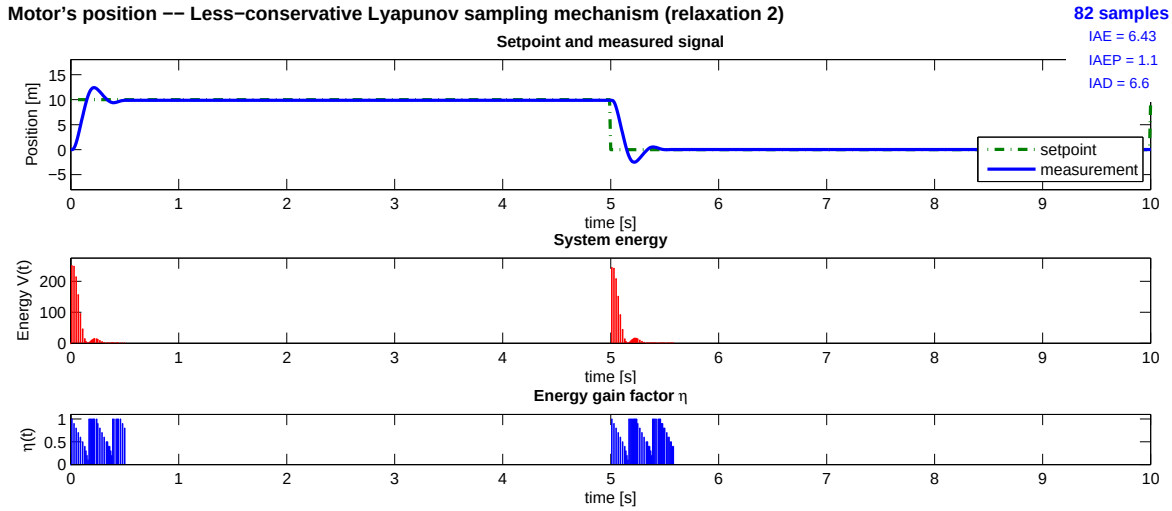


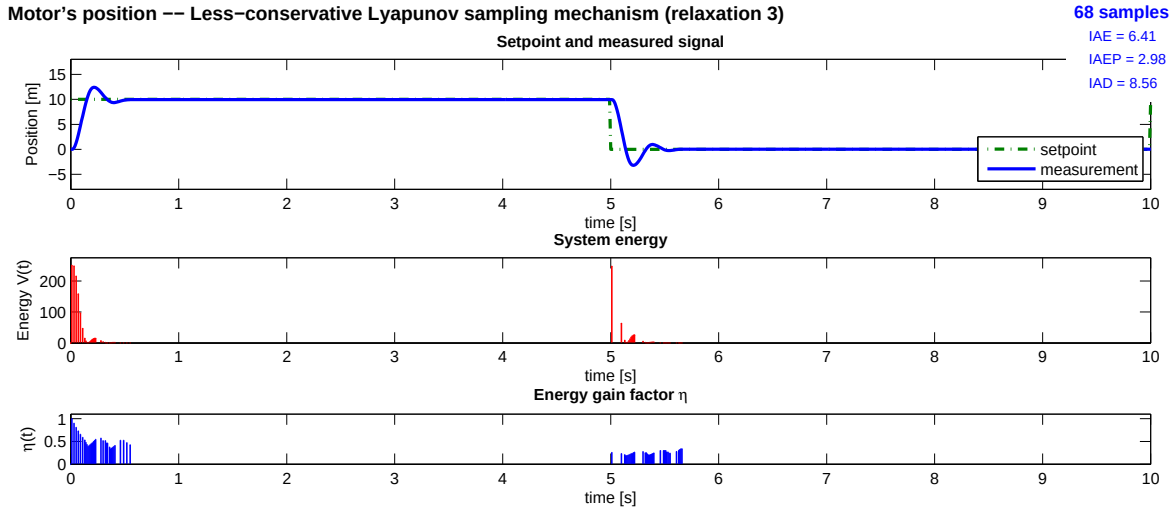
Figure 8.13: Experimental results: control of the position with an asynchronous state-feedback controller using the Lyapunov sampling mechanism with improvement.

reduction of computation is achieved since only (less than) 10 % of samples allow to obtain almost the same performance than the classical scheme. The IAEP index for instance - which compares the event-based system response with the time-triggered one - is always lower.

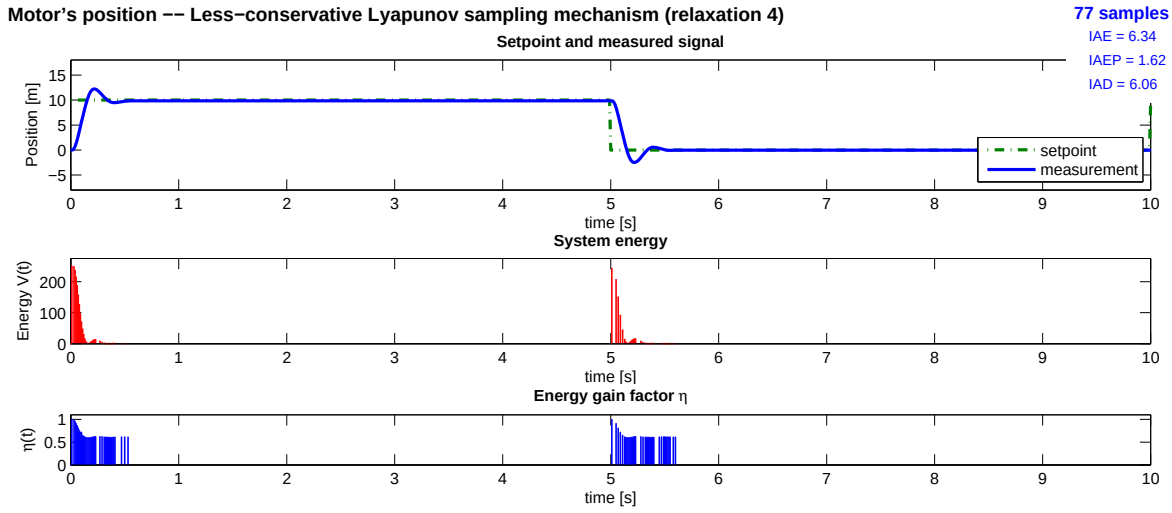
- The first relaxation consists in slowly decreasing the energy gain factor when the Lyapunov function decreases, and drastically going back to η^* as soon as the system becomes unstable. However, this solution still needs to know η^* and, consequently, to run the off-line algorithm.
- The second relaxation is an immediate improvement to have an on-line running. The previous relaxation is modified, replacing η^* by the maximum possible value of η, that is $1-\varepsilon$ by construction. The results are drawn in figure 8.14(a). The system response is quite good, as one can notice looking at the different indexes of performance which are all very low.
- A third relaxation then slowly decreases and slowly increases $\eta(t)$. The results are represented in figure 8.14(b) and lead to reduce the number of samples again.
- Finally, a more formal variation of the energy gain factor is proposed, considering this parameter as a state variable in the control algorithm, which now evolves with respect to the variation of the Lyapunov function and the sampling interval. The results, depicted in figure 8.14(c), are still close to the time-triggered response.



(a) Relaxation 2: slowly decrease/drastically increase $\eta(t)$ in an on-line running

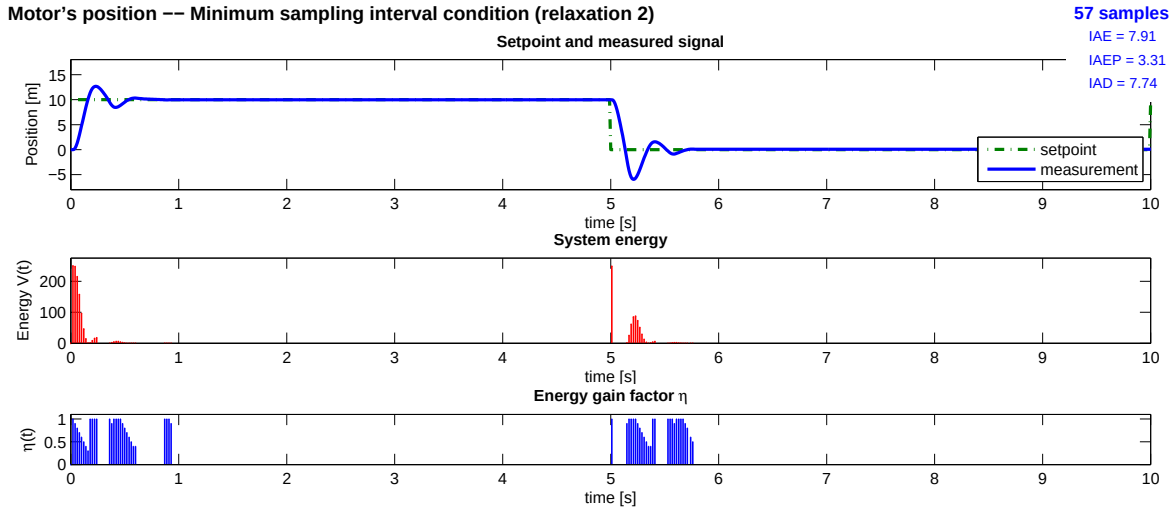


(b) Relaxation 3: slowly decrease/slowly increase $\eta(t)$

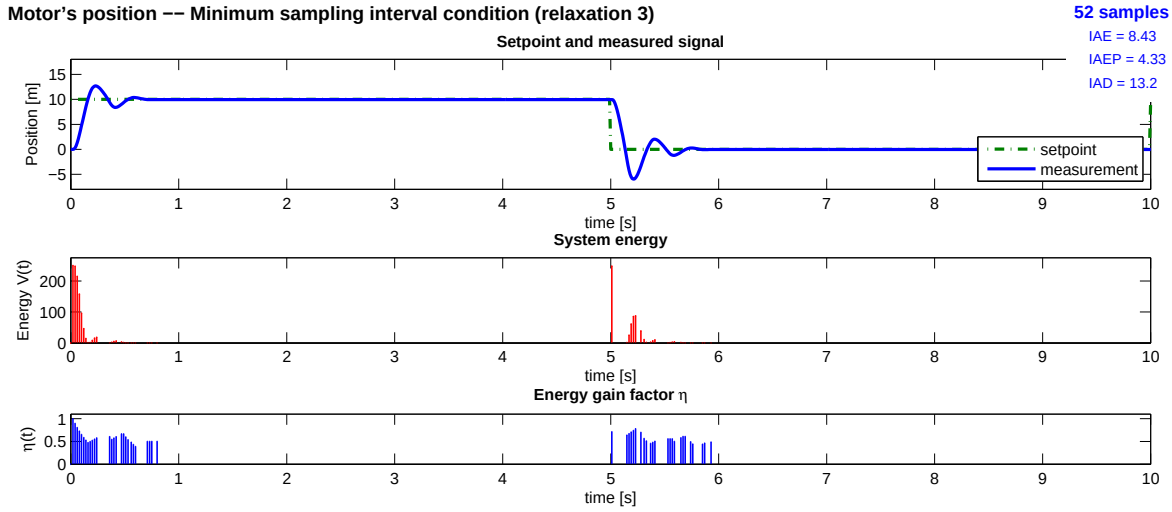


(c) Relaxation 4: a more formal variation of $\eta(t)$

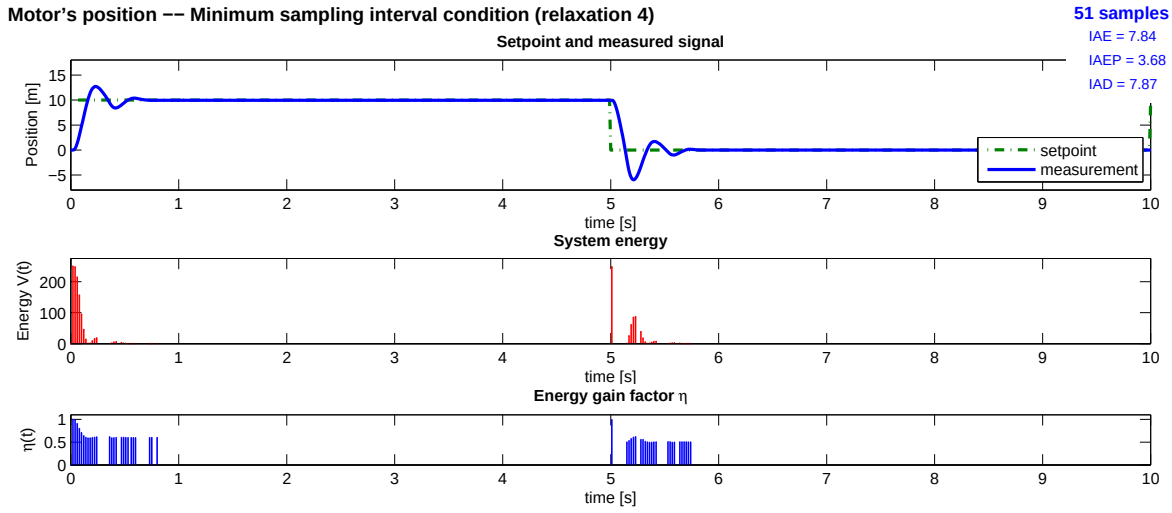
Figure 8.14: Experimental results: control of the position with an asynchronous state-feedback controller using a less-conservative Lyapunov sampling mechanism.



(a) Relaxation 2: slowly decrease/drastically increase $\eta(t)$ in an on-line running



(b) Relaxation 3: slowly decrease/slowly increase $\eta(t)$



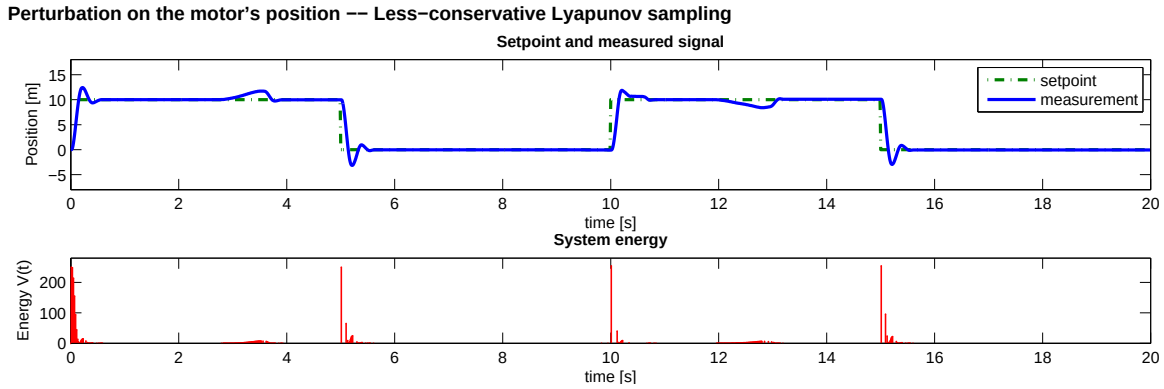
(c) Relaxation 4: a more formal variation of $\eta(t)$

Figure 8.15: Experimental results: control of the position with an asynchronous state-feedback controller using a less-conservative Lyapunov sampling mechanism with a minimum sampling condition.

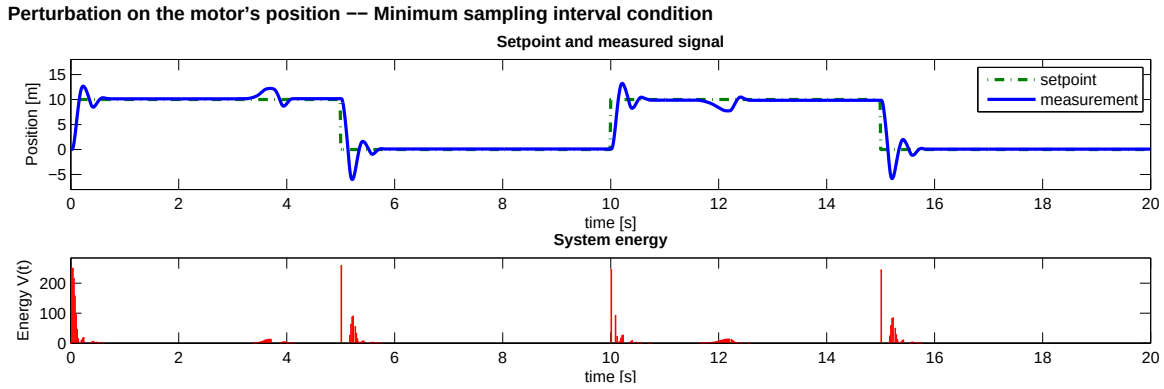
These encouraging results are completed with the ones obtained in figure 8.15, where a minimum sampling condition is added. The principle was explained in subsection 7.2.4. The idea is to update the control signal only if the elapsed amount of time since the last sample is larger than a given value h_{min}. This allows to reduce again the number of samples (about 30 % less than with out initial proposals), still with some acceptable performance.

8.2.2.2 Perturbations and robustness

Eventually, some perturbations are introduced during some experiments, as one can see in figure 8.16 at 3 and 12 s. Two strategies are tested: the less-conservative Lyapunov sampling scheme and when adding a minimum sampling interval condition. In both cases, the system is robust and the position continues to track the setpoint. Actually, some events occur during the perturbation because the Lyapunov function changes.



(a) Less conservative Lyapunov sampling mechanism



(b) Less conservative Lyapunov sampling mechanism (with a minimum sampling condition)

Figure 8.16: Experimental results: robustness to some perturbations on the system (slowing down the drive-shaft of the motor).

8.3 Further results in stabilizing the inverted pendulum

In the previous section, we tested the event-based proposals on some linear systems. The experimental results were very encouraging and **we propose to apply the asynchronous control schemes to an unstable nonlinear system, the inverted pendulum**. Such a system requires to update the control law very often else its arms fall down. Only the state-feedback control strategies will be tested on this four-order system.

As explained in subsection 8.1.2, controlling the inverted pendulum is divided into two parts: ***i)*** the balancing to achieve an upright position and ***ii)*** the stabilization of the pendulum when it is close to its equilibrium point. Only the second part is studied here. The control principle is detailed in subsection 8.2.2. The classical state-feedback strategy $u = -K \cdot x$ is used and an extra term is added for friction compensation, that yields $u = -K \cdot x + K_f$. Then, the control parameters which will be used by the different state-feedback strategies are obtained. They are

$$P = \begin{bmatrix} 2.43 & -9.29 & 2.45 & -2.30 \\ -9.29 & 173.50 & -20.24 & 42.54 \\ 2.45 & -20.24 & 5.00 & -5.02 \\ -2.30 & 42.54 & -5.02 & 10.56 \end{bmatrix} \quad \text{and} \quad K = \begin{bmatrix} -1 & 37.32 & -2.43 & 9.27 \end{bmatrix}.$$

Moreover, the friction parameter is ± 0.1 when the velocity is positive or negative, 0 otherwise. The time-triggered sampling period is $h_{nom} = 0.01\,s$. Only the third relaxation will be tested here. The parameters required in this strategy are $\overline{\nu} = \underline{\nu} = 0.1$ (see the recap in section 7.3 for further details).

The experimental results for the classical approach are represented in figure 8.17 where the four top plots show the states of the system: the position and the velocity of the cart ; the angle and the angular velocity of the pendulum. The bottom plot shows the control signal. Note that the number of samples required to perform the bench is also indicated. One could see two distinctive parts in the experimental results. The first part is the balancing of the pendulum until it achieves its equilibrium state. Thus, from 0 to about $20\,s$, the angle of the pendulum oscillates until achieving the origin. This balancing can be done manually or in using a special strategy, as explained in subsection 8.1.2. Once the equilibrium point is achieved, the inverted pendulum is stabilized thanks to the state-feedback controller and remains stabilized during the whole test bench. On the other hand, the cart evolves in order to keep this stability and one can hence see the position slowly tending to the origin too. Only the second part will be analyzed and, therefore, a zoom is performed in figure 8.18. The number of samples is finally indicated only for the stabilizing part.

Then, the system is tested with an event-driven controller using a Lyapunov sampling mechanism. We directly implement the less-conservative Lyapunov sampling mechanism, and only the third relaxation is tested (due to a lack of time). Thus, the control signal is updated when the system trajectory achieves some given levels. The experimental results are shown in figure 8.19. Two extra plots are represented: the system energy (that is the Lyapunov function) and the dynamics of the energy gain factor $\eta(t)$ used for event detection. One could already notice that a real reduction of the number of samples is achieved (about 55 % less) while similar performance. One could also note that the energy gain factor evolves with respect to the Lyapunov function. Eventually, the results confirm the interest of an event-based setup, since an important gain is performed, even in the case of an unstable system which requires to update the control law very often. Furthermore, the control signal is correct since it is not always saturated.

At the end, the same asynchronous state-feedback controller is tested with some other parameters. It is now implemented with

$$P = \begin{bmatrix} 0.12 & 0 & 0 & 0 \\ 0 & 0.87 & 0 & 0 \\ 0 & 0 & 3.63 & 0 \\ 0 & 0 & 0 & 186.86 \end{bmatrix}$$

where a lot of zero compose the P matrix in order to reduce even more the control computational cost. The results are drawn in figure 8.20. Finally, this second scheme also gives good results.

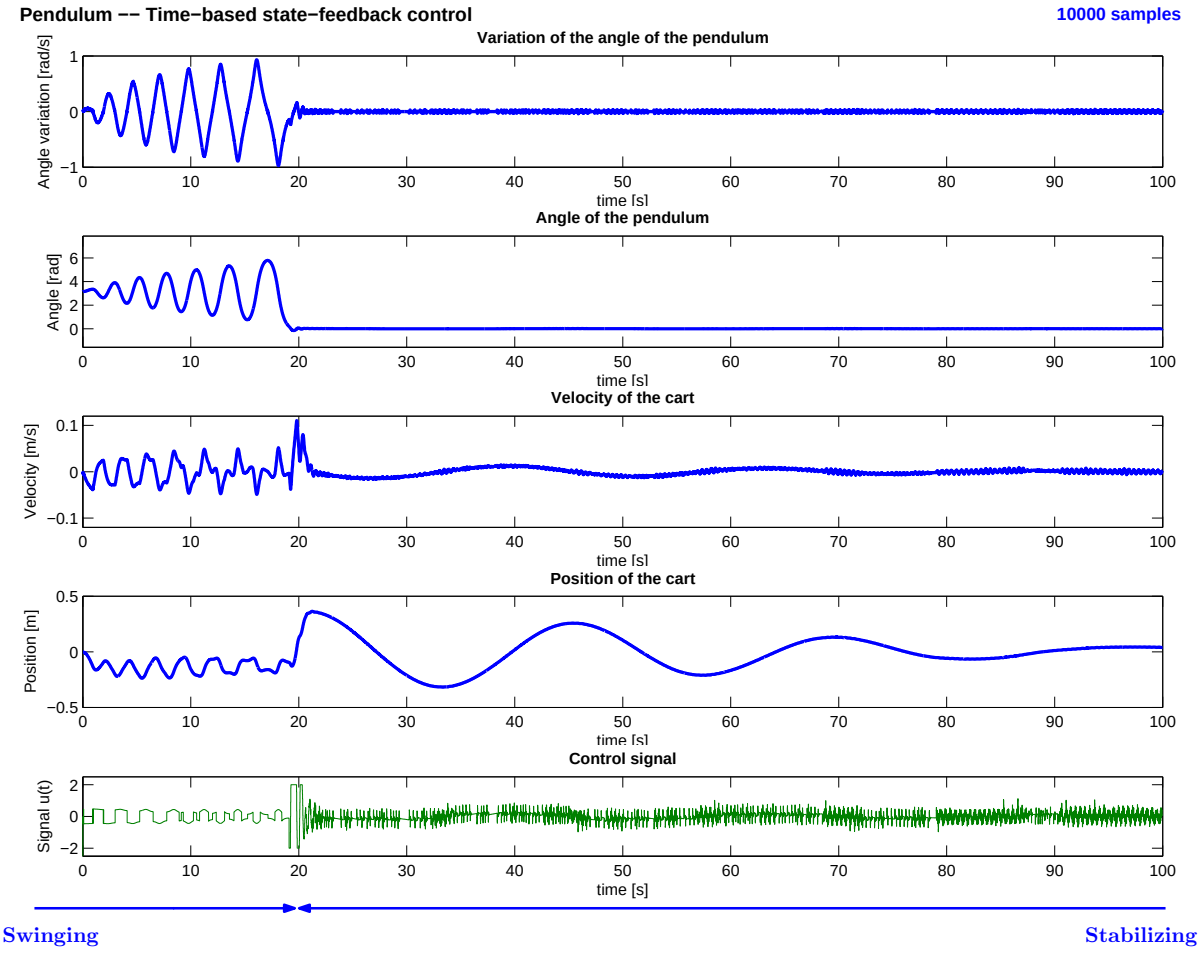


Figure 8.17: Experimental results: the two parts - balancing and stabilizing - in controlling the inverted pendulum (using time-triggered control laws).

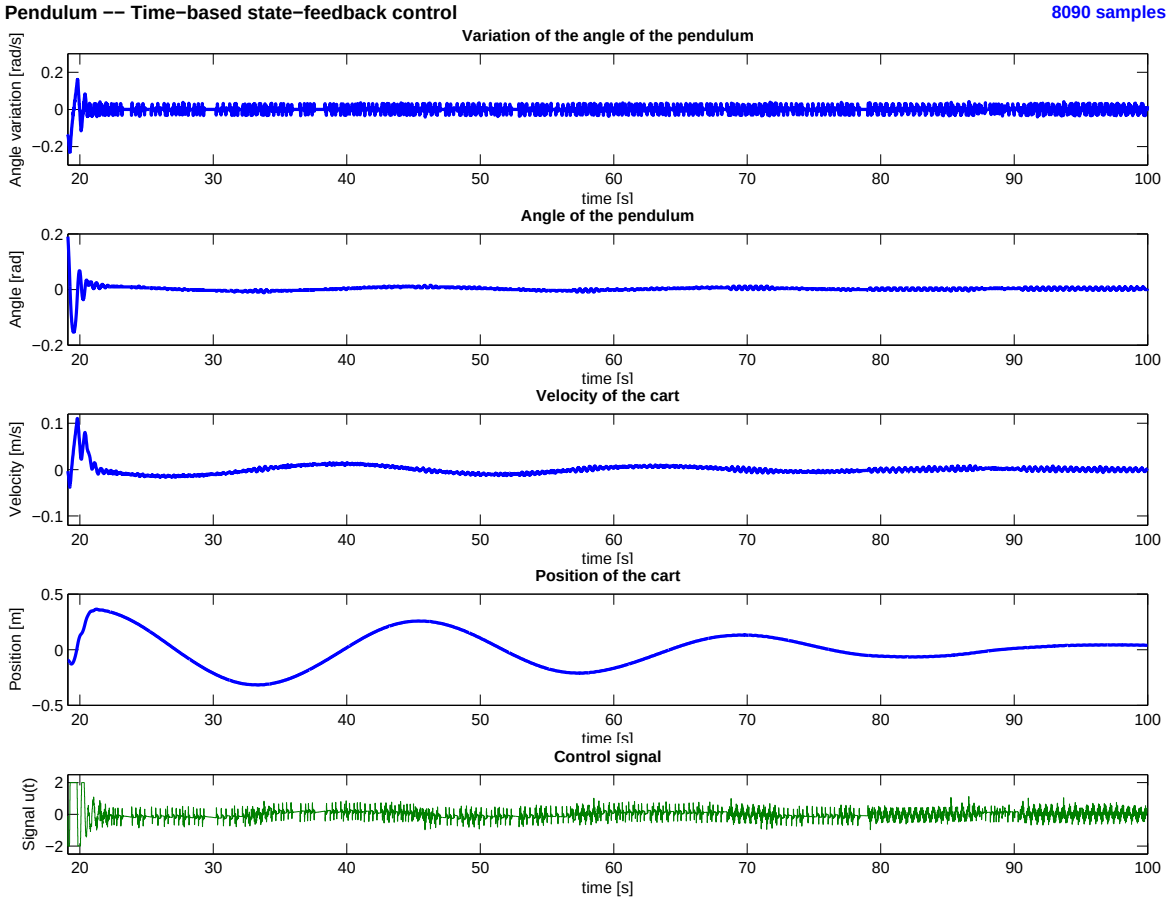


Figure 8.18: Experimental results: stabilizing of the inverted pendulum with a conventional (time-triggered) state-feedback controller.

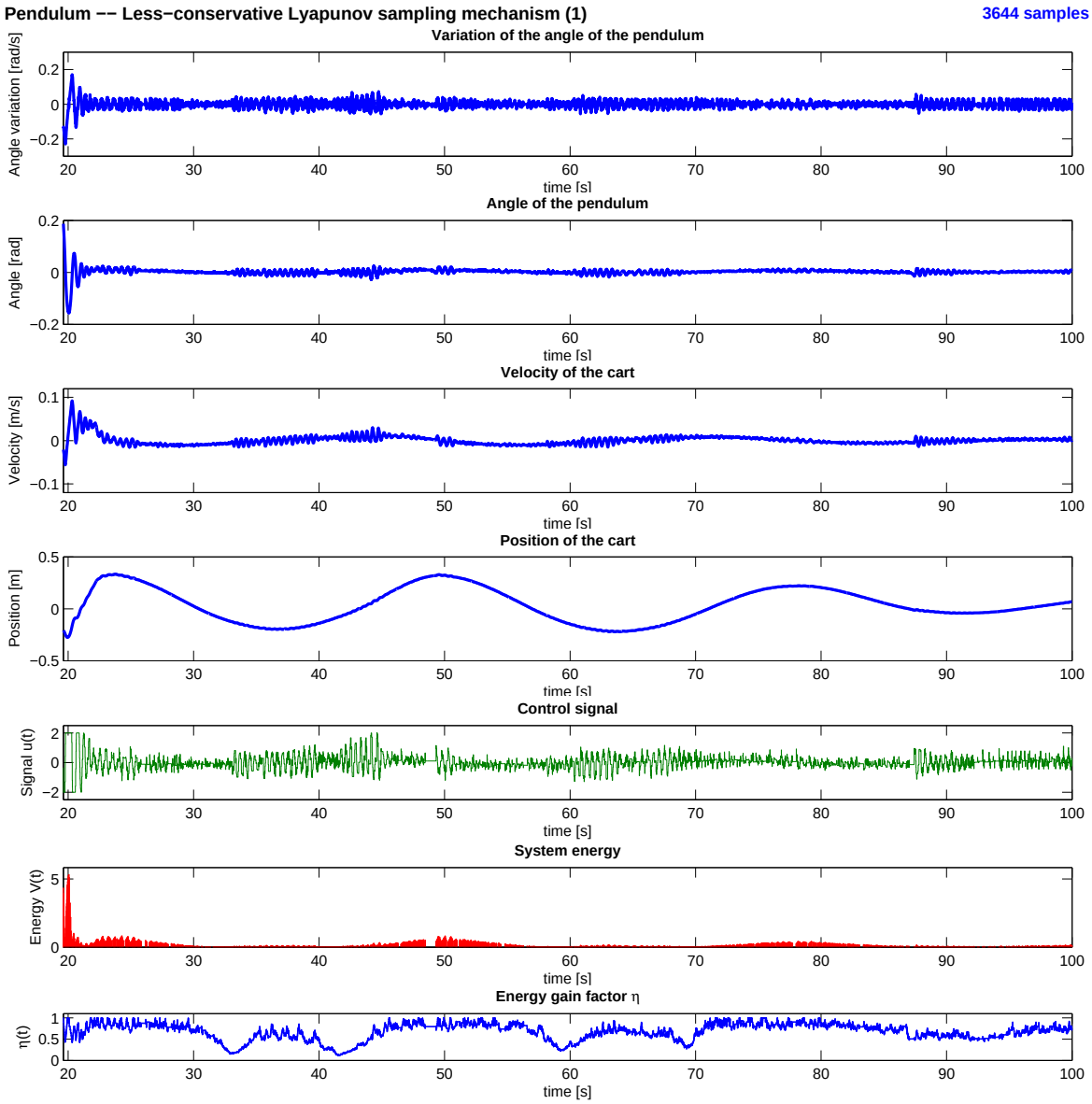


Figure 8.19: Experimental results: stabilizing of the inverted pendulum with an asynchronous state-feedback controller using the less-conservative Lyapunov sampling mechanism (relaxation 3).

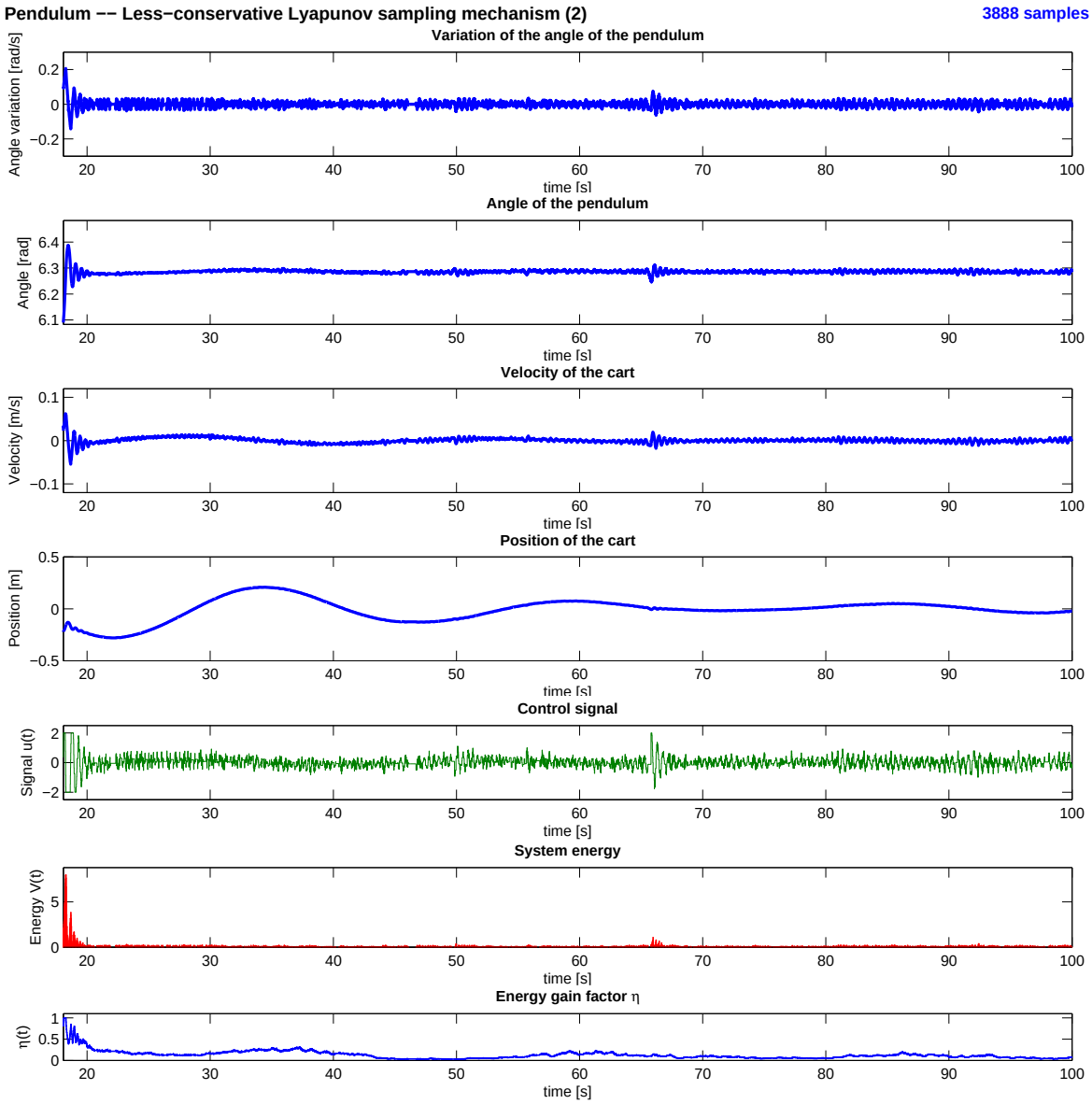


Figure 8.20: Experimental results: stabilizing of the inverted pendulum with another asynchronous state-feedback controller using the less-conservative Lyapunov sampling mechanism (relaxation 3).

Moreover, the energy gain factor is clearly decreased. One could note that the angle of the pendulum is stabilized at 2π here (instead of 0) which is another equilibrium point.

8.4 Synthesis

In this chapter, we proposed to implement for the first time some event-based control strategies in a real-time system, in order to highlight the advantage of such an asynchronous control scheme. Different algorithms were tested in practice. They can be divided into two parts, which refer to chapter 6 and 7 respectively:

1. Six event-based PID controllers using a level-crossing detection mechanism allow to reduce the number of samples with similar performance of the closed-loop system. These algorithms are based on the work of Årzén in [10], where a safety maximum period is added forcing the control to be recomputed even if the measured signal remains unchanged (for stability reason). We shown that this re-computation is not needed thanks to the level detection.

    - The different approaches were implemented on a real testbed in this chapter, where the velocity and the position of an electric motor are controlled. The experimental results confirmed the ones obtained in simulation, and highlighted the robustness of an event-based control law to system errors.
    - Event-triggered controllers with a minimum sampling condition were also tested. This scheme consists in performing a new control signal only if a given amount of time was elapsed since the last sample. This principle leads to reduce again the number of samples.
    - Finally, some extra samples are added after the transients. This results in decreasing the system error even more in order to have a better steady-state interval.

2. On the other hand, four asynchronous state-feedback controllers driven by a Lyapunov sampling mechanism were also implemented in real-time. Lyapunov sampling consists in defining the events related to the value of a Lyapunov function. Existing work in [68] relies on the heavy off-line computation of a parameter, that was removed while still ensuring stability.

    - This approach was tested to control the position of an electric motor and to stabilize an inverted pendulum. Besides a noticeable reduction of the mean control computation cost, robustness to some perturbations was also demonstrated.
    - Eventually, the minimum sampling condition was applied to the Lyapunov-based scheme. It allows to reduce both the number of samples and the control computational cost.

As a result, the advantage of an asynchronous scheme is clearly highlighted for linear and nonlinear systems. Therefore, the encouraging results strongly motivate to continue developing some event-based control strategies (for networked controlled systems for instance).

Conclusion and future works

Summary of the thesis and main contributions

This thesis mainly contributes to reduce both the system energy consumption and the control computational cost in embedded electronic devices. This thesis deals more especially with the problems occurring in sub-micrometric technologies and the benefit in using some asynchronous control approaches.

- The first part of the thesis is devoted to some parasitic physical phenomena, which do not influence the circuit at a higher scale, but become very annoying at a nanometric one. New architectural designs are required and some control loops become essential in such architectures. The chips have to be ***i)*** *low-power* and ***ii)*** *highly robust to process variability* (one of the leading causes for chip failures and delayed schedules). On the first hand, a dynamic power management technique is possible using a voltage scalable processor. Indeed, decreasing the supply voltage and/or the clock frequency allows to reduce the computational activity of the device and, consequently, its power consumption. However, the maximum delay over the critical path of the circuit has to be ensured. This involves restricting both control variables together. Therefore, the key point is to control the energy-performance tradeoff. On the other hand, the process variability affects the entire physical design environment, from power management through timing and signal integrity. It introduces an uncertainty about how a fabricated system will perform. Thus, although a circuit is designed to run at a nominal clock frequency, the fabricated implementation may vary far from this expected performance. As a result, the control law has to adapt itself to this inherent dispersion.

- The asynchronous scheme provides a promising solution for a chip design. For instance, the globally asynchronous locally synchronous systems are chips split into multiple frequency domains, where each domain is synchronous with respect to its clock. Such an architecture can mitigate the impact of process and temperature variations, because a globally asynchronous system does not require that the global frequency was dictated by the longest path delay of the whole chip. Nevertheless, the asynchronous approaches depicted in this thesis are from a control theory point of view. Actually, contrary to the classical scheme which samples the controlled system uniformly in time, an asynchronous (or event-based) framework updates the control signal only when the measurement *sufficiently* changes. This theoretically enables to reduce the number of samples and, consequently, to save computations in the control task. This point is important in all embedded systems with low allocated resources. However, only some primary works exist on event-driven controllers. Moreover, it is difficult to untie the well-established paradigms of the classical

approach. Nevertheless, some encouraging results lead to develop new asynchronous control strategies.

Various strategies have been studied in this thesis, to address the depicted control problems. The most important contributions are highlighted just below.

Contribution on the system energy consumption reduction

The main contributions on controlling the energy-performance tradeoff in electronic chips are presented in part I. They can be summarized as follows:

1. A monocore architecture was proposed to control the voltage and the frequency of a single voltage scalable processor. The main component in this proposal is a controller which monitors the computational activity of the device (its speed) and then calculates the control variables to send to the actuators. They are a Vdd-hopping and a ring oscillator which respectively provide the voltage and the frequency. Finally, this closed-loop architecture makes possible to decrease the power consumption in reducing the supply voltage and/or the clock frequency. Furthermore, the control strategy will have to take care of the computational performance of the device.

2. An intuitive frequency and voltage control strategy was immediately suggested. It allows the measured computational speed to track an average speed setpoint (the speed required to fit the task with its deadline) in order to ensure some good system performance. In fact, a simple PI control law is applied on the integral of the error and gives the (continuously varying) frequency dynamics. The voltage level is then deduced between two possible values from a hysteresis function. At the end, the maximum delay over the critical path is ensured restricting both control variables together (four approaches were proposed for this restriction).

3. A computational speed control strategy was also proposed. Since a task can be executed with two voltage levels, the control law aims at building a more energy-efficient speed setpoint to minimize the penalizing high voltage running time. A robust fast predictive control law (with a low computational cost) is applied to build this new reference. Eventually, the frequency and voltage control is adapted in order to still guarantee some good computational performance.

4. The fast predictive control strategy was then extended to a fully discrete architecture, where the frequency becomes discretely varying. Thus, M voltage and N frequency levels are now possible. This scheme reduces the control computational cost. Moreover, an estimation of the possible computational speeds (when the system runs with a given voltage and frequency levels) allows to calculate the control variables without any information on the system parameters. As a result, this proposal is strongly robust to process variability which occurs in sub-micrometric electronic chips.

5. Some tricks were finally proposed to reduce the control computational cost of the different strategies.

6. An approximated stability analysis was performed to intuitively show that the different control strategies make the system stable.

7. Based on these seminal results, a multicore architecture was suggested too. Several processing nodes, working together on a single chip, are all power supplied with the same supply

voltage and clock frequency. However, a certain degree of freedom is possible thanks to some frequency ratios which allow the different devices to work with a ratio of the clock.

8. An intuitive multicore control strategy was suggested. It is based on the full duplication of the monocore scheme and allows to easily obtain a control algorithm. However, this first strategy multiplies the control computational cost as many times as devices, but it has the advantage to keep the robustness and stability properties of the monocore proposals.

9. A second multicore strategy, based on partial duplication, then consists in not repeating the monocore scheme as many times as devices in order to reduce the control cost. The proposal calculates the frequency ratios focusing on the critical task to treat (the task which requires the maximal frequency/speed to fit with its deadline). At the end, the penalizing high voltage running time of the corresponding device is minimized - to reduce the energy consumption - while guaranteeing the computational performance of the whole system. Furthermore, this control law is still robust to process variability and stable anyway.

10. A fully discrete multicore control scheme was also proposed for an architecture working with M voltage and N frequency levels.

11. Another multicore architecture was suggested. It consists in controlling several chips working with the same supply voltage but with their own clock frequency. The resulting control strategies were not developed for this second architecture but an extension from the previous proposals seems easy.

12. At the end, some simulation results of the different monocore and multicore control strategies were presented. They demonstrate that a fast predictive control technique allows to minimize the energy consumption while guaranteeing some computational performance. The different control strategies give an important reduction of the energy consumption in comparison with a system without DVFS or DVS mechanism. Furthermore, the proposals lead to low control computational needs, more especially for the fully discrete proposals. The control strategies are also highly robust in the case of high dispersion phenomena.

Contribution on the control computational cost reduction

The main contributions on an asynchronous control scheme are detailed in part II. They can be summarized as follows:

1. Different event-based PID control algorithms were proposed. The proposals avoid to recompute the control law after a large amount of time was elapsed while the measurement remains unchanged, as done in the original work of Årzén in [10]. To compensate this safety maximum period, a forgetting factor was imagined. It allows to reduce the impact of the integral gain in the integral part of the control strategy. This approach is somehow similar to the anti-windup one, where the error induced by the saturation has to be compensated. Based on this idea, six controllers without safety limit condition were proposed:

   - algorithm only without safety limit condition,
   - algorithm with saturation of the integral gain,
   - algorithm with an exponential forgetting factor of the sampling interval,
   - algorithm using a hybrid strategy,
   - exponential algorithm with low-cost implementation,

- hybrid algorithm with low-cost implementation.

2. A low computational cost scheme was proposed, adding a minimum sampling interval condition. This was done in order to lighten the transients. Thus, a new control signal is enforced only if a given (minimum) amount of time was elapsed since the last sample.

3. In order to reduce even more the error margin during the steady-state intervals, we also suggested to add some extra samples just after the transients.

4. All the event-based PID algorithms (without safety limit condition, with minimum sampling condition and with extra samples) were compared in simulation, both with the conventional time-triggered controller and the original Årzén's event-based controller. Besides a noticeable reduction of the mean control computation cost, the performance of the closed-loop system was also improved.

5. Different asynchronous state-feedback control law based on Lyapunov sampling were then suggested. Actually, such a stable mechanism, initially proposed in [68], is based on restricting the energy gain factor in the Lyapunov sampling condition (which defines the length of the level detection), and yet, calculating this parameter requires to execute a computationally heavy off-line algorithm. As a result, four less-conservative methods were developed relaxing the constraints on the energy gain factor and making varying this parameter:

   - slowly decrease/drastically increase the energy gain factor,
   - improvement for an on-line running,
   - slowly decrease/slowly increase the energy gain factor,
   - a more formal variation of the energy gain factor.

   Three of the last proposals allow a fully on-line running, which results in a very low control computational cost.

6. The minimum sampling interval condition originally introduced for some PID controllers (to lighten the transients) was also extended to the asynchronous state-feedback controllers.

7. Eventually, the Lyapunov theory allows to prove that an asynchronous scheme can decrease the computational cost while ensuring the stability of the system, even if the system is not sampled during a long amount of time.

8. At the end, the different event-driven (PID and state-feedback) control strategies were implemented for the first time in some real-time systems. The experimental results strongly highlight the advantage of the asynchronous control scheme.

List of publications

International conference papers with proceedings

1. S. Durand, N. Marchand. "Further results on event-based PID controller". In proceedings of the ***10th European Control Conference (ECC'09)***, Budapest, Hungary (2009).

2. S. Durand, N. Marchand. "An event-based PID controller with low computational cost". In proceedings of the ***8th International Conference on Sampling Theory and Applications (SampTA'09)***, *special session on Sampling and Industrial Applications*, Marseille, France (2009).

3. S. Durand, N. Marchand. "Fast Predictive Control of Micro Controller's Energy-Performance Tradeoff". In proceedings of the ***3rd IEEE Multi-conference on Systems and Control, 18th IEEE International Conference on Control Applications (CCA'09)***, *tutorial session on Low Power Electronic*, Saint Petersburg, Russian Federation (2009).

4. S. Durand, N. Marchand. "Energy Consumption Reduction with Low Computational Needs in Multicore Systems with Energy-Performance Tradeoff". In proceedings of the ***48th IEEE Conference on Decision and Control (CDC'09)***, Shanghai, China (2009).

5. H. Zakaria, S. Durand, L. Fesquet, N. Marchand. "Integrated Asynchronous Regulation for Nanometric Technologies". In proceedings of the ***1st European workshops on CMOS Variability (VARY'10)***, Montpellier, France (2010).

6. S. Durand, N. Marchand. "Fully Discrete Control Scheme of the Energy-Performance Tradeoff in Embedded Electronic Devices". In proceedings of the ***18th World Congress of IFAC***, Milano, Italia (2011).

7. S. Durand, N. Marchand, J.F. Guerrero Castellanos. "Simple Lyapunov Sampling for Event-Driven Control". In proceedings of the ***18th World Congress of IFAC***, Milano, Italia (2011).

Patents

1. S. Durand, N. Marchand. "Dispositif de commande d'alimentation d'un calculateur". ***Patent nř 09/004686*** (March 2009).
2. S. Durand, N. Marchand. "Dispositif de commande d'alimentation d'un calculateur". ***Patent nř 09/01576*** (October 2009).

Journals **(under preparation)**

1. S. Durand, N. Marchand. "Control of the Energy-Performance Tradeoff in Embedded Electronic Devices: a Robust Approach to Process Variability".
2. S. Durand, N. Marchand, J.F. Guerrero Castellanos. "Event-Based PID Control Strategies without Safety Limit Condition".
3. S. Durand, N. Marchand, J.F. Guerrero Castellanos. "Simple Lyapunov Sampling".

Miscellaneous

1. H. Zakaria, L. Fesquet, S. Durand, C. Albea-Sanchez, Y. Thonnart, C. Canudas-de-Wit, N. Marchand. "Integrated Asynchronous Regulation for Nanometric Technologies: Application to an Embedded Parallel System". In ***MINATEC CROSSROADS***, Grenoble, France (2008).

Perspectives

Following the investigations described in this thesis, many perspectives can be considered to complete and improve this work. Here some possible issues are given:

1. A physical implementation of the energy-performance tradeoff control (in part I) is important in order to test the real performance of the proposed control strategies and, therefore, validate our work. This will be performed on a nanometric chip, in collaboration with the CEA-LETI laboratory in a post-doctoral work. In addition, the results will demonstrate the high robustness to process variability before implementing it in some manufactured SoCs.

2. The encouraging results on the asynchronous control scheme (in part II) strongly motivate to develop some new event-driven control strategies for some more general systems:

   - Actually, the value of the control parameters in the different PID controllers were obtained by pole placement of the closed-loop system in the time-triggered case. The event-based controllers are designed with the same values and the aim is then to be as close as possible of the time-triggered closed-loop shaping. However, a better solution would be to directly calculate these parameters for the asynchronous controllers. New tools have to be developed in this sense.
   - An extension of the proposed asynchronous state-feedback control laws can be performed considering that all the states of the system are not measured. This will lead to use an output-feedback strategy.
   - In order to reduce even more the control computational cost of asynchronous controllers based on Lyapunov functions, it would be interesting to choose a simple matrix P (as quickly presented in section 8.3 when stabilizing the inverted pendulum). Indeed, a matrix with a lot of null values reduces the number of operations required to detect the events. A diagonilization of this matrix seems to be a suitable solution and has to be more analyzed.
   - The Lyapunov sampling mechanism would have to be adapted to some Control-Lyapunov functions or Input-to-State Stability (ISS) analysis.
   - The asynchronous strategies could also be adapted for networked controlled systems for instance, where allocated resources are very low. The event-based scheme would also be extended in the communication decisions.

- It would be attractive to develop a physical demonstrator. This could be used to highlight the advantage of an event-based technique in order to convince industrial firms. Such a system could be a cruise control mechanism implemented on a remote control car for example.

3. Eventually, it could be interesting to develop an asynchronous control of the energy-performance tradeoff, mixing both parts of the thesis. A first solution could be to simply make the clock of the controller varying - and consequently its sampling period - with respect to the clock frequency of the system. A second solution would be to specifically build an event-based approach to control the energy consumption on a globally asynchronous locally synchronous chip.

References

[1] F. Aeschlimann, E. Allier, L. Fesquet, and M. Renaudin. Asynchronous FIR filters: towards a new digital processing chain. In *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems*, pages 198–206, 2004.

[2] F. Akopyan, R. Manohar, and A. Apsel. A level-crossing flash asynchronous analog-to-digital converter. In *International Symposium on Asynchronous Circuits and Systems*, 2006.

[3] M. Alamir. *Stabilization of Nonlinear Systems Using Receding-Horizon Control Schemes: A Parametrized Approach for Fast Systems.* Lecture Notes in Control and Information Sciences. Springer-Verlag, London, 2006.

[4] C. Albea Sánchez. *Nonlinear Control Design for Inverter And Converter.* PhD thesis, University of Grenoble (France) and University of Seville (Spain), 2010.

[5] E. Allier, G. Sicard, L. Fesquet, and M. Renaudin. A new class of asynchronous a/d converters based on time quantization. In *Ninth International Symposium on Asynchronous Circuits and Systems*, pages 196–205, 2003.

[6] E. Allier, G. Sicard, L. Fesquet, and M. Renaudin. Asynchronous level crossing analog to digital converters. In *Measurement (Special Issue on ADC Modelling and Testing)*, volume 37, pages 286–309, 2004.

[7] A.-M. Alt and D. Simon. Control strategies for H.264 video decoding under resources constraints. In *Fifth International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks (FeBID 2010)*, 2010.

[8] A. Anta and P. Tabuada. Self-triggered stabilization of homogeneous control systems. In *Proc. of the IEEE American Control Conference*, 2008.

[9] A. Anta and P. Tabuada. To sample or not to sample: Self-triggered control for nonlinear systems. *IEEE Transactions on Automatic Control*, 55:2030 – 2042, 2010.

[10] K.-E. Årzén. A simple event-based PID controller. In *Preprints of the 14th World Congress of IFAC*, Beijing, P.R. China, 1999.

[11] K. Åström and B. Bernhardsson. Comparison of periodic and event based sampling for first-order stochastic systems. In *Preprints 14th World Congress of IFAC*, page 301Ű306, 1999.

[12] K. Åström and B. Bernhardsson. Comparison of Riemann and Lebesque sampling for first order stochastic systems. In *Proceedings of the 41st IEEE Conference on Decision and Control*, 2002.

[13] K. Åström and K. Furuta. Swinging up a pendulum by energy control. *Automatica*, 36:287–295, 2000.

[14] K. Åström and T. Hägglund. *PID controllers: theory, design, and tuning, 2nd Edition.* The Instrumentation, Systems, and Automation Society, 1995.

[15] K. Åström and R. Murray. *Feedback Systems: An Introduction for Scientists and Engineers.* Princeton University Press, 2008.

[16] K. Åström and B. Wittenmark. *Computer Controlled Systems, 3rd Edition.* Prentice Hall, 1997.

[17] B. Bidégaray-Fesquet and L. Fesquet. Signal processing for asynchronous systems (spass) toolbox v1.0.0, 2006. http:// ljk.imag.fr/ membres/ Brigitte.Bidegaray/ SPASS/.

[18] B. Bidégaray-Fesquet and L. Fesquet. A fully nonuniform approach to fir filtering. *International Conference on Sampling Theory and Applications*, 2009.

[19] D. Bland and A. Tarczynski. Optimum nonuniform sampling sequence for alias frequency suppression. In *International Symposium on Circuits and Systems, ISCAS*, volume 4, pages 2693–2696, 1997.

[20] T. Burd and R. Brodersen. Processor design for portable systems. In *The Journal of VLSI Signal Processing*, volume 13, pages 203–221, 1996.

[21] T. Burd, T. Pering, A. Stratakos, and R. Brodersen. A dynamic voltage scaled microprocessor system. In *IEEE International Solid-State Circuits Conference Digest of Technical Papers*, volume 35, pages 1571–1580, 2000.

[22] C. Canudas De Wit, K. Crisanto Vega, and J. Jaglin. Non uniform sampling and entrohopy coding in networked controlled linear systems. In *Proceedings of the European Control Conference*, 2007.

[23] C. Canudas De Wit, J. Jaglin, and C. Siclet. Energy-aware 3-level coding and control co-design for sensor network systems. In *16th IEEE International Conference on Control Applications*, 2007.

[24] A. Chandrakasan and R. Brodersen. Minimizing power consumption in digital cmos circuits. In *Proceedings of the IEEE*, volume 83, pages 498–523, 1995.

[25] M. Donkers and W. Heemels. Output-based event-triggered control with guaranteed l_{inf}-gain and improved event-triggering. In *Proc. of the IEEE Conference on Decision and Control*, 2010.

[26] A. Eqtami, D. Dimarogonas, and K. Kyriakopoulos. Event-triggered control for discrete-time systems. In *Proc. of the IEEE American Control Conference*, 2010.

[27] S. Fairbanks and S. Moore. Analog micropipeline rings for high precision timing. In *Proceeding of the International Symposium on Advanced Research in Asynchronous Circuits and Systems*, pages 41–50, 2004.

[28] L. Fesquet and H. Zakaria. Controlling energy and process variability in system-on-chips: needs for control theory. In *Proceedings of the 3rd IEEE Multi-conference on Systems and Control - 18th IEEE International Conference on Control Applications*, 2009.

[29] K. Flautner, D. Flynn, D. Roberts, and D. Patel. An energy efficient soc with dynamic voltage scaling. In *proceedings of the Design, Automation and Test in Europe Conference and Exhibition*, pages 324 – 327, 2004.

[30] K. Flautner, S. K. Reinhardt, and T. N. Mudge. Automatic performance setting for dynamic voltage scaling. In *Mobile Computing and Networking*, pages 260–271, 2001.

[31] F. Gomez Estern, C. Canudas De Wit, F. Rubio, and J. Fornes. Adaptative delta-modulation coding for networked controlled systems. In *IEEE American Control Conference*, 2007.

[32] J. Hamon, L. Fesquet, B. Miscopein, and M. Renaudin. High-level time-accurate model for the design of self-timed ring oscillators. In *14th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 29–38, 2008.

[33] W. Heemels, R. J. A. Gorterb, A. van Zijla, P. P. J. van den Boscha, S. Weilanda, W. H. A. Hendrixa, and M. R. Vonder. Asynchronous measurement and control: a case study on motor synchronization. *Control Engineering Practice*, 7:1467–1482, 1999.

[34] W. Heemels, J. Sandee, and P. van den Bosch. Analysis of event-driven controllers for linear systems. *International journal of control*, 81:571–590, 2009.

[35] E. Hendricks, M. Jensen, A. Chevalier, and T. Vesterholm. Problems in event based engine control. In *Proc. of the IEEE American control conference*, 1994.

[36] T. Ishihara and H. Yasuura. Voltage scheduling problem for dynamically variable voltage processors. In *Proceedings of the International Sympsonium on Low Power Electronics and Design*, pages 197–202, 1998.

[37] J. Jaglin, C. Canudas De Wit, and C. Siclet. Delta modulation for multivariable centralized linear networked controlled systems. In *Conference on Decision and Control*, 2008.

[38] M. Krstic, E. Grass, F. Gurkaynak, and P. Vivet. Globally asynchronous, locally synchronous circuits: Overview and outlook. *IEEE Design and Test of Computers*, 24:430–441, 2007.

[39] W. Kuzmicz, E. Piwowarska, A. Pfitzner, and D. Kasprowicz. Static power consumption in nano-cmos circuits: Physics and modelling. In *Proceeding of the 14th International Conference Mixed Design of Integrated Circuits and Systems*, 2007.

[40] I. Lopez, C. Abdalla, and C. Canudas De Wit. Gain-scheduling multi-bit delta-modulator for networked controlled system. In *Proceedings of European Control Conference*, 2007.

[41] Y. Lu and G. De Micheli. Comparing system-level power management policies. *IEEE Design and test of Computers*, 18:10–19, 2001.

[42] J. Lunze and D. Lehmann. A state-feedback approach to event-based control. *Automatica*, 46:211–215, 2010.

[43] N. Marchand. Stabilization of Lebesgue sampled systems with bounded controls: the chain of integrators case. In *Proceedings of the 17th IFAC World Congress*, 2008.

[44] D. Marculescu and E. Talpes. Energy awareness and uncertainty in microarchitecture-level design. *IEEE Micro*, 25:64–76, 2005.

[45] F. Marvasti. *Nonuniform Sampling: Theory and Practice.* Kluwer Academic/Plenum Publishers, 2001.

[46] D. Q. Mayne and H. Michalska. Receding horizon control of nonlinear systems. *IEEE Trans. on Automatic Control*, 35(7):814–824, 1990.

[47] M. J. Mazo, A. Anta, and P. Tabuada. On self-triggered control for linear systems: Guarantees and complexity. In *Proceedings of the 10th European Control Conference*, 2009.

[48] S. Miermont, P. Vivet, and M. Renaudin. A power supply selector for energy- and area -efficient local dynamic voltage scaling. In *PATMOS'07: 17th International Workshop on Power and Timing Modeling, Optimization and Simulation*, pages 556–565, 2007.

[49] T. Minka. The lightspeed matlab toolbox v2.2, 2009. http://research.microsoft.com/en-us/um/people/minka/software/lightspeed/.

[50] R. Murueta Fortiz. Estabilización de un péndulo invertido sobre base móvil a partir del equilibrio estable mediante control híbrido (in Spanish). Master's thesis, Benemérita Universidad Autónoma de Puebla y Facultad de Ciencias Físico-Matemáticas, 2009.

[51] A. Nicoli. Achieving yield in the nanometer age. In *Mentor Graphics Corp.*, 2007.

[52] B. Pangrle and K. Shekhar. Leakage power at 90nm and below. In *Synopsis Inc.*, 2005.

[53] T. Pering, T. Burd, and R. Brodersen. Voltage scheduling in the lparm microprocessor system. In *Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED)*, pages 96–101, 2000.

[54] J. Pouwelse, K. Langendoen, and H. Sips. Dynamic voltage scaling on a low-power microprocessor. In *Proceedings of the 7th Annual International Conference on Mobile Computing and Networking*, pages 251–259, 2001.

[55] B. Romanescu, M. Bauer, D. Sorin, and S. Ozev. Reducing the impact of process variability with prefetching and criticality-based resource allocation. In *Proceedings of the 16th International Conference on Parallel Architecture and Compilation Techniques*, 2007.

[56] J. Sánchez, M. Guarnes, and S. Dormido. On the application of different event-based sampling strategies to the control of a simple industrial process. *Sensors*, 9:6795–6818, 2009.

[57] J. Sánchez, M. Guarnes, S. Dormido, and A. Visioli. Comparative study of event-based control strategies: An experimental approach on a simple tank. In *Proceedings of the 10th European Control Conference*, 2009.

[58] J. Sandee, W. Heemels, and P. van den Bosch. Event-driven control as an opportunity in the multidisciplinary development of embedded controllers. In *Proceedings of American Control Conference*, pages 1776–1781, 2005.

[59] O. Sename, D. Simon, and D. Robert. Feedback scheduling for real-time control of systems with communication delays. In *Proceedings of the IEEE Conference on Emerging Technologies and Factory Automation*, volume 2, 2003.

[60] D. Simon, D. Robert, and O. Sename. Robust control/scheduling co-design: application to robot control. In *Proceedings of the IEEE Symposium on Real-Time and Embedded Technology and Applications*, pages 118–127, 2005.

[61] E. D. Sontag. *Mathematical Control Theory: Deterministic Finite Dimensional Systems. Second Edition*. Springer-Verlag, 1998.

[62] K. Sultan. Inverted pendulum: Analysis, design and implementation, 2003.

[63] P. Tabuada. Event-triggered real-time scheduling of stabilizing control tasks. *IEEE Transactions on Automatic Control*, 52:1680 – 1685, 2007.

[64] C. Van Berkel, M. Josephs, and S. Nowick. Scanning the technology: Applications of asynchronous circuits. *Proceedings of the IEEE*, 87(2):223–233, 1999.

[65] K. Van Berkel, R. Burgess, J. Kessels, M. Roncken, F. Schalij, and A. Peeters. Asynchronous circuits for low power: a DCC error corrector. *IEEE Design and Test of Computers*, 11(2):22–32, 1994.

[66] H. Van Gageldonk, K. van Berkel, A. Peeters, D. Baumann, D. Gloor, and G. Stegmann. An asynchronous low-power 80C51 microcontroller. In *Proceedings of the 4th International Sympsonium on Advanced Research in Asynchronous Circuits and Systems*, pages 96–107, 1998.

[67] A. Varma, B. Ganesh, M. Sen, S. Choudhury, L. Srinivasan, and J. Bruce. A control-theoretic approach to dynamic voltage scheduling. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems*, pages 255–266, 2003.

[68] M. Velasco, P. Martí, and E. Bini. On lyapunov sampling for event-driven controllers. In *Proceedings of the 48th IEEE Conference on Decision and Control*, 2009.

[69] M. Velasco, P. Martí, and J. M. Fuertes. The self-triggered task model for real-time control systems. In *Proc. of the 24th IEEE Real-Time Systems Symposium*, 2003.

[70] K. von Arnim, E. Borinski, P. Seegebrecht, H. Fiedler, R. Brederlow, R. Thewes, J. Berthold, and C. Pacha. Efficiency of body biasing in 90-nm cmos for low-power digital circuits. *IEEE Journal of Solid-state Circuits*, 40:1549Ũ1556, 2005.

[71] X. Wang and M. Lemmon. State based self-triggered feedback control systems with l_2 stability. In *Proc. of the IFAC World Congress*, 2008.

[72] X. Wang and M. Lemmon. Self-triggered feedback control systems with finite-gain l_2 stability. *IEEE Transactions on Automatic Control*, 54:452, 2009.

[73] E. Yahya, O. Elissati, H. Zakaria, L. Fesquet, and M. Renaudin. Programmable/stoppable oscillator based on self-timed rings. In *15th IEEE International Symposium on Asynchronous Circuits and Systems*, 2009.

[74] H. Zakaria. *Integrated asynchronous regulation for decananometric technologies: Application to an embedded reconfigurable parallel system.* PhD thesis, University of Grenoble (France), 2010.

[75] V. Zebilis and C. P. Sotiriou. Controlling event spacing in self-timed rings. In *11th IEEE International Symposium on Asynchronous Circuits and Systems*, pages 109 – 115, 2005.

[76] Y. Zhu and F. Mueller. Feedback dynamic voltage scaling dvs-edf scheduling: Correctness and pid-feedback. In *Workshop on Compilers and Operating Systems for Low Power*, 2003.

UNIVERSITÉ DE GRENOBLE

Reduction of the energy consumption in embedded electronic devices with low control computational cost

Abstract: The demand of electronic components in all embedded and miniaturized applications encourages to develop low-cost components, in term of energy consumption and computational resources. Actually, the power consumption can be reduced when decreasing the supply voltage and/or the clock frequency, but with the effect that the device runs more slowly in return. Nevertheless, a fast predictive control strategy allows to dynamically manage this tradeoff in order to minimize the energy consumption while ensuring good performance of the device. Furthermore, the proposals are highly robust to tackle variability which is a real problem in nanometric systems on chip. Some issues are also suggested in this thesis to reduce the control computational cost. Contrary to a time-triggered system where the controller calculates the control law at each (constant and periodic) sampling time, an event-based controller updates the control signal only when the measurement sufficiently changes. Such a paradigm hence calls for resources whenever they are indeed necessary, that is when required from a performance or stability point of view for instance. The idea is to soften the computational load by reducing the number of samples and consequently the CPU utilization. Some simulation and experimental results eventually validate the interest of such an approach.

Keywords: Energy/performance tradeoff, nanometric electronic circuits, process variability, event-based control

Commande faible coût et réduction de la consommation d'énergie dans les systèmes électroniques embarqués

Résumé: La course à la miniaturisation des circuits électroniques pousse à développer des systèmes faible coût, que ce soit en terme de consommation d'énergie ou de ressources de calcul. Il est ainsi possible de réduire la consommation en diminuant la tension d'alimentation et/ou la fréquence d'horloge, mais ceci a pour conséquence de diminuer aussi la vitesse de fonctionnement du circuit. Une commande prédictive rapide permet alors de gérer dynamiquement un tel compromis, de manière à ce que la consommation d'énergie soit minimisée tout en garantissant de bonnes performances. Les stratégies de commande proposées ont notamment l'avantage d'être très robustes aux dispersions technologiques qui sont un problème récurrent dans les nanopuces. Des solutions sont également proposées afin de réduire le coût de calcul du contrôleur. Les systèmes à échantillonnage non-uniforme, dont la loi de commande est calculée et mise à jour lorsqu'un événement est déclenché, sont ainsi étudiés. Ce principe permet de réduire le nombre d'échantillons et, par conséquent, d'économiser des ressources de calcul, tout en garantissant de bonnes performances du système commandé. Des résultats de simulation, et surtout expérimentaux, valident finalement l'intérêt d'utiliser une telle approche.

Mots-clé: Compromis énergie/performance, circuits électroniques nanométriques, variabilité du procédé de fabrication, commande déclenchée par événements

INRIA Grenoble - Rhône-Alpes
Inovallée
655 avenue de l'Europe,
Montbonnot
38 334 Saint Ismier Cedex, France

GIPSA-Lab, département
Automatique
Bâtiment ENSE3
961 rue de la Houille Blanche
BP 46
38 402 Grenoble Cedex, France