Modélisation Incrémentale des Processeurs Embarqués pour l'Estimation des Caractéristiques et le Diagnostic

Oussama Djedidi

Thèse de Doctorat

de

l'Université d'Aix-Marseille

préparée au

Laboratoire d'Informatique et Systèmes
(LIS UMR 7020)

École Doctorale en Mathématiques et Informatique de Marseille
(ED 184)
Spécialité de doctorat : Génie informatique, Automatique et Traitement du Signal

Par
Oussama DJEDIDI

Modélisation Incrémentale des Processeurs Embarqués pour l'Estimation des Caractéristiques et le Diagnostic

---

Soutenue publiquement, le 10 Décembre 2019
Devant le jury composé de :

M. Abdel AITOUCHE	Professeur YNCREA-HEI, CRIStAL, Université de Lille	Rapporteur
M. Moamar SAYED MOUCHAWEH	Professeur des Universités École des Mines de Douai	Rapporteur
Mme Karen GODARY-DEJEAN	Maître de Conférences Polytech Montpellier, LIRMM 5506	Examinatrice
M. Ghaleb HOBLOS	Professeur des Universités ESIGELEC, IRSEEM	Examinateur
M. Aziz NAAMANE	Maître de Conférences, HDR Polytechnique Marseille, LIS UMR 7020	Examinateur
M. Nacer M'SIRDI	Professeur des Universités Polytechnique Marseille, LIS UMR 7020	Directeur de thèse
M. Mohand DJEZIRI	Maître de Conférences, HDR Aix-Marseille Université, LIS UMR 7020	Co-directeur de thèse

A mes parents.

ABSTRACT

Systems on Chip are increasingly embedded in safety-critical systems, such as aeronautical systems and energy production equipment. Such technological evolution allows for significant improvements in performance but presents limits in terms of reliability and security. Therefore, the development of new tools for the monitoring and diagnosis of embedded electronic systems—Systems on Chip, in particular—is currently one of the scientific challenges to overcome, in order to ensure a broader and safer use of these systems in safety-critical equipment.

The work presented in this thesis aims to develop an approach for detecting and identifying drifts in embedded Systems of Chips characteristics and performance. The proposed approach is based on an incremental model built from reusable and exchangeable modules able to adapt and accommodate the broad range of Systems on Chips available on the market. This model is then used to estimate a set of characteristics relating to the state of operation of the SoC.

The diagnostic algorithm developed in this work consists of generating drift signals though the online comparison of the estimated characteristics to those measured. Then, the assessment of residuals and decision making are performed by statistical methods appropriate to the nature of each drift.

The developed approach has been experimentally validated on different Systems on Chip, as well as on a demonstrator developed as part of this work. The obtained experimental results validate and show the efficiency and robustness of the incremental model and the monitoring algorithm.

RESUMÉ

Les Systèmes-sur-Puce (*Systems on Chip*, SoC) sont de plus en plus embarqués dans des systèmes à risque comme les systèmes aéronautiques et les équipements de production d'énergie. Cette évolution technologique permet un gain de temps et de performance, mais présente des limites en termes de fiabilité et de sécurité. Ainsi, le développement d'outils de surveillance et de diagnostic des systèmes électroniques embarqués, en particuliers les SoC, est devenu l'un des verrous scientifiques à lever pour assurer une large utilisation de ces systèmes dans les équipements à risque en toute sécurité.

Ce travail de thèse s'inscrit dans ce contexte, et a pour objectif le développement d'une approche de détection et identification des dérives des performances des SoC embarqués. L'approche proposée est basée sur un modèle incrémental, construit à partir de modules réutilisables et échangeables pour correspondre à la large gamme de SoC existants sur le marché. Le modèle est ensuite utilisé pour estimer un ensemble de caractéristiques relatives à l'état de fonctionnement du SoC.

L'algorithme de diagnostic développé dans ce travail consiste à générer des indices de dérives par la comparaison en ligne des caractéristiques estimées à celles mesurées. L'évaluation des résidus et la prise de décision sont réalisées par des méthodes statistiques appropriées à la nature de chaque indice de dérive.

L'approche développée a été validée expérimentalement sur des SoC différents, ainsi que sur un démonstrateur développé dans le cadre de ce travail. Les résultats expérimentaux obtenus, montrent l'efficacité et la robustesse de l'approche développée.

TRAVAUX PUBLIÉS

2019	[1]	**Oussama Djedidi**, Nacer M'Sirdi, and Aziz Naamane. **Adaptive Estimation of the Thermal Behavior of CPUGPU SoCs for Prediction and Diagnosis**. In *IMAACA 2019 – Proceedings of the International Conference on Integrated Modeling and Analysis in Applied Control and Automation*, 2019, pages 93–98, Lisbon, Portugal, September 2019. Dauphin-Tanguy Bruzzone and Junco Eds.
	[2]	**Oussama Djedidi**, Mohand Djeziri, and Samir Benmoussa. **Failure Prognosis of Embedded Systems Based on Temperature Drift Assessment**. In *IMAACA 2019 – Proceedings of the International Conference on Integrated Modeling and Analysis in Applied Control and Automation*, 2019, volume 1, pages 11–16, Lisbon, Portugal, September 2019. Bruzzone, Dauphin-Tanguy and Junco Eds.
	[3]	**Oussama Djedidi**, Mohand DJEZIRI, and Kouider nacer M'SIRDI. **Data-driven monitoring of Systems On Chip for Multifunction Modular Cockpit Display**. *5th Summer School on INtelligent signal processing for FrontIEr Research and Industry (INFIERI 2019)*, May 2019. Poster.
2018	[4]	**Oussama Djedidi**, Mohand Djeziri, and Nacer M'SIRDI. **Data-Driven Approach for Feature Drift Detection in Embedded Electronic Devices**. *IFAC-PapersOnLine*, 51(24) :1024 – 1029, 2018 .
	[5]	**Oussama Djedidi**, Mohand DJEZIRI, Nacer M'SIRDI, and Aziz Naamane. **Constructing an Accurate and a High-Performance Power Profiler for Embedded Systems and Smartphones**. In *MSWIM '18 21st ACM Int'l Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems*, Montreal, Canada, October 2018. ACM Press.
	[6]	**Oussama Djedidi**, Mohand DJEZIRI, Nacer M'SIRDI, and Aziz Naamane. **A Novel Easy-to-construct Power Model for Embedded and Mobile Systems - Using Recursive Neural Nets to Estimate Power Consumption of ARM-based Embedded Systems and Mobile Devices**. In *15th International Conference on Informatics in Control, Automation and Robotics*, Porto, Portugal, July 2018. SCITEPRESS - Science and Technology Publications.

	[7]	**[Vidéo] Oussama Djedidi**, Mohand Arab Djeziri, and Kouider nacer M'SIRDI. **Prototyping of a Data-driven monitoring of Systems On Chip for Multifunction Modular Cockpit Display (MMCD) Project**, June 2018.
	[8]	**Oussama Djedidi**, Mohand Djeziri, and Nacer M'Sirdi. **Modélisation orientée diagnostic des composants électroniques embarqués**. In *Congrès National de la Recherche des IUT (CNRIUT'2018)*, Aix en Provence, France, June 2018.
2017	[9]	**Oussama Djedidi**, Mohand Arab Djeziri, Nacer M'SIRDI, and Aziz Naamane. **Modular Modelling of an Embedded Mobile CPU-GPU Chip for Feature Estimation**. In *14th International Conference on Informatics in Control, Automation and Robotics*, volume 1, pages 338–345, Madrid, Spain, July 2017. SCITEPRESS - Science and Technology Publications.

«Je suis comme tous les scientifiques, un footballeur raté.»

—Tarek Khaled[1]

REMERCIEMENT

Les travaux de cette thèse font partie du projet MMCD soutenu et financé par la Banque Publique d'Investissement (Bpifrance), à qui nous adressons nos remerciements. Ils ont été réalisés au sein de l'équipe SASV du Laboratoire d'Informatiques et Systèmes à l'Université d'Aix-Marseille, ou j'ai eu le privilège d'être entouré par des collègues généreux et très agréable.

Je tiens à exprimer mes vifs remerciements à mes deux Directeurs de thèse Dr. Mohand A. DJEZIRI et Prof. Nacer K. M'SIRDI, pour leurs conseils, leur soutient et leur grande patience dont ils ont su faire preuve malgré leurs charges. Ils ont toujours été à l'écoute et très disponibles tout au long de la réalisation de cette thèse.

J'exprime ma gratitude à Prof. Abdel AITOUCHE et Prof. Moamar SAYED MOUCHAWEH de m'avoir fait l'honneur d'accepter l'évaluation de mon travail. Je tiens à remercier aussi Dr. David ANDREU, Dr. Karen GODARY-DEJEAN, Prof. Ghaleb HOBLOS et Dr. Aziz NAAMANE pour avoir accepté l'invitation pour faire partie du jury et examiner mes travaux de recherche.

J'adresse mes remerciements également à Mme. Nadia Creignou et Mr. Thierry Gallouet, directrice et directeur adjoint de l'ED 184 à Marseille, pour leur soutien.

Enfin, j'adresse mes plus sincères remerciements, à mes parents sans eux rien de tout cela ne serait possible. Je remercie aussi tous mes proches, ma famille et amis, qui m'ont toujours soutenu et encouragé, particulièrement A.B. ANATHALLEE, A. FEDDAOUI-PAPIN, H. HADDOU-BENDERBAL, et T. KHALED.

1. Le 8 mars 2018 durant le match Olympique Marseille Vs Athletico Bilbao [3-1].

TABLE DES MATIÈRES

TABLE DES FIGURES

LISTE DES TABLEAUX

NOMENCLATURE

Abréviation

ADB Android Debug Bridge

API Application Programming Interface (Interface de programmation)

APU Accelerated Processing Unit (Unité de Calcul Accélérée)

ARMAX Autoregressive–Moving-Average with Exogenous Inputs [Model] ([Modèle] Autorégressif et moyenne-mobile à entrée exogène)

CPU Central Processing Unit (Unité centrale de traitement—Processeur)

DVFS Dynamic Voltage and Frequency Scaling (Ajustement dynamique du la tension et la fréquence)

FDI Fault Detection and Isolation (Détection et d'Isolation de défauts)

FSM Finite State Machines (Machines à état finis)

GPU Graphics Processing Unit (Processeur graphique)

HSA [The] Heterogeneous System Architecture (L'Architecture des Systèmes Hétérogènes)

IDE Integrated Development Environment (Environnement de développement)

IoT Internet of Things (Internet des Objets)

k-NN *k* Nearest Neighbor (*k* plus proches voisins)

MAE Mean Absolute Error (Erreur moyenne absolue)

MMCD Multi-Modular Cockpit Display

MOR Memory Occupation Rate (Taux d'occupation de mémoire)

NARX Nonlinear AutoRegressive [Model] with eXogenous inputs ([Modèle] autorégressif non-linéaire à entrées exogènes)

OS Operating System (Système d'exploitation)

RAM Random Access Memory (Mémoire vive)

RISC Reduced Instruction Set Computer (Ordinateur à jeu d'instructions réduit)

RNA	Résau de Neurones Artificiels
SDK	Software Developement Kit (Kit de développement)
SIMD	Single Instruction, Multiple Data (Une instruction, Multiples données)
SIMT	Single Instruction, Multiple Threads (Une instruction, Multiples fils d'exécutions)
SoC	System on Chip (Système sur une puce)
SSE	Sum of Squared Errors (Somme quadratique des erreurs)
SVR	Support Vector Machines
TDL	Time Delay Line (Ligne de temporisation)

Symboles grecs

μ	La moyenne	
σ	L'écart type	
τ	Le retard des estimations	
ε	L'erreur d'estimation	
ς	La fonction sigmoïde	
ϑ	La température en degrés Kelvin	K

Symboles latins

d	Le retard à l'entrée des modèles.	
f	La fréquence du processeur	Hz
I	Le courant électrique	A
P	La puissance	W
R	La régression des estimations et mesure d'un modèle	
r	Les résidus bruts	
R^2	Le coefficient de détermination	
r_m	Les résidus moyennés	
R_x	Les résidus normalisés [de la variable x]	
T	La température	°C
t	Le temps	s
V	La tension électrique	V

INTRODUCTION GÉNÉRALE

Les systèmes sur puce (*System-on-Chip*, SoC) hétérogènes combinent plusieurs types de processeurs, généralement des unités centrales de traitement (*Central Processing Units*, CPU) et des unités de traitement graphiques (*Graphics Processing Units*, GPU) [16]. Au début des années 2000, ces SoC sont devenus populaires grâce à leur intégration dans les ordinateurs personnels (*Personnel Computer*, PC). Leur popularité s'est accrue avec leur intégration dans les smartphones et les tablettes, au cours de la dernière décennie.

De nos jours, les SoC hétérogènes sont au cœur des systèmes informatiques et des appareils portables les plus modernes, y compris ceux utilisés dans les systèmes critiques pour la sécurité (militaires, aérospatials, automobiles, etc.) [17]. Dans ces systèmes, qui sont de plus en plus intégrés et mobiles, les fabricants sont confrontés au défi croissant de concevoir des puces offrant des performances élevées, tout en minimisant la consommation électrique, et maintenant une dissipation thermique gérable.

Ce travail de thèse s'inscrit dans le cadre du projet *Multi-Modular Cokpit Display* (MMCD) dont l'objectif est de développer des écrans tactiles qui serviront à la fois pour l'affichage et à la commande dans les cockpits du future. Les SoC à CPU-GPU embarqués sont le cœur de ces systèmes critiques et doivent de ce fait être fiables et résistants aux pannes.

Dans ce travail, nous abordons cette problématique en proposant une approche générique de surveillance qui permet de détecter des dérives liées aux caractéristiques des systèmes CPU-GPU embarqués dans les écrans de pilotage. Cette approche est composée d'un modèle incrémental associé à un algorithme de détection et d'isolation des défauts (*Fault Detection and Isolation*, FDI). Elle repose sur la détection précoce des dérives liées aux caractéristiques des systèmes CPU-GPU causées par l'usure, des conditions de fonctionnement extrêmes, ou une sollicitation excessive.

Pour ce faire, nous commençons par la construction du modèle incrémental qui générera les estimations de référence du système.

La dynamique des systèmes CPU-GPU est assez diversifiée et peut être étudiée sous différents angles. Elle peut être décrite par des variables discrètes (comme la charge de calcul du CPU), des variables continues (telle que la température), ou des systèmes à événement discrets (comme pour les gouverneurs de fréquence). Afin de modéliser l'ensemble de ces dynamiques, le système CPU-GPU est vu, dans ce travail, comme un système à structure variable, pour lequel la fréquence de fonctionnement, la puissance consommée et la chaleur générée dépendent à la fois de la charge logicielle et des modes de fonctionnement (mode économie d'énergie, mode performance, etc.). Par

conséquent, le modèle obtenu est par conception un modèle *générique*, car il correspondra à une large gamme de SoC hétérogènes.

L'idée – et l'originalité de ce modèle de CPU-GPU – est de construire une structure de modélisation incrémentale et interconnectée, composée de sous-systèmes construits dans l'intention que chacun de ces sous-systèmes génère l'estimation d'une des variables du système, tout en tenant compte des interactions avec les autres sous-systèmes. Cette approche de modélisation constitue la première contribution de ce travail par rapport aux travaux existants, et offre les avantages suivants :

— Le modèle considère naturellement la variabilité de la structure du système.

— Il permet d'estimer simultanément les variables continues et les variables discrètes du système.

— Le modèle est facilement adaptable aux changements de composants et de modes de fonctionnement.

— Cette approche incrémentale et modulaire offre à ses utilisateurs la possibilité de construire une bibliothèque de modèles décrivant les variables et les multiples modes de fonctionnement du système, puis d'adapter le modèle incrémental aux besoins de la modélisation.

  Par exemple, pour l'estimation de la puissance consommée par le CPU, l'utilisateur peut soit construire de nouveaux modèles [5, 9], soit avoir recours à des modèles déjà établis dans la littérature comme le *PowerBooter* [18], et ainsi construire sa bibliothèque. Cette approche permet d'adapter le modèle incrémental aux besoins de la modélisation, aux variations dans les structures et aux différents pilotes des systèmes embarqués existants ou futurs.

— Les variables estimées ont un sens physique clair qui permet de les lier par des relations de cause à effet aux processus de dégradation des SoC.

La plupart des techniques de diagnostic de défauts appliquées aux systèmes à microprocesseurs sont basées sur la théorie de la dépendance, de la redondance matérielle et logicielle et sur les tests de vérification en ligne. Alternativement, la méthode proposée dans ce manuscrit est basée sur le principe de la redondance analytique. Cette technique est largement utilisée pour le diagnostic de défaillance dans les systèmes sans composants logiciels. Elle n'a cependant pas été utilisée pour la surveillance de systèmes possédant des composants matériels et logiciels.

La redondance analytique ici est créée par l'adoption du modèle incrémental décrit ci-dessus comme modèle de référence du système. La méthode consiste ensuite à comparer les sorties réelles du système aux références générées par ce modèle. En présence de défauts, les sorties du système s'écartent de leurs valeurs de référence, générant ainsi des indicateurs de défauts. Ces derniers, communément appelés résidus, sont évalués par une méthode probabiliste pour tenir compte des incertitudes de modélisation dans le processus de prise de décision.

Cette thèse est organisée en cinq chapitres.

Le Chapitre I commence par un état de l'art des SoC, leurs propriétés et leur principe de fonctionnement, afin de comprendre les interactions entre la partie logicielle et la partie matérielle du système, et définir par la suite les variables caractéristiques du fonctionnement des SoC. Nous abordons ensuite

la modélisation des SoC dans la deuxième partie du chapitre, avec une analyse fine des travaux existants dans la littérature, et présentons une étude comparative de ces travaux pour proposer une classification des approches existantes en se basant sur des critères bien définis. Le chapitre se conclut par une méthodologie de construction d'outils d'acquisition et modélisation des SoC, dont l'organigramme est suivi dans les chapitres II et III pour la construction du modèle incrémental des SoC.

Le Chapitre II introduit la première étape du processus de modélisation. Il est consacré au développement d'un outil d'acquisition, de traitement et d'enregistrement des données issues des systèmes embarqués en général, et des SoC en particulier. En effet, l'instrumentation de ces systèmes est particulière, composée d'une part de capteurs, requis pour l'enregistrement de variables physiques telles que la température et la puissance, et d'autre part des traces du système. Ces dernières sont nécessaires au fonctionnement du système d'exploitation (*Operating System*, OS) et décrivent la partie logicielle du SoC, dont la charge des processeurs et la fréquence d'opération. Ce chapitre constitue le socle de notre travail de recherche ; les performances du modèle et de l'algorithme de surveillance des SoC dépendront directement de la qualité des données acquises, traitées et enregistrées par ce système d'acquisition.

Le Chapitre III présente de façon détaillée la modélisation incrémentale des SoC. Il s'agit d'une contribution principale de ce travail de recherche, puisque notre objectif est de concevoir un modèle générique s'adaptant à la majorité des SoC existants ainsi qu'à tous leurs modes de fonctionnement. De plus, nous souhaitons que le modèle n'impose pas une instrumentation supplémentaire aux SoC, et ses sorties doivent avoir un sens physique clair. Le modèle que nous proposons donc dans ce chapitre est un modèle incrémental, constitué de modules (sous-systèmes) interconnectés et présentés dans une bibliothèque de modèles ouverts, interchangeables et modifiables. Afin de garantir des relations causales lors de l'interconnexion des modules, et permettre une interprétation explicite du sens physique des indicateurs de fautes, les entrées et les sorties de chaque module sont bien définies physiquement.

Face à la complexité des SoC et grâce à la disponibilité des traces du système, notre choix s'est naturellement porté sur des approches guidées par les données pour la construction de sous-systèmes du modèle incrémental. Le rôle de ces modules est d'estimer une variable physique ou logicielle caractérisant l'état de fonctionnement du SoC et dont la dérive est clairement associée à une dégradation. Pour finir, les résultats de chaque module sont analysés et validés sur différents systèmes.

Le Chapitre IV décrit notre approche de surveillance des SoC, basée sur la détection et l'identification des dérives de leurs caractéristiques. La méthode consiste à comparer le fonctionnement du système à un fonctionnement de référence généré par le modèle incrémental développé dans le Chapitre III. Chaque sous-système génère donc un indicateur de dérive d'une des caractéristiques du système. L'évaluation des résidus et la prise de décision sont ensuite réalisées par des outils statistiques appropriées à la nature de chaque résidu. Cet algorithme de surveillance est validé expérimentale-

ment sur des systèmes différents, pour des scénarios de fonctionnement normal ou présentant des défaillances. Ces expériences de défaillances sont représentatives des situations pouvant se présenter au cours du fonctionnement du système dans des conditions réelles.

Pour finir, le Chapitre V est consacré au déploiement des modèles et algorithmes développés dans les chapitres II, III et IV sur la carte de développement sélectionnée dans le cadre du projet MMCD. Dans ce chapitre, nous méttons en œuvre un démonstrateur, et détaillons la mise en place du système de surveillance, ainsi que les scénarios de test et de validation. L'implémentation des algorithmes sur la carte de développement du projet MMCD montre la généricité et la portabilité de ces algorithmes et leur capacité d'adaptation à différents SoC, quels que soient leurs niveaux de performance ou leur génération. Les résultats expérimentaux sont obtenus suite à un scénario de validation d'un fonctionnement normal et deux scénarios de test de fonctionnements défaillants. Ces résultats valident bien l'approche de surveillance. Les processus de dégradation introduits pendant les deux scénarios de défaillance sont étudiés pour correspondre aux types de dégradations qui peuvent survenir lors du fonctionnement du SoC dans le domaine de l'aéronautique.

Ce manuscrit s'achève sur une conclusion générale, qui dresse une synthèse des travaux réalisés, des verrous levés ainsi que des résultats obtenus. De plus, les contributions principales décrites dans chaque chapitres sont mises en évidence, autant du point de vue de la recherche que de celui du développement et de l'adaptation au contexte du projet MMCD. Pour conclure, des perspectives de recherches concernant l'utilisation des indices de dérives pour le pronostic de défaillances sont exposées.

Chapitre I

État de l'Art de la Modélisation des Processeurs Embarqués et Mobiles Hétérogènes

I.1 Introduction

De leur introduction en 1971 comme simples calculateurs, les microprocesseurs n'ont cessé d'évoluer jusqu'à être au centre de toutes les innovations technologiques [19]. Ces puces électroniques ont permis le développement des systèmes d'information et l'informatique personnelle et mobiles grâce à leur versatilité, vitesse, et surtout leur évolution rapide.

Les microprocesseurs sont composés d'une ou plusieurs unités centrales de traitement (CPU), ainsi que d'autres modules nécessaire à leur fonctionnement tels que des contrôleurs de mémoire, une mémoire cache et des contrôleurs entrées-sorties. Toutefois, dans certains systèmes, le circuit intégré contenant le microprocesseur inclut tout ou la majorité des composants nécessaires à leur fonctionnement sur la même puce, à l'image des microcontrôleurs, GPU, etc. Un tel système est appelé système sur une puce (*System on Chip*, SoC). De nos jours, les SoC sont omniprésents par leur utilisation dans les systèmes mobiles (smartphones et tablettes). Ils occupent moins d'espace et consomment moins de puissance, ce qui les rends appropriés à ces systèmes. De plus, les SoC intègrent intègrent davantage de modules à chaque nouvelle génération, apportant ainsi de nouvelles fonctionnalités. Par exemple, un SoC moderne typique contient les CPU, le GPU, les modules de communication (Wi-Fi, Bluetooth, Modem cellulaire...), un module pour la localisation, ainsi que d'autre sous-systèmes et coprocesseurs assurant diverses fonctions comme la sécurité de l'appareil [20].

En outre , les SoC sont utilisés dans des systèmes informatiques appliqués appelés généralement *Systèmes embarqués*. Bien qu'ils n'existe pas de définition formelle de ces derniers, il s'agit généralement de systèmes d'information conçus pour des tâches bien définies [21] et sont intégrés dans d'autres produits [22]. Cette intégration touche pratiquement à tous les systèmes informatiques, de ceux destinés au grand public (PC, smartphones, appareils photos, GPS, etc.), aux systèmes de grade industriel et critiques pour la sécurité (automates programmables, systèmes de contrôle des automobiles, etc.). Par conséquent, une grande partie de ces systèmes a des exigences de fiabilité et de durée de vie plus élevées que les systèmes informatiques destinés au grand public.

A présent que les SoC et les systèmes embarqués ont été introduits, nous allons dans un premier temps présenter les systèmes embarqués et les SoC hétérogènes dans la Section I.2. Puis nous résumerons l'état de l'art de la modélisation de ces systèmes dans la Section I.3. Pour finir, nous présenterons dans la Section I.4 une méthodologie de la construction de modèles et de « *profilers* » pour les systèmes embarqués en général et les SoC hétérogènes en particulier.

I.2 Architecture des systèmes hétérogènes embarqués et mobiles

Les systèmes informatique à GPU comprennent deux types de microprocesseurs, un processeur central de calcul et un processeur graphique. Ce type de système – contenant deux processeurs de nature différente – est dit *hétérogène* [23].

I.2.1 Les systèmes hétérogènes

Les systèmes hétérogènes sont décrits par deux caractéristiques principales : le nombre des sous-systèmes fonctionnels et des circuits processeurs qu'ils contiennent, et les sous-systèmes de mémoires dédiées à chacun des sous-systèmes fonctionnels [24]. L'un des exemples les plus célèbres des SoC hétérogènes sont les SoC conçus sous les normes de la fondation « *The Heterogeneous System Architecture* » (L'Architecture des Systèmes Hétérogènes, HSA), une fondation créée par de multiples fournisseurs de semi-conducteurs comme AMD et ARM [10, 25].

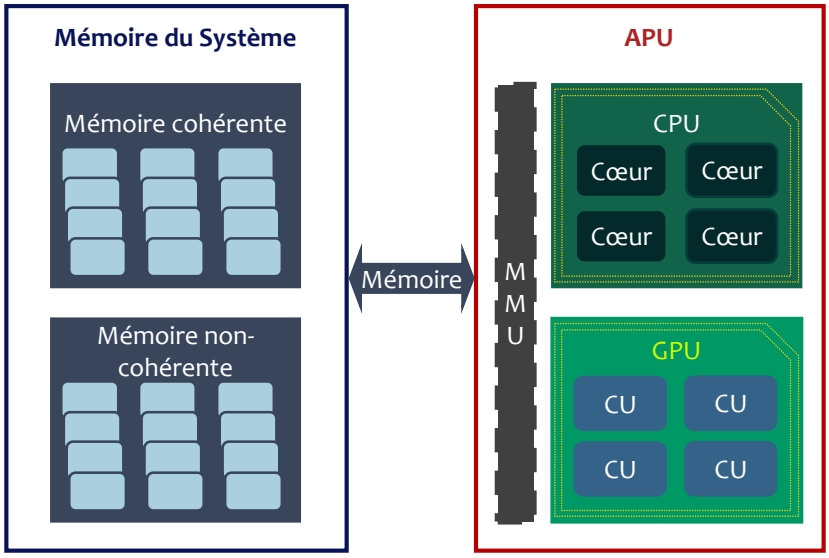


FIGURE I.1 – L'APU AMD : un exemple des systèmes hétérogènes (Source : [10]). MMU : Memory Management Unit (Unité de gestion mémoire). CU : Compute Unit (Unité de calcul).

La Figure I.1 montre les éléments principaux d'une Unité de Calcul Accélérée (*Accelerated Processing Unit*, APU) de AMD, utilisant l'architecture HSA. Cette APU contient deux sous-systèmes fonctionnels : un CPU et un GPU. Les deux processeurs sont placés sur le même SoC avec un contrôleur de mémoire (*Memory Management Unit*, MMU) et partagent la même mémoire vive (*Random Access Memory*, RAM). L'une des particularités de cette architecture est qu'elle permet aux programmes d'utiliser le GPU pour les calculs en virgule flottante, sans ordonnancement particulier ou séparation de mémoire [10].

Dans le projet MMCD, les systèmes embarqués envisagés utilisent un SoC hétérogène basé sur l'architecture de ARM. Dans la suite de cette section, nous allons donner une brève description de l'architecture de cette famille de SoC.

I.2.2 Les SoC basés sur l'achitecture ARM

La Figure I.2 montre le schéma blocs d'un SoC d'application basé sur les processeurs ARM avec deux processeurs : Un Cortex-A9 et un Cortex-M4. La figure montre aussi tous les modules et composants

inclus dans le SoC, comme les modules de communication (Ethernet, USB...), les mémoires (RAM, ROM), et les modules de contrôle.

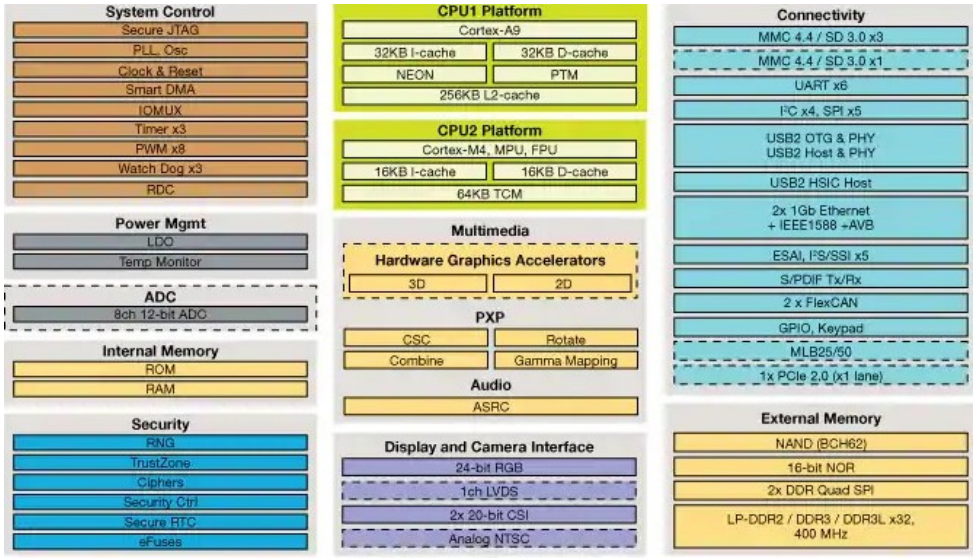


FIGURE I.2 – Le schéma blocs d'un SoC basé sur l'architecture ARM à deux processeurs : Un Cortex-A7 et un Cortex-M4 [11].

I.2.2.1 Les processeurs basés sur l'architecture ARM

Les SoC ARM utilisent des processeurs basés sur l'architecture du même nom. Ces processeurs ont un jeu d'instructions réduit (*Reduced Instruction Set Computers*, RISC). Dans cette architecture, les CPU sont conçus pour exécuter un nombre réduit d'instruction de base avec pour objectif – dans la majorité des cas – d'exécuter une instruction par un cycle d'horloge. Ceci est obtenu en fixant la longueur des champs d'instructions afin de simplifier le décodage de ces dernières. En outre, ces processeurs ont un mode d'adressage simplifié avec l'architecture chargement/stockage (*load/store architecture*), où toutes les opérations sur les données sont exécutées uniquement sur des registres.

En plus de l'architecture RISC de base, les processeurs ARM sont aussi dotés de fonctionnalités permettant de contrôler les unités arithmétiques et logiques (UAL), le chargement et le stockage de plusieurs instructions, ainsi que l'exécution d'une instruction sur plusieurs données (*Single instruction Multiple Data*, SIMD). Ces améliorations apportées à l'architecture RISC de base permettent aux processeurs ARM un meilleur équilibre entre les performances, la consommation d'énergie, et la taille de la puce. Grâce à cet équilibre, les SoC à base de processeurs ARM dominent le marché des systèmes embarqués, notamment dans le secteur des systèmes mobiles, et l'Internet des Objets (*Internet of Things*, IoT).

Les processeurs ARM sont divisés en trois familles : Les Cortex-A, Cortex-M, et Cortex-R.

Les Cortex-A sont des processeurs d'application, généralement destinés au grand public. Ils sont conçus pour offrir des performances élevées avec une consommation d'énergie relativement faible. Comme leur nom l'indique, ces processeurs sont destinés à être utilisés en application (Smartphones,

Tablettes, appareils photos, etc.). Par conséquent, les systèmes utilisant ces processeurs fonctionnent généralement sous un système d'exploitation, et sont aussi dotés d'une unité de gestion de la mémoire (MMU).

La seconde famille des processeurs ARM est celle des microcontrôleurs Cortex-M. Ces processeurs sont utilisés dans des systèmes fortement embarqués, qui nécessitent des traitements en temps réel et une consommation minime d'énergie comme les capteurs, les transmetteurs et les appareils utilisés dans l'IoT.

La dernière famille regroupe les Cortex-R. Ces processeurs sont dits déterministes en raison de leur latence d'interruption faible. Cette propriété rend ces processeurs les plus adaptés aux systèmes à temps réel et critiques. Ils sont généralement utilisés dans les systèmes de contrôle et commandes avioniques, les automates programmables industriels, et les appareils médicaux.

I.2.2.2 Les processeurs graphiques embarqués et mobiles

Un GPU, par définition, est un processeur spécialisé pour le rendu graphique. Il est utilisé pour l'accélération des opérations visuelles [26]. Les GPU existent en plusieurs familles suivant leur implémentation et leur puissance de calcul, allant des processeurs graphiques de base utilisés pour les tâches d'affichage de l'interface utilisateur, le décodage des vidéos, et les simples rendu 3D, jusqu'aux processeurs dédiés aux jeux vidéo et aux stations de travail, capables de rendre des synthèses plus complexes en 3D [23, 27].

Contrairement aux CPU conçus pour avoir la plus petite latence possible et optimisés pour le branchement et les boucles (Figure I.3a), les GPU ont été et sont toujours conçus pour le traitement des graphismes, et leur architecture est optimisée à cette fin. Par conséquent, ils sont adaptés à l'exécution d'une seule instruction sur plusieurs données (SIMD), et une seule instruction sur plusieurs fils d'exécution : (*Single Instruction, Multiple Threads*, SIMT). Ainsi, les GPU sont plutôt conçus pour avoir *throughput* maximum [23].

Le throughput est défini comme la quantité totale de travail exécutée sur un temps donné [27]. Ainsi, un processeur conçu pour délivrer un grand throughput, comme les GPU, doit être capable d'exécuter plusieurs travaux en parallèle [24]. Pour ce faire, les GPU sont dotés d'un grand nombre d'unités de calcul (Appelées : *CU*, ou *CUDA Core*, ou *cœur*), en comparaison avec le nombre de cœurs dans les CPU. Toutefois, le grand throughput et la parallélisation des calculs, réduisent la nécessité d'une faible latence puisque la quantité de travail est ce qui prévaut. Ainsi, la plupart des GPU fonctionnent à des fréquences d'horloge modestes comparées à celles des CPU.

La Figure I.3b explique la parallélisation des calculs et l'ordonnancement des travaux dans un GPU. Quand des travaux sont initialisés sur un GPU, ils sont organisés de sorte que ce dernier commence le premier travail et que durant le temps d'attente des données, il commence le second, et ainsi de suite jusqu'à la fin de tous les travaux. Une fois ceci fait, il reprend le premier travail dont les données sont disponibles et poursuit la tâche, et ainsi de suite. Cet ordonnancement, s'il est bien calculé, permet d'éliminer l'effet de la grande latence, parce que le processeur sera toujours occupé avec des calculs.

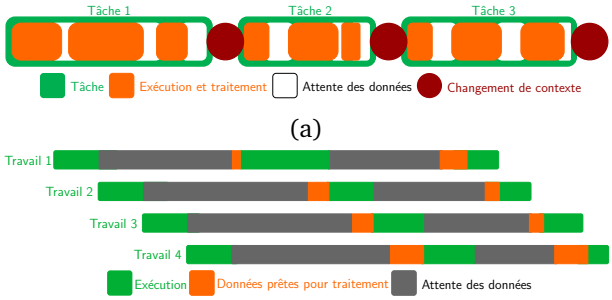


FIGURE I.3 – La différences d'exécution des programme entre les CPU et les GPU : L'influence du temps de traitement et le temps d'attente. (a) L'exécution des tâches sur un CPU : (b) L'exécution des tâches sur un GPU.

Les performances des GPU augmentent pour chaque nouvelle architecture, en particulier celles conçues par les plus grands constructeurs tels que NVIDIA et AMD. Cette course à la performance est surtout motivée par le rôle que jouent les GPU dans l'industrie du divertissement (Jeux vidéo, conceptualisation 3D, édition et rendu des vidéos). Néanmoins, La caractérisation les GPU uniquement par leurs tâches graphiques est limitante et dépassée. En effet, les tâches des GPU dépassent désormais le monde des graphismes et les rendus vidéos, et ils sont utilisés pour les calculs scientifiques, les simulations, et dans le domaine de l'Intelligence Artificielle (IA) grâce au nombre élevé des cœurs dont ils disposent [23, 27, 28].

Il n'existe pas de classification standard pour les GPU. Toutefois, nous parlons généralement de GPU intégrés et de GPU discrets (ou dédiés). Les GPU intégrés sont des GPU disposés sur le même SoC que le CPU et ils partagent généralement la même RAM. Par contre, les GPU discrets sont le plus souvent disposés sur leur propre circuit (dit carte graphique) et ont une mémoire séparée du reste du système.

Dans les systèmes embarqués, la quasi-majorité des GPU sont intégrés. Ces derniers sont conçu pour les tâches générales d'affichage et de rendus 2D et 3D, ainsi que la lecture vidéo. Ils dominent le marché du grand public, notamment le secteur mobile (smartphone et tablette) où ils sont pratiquement les seuls à exister. L'un des plus grands avantages de ce genre de GPU est la mémoire partagée, qui permet un interfaçage facile avec les CPU. En outre, avec les outils logiciels comme l'OpenCL, ces GPU deviennent des ressources directement disponibles pour effectuer des tâches générales en plus des tâches graphiques. Ils ont aussi une consommation très faible en puissance comparée à celle des GPU dédiés, et deviennent le choix idéal pour les systèmes utilisant des batterie et les usages qui ne requiert pas une grande puissance graphique.

Les GPU intégrés les plus connus sur les systèmes embarqués – dont les smartphones et les tablettes – sont les GPU intégrés ARM et Qualcomm. ARM propose la famille de GPU *Mali* avec les processeurs cortex-A allant jusqu'à 16 cores [29, 30]. De son côté, Qualcomm commercialise dans les SoC Snapdragon, les GPU *Adreno*. Ces GPU sont ceux utilisés dans la plupart des smartphones haut de gamme [31]. En plus de ces deux compagnies, citons également Vivante, PowerVR, et NVIDIA qui

propose ses GPU Tegra pour les applications embarquées et intégrées nécessitant des performances plus élevées [32, 33].

I.3 La modélisation des systèmes embarqués et mobiles

La littérature est riche de modèles consacrés aux systèmes embarqués et mobiles. Toutefois, à notre connaissance, il n'existe pas de travaux traitant le problème de la surveillance en ligne de l'état de fonctionnement des puces CPU-GPU. Ceci confirme l'originalité de notre proposition et l'intérêt de pallier ce manque. Néanmoins, certains travaux traitent de problématiques assez proches des nôtres. Par exemple, Mercati et al. [34] ont créé des modèles de puissance et de température pour les systèmes fonctionnant sous Linux et Android dans le but d'améliorer leur fiabilité [34, 35]. Ces modèles sont des modèles d'optimisation, et sont utilisés hors ligne [35]. De manière similaire, Li et al. [36] ainsi que Zhou et al. [37] ont créé des modèles pour l'estimation de la puissance et de la température afin d'optimiser les niveaux thermiques des processeurs, mais davantage pour des MPSoC hétérogènes. De plus, Dousti et al. [38] ont présenté un analyseur appelé *ThermTap* afin d'estimer la consommation d'énergie et la température des appareils mobiles.

La littérature regorge également de travaux traitant au moins de l'un des axes de modélisation présentés dans ce manuscrit. Dans le paragraphe qui suit, nous explorons les principaux travaux de recherches effectuées dans ces domaines, en nous limitant aux travaux effectués sur des SoC embarqués ou mobiles.

I.3.1 La modélisation de la puissance et de l'énergie consommée

Dès l'introduction des smartphones modernes en 2007, les difficultés rencontrées ont été l'estimation de l'autonomie [39] et la prédiction de la durée de vie restante de la batterie [40]. Face à ces appareils livrés avec une source d'énergie limitée, des développeurs ont étudié leur autonomie [41, 42] et construit des modèles de consommation de puissance [43].

La problématique majeure concernant ces appareils concerne la dualité performance-consommation de puissance. Cela consiste à augmenter les performances des systèmes tout en maintenant une consommation d'énergie minimale. Ainsi, l'amélioration de l'efficacité énergétique est devenue l'objectif majeur des industriels et académiciens [44, 45], ce qui explique les nombreux travaux sur le sujet [46–52]. En effet, les deux secteurs – scientifique et industriel – travaillent activement pour atteindre cet objectif [52–54]. Les solutions proposées sont variées. Les articles [55–58] proposent des solutions algorithmiques telles que l'ajustement dynamique de la tension et de la fréquence des CPU (*Dynamic Voltage and Frequency Scaling*, DVFS), tandis que [59, 60] s'intéressent au DVFS dans les GPU. D'autres travaux proposent la limitation de puissance consommée [49] ou des algorithmes améliorant l'utilisation des ressources ainsi que la distribution des tâches[61, 62]. Citons de plus Marz et al. [63] qui se penchent sur l'optimisation des Frameworks des applications, et Lu et al. [64] qui se focalisent sur l'amélioration de la consommation électrique des écrans et proposent des solutions

techniques tels que la réduction des résolutions des écrans. Enfin, relevons les travaux proposant des études de l'influence des applications et des interactions des utilisateurs avec le système sur la consommation d'énergie [34, 65–68], parfois à l'aide d'approches statistiques à grande échelle [69, 70].

Cette problématique continue d'être étudiée afin d'améliorer la précision des modèles d'estimation, et d'intégrer les nouvelles technologies [41, 71]. En outre, avec l'émergence de l'IoT et l'omniprésence des systèmes embarqués, surtout dans les systèmes critiques, l'intérêt porté à l'étude et la modélisation de ces systèmes ne se limite plus aux systèmes mobiles, mais englobe désormais la plupart des systèmes électroniques embarqués [49, 72–74].

L'étude de la consommation d'énergie des systèmes embarqués a également permis aux chercheurs d'identifier et de résoudre des bugs entraînant une consommation excessive d'énergie [75–77], et de créer des logiciels écoénergétique (optimisés pour avoir une consommation réduite) [78]. En outre, Suarez-Tangil et al. [79] ont utilisé le profil de la puissance consommée pour la détection d'anomalie dans le système, tandis que [80] l'ont employé afin de développer une méthode pour la détection de logiciels malveillants.

Afin d'estimer ou de mesurer la puissance consommée par les systèmes embarqués ou mobiles, les concepteurs sont généralement amenés à construire des *profilers*. Ces derniers sont des outils permettant la collection cyclique des mesures ou l'estimation des variables désirées (dans ce cas, il s'agit de la puissance). Ils varient en complexité et en fonctions, du simple enregistreur [81] au profilers basés sur des modèles complexes capables d'estimer l'énergie consommée par les instructions [54].

Il existe trois revues principales dans la littérature détaillant les profilers et la modélisation de la puissance dans les smartphones [82, 43, 83]. La première est le travail de Hoque et al. [82], dans lequel, les auteurs présentent les différents mécanismes de mesure de l'énergie (instruments externes, capteurs internes, etc.), les types de modèles (modèles basés sur l'utilisation, les événements et l'analyse du code), les philosophies de modélisation (boîte blanche ou boîte noire) et les approches de *profiling*. Les auteurs ont de plus proposé une taxonomie des profilers et modèles étudiés en fonction de leur déploiement et le lieu de la construction du modèle, soit en interne sur le système, soit en externe (en de hors du système) [82].

La deuxième revue mentionne essentiellement les mêmes travaux, mais elle les examine dans l'optique de leur mise en œuvre (matérielle ou logicielle) [43], tandis que la troisième porte sur la simulation des réseaux [83].

Parmi les profilers et modèles cités dans ces revues, plusieurs relèvent de notre cadre de travail en fonction du niveau auquel ils peuvent estimer la consommation d'énergie, ainsi que leurs utilisations. Parmi ces profilers, certains sont construits par des fournisseurs de système tels que *Android Power Profiler* de Google [84, 85], et *Trepn Profiler* de Qualcomm [86]. Nous ajoutons également des travaux publiés dans la littérature tels que le célèbre *PowerBooter* [18], *Sesame* [87], *DevScope* [88], *Eprof* [89] et les modèles construits par Banerjee et al. [90] et Shye et al. [91].

Outre les travaux mentionnés dans ces revues, Kim et al. [92] ont également proposé une approche pour la modélisation de la consommation d'énergie, dans laquelle le modèle est un polynôme dont chaque terme reflète la consommation d'énergie d'un composant du smartphone [92]. Les auteurs ont ensuite continué à améliorer leur modèle avec une meilleure estimation de la puissance pour le GPU [13]. Plus récemment, plusieurs nouveaux modèles ont également été publiés, soit pour le système dans son ensemble [35, 34, 93–98], ou tout simplement pour le CPU [99, 100].

Le flux de nouveaux modèles est dû aux différences générationnelles entre les composants (et donc leurs profils de puissance) [81, 101], et le changement des interactions utilisateur-appareil [102]. Ces changements rendent les anciens modèles obsolètes [99], et poussent les concepteurs à établir de nouveaux modèle plus précis [13, 93–95, 100].

La plupart des modèles d'estimation de la consommation d'énergie, sont conçus soit en utilisant des machines à états finis (*Finite State Machines*, FSM) telles que *Eprof* [89] et *DevScope* [88], soit à l'aide d'approches plus courantes basées sur des régressions, comme c'est le cas pour au moins Walker et al. [100], Dong and Zhong [87], Kim et al. [92], Kim et al. [13], Shukla et al. [94], et Xu et al. [103], soit en utilisant la combinaison des deux techniques [88, 18]. Nous notons de plus une nouvelle tendance consistant à utiliser des méthodes non linéaires, telles que les réseaux de neurones artificiels (RNA) afin d'améliorer la précision des modèles [104, 105, 9, 4].

La modélisation boîte noire, à l'image de la régression et l'ajustement de données (*Data-fitting*), est populaire pour les systèmes embarqués, principalement en raison de leur complexité et de celle des phénomènes physiques dans ces systèmes. Les modèles conçus en boîte noire permettent de simplifier le processus de modélisation, étant donné que les modèles construits ne nécessitent pas de preuves formelles de l'interaction avec les sorties estimées [18]. Le choix des entrées du modèle est souvent le résultat d'observations et de tests expérimentaux dans diverses conditions de fonctionnement [82].

I.3.2 La modélisation de la température

La température du SoC est une variable importante pour le bon fonctionnement et la durée du cycle de vie des SoC. Pour cette raison, la modélisation de la température est généralement réalisée à des fins d'optimisation. Tout d'abord, durant la phase de conception, elle est établie afin de définir les seuils de température [106] ou dimensionner les dispositifs d'évacuation de chaleurs (Radiateurs, caloducs, etc.). Ensuite, durant la phase d'exploitation, ces modèles sont construits afin d'avoir de meilleurs comportements thermiques [36, 56, 107, 108]. D'autres modèles se penchent sur la fiabilité du système en étudiant les effets des températures interne et externe sur le système [109], ou sur la durée de vie du système [110–112] afin de pouvoir l'améliorer [34, 113–115].

En plus des modèles conçus pour l'optimisation, il existe des travaux proposant des modèles axés sur la gestion de l'énergie et la gestion thermique. Par exemple, Mercati et al. [35] proposent une méthode pour adapter les conditions de fonctionnement (fréquences des processeurs, luminosité de l'écran, etc.) en fonction des besoins de l'utilisateur (utilisation, application en cours d'exécution)

[35]. D'autre travaux visent ce même objectif en proposant des algorithmes de planification de tâches [116, 37, 117, 118] ou de DVFS [36].

En ce qui concerne les SoC embarqués et mobiles, les travaux les plus proches de notre cadre de travail – à savoir la modélisation pour la surveillance – portent sur le simulateur *Therminator* [119] et sa deuxième version *ThermTap* [38]. Ce sont des logiciels développés pour l'estimation en ligne de la température des appareils mobiles pour le débogage. Dans un autres travail de modélisation, Fu et al. [120] ont utilisé un modèle boite-grise basée sur un filtre de Kalman afin prédire la température d'un réseau sur puce (Network-on-Chip). Ils ont ainsi pu réduire considérablement le bruit des capteurs thermiques.

I.4 Conception et construction des profilers pour les SoC embarqués

Tous les profilers existant dans la littérature ont la tâche commune de fournir des valeurs décrivant une ou plusieurs variables telles que la consommation d'énergie [43] ou la température [9, 38], même lorsqu'ils diffèrent par leur construction ou approche. Certains prennent des mesures [81, 121] tandis que d'autres génèrent des estimations [6, 100]. Toutefois, en étudiant ces profilers, d'autres aspects communs apparaissent. Ces aspects partagés entre les profilers permettent de les classifier en sous-catégories selon les approches et choix de design utilisés.

Dans ce qui suit, en utilisant cette classification, nous détaillerons le processus de construction des profilers en plusieurs étapes où chaque étape correspond à un choix entre plusieurs approches. Nous expliquons les choix de conception à effectuer pour chacune d'elles, en commençant par la définition de l'objectif et de la mise en œuvre du profiler.

I.4.1 Définition de l'objectif du profiler et sa granularité

Les profilers sont conçus pour accomplir une tâche spécifique vis-à-vis d'une ou plusieurs variables. Afin de choisir la meilleure approche et la meilleure structure pour le *profiling*, la première étape de sa construction consiste donc à définir l'objectif principal que doit atteindre ce profiler. Les profilers d'énergie, par exemple, peuvent être utilisés pour mesurer l'énergie consommée par le système [94, 122], ou par un composant spécifique [123, 124], ainsi que pour surveiller ce système et détecter des anomalies [6, 76].

Définir l'objectif du profiler permet également de déduire le niveau de granularité nécessaire afin d'atteindre cet objectif – la granularité étant le niveau auquel le profiler peut délivrer des mesures ou générer des estimations. En reprenant les exemples du paragraphe précédent, un profiler conçu pour estimer la puissance consommée par le système possède la granularité la moins fine (niveau système) [94]. Il en est de même un modèle conçu pour détecter des intrusions de sécurité [76]. Par contre, pour la surveillance du SoC, On requiert un niveau de granularité plus fin (composants) [4, 5], et un niveau de granularité encore plus fin (application), pour un modèle dont l'objectif est d'aider les

développeurs à optimiser la consommation de puissance de leurs applications [125]. La Figure I.4 montre les niveaux de granularité les plus courants dans la littérature, du moins au plus fin.

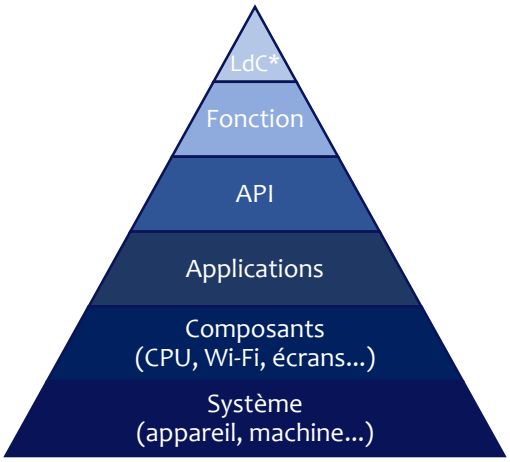


FIGURE I.4 – Les niveaux de granularité des profilers de puissance dans la littérature. *LdC* : Lignes de code.

Choix de conception à faire :

— **Objectif :** Surveillance, diagnostic, optimisation...

— **Granularité :** Système, composants, application...

I.4.2 Choix du type de *profiling* et des sources de mesure

Une fois que l'objectif du profiler et sa granularité sont définis, l'étape suivante consiste à déterminer le type de *profiling* à utiliser. Il en existe deux principaux : la méthode matérielle (*hardware-based profilers*) et la méthode logicielle (*software-based profilers*) [43].

Comme leurs noms l'indiquent, les profilers basés sur la méthode matérielle utilisent des capteurs et des instruments externe pour mesurer, rassembler et enregistrer les valeurs nécessaires [126]. L'utilisation du matériel externe nécessite, généralement, que l'appareil soit ouvert et connecté en permanence à l'instrument de mesure. En revanche, le *profiling* logiciel utilise un programme pour collecter des mesures (ou des lectures) et générer des estimations. Ces profilers s'appuient sur plusieurs techniques telles que l'auto-mesure (*self-metering*) [82], ou la courbe de décharge de la batterie [43], pour obtenir les résultats souhaités.

L'objectif du profiler et notamment son niveau de granularité dicteront également la méthode que le profiler utilisera pour fournir ses mesures et estimations. Les profilers matériels ont l'avantage d'offrir les mesures les plus précises mais ne sont applicables que pour les granularités les moins fines (systèmes [126], et dans certains cas, composants [60]).

Réciproquement, les profilers logiciels permettent d'atteindre des niveaux de granularité plus fins. En effet, comme il est très difficile d'obtenir des mesures physiques des variables au-delà des

composants, la meilleure alternative est d'utiliser des modèles mathématiques afin de générer des estimations. Ces profilers sont dits basés sur des modèles [82]. Ils peuvent atteindre des niveaux de granularité très fins. Toutefois, ils dépendent de la précision et de la fiabilité des capteurs internes et des estimations fournies par les API [43].

L'utilisation des profilers matériels a diminuée au fil des ans, car leur avantage principal – mesures précises ne justifie plus leur principal inconvénient qui, est l'intrusion physique du périphérique. De plus, avec les progrès récents, les mesures logicielles (API, capteurs internes, estimations...) ont atteint une précision équivalente à celle des mesures matérielles [121]. Ainsi, le *profiling* logiciel a gagné en popularité d'autant plus que ses résultats sont comparables au *profiling* matériel sans aucune intrusion [5, 121, 125].

Choix de conception à faire :

— **Types de *profiling* :** matériel ou logiciel.

— **Sources des mesures :**Instruments externes, API du système, courbe de décharge de la batterie...

— **Génération des valeurs :** Mesures physiques, estimations fournies par des modèles, analyses de code...

I.4.3 Modélisation

Dans le paragraphe précédent, nous avons décrit la méthode matérielle (enregistrement de mesures) et le *profiling* basé sur des modèles. La construction et la configuration d'un profiler pour l'enregistrement de mesures ne nécessite aucune étape supplémentaire. Par conséquent, dans tout ce qui suit, nous nous concentrons sur les profilers basés sur des modèles.

Toutes les approches de modélisation nécessitent un certain niveau de connaissances et d'expertise sur le système à modéliser. Ainsi, bien que les étapes de ce paragraphe soient toutes nécessaires, leur ordre peut être interchangeable, notamment l'étape du choix de l'approche de modélisation et celle du choix des entrées.

I.4.3.1 L'approche de modélisation

Il existe trois approches principales de modélisation : la modélisation boîte blanche, la modélisation boîte noire, et la modélisation boîte grise [127, 128]. Dans la modélisation boîte blanche, le fonctionnement interne du modèle ainsi que ses paramètres sont connus [129]. Un tel modèle utilise typiquement des machines à états finis [82, 127], ou simule des équations différentielles [38], ce qui nécessite, dans le cas de systèmes embarqués, une connaissance de la conception de tous les composants internes du SoC [18].

Par ailleurs, les techniques de modélisation boîte noire – telles que l'identification et la régression – entraînent un modèle en ajustant ses paramètres afin de générer des estimations correspondant aux

observations et mesures [82]. Dans ces modèles, une bonne connaissance des facteurs qui influencent les sorties est appréciée, mais pas requise, il en va de de même pour la preuve formelle de la relation entre les entrées et les sorties [18, 82]. Néanmoins, contrairement aux modèles boîte blanche, ces méthodes nécessitent de larges bases de données qui couvrent toutes les variations de la sortie pour l'apprentissage et la validation [82].

Enfin, les modèles boîte grise constituent le juste milieu entre les modèles boîte blanche et les modèles boîte noire, puisqu'une partie du modèle est construite à partir des connaissances du système et le reste est estimé via une identification des paramètres ou un apprentissage [120]. Par exemple, dans un premier temps, *PowerBooter* décrit la puissance consommée par les composants comme un ensemble d'états déterminés dans un modèle FSM, puis utilise la régression des données pour déterminer le niveau de consommation de puissance pour chacun de ces états [130, 129].

I.4.3.2 Choix des entrées du modèle

Hoque et al. [82] indiquent que dans les systèmes embarqués et mobiles, il existe deux types d'entrées qui caractérisent l'activité des composants et des applications : les entrées basées sur l'utilisation du dit composant (*utilization-based*) et celles basées sur les événements (*event-based*). Les modèles avec des entrées basées sur l'utilisation se concentrent sur la corrélation directe entre l'utilisation d'un périphérique ou d'un composant et l'énergie consommée [82]. Pour estimer la consommation de puissance du module Wi-Fi, dans les modèle basés sur l'utilisation, on observe son débit transféré. En revanche, dans les modèles basés sur les événements, c'est plutôt l'état (activé / désactivé) du module qui est observé [127]. Néanmoins, les modèles publiés dans les travaux récents n'utilisent plus un seul type d'entrée, chacun des deux types ayant des limites [131], et utilisent plutôt une combinaison de ces derniers [5, 93, 105, 132].

Le choix des entrées dépend de deux facteurs principaux ; les données disponibles sur l'appareil (les mesures provenant des capteurs, fichiers systèmes, etc.) et la granularité souhaitée. En premier lieu, les profilers – notamment ceux basés sur des modèles – doivent au moins inclure des données provenant des composants nécessaires à la fonction du système, tels que les processeurs et les composants actifs durant le fonctionnement (écrans, communication, etc.) [83]. Chaque composant doit être décrit par au moins une variable en entrée (la fréquence pour le CPU, par exemple). Généralement, il faut plus d'une variable pour générer des estimations précise (la fréquence et la charge pour le cas du CPU, par exemple). Chen et al. [101] ont démontré, dans leur étude détaillant la consommation de puissance, comment chaque composant matériel est pris en compte dans le décompte d'énergie. En outre, Ardito et al. [81] montrent également l'écart entre les différentes générations de matériels et de technologies en terme de consommation d'énergie.

Enfin, une granularité plus fine nécessite d'avantage de données pour générer des estimations correctes. Pour estimer la consommation de puissance d'une application, il est indispensable de dater son lancement et exécution, ainsi qu'enregistrer les fonctions auxquelles elle fait appel. Ceci nécessite l'utilisation des horodatages et des traces du système. Ces deux éléments sont essentiels à l'estimation

de la consommation énergétique des composants logiciels, encourageant ainsi l'utilisation des séries temporelles (*Timeseries*) [65, 103].

I.4.3.3 Construction et implémentation du profiler

Les profilers matériels, par définition, sont conçus pour fournir des mesures et des estimations en ligne. Les profilers logiciels, quant à eux, peuvent soit fonctionner en ligne et générer des estimations à la volée, soit générer des estimations à partir de données collectées au préalable. Pour une étude statistique à grande échelle des profils de consommation de puissance, l'estimation hors ligne convient mieux à la tâche [133]. Toutefois, si le profiler a pour objectif de surveiller ou déboguer la consommation de l'appareil, l'estimation en ligne est une meilleure solution [134, 9].

Les modèles des profilers sont également caractérisés par l'endroit de leur construction et apprentissage. Les modèles construits et entraînés sur l'appareil même ne requièrent aucun apprentissage ou réglage supplémentaire [18]. Ils génèrent des estimations précises et s'adaptent au profil spécifique de l'appareil [82]. Cependant, ils nécessitent une collecte de données et un temps d'apprentissage qui peut entraîner des coûts de calculs élevés. Les modèles construits et entraînés hors ligne éviteraient les inconvénients de ceux décrits plus haut [9], mais ils sont toujours spécifiques à un appareil et leur précision varie d'un appareil à l'autre [82].

La dernière catégorie regroupe les profilers hors-périphérique. Ces derniers sont construits et entraînés sur une machine secondaire, à l'exemple le *Snapdragon Profiler* [135]. Ces profilers fournissent des estimations précises, notamment dans les cas de fine granularité [43], mais nécessitent un lien permanent entre l'appareil et la machine secondaire.

Choix de conception à faire :

— **Approche de modélisation :** Boîte blanche, grise, ou noire

— **Implémentation du profiler :** Sur le système ou en dehors du système

— **Disponibilité des mesures et estimation :** En-ligne ou hors-ligne

I.4.4 Évaluation et mise au point du profiler

Pour les profilers basés sur des modèles, une fois ce dernier construit et entraîné, il doit être validé. En général, l'estimation des variables dans les systèmes embarqués est faite avec des modèles de régression (comme dans le cas de ce travail). Ainsi, nous nous concentrons sur la validation de ce type de modèle.

Dans cette étape, le modèle est testé en comparant ses estimation $\hat{y}$ avec les mesures y. Durant cette comparaison, la qualité de l'ajustement (*Goodness of fit*), la précision des estimations, et analyser les résidus sont vérifiées.

La qualité de l'ajustement est une comparaison directe entre les estimations du modèle et les mesures. Généralement, elle peut être évaluée à travers la régression linéaire R des estimations et des

mesures. La régression R est calculée par l'ajustement :

$$\hat{y} = ay + b \tag{I.1}$$

où a et b sont des coefficients à déterminer (R est calculé pour le cas $b = 0$). Dans le cas idéal, $\hat{y}$ correspond parfaitement à y, et l'on obtient $R = 1$. Par conséquent, en pratique, la valeur R doit se rapprocher de 1 le plus possible.

La qualité de l'ajustement peut être également évaluée par le calcul du coefficient de détermination R^2 (*R-squared*). Pour n échantillons, R^2 prend la valeur :

$$R^2 = 1 - \frac{\sum_{k=1}^{n}(y(k) - \hat{y}(k))^2}{\sum_{k=1}^{n}(y(k) - \mu_y)^2} = 1 - \frac{\text{SSE}}{\sum_{k=1}^{n}(y(k) - \mu_y)^2}, \tag{I.2}$$

où μ_y, la moyenne des mesures $y(k)$ sur n échantillons s'écrit :

$$\mu_y = \frac{1}{n}\sum_{k=1}^{n} y(k), \tag{I.3}$$

et SSE, la somme quadratique des erreurs (*Sum of Squared Errors*), a pour expression :

$$\text{SSE} = \sum_{k=1}^{n}(y(k) - \hat{y}(k))^2. \tag{I.4}$$

Une correspondance parfaite entre $y(k)$ et $\hat{y}(k)$, résulte pour $R^2 = 1$ (SEE $= 0$). De même que pour R, les meilleures estimations résultent pour R^2 le plus proche possible de 1.

Ensuite, il faut évaluer la précision des estimations. Le calcul de R ou R^2 est un indicateur de cette dernière, mais elle peut être davantage évaluée en calculant les erreurs d'estimation :

$$\varepsilon(k) = y(k) - \hat{y}(k) \tag{I.5}$$

L'étude de ces résidus permet d'affirmer si $y(k)$ correspond bien à $\hat{y}(k)$. Ces résidus doivent avoir un profil aléatoire. Le profil aléatoire des ces résidus une indication que les entrées et les sorties du modèle sont corrélées. Par conséquent, il faut tester la distribution de ces résidus. Parmi les différents test possible pour déterminer si les résidus sont aléatoires, nous citons le test de normalité [136], et le test de Kolmogorov–Smirnov (K–S test) [137]. Ces deux tests permettent de déduire si les résidus sont normalement distribués, et centrés autour la moyenne μ_ε avec un écart type σ. Pour un vecteur $\varepsilon(k)$ de n échantillons, l'écart type est égal à :

$$\sigma_\varepsilon = \sqrt{\frac{1}{n-1}\sum_{k=1}^{n}(\varepsilon(k) - \mu_\varepsilon)^2} \tag{I.6}$$

Cette variable est aussi affectée par l'hétéroscédasticité, qui indique l'existence d'une relation entre la variance des données (σ^2) et le nombre des échantillons des variables d'entrées. Ainsi, afin de vérifier l'homoscédasticité des résidus, des tests comme celui de White [136] ou ARCH d'Engle (Engle ARCH test) [138] peuvent être utilisés.

En outre, l'évaluation de la précision des estimations du modèle peut se faire en calculant l'erreur absolue moyenne (*Mean Absolute Error*, MAE). Celle-ci, pour n échantillons, est égale à :

$$\mathrm{MAE} = \frac{1}{n}\sum_{k=1}^{n}|y(k)-x(k)| = \frac{1}{n}\sum_{k=1}^{n}|\varepsilon(k)|\,, \tag{I.7}$$

et l'erreur quadratique moyenne (*Mean Squared Error*, MSE) vaut :

$$\mathrm{MSE} = \frac{1}{n}\sum_{k=1}^{n}(y(k)-\hat{y(k)})^2 = \frac{1}{n}\sum_{k=1}^{n}\varepsilon(k)^2. \tag{I.8}$$

Ces deux informations permettent surtout de comparer les résultats de différentes instances du modèle où les résultats des modèles différents.

Dès lors que la qualité de l'ajustement est convenable, les résidus sont analysés, et les erreurs d'estimation sont suffisamment faibles pour satisfaire les exigences des concepteurs (par exemple MAE inférieur à la résolution des capteurs), le modèle est validé. Si le modèle ne peut pas être validé, le concepteur doit alors répéter les étapes à partir du choix de l'approche de modélisation. Ce cas de figure a été abordé dans [5]. La précision du modèle de puissance y a été considérablement améliorée par rapport à celle proposée dans une première étude [9] et ce, en changeant le type du modèle.

Après la validation du modèle, il faut s'assurer des choix de conception, notamment le niveau de granularité choisi et la faisabilité concernant le lieu de construction, l'apprentissage et la mise en œuvre du profiler. Enfin, Il est nécessaire de vérifier si les performances du profiler sont satisfaisantes. Il convient d'examiner la période d'échantillonnage et la charge de travail supplémentaire ajouté par le profiler au système. Un profiler qui augmente la charge du CPU par 25% et nécessite 2s pour générer une estimation de la puissance, n'est pas satisfaisant pour un appareil mobile. Il convient également d'analyser si l'adoption de certains choix pendant la conception l'emporte sur leurs inconvénients, par exemple le temps nécessaire pour générer une estimation lente contre une granularité très fine.

I.4.5 Analyse comparative et synthèse de méthodologie

La Figure I.5 montre un organigramme décrivant une méthodologie générique de la construction des profilers pour les systèmes embarqués hétérogènes. Cet organigramme montre toutes les étapes à suivre, et indique également, en cas de résultats non satisfaisants, quelles étapes devraient être réexaminées en fonction des contraintes sous-jacentes.

Enfin, le tableau I.1 affiche une compilation de profilers et de modèles trouvés dans la littérature classés selon les étapes expliquées ci-dessus.

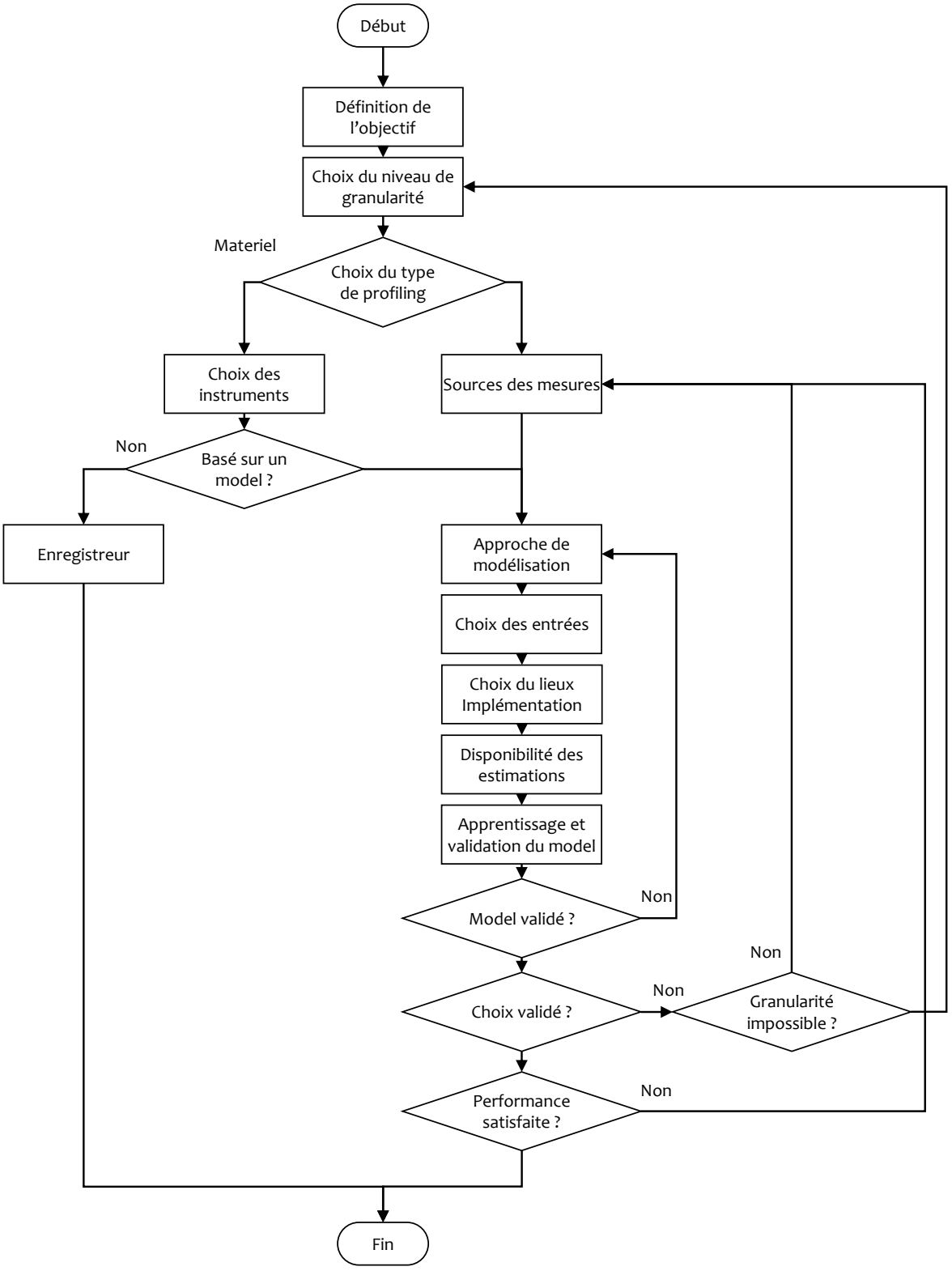


FIGURE I.5 – Un algorithme générique pour la construction de profilers pour les systèmes embarqués et mobiles.

TABLE I.1 – Résumé descriptif de certains profilers disponibles dans la littérature.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Snap-Dragon Profiler [135]	Estimation de la consommation électrique et logging	Système	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	HL	HS
Trepn [86, 139]	Estimation de la consommation électrique et logging	Système	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	HL	SS
Ahmad et al. [140]	Estimation de la consommation électrique	Instruction	Mixte	Matériel	Oui	Régression linéaire	HL	HL	HS
Huang et al. [47]	Optimisation de la consommation d'énergie	CPU	Logiciel	Traces du systèmes	Oui	Polynomial (non-linéaire)	HL	HL	HS
Niu and Zhu [116]	Amélioration de la fiabilité	CPU	Logiciel	Traces du systèmes	Oui	Régression linéaire	HL	HL	—

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Bokhari et al. [53, 141]	Optimisation de la consommation d'énergie	Application	Logiciel	API de la batterie	Oui	Spécifique à l'application	En / HL	En / HL	En / HS
Alawnah and Sagahyroon [105]	Estimation de la consommation électrique de la consommation d'énergie	Système	Logiciel	• API • Traces du systèmes	Oui	RNA	HL	HL	HS
Yoon et al. [99]	Estimation de la consommation électrique	Processeurs	Logiciel	Traces du systèmes	Oui	Régression linéaire	—	—	—
Walker et al. [100]	Estimation de la consommation électrique	CPU	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	EL	SS
PETrA [93]	Estimation de la consommation électrique	Application	Logiciel	Traces du systèmes	Oui	Régression linéaire	HL	EL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Djedidi et al. [9]	Surveillance du SoC (CPU, GPU, RAM)	Logiciel	Traces du systèmes	Oui	Neural networks	EL	HL	HS	
Deep Green [65]	Prédiction	Système	Mixte	Traces du systèmes + *GreenMiner*[142]	Oui	SVR	HL	HL	HS
pProf [94]	Profiling de puissance optimisé	Système	Logiciel	API	Oui	Régression linéaire	EL	HL	SS
Bokhari and Wagner [44]	Optimization	Application	Logiciel	• API de la batterie • Traces du systèmes	Non	—	—	—	—
Carvalho et al. [104]	Prédiction	Système	Mixte	• Matériel • Traces du systèmes	Oui	• RNA • Régression k-NN	HL	HL	HS
Kim et al. [41]	Prédiction de l'a	Application	Logiciel	Traces du systèmes	Oui	AppScope [143]	EL	HL	SS
Lu et al. [64]	Modélisation des écrans	Écran	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	EL	SS
Caviglione et al. [80]	Détection de logiciels malveillants	Système	Logiciel	• API de la batterie • Traces du systèmes	Oui	version modifiée de PowerTutor [18, 144]	EL	EL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Chowdhury and Hindle [95]	Estimation de la consommation électrique	Application	Logiciel	• API de la batterie • Traces du systèmes	Oui	• SVR • Régression linéaire	EL	EL	SS
Green-Oracle [95]	Estimation de la consommation électrique	Applications	Logiciel	Traces du systèmes	Oui	• Régression linéaire (Ridge) • Lasso • SVR • Bagging	HL	HL	HS
Dzhagaryan et al. [122]	Mesure	Applications	Matériel	—	Non	—	EL	—	HS
Linares-Vásquez et al. [145]	Mesure	CPU	Logiciel	Traces du systèmes	Non	—	EL	—	HS
Li and Mishra [121]	Mesure	CPU	Logiciel	Traces du systèmes	Non	—	EL	—	SS
Altamimi and Naik [146]	Estimation de la consommation électrique	• CPU • Mémoire de stockage	Mixte	• Matériel • Traces du systèmes	Oui	Régression linéaire	—	—	HS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
PowerTap [38]	Estimation de la consommation électrique	composants	Logiciel	• Matériel • Traces du systèmes	Oui	Régression linéaire	EL	HL	HS
WattsOn [97]	Estimation de la consommation électrique	Composants	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	HL	HS
Li and Mishra [121], Li et al. [147]	Estimation de la consommation électrique	Wi-Fi	Logiciel	• Matériel • Traces du systèmes	Oui	FSM	HL	HL	HS
Merlo et al. [76]	Detection d'intrusion	Système	Logiciel	Traces du systèmes	Oui	PowerTutor [18, 144]	EL	EL	SS
Jin et al. [148]	Estimation de la consommation électrique	GPU	Logiciel	Traces du systèmes	Oui	Régression linéaire	—	—	—
Kim et al. [13]	Estimation de la consommation électrique	GPU	Logiciel	Traces du système	Oui	Régression linéaire	EL	—	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Demirbilek et al. [54]	Amélioration de l'autonomie	Système	Matériel	—	Non	—	EL	—	HS
Martinez et al. [74]	Estimation de la consommation électrique (pour les nœuds IoT)	Système	Logiciel	System readings	Oui	Régression linéaire	EL	—	—
Sun et al. [123]	Estimation de la consommation électrique	Wi-Fi	Logiciel	Traces du systèmes	Oui	Régression linéaire	—	—	—
Lee et al. [149]	Modélisation automatique de consommantion en puissance	Application	Logiciel	Traces du systèmes	Oui	• Power Doctor [150] • Régression linéaire	—	—	—
FEPMA [98]	Enregistrement des mesures	Applications	Logiciel	Traces du systèmes	Oui	ANN	EL	—	—
Kamiyama et al. [132]	Optimisation de la consommation en puissance	Application	Logiciel	—	Oui	Régression linéaire	EL	—	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Park et al. [60]	Optimisation de la consommation en puissance	Composants	Matériel	• Traces du systèmes • Instrument de mesure	Non	EL	HL	—	HS
Arpinen et al. [151]	Gestion de la consommation	Composants	Logiciel	Traces du systèmes	Oui	FSM	EL	EL	SS
Holleis et al. [126]	Gestion de la consommation	Système	Matériel	—	Non	—	EL	—	HS
König et al. [152]	Mesure	Capteurs	Matériel	—	Non	—	EL	—	HS
Shin et al. [134]	Estimation de la consommation électrique de l'autonomie	Traces du système	Logiciel	Composants	Oui	Régression linéaire	EL	HS	HS
eLens [153]	Estimation de la consommation électrique	Ligne de code	Logiciel	Application	Oui	Regression	EL	EL	SS
Nacci et al. [154]	Estimation de la consommation électrique	Système	Logiciel	Traces du systèmes	Oui	Régression linéaire (ARX)	EL	EL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Kim and Chung [155]	Estimation de la consommation électrique	Composants	Logiciel	Traces du systèmes	Oui	Régression linéaire	—	—	—
Ardito et al. [81]	Enregistrement des valeurs de la puissance	Composants	Matériel	—	Non	—	HL	—	HS
V-edge [103]	Enregistrement des valeurs de la puissance	Composants	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	EL	SS
Kim et al. [92]	Power modeling	Composants	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	—	—
Power Doctor [150]	Estimation de la consommation électrique	• CPU • Ecran	Logiciel	• API • Traces du systèmes	Oui	Régression	EL	EL	SS
Murmuria et al. [129]	Measurement of power consumption	• Composants • Application	Logiciel	Traces du systèmes	Oui	Régression linéaire	HL	EL	SS
DevScope [88]	Analyse en ligne de la puissance consommée	Composants	Logiciel	• API de la batterie • Traces du systèmes	Oui	Régression linéaire	EL	HL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	*Objectif*	*Granularité*	*Type de Profiling*	*Source des mesures*	*Basé sur un modèle*	*Approche de Modélisation*	*Estimations*	*Apprentissage*	*Implémentation*
Minyong Kim et al. [156]	Estimation de la consommation électrique	Composants	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	—	—
eProf [127, 89]	Modélisation et Estimation de la consommation électrique	fil d'exécution (Thread)	Logiciel	Traces du systèmes	Oui	FSM	EL	EL	SS
Sesame [87]	Modélisation automatique et Estimation de la consommation électrique	Système	Logiciel	API de la batterie	Oui	Régression linéaire	EL	HL	HS
Power-Booter [18]	Modélisation et Estimation de la consommation électrique	Composants	Logiciel	• Capteurs de la batterie • Traces du systèmes	Oui	PowerTuTor (FSM + Régression)	EL	EL	SS
Yusuke et al. [157]	Surveillance de la consommation électrique	CPU	Logiciel	Traces du systèmes	Oui	Régression linéaire	EL	HL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

Travail	Objectif	Granularité	Type de Profiling	Source des mesures	Basé sur un modèle	Approche de Modélisation	Estimations	Apprentissage	Implémentation
Gurun and Krintz [158]	Estimation de la consommation électrique de la consommation électrique	• CPU • Réseaux	Logiciel	• Traces du systèmes • Capteurs de la Batterie	Oui	Régression linéaire	EL	HL	SS
AppScope [143]	Estimation de la consommation électrique de l'énergie consommée	Composants	Logiciel	Traces du systèmes	Oui	DevScope [88]	EL	EL	SS

— : Information non-disponible ou non-applicable. HS : Hors-système. SS : Sur le système. HL : Hors-ligne. EL : En-ligne.

I.5 Conclusion

Dans ce chapitre état de l'art, nous avons présenté une vue d'ensemble sur les SoC hétérogènes, leur instrumentation et leur modélisation, dans le but de constituer une base de connaissance suffisante pour entamer le travail de modélisation et de surveillance de ces systèmes. En effet, ce travail de recherche bibliographique nous a permis d'avoir une idée claire sur le fonctionnement des SoC, les interactions entre les composants matériels et logiciels, et les phénomènes physiques résultants (consommation électrique, transfert thermique...).

L'étude des travaux de modélisation existants dans la littérature a abouti à une analyse comparative, basée sur des critères de comparaison définis en termes d'objectifs, de granularité, le type d'instrumentation utilisée et l'approche de modélisation. Nous avons utilisé ce travail d'analyse, ensuite, comme support pour la synthèse d'une méthodologie de construction de profilers et modèles de SoC en fonction des objectifs à atteindre et de l'instrumentation choisie.

Enfin, les étapes de la procédure de modélisation définies dans ce chapitre sont suivies dans les chapitres II et III, pour la construction du modèle incrémental du SoC. En premier lieu, dans le Chapitre II, nous commençons par la définition de l'objectif du profiler, et choisissons en conséquence le type d'instrumentation permettant d'atteindre cet objectif. Puis, nous procédons au développement d'un outil d'acquisition et d'analyse de données provenant de l'instrumentation du système. Ensuite, dans le Chapitre III, nous entamons le reste des étapes de la procédure en passant à la construction et la validation du modèle incrémental.

CHAPITRE II

DESCRIPTION DU PROFILER ET ACQUISITION DES DONNÉES

II.1 Introduction

L'un des objectifs principaux de ce travail est de créer un modèle de référence pour l'estimation des variables caractérisant l'état de fonctionnement des SoC des systèmes embarqués. Pour ce faire, nous avons opté pour la création d'un profiler. Un profiler est un outil qui permet la mesure ou l'estimation périodique d'une ou plusieurs variables, dans un système informatique (voir I.3.1). Dans le chapitre précédent, nous avons aussi détaillé les étapes de la construction des profilers pour les SoC des systèmes embarqués (Figure I.5). Toutefois, les étapes à entreprendre et leur nombre dépendent fortement de la tâche que ces profilers doivent effectuer ainsi que l'approche de mesure ou d'estimation des variables. Étant donné que nous avons déjà établi que notre profiler générera les estimations de référence grâce à un modèle incorporé, sa construction sera donc limitée aux étapes qui concerne les profilers basés sur des modèles.

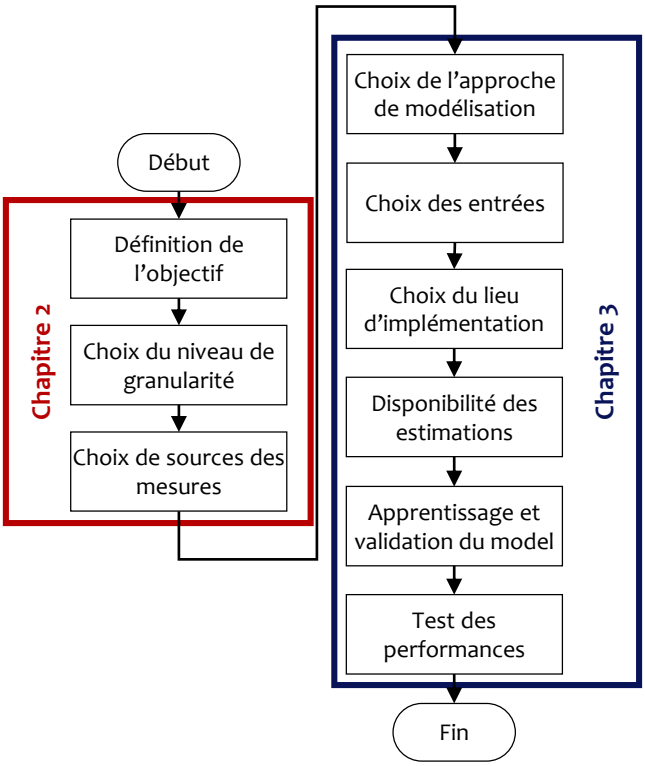


FIGURE II.1 – Un organigramme des étapes à entreprendre pour la construction des profilers basée sur des modèles.

Afin de bien présenter les choix de conception et mieux mettre en valeur la contribution de la modélisation, nous divisions la mise en œuvre du profiler en deux parties (Figure II.1).

La première partie, détaillée dans ce chapitre, sera dédiée à la définition des tâches du profiler, au choix des sources de mesures ainsi qu'au développement d'un outil adapté aux systèmes embarqués à modéliser pour l'acquisition et l'organisation de ces mesures. La seconde partie, développée dans

le Chapitre III sera dédiée aux étapes de modélisation, de validation des modèles et de test des performances.

Dans les prochaines sections de ce chapitre, nous commençons par la présentation des deux systèmes embarqués sur lesquels nous testons notre approche de modélisation et surveillance. Ensuite, nous définissons l'objectif du profiler ainsi que le niveau de granularité requis, dans la Section II.3. Enfin, dans la Section II.4, nous justifions le type de *profiling* choisi, et détaillerons le processus d'acquisition des données et mesures.

II.2 Les systèmes électroniques étudiés

Dans le dernier chapitre de ce manuscrit, nous présenterons un prototype servant de preuve de concepts pour les méthodes que nous développons dans les deux prochains chapitres. Toutefois, avant de tester et formaliser notre modélisation et notre algorithme de surveillance sur le prototype, nous avons choisi d'abord de tester ces méthodes sur des systèmes embarqués déjà établis et commercialisés. Notre choix s'est porté sur les smartphones.

Ce choix, non limitatif de supports d'application, est justifié par la similitude architecturale entre ces dispositifs et le système que notre partenaire industriel prévoie pour le produit final du projet MMCD. Les deux systèmes sont dotés de SoC basés sur l'architecture ARM et fonctionnent sous le même noyau (Linux), facilitant ainsi la migration de tout travail effectué sur le smartphone vers la carte prototype. De plus, l'omniprésence des smartphones ainsi que la facilité relative avec laquelle ils peuvent être programmés font d'eux les candidats les plus appropriés pour les essais préliminaires de la modélisation et de la surveillance. En outre, les smartphones sont ce qui se fait de mieux en termes de systèmes embarqués et fonctionnalités intégrées. Ceci pose des problèmes particuliers en termes de gestion de la consommation de puissance et d'évacuation de la chaleur. De ce fait, leur utilisation comme support d'application nous permettra de prendre en considération les différents algorithmes utilisés pour contourner ces problèmes tel que le DVFS, et l'étranglement thermique (*Thermal throttling*), et ainsi généraliser l'application de nos modélisations et algorithmes de surveillance sur tous les systèmes embarqués ayant la même architecture.

Nous avons utilisé deux smartphones dans cette étude de modélisation : un Samsung Galaxy S5 [14] et un Saumsung Galaxy S8+ [15]. Le premier est équipé d'un SoC *Snapdragon 801* de *Qualcomm* hébergeant un processeur quadricœur à fréquences variables – via DVFS – variant entre 0.3 –2.45 GHz. Il possède également un GPU *Adreno 330* avec des fréquences comprises entre 200 –578 MHz. Le SoC est couvert par la RAM du système, 2 GB DDR3 à faible consommation (Figure II.2).

Le second appareil est doté d'un SoC *Exynos 8895* plus moderne. Il est équipé d'un processeur octa-core agencé dans une configuration big.LITTLE [159] et fonctionnant à des fréquences allant jusqu'à 2.3 GHz, un GPU *Mali-G71 MP20* et 4 GB de RAM (Table II.1).

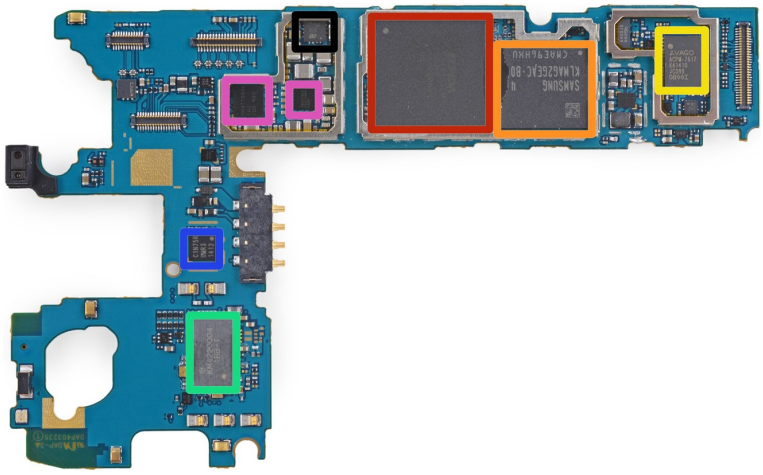


FIGURE II.2 – Le SoC *Snapdragon 801* (entouré en rouge) sur la carte mère du smartphone n° 1 (Source : [12]).

TABLE II.1 – Les spécifications techniques essentielles des deux smartphones utilisés dans cette étude [14, 15].

	Appreil Mobile n° 1	Appreil Mobile n° 2
OS	Android 6.0.1 (Marshmallow)	Android 8.0.1 (Oreo)
SoC • CPU • GPU • RAM	Snapdragon 801 Quad-cœur : — 2.5 GHz Krait 400 Adreno 330 2 GB	Exynos 8895 big.LITTLE Octa-cœur : — 4 × 2.3 GHz Mongoose M2 — 4 × 1.7 GHz Cortex-A53 Mali-G71 MP20 4 GB
Communication • Cellulaire • Wi-Fi • Blutooth • GPS	 GSM/HSPA/LTE Wi-Fi 802.11 a/b/g/n/ac 4.0 Oui	 GSM/HSPA/LTE Wi-Fi 802.11 a/b/g/n/ac 5.0 Oui
I/O • Écran tactile • Haut-parleurs • Appareils photos • Microphone • Vibration	 FullHD Super AMOLED 2 2 (Face avant et arrière) Oui Oui	 QHD+ Super AMOLED 2 2 (Face avant et arrière) Oui Oui
Batterie	2800 mA · h	3500 mA · h
Date de sortie	Février 2014	Mars 2017

II.3 Objectif du profiler : Monitoring du SoC

Dans ce travail, la méthode de surveillance que nous proposons vise à détecter rapidement les dérives des caractéristiques du système. La tâche principale de l'algorithme de surveillance sera d'assurer que le SoC se comporte correctement et fonctionne dans des conditions optimales, étant donné que ce dernier sera déployé dans des systèmes et des environnements critiques pour la sécurité (systèmes avioniques). Pour ce faire, nous étudierons le profil des signaux indicateurs de l'état de fonctionnement du SoC. Le profiler que nous développons dans ce chapitre à pour but de générer ces signaux. Il servira d'estimateur des variables qui caractérisent l'état de fonctionnement du système.

II.3.1 Variables caractéristiques du SoC

Dans les SoC, l'état de fonctionnement doit être décrit à la fois par des aspects logiciels et des aspects matériels. De plus, les variables sélectionnées doivent être mesurables, ou lisibles directement depuis le système pour faciliter leur acquisition et garantir la généricité de l'approche. Ainsi, nous avons sélectionné les variables suivantes pour décrire l'état de fonctionnement du SoC :

— **Charges de travail des cœurs du CPU (Load) :** Appelée aussi *utilisation* du CPU [160], la charge est la somme des temps que le processeur passe en exécution ou en attente (par exemple pour les entrées et les sorties) au cours d'une période d'échantillonnage, par rapport à cette période d'échantillonnage en pourcentage [161]. La charge du processeur ne doit pas être confondue avec la charge moyenne [162] et l'utilisation par rapport au processus [160]. Bien que l'industrie hésite à utiliser un nom, nous avons opté pour le terme utilisé par le fabricant des SoC utilisés dans ce travail, c'est-à-dire la charge (CPU Load) [139].

— **Charge du GPU :** Même définition que pour la charge du processeur, il s'agit de la somme des temps que le processeur passe occupé ou en attente pendant une période d'échantillonnage, par rapport à cette période d'échantillonnage en pourcentage [139].

— **Taux d'occupation de la mémoire (MOR) :** L'utilisation de la mémoire joue un rôle important dans la consommation d'énergie [163] et dans le comportement thermique du SoC [164]. Pour caractériser l'influence de la RAM sur la température du SoC et sa consommation d'énergie, nous devons inclure sa valeur en tant qu'entrée dans ces modèles. Cependant, la valeur de la RAM seule n'indique ni la valeur de base utilisée ni le maximum. Ainsi, nous définissons le «*Memory Occupation Rate*» (MOR) comme le ratio de la RAM occupée (mémoire) par rapport à sa taille maximale.

— **Mesures physiques du SoC :**
  — Les fréquences par cœur
  — La fréquence du GPU
  — Les tensions par cœur
  — La tension du GPU

— La température du SoC

— La puissance consommée par le SoC et le système.

II.3.2 Granularité du profiler

Dans le paragraphe précédent, nous avons établi la liste des variables à surveiller pour déterminer l'état de fonctionnement du SoC. Le *profiling* de ces variables ne nécessite pas un niveau de granularité particulièrement fin. Toutefois, l'estimation de la puissance consommée par le SoC nécessite aussi l'estimation de la puissance des autres composants, étant donné que les mesures directes ne délivreront que la consommation de tout l'appareil (voir I.4.1). Par conséquent, le profiler aura un niveau de granularité *«composants»*.

Dans ce travail nous nous sommes limités aux composants suivants :

— **Le système sur puce (SoC)** : dans la plupart des systèmes embarqués, il est composé du processeur, de la mémoire vive (RAM) et d'un processeur graphique (GPU) en cas de besoin.

— **Le modem cellulaire**

— **Le module GPS**

— **Le module sans fil :** Wi-Fi et Bluetooth.

— **L'écran tactile**

— **Les haut-parleurs et le microphone**

— **Les modules des caméra** : contenant la caméra frontale, la caméras de l'arrière et le flash.

Cette limitation vient du fait que ces composants sont les principaux facteurs en matière de consommation d'énergie [83, 66]. Bien que les smartphones contiennent bien d'autres périphériques, nous considérons que leurs consommations électrique est négligeable [66] et fait partie de l'énergie de mise en marche de l'appareil (dite statique). Par exemple, la couche tactile de l'écran consomme une quantité d'énergie négligeable par rapport à l'affichage [165, 92].

Durant nos expérimentations préliminaires, nous avons pu tester et confirmer cette hypothèse. La Figure II.3 montre le résultat de l'un de ces tests, où la consommation d'énergie de l'appareil mobile n° 2 est mesurée durant plusieurs étapes. Dans la première étape (points bleus), la fréquence est minimale et l'écran est éteint. Au cours de la deuxième étape (ligne grise), le capteur d'empreintes digitales est activé et utilisé avec l'écran toujours éteint. Durant cette phase, malgré l'activation du capteur, la consommation reste inchangée. La même consommation est toujours observée dans la troisième étape (tirets rouges), durant laquelle nous avons utilisé le capteur infra-rouge du rythme cardiaque. Par-contre, la consommation électrique augmente considérablement durant la dernière étape (tirets verts), où la fréquence n'est plus limitée et l'écran est allumé.

Ainsi, le profiler collectera les données, générera une estimation de la puissance avec son modèle de puissance incorporé pour l'appareil dans son ensemble et ses composants individuellement (voir III.4.1), puis enregistrera ces données.

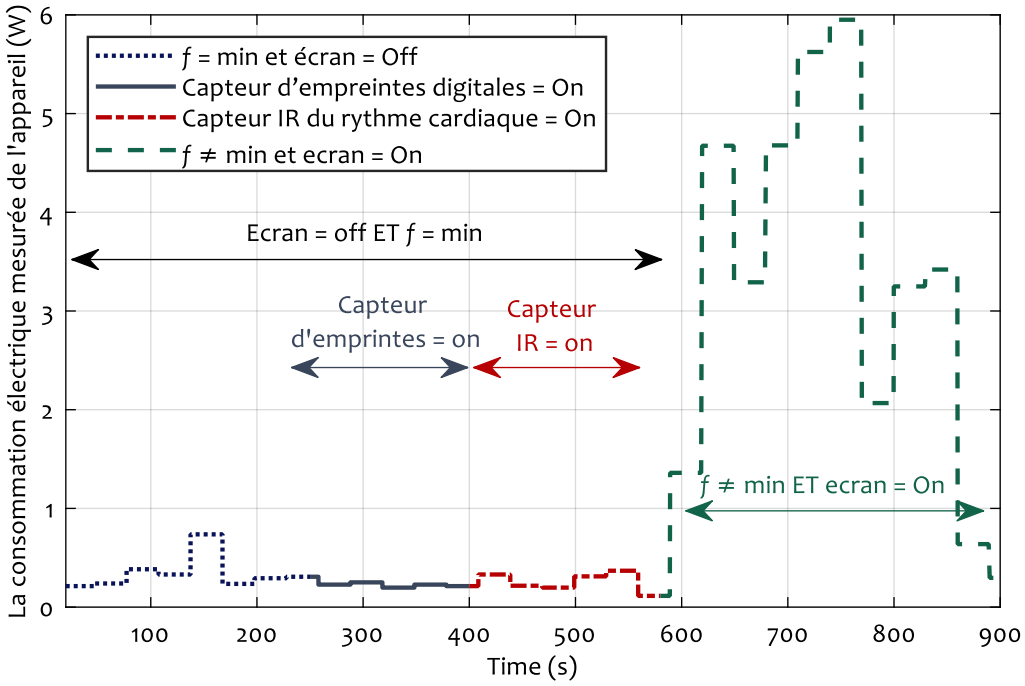


FIGURE II.3 – La consommation électrique globale mesurée de l'appareil, étape par étape. Des composants différents sont activés à chaque étape.

II.4 Type du *profiling* : *Profiling* logiciel

L'abondance des smartphones, ainsi que la facilité de leur programmation, permet de développer des applications dédiées facilement et pratiquement sans frais ou risque. Par conséquent, nous avons opté pour un profiler logiciel. Ce type de profiler est une solution non-intrusive offrant un rendu optimal pour les smartphones modernes. Le profiler utilisera ainsi les fichiers et les traces du système pour ses lectures et mesures.

Ce type de *profiling* et cette source de mesure conviennent parfaitement au niveau de granularité choisi. De plus, les résultats obtenus avec ces profilers ont une précision comparable à celles obtenues avec des instruments de mesures [125, 95].

Le profiler sera composé de deux parties : un enregistreur de données et un modèle d'estimation. Tout d'abord, l'enregistreur de données collectera les informations pertinentes à partir de l'appareil. Ensuite, le modèle utilisera ces informations comme entrées pour générer une estimation des variables caractéristiques (les fréquences, la puissance consommée, la température...). Enfin, le profiler organisera et écrira les données acquises et les estimations dans un ficher texte sous forme de valeurs séparées par des virgules (*Comma-separated values*, CSV).

Nous avons nommé ce profiler N^3. Ce nom a été choisi pour illustrer la version du profiler (3ème version), ainsi que son utilisation des réseaux de neurones (voir III.4.1).

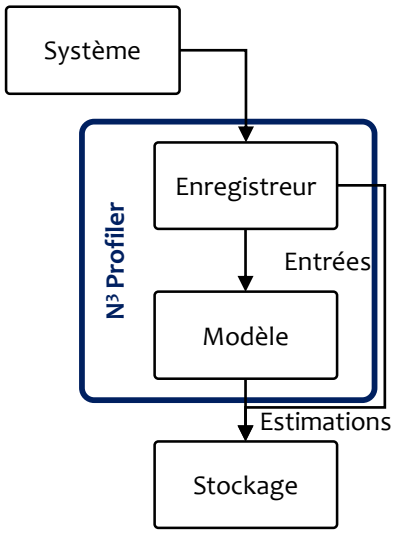


FIGURE II.4 – Architecture générale du profiler N^3.

II.4.1 Instrumentation des appareils étudiés

Le modèle que nous aborderons dans la prochain chapitre, est un modèle comportemental, décrivant l'état de fonctionnement du système à l'aide de l'estimation de la dynamique de certaines variables, c'est-à-dire : la charge de travail, les fréquences, les tensions, la puissance consommée et la température du SoC. Ces variables n'étant pas directement accessibles à la mesure, nous développerons dans un premier temps une application permettant leur lecture. Ce paragraphe sera donc consacré à la mesure et à la lecture des variables dont l'évolution nous intéresse.

II.4.1.1 Sources des mesures

Les smartphone sont des systèmes fournis avec toutes les mesures nécessaires à leurs fonctionnements déjà pré-installés et programmés. L'accès au données délivrées par ces capteurs se fait généralement via deux méthodes (voir I.4.2) : le suivi des fichiers et traces du systèmes et les classes dédiées dans le kit de développement (*Software Development Kit*, SDK) de l'OS.

Généralement, il est conseillé d'utiliser les interfaces de programmation (*Application Programming Interface*, API) de l'OS, car elles offrent plus de stabilité et sont plus faciles à programmer. Toutefois, nous avons opté pour la lecture des fichiers système pour deux raisons. Premièrement, par défaut, l'OS génère des traces pour indiquer son état actuel et les états des périphériques. Ces traces contiennent toutes les variables que nous utilisons, contrairement aux API (Table II.2). Alors, au lieu d'encombrer le système avec des requêtes d'interrogations sur l'état d'un périphérique ou sur la valeur d'une variable, nous les cherchons directement dans les traces déjà générées en parallèle. Deuxièmement, en testant les temps d'acquisition, nos expérimentations ont également montré qu'il est en réalité plus rapide de lire les traces du système que d'utiliser les API système (Tableau II.2).

TABLE II.2 – Une comparaison entre les temps nécessaires en moyenne pour accéder à la valeur de certaines variables via les API d'Android et les fichiers systèmes, sur l'appareil mobile n° 1. *ND* : Non-disponible.

Variable	**API** (ms)	**Traces du systèmes** (ms)
Fréquence d'un core du CPU	*ND*	~ 2
Fréquence du GPU	*ND*	>2
Etat de l'écran	8	2
Luminosité de l'écran	8	>2
Etat de l'appel	15	~ 2
La tension de la batterie	15	5
Le courant de la batterie	10	2

Dans la suite de ce paragraphe, nous listerons les variables incluses dans le processus d'acquisition ainsi que les traces et fichiers systèmes utilisés pour obtenir leurs valeurs. Néanmoins, nous notons que les liens des fichiers que nous donnons sont à titre d'exemple, car ils varient d'un constructeur à l'autre et sont le résultat de l'exploration et des recherches de données que nous avons effectués sur les appareils étudiés.

II.4.1.2 Acquisition des variables du SoC

II.4.1.2.1 Calcul de la charge de calcul du processeur (*Load*) : Dans le Paragraphe II.3.1, nous avons décrit la charge de calcul des processeurs. Le calcul de cette variable se fait pour chaque coeur individuellement, à partir des statistiques du processeur. Le noyau de Linux propose un fichier de statistiques :

`/proc/stat`

```
cpu  2441381 1805748 2493148 29546117 332086 229 329235 0 0 0
cpu0 1026646 729930 1198563 14555820 181051 210 199379 0 0 0
cpu1 233639 117240 188443 554999 28096 1 17748 0 0 0
cpu2 330768 210289 292826 806789 39276 8 26967 0 0 0
cpu3 850328 748289 813316 13628509 83663 10 85141 0 0 0
intr 179355647 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 71692151 0 0 0 0 0 0 0 0 0 0 0 0 0 500366 0 0 0 0 0 0 3982833 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 35760 90564 5 0 5736364 0 0 0 1207849 0 0 0 0 0 0 0 0 0 0 400039 0 0 0 0 293 3216 8 0 10 0 31
6 0 652 1230 0 0 0 0 0 0 0 0 0 0 0 0 0 1166246 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 38 1960519 0 0 22 0 44 0 0 21
78 2496060 0 0 5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20339633 0 172205 0 366746 0 0 0 19495313 0 0 0 0 0 0 3805 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 480551 0 5 0 0 0 0 35458 8750 0 0 0 3486074 0 0 486 0 0 0 0 0 0 0 0 0 0 0 0 1108 0 0 0 0 0 161977 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 328888 0 0 0 0 0 0 0 0 0 0 0 0 17 0 0 6411 0 0 0 0 0 0 0 0 0 0 0 0 0 252 2132 0 0 0 0 0 0
0 0 2 0 0 0 0 0 0 0 501 0 418 14 0 0 0 0 0 0 0 233 0 0 1 0 0 3133 0 0 0 0 0 0 0 0 26 0 0 12 0 3032 0 0 0 0 45 68 0 0 45
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 20847 1
304179 894 159707 0 0 0 0 12 10 369 20 29 178 501 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
ctxt 483538508
btime 1465683377
processes 545359
procs_running 5
procs_blocked 0
softirq 106463264 33347 11783738 228813 7009703 33347 33347 11590806 12548259 41403 63160501
```

FIGURE II.5 – La lécture du fichier `Proc/stat` pour l'appareil mobile n° 1.

La Figure II.5 montre l'exemple d'une lecture du fichier `stat`. Les statistiques nécessaires pour le calcul de la charge se trouvent dans les lignes taguées «cpu ». La première ligne de ce ficher

montre les statistiques du CPU dans son ensemble, tandis que les n lignes suivantes – ou n est le nombre de coeurs dans le CPU – mettent en évidence les statistiques de chaque coeur. Dans l'exemple de la Figure II.5, n est égale à 4, et les lignes pertinentes sont les lignes taguées `cpu0` à `cpu3`. Ces statistiques correspondent à la durée (en milliseconde) des périodes que le coeur CPU a passé dans l'état que représente chaque colonne. Ces états sont :

1. **`user`** : le temps que le processeur passe à exécuter des tâches pour l'utilisateur.
2. **`nice`** : idem.
3. **`system`** : le temps que le CPU passe à exécuter des tâches pour le noyau du système.
4. **`idle`** : le temps d'inoccupation.
5. **`iowait`** : le temps d'attente pour qu'une opération sur les entrées et les sorties soit finie.
6. **`irq`** : le temps pris pour la gestion et l'exécution des interruptions.
7. **`softirq`** : idem.

Les trois dernières colonnes consacrées aux serveurs et aux machines virtuelles et ne concernent pas le noyau d'Android.

La charge est définie comme le rapport (en %) entre le temps d'occupation (de travail) et le temps total de la période d'échantillonnage. Pour la calculer, le fichier `stat` est lu une première fois pour initialiser les valeurs des temps à leurs valeurs actuelles. Puis, après une période d'échantillonnage, ce fichier est lu une seconde fois. Puis, nous calculons la différence entre les valeurs des deux lectures. Ces différences représentent les périodes que le CPU et ses coeurs ont passé dans chaque état pendant la période d'échantillonnage. La formule suivante est ensuite appliquée :

$$Load_{CPU_x} = \frac{t_{user} + t_{nice} + t_{system} + t_{iowait} + t_{irq}t_{softirq}}{t_{user} + t_{nice} + t_{system} + t_{iowait} + t_{irq}t_{softirq} + t_{idle}} \times 100 \tag{II.1}$$

II.4.1.2.2 Lecture des fréquences des coeurs CPU et GPU : En suivant l'exemple de la charge, les valeurs de la fréquence peuvent être obtenues en lisant les fichiers système. ceux-ci se trouvent dans le répertoire :

```
/sys/devices/system/cpu/
```

Ce répertoire contient des sous-répertoires pour chaque coeur, et ces derniers comportent, entre autres, des fichiers contenant les valeurs des fréquences minimales (`cpuinfo_min_freq`), et maximales (`cpuinfo_max_freq`), ainsi que la fréquence actuelle (`cpuinfo_cur_freq`). On trouve aussi la table des fréquences (`scaling_available_frequencies`) et le gouverneur actuellement utilisé. Ce dernier est le programme utilisé pour l'ajustement de la fréquence. Nous détaillerons ces gouverneurs dans le prochain chapitre traitant de la modélisation.

De la même façon, nous pouvons calculer la charge et lire la fréquence du GPU dans le répertoire correspondant. Par contre, la localisation de celui-ci varie d'un constructeur à l'autre. Par exemple dans le cas de l'appareil mobile n° 1, la fréquence du GPU (et même la charge pré-calculée) se trouvent dans le répertoire :

```
/sys/devices/platform/13900000.mali/
```

II.4.1.2.3 Lecture des tensions des processeurs : Sous Android, la régulation de la tension des composants est gérée par des programmes appelés «*regulators* ». Les traces relatives à ces programmes se trouvent dans le répertoire :

```
/sys/class/regulator/
```

Les sous-répertoires de ce dossier sont des régulateurs dédiés chacun à un composant spécifiques. La difficulté devient donc l'identification des composants associés à chaque régulateur. Dans plusieurs cas, le nom donné au régulateur donne une indication du composant associé, comme dans le cas du CPU de l'appareil mobile n° 1 (Table II.3). Cependant, le plus souvent, le nom n'indique pas directement le composant associé. Dans ce cas, les changements des valeurs des composants (par exemple la fréquence du GPU) sont surveillés en même temps que les changements des valeurs de la tension indiquées dans les traces des répertoires régulateurs pour associer le composant à son régulateur.

La Table II.3 montre un exemple des tensions inscrites dans les traces des *regulators*. Les régulateurs des coeurs du CPU sont faciles à associer à leurs composants grâce à leurs noms (`Regulator.47` à `Regulator.50`), contrairement à celui du GPU (`Regulator.34`).

TABLE II.3 – Un exemple des données disponibles dans les traces des *regulators* pour le GPU (`Regulator.34`) et les quatre coeurs du CPU de l'appareil mobile n° 1 (`Regulator.47` à `Regulator.50`).

Répertoire	Name	Type	Max microvolts	Min Microvolts	Microvolts
Regulator.34	8084_l19	voltage	3300000	2900000	3300000
Regulator.47	krait0	voltage	1120000	500000	1050000
Regulator.48	krait1	voltage	1120000	500000	500000
Regulator.49	krait2	voltage	1120000	500000	835000
Regulator.50	krait3	voltage	1120000	500000	875000

II.4.1.2.4 Acquisition de la température du SoC : La température du SoC est nécessaire pour la régulation de la fréquence en cas de forte température. Par conséquent, elle est mesurable sur la grande majorité des systèmes, mais l'emplacement de la mesure varie d'un constructeur à un autre, et peut même varier entre deux systèmes conçus par un même constructeur. La Table II.4 montre une liste de fichiers utilisés pour la lecture de la valeur de la température de plusieurs systèmes montrant la diversité de ces fichiers. Le fichier est généralement localisé à l'aide d'une fouille de données. Une fois le fichier localisé, la valeur de la température est lue comme dans les cas précédents.

II.4.1.2.5 Acquisition du MOR : Dans le Paragraphe II.3.1 de ce chapitre, nous avons défini le MOR comme le rapport de la RAM utilisée sur sa valeur totale. La valeur totale de la RAM du système est une constante. Par contre, la valeur de la RAM utilisée change en fonction des programmes en

cours d'exécution. Le fichier `/proc/meminfo` donne dans ces trois premières lignes, la valeur totale de la RAM ($MemTotal$), la quantité de RAM libre ($MemFree$), et celle disponible ($MemAvailable$) respectivement (Figure II.6).

```
beyond2:/proc $ cat meminfo
MemTotal:        7585856 kB
MemFree:         1115680 kB
MemAvailable:    2195084 kB
```

FIGURE II.6 – Un exemple de la lecture du fichier `/proc/meminfo`, avec les trois premières lignes qui indiquent la quantité totale de la RAM et les quantités libres et disponibles.

Une fois ce fichier `/proc/meminfo` est lu, le calcul du MOR est facilement établi par la relation suivante :

$$MOR = \frac{MemTotal - (MemFree + MemAvailable)}{MemTotal} \tag{II.2}$$

II.4.1.3 Acquisition des variables du système

En plus des variables décrivant le SoC, il faut aussi faire l'acquisition des mesures et lectures relatives à la consommation de puissance. Toutefois, les traces fournies par le système donnent les mesures délivrées par la batterie, qui concernent donc tout le système. Par conséquent, l'acquisition d'autres variables supplémentaires est indispensable pour atteindre la granularité nécessaire à l'estimation de la puissance au niveau des composants. Ces variables serviront à caractériser la consommation de chaque composant de l'appareil, en indiquant si le composant est actif ou pas, ainsi que le type d'activité (téléchargement pour le module Wi-Fi, par exemple).

La plate-forme utilisée, comme pour tout les téléphone Android n'offre pas une lecture directe de la puissance consommée. Néanmoins, les traces du système fournissent les mesures du courant I

TABLE II.4 – Une liste d'exemples des fichiers utilisés pour la lecture de la température dans les systèmes embarqués et mobiles. Cette liste montre la différence entre les constructeurs de ces systèmes.

	Chemin du fichier de la température
1.	`/sys/devices/platform/omap/omap_temp_sensor.0/temperature`
2.	`/sys/kernel/debug/tegra_thermal/temp_tj`
3.	`/sys/devices/system/cpu/cpu0/cpufreq/cpu_temp`
4.	`/sys/class/thermal/thermal_zone0/temp`
5.	`/sys/class/thermal/thermal_zone1/temp`
6.	`/sys/devices/platform/s5p-tmu/curr_temp`
7.	`/sys/devices/virtual/thermal/thermal_zone0/temp`
8.	`/sys/devices/virtual/thermal/thermal_zone1/temp`
9.	`/sys/devices/system/cpu/cpufreq/cput_attributes/cur_temp`
10.	`/sys/devices/platform/s5p-tmu/temperature`
11.	`/sys/bus/i2c/devices/9-004c/temperature`

débité par la batterie, ainsi que la tension V à ses bornes, permettant le calcul de la puissance P (W) :

$$P = VI \tag{II.3}$$

Ces traces permettent aussi d'obtenir le niveau de batterie (%), son état (en charge ou en décharge) et sa température.

Ces mesures, ainsi que les lectures qui concernent les autres composants (détaillés dans le Chapitre III), peuvent êtres localisées dans les répertoires `/sys/class` et `/sys/devices`. Ces derniers contiennent des sous-répertoires dédiés à chacun des composants. La tables II.5 montre une liste des répertoires trouvés dans `/sys/class` de l'appareil n° 2, et les composants associés aux traces contenues dans ces répertoires.

TABLE II.5 – Un exemple d'une liste de composants et les répertoires des traces qui leurs sont associées dans le répertoire `/sys/class` pour l'appareil mobile n° 2.

Composant	Chemin du répertoire
Batterie	`/sys/class/power_supply`
Bluetooth	`/sys/class/bluetooth`
Capteurs photos	`/sys/class/camera`
Capteur d'emprintes	`/sys/class/fingerprint`
Ecran	`/sys/class/lcd`

II.4.2 Développement de l'application d'acquisition sous Android

Dans le paragraphe précédent, nous avons exploré les fichiers sources des mesures essentielles pour la modélisation. Dans qui suit, nous présenterons l'application que nous avons construit pour les lectures, l'enregistrement – et éventuellement l'estimation – de ces données. Ce type d'application est la version la plus commune des profilers existants sur le marché, à l'exemple du célèbre profiler venant de chez Qualcomm, le *Trepn Profiler*. Toutefois, la plupart des applications qui existent déjà permettent d'afficher les variables désirées, mais pas de les enregistrer pour une analyse ou traitement ultérieure. En plus, celles qui permettent le transfert ou la sauvegarde des données ne permettent pas d'ajuster la période d'échantillonage, ou de choisir le protocole de communication avec un matériel externe. Afin de contourner ces lacunes, nous avons développé une nouvelle application, et l'avons nommé *CommunicatingProfiler*.

Cette application est elle-même la première version du profiler que nous développons. Elle est composée de deux grande partie, d'une interface graphique et un service d'écriture et d'enregistrement. L'interface graphique, comme son nom l'indique, est nécessaire pour l'affichage des lectures en temps réel, alors que le service est un processus qui tourne par défaut en arrière plan [166].

II.4.2.1 Le service d'acquisition et enregistrement des données

Après avoir identifié les traces nécessaires dans les fichiers systèmes, nous avons procédé au développement des classes granulaires qui ne se spécialisent que dans la lecture des variables d'un seul composant. Par exemple, la classe `CpuVoltagesAndFrequencies` fait exactement ce que son nom le présume : lire les fréquences et les tensions du CPU.

Ensuite, nous avons créé le service `RecordingService` pour lire et enregistrer ou transférer les données nécessaires. Ce service lit à chaque période de temps prédéfinie (T_s) les valeurs des variables.

Afin d'avoir une corrélation correcte entre les entrées et la sortie, le service lit les données simultanément pour minimiser le temps de lecture. Il divise les tâches de lecture en deux moitiés et lances ainsi deux threads d'exécution. Chacun des threads est instancié par un horodatage, puis il procède à l'acquisition des données à partir des fichiers système. Une fois la lecture terminée, chacun des threads clôture la liste des données acquises par un second horodatage (Figure II.7).

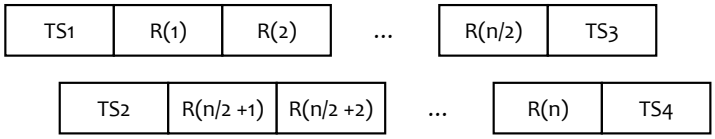


FIGURE II.7 – Division de la tâche d'acquisition des données : La lectures simultanées de n éléments de données avec des horodatages (*Timestamps*, TS).

L'horodatage des threads nous permet de garantir la validité des données, car l'application compare le temps du début et de la fin de chaque thread :

$$\begin{matrix} TS3 - TS1 \\ TS4 - TS2 \end{matrix} \tag{II.4}$$

Ensuite, Elle compare le début du thread qui a commencé en premier avec la fin du second, pour calculer le temps de l'ensemble du processus ($TS4 - TS1$). Tout le processus ne devrait pas prendre plus de 20 ms (la moyenne est autour de 10 ms), avec chaque thread prenant au plus 10 ms. Si ces délais sont dépassés, les données sont considérées comme corrompues et ignorées.

Le choix de 10 ms pour chaque thread est basé sur le temps que le noyau Linux accorde à chaque processus, avant de le suspendre et de passer à un autre, à cause des contraintes de la gestion des tâches et pour être multitâche (*multitasking*). Alors que la limite 20 ms est la période d'échantillonnage que le service doit respecter.

Le système sur lequel nous travaillons est très rapide en matière de changements et de variations des variables, notamment la charge et la fréquence. Par conséquent, la qualité des résultats dépend directement de la période d'échantillonnage choisie. Cette dernière dépend de l'algorithme du DVFS qui calcul la charge et la fréquence chaque 20 ms [167]. Nous avons donc choisi cette période $Ts = 20\,\text{ms}$.

$$\begin{cases} TS3 - TS1 & \leq & 10\,\text{ms} \\ TS4 - TS2 & \leq & 10\,\text{ms} \\ TS4 - TS1 & \leq & T_s,\ T_s = 20\,\text{ms} \end{cases} \tag{II.5}$$

Enfin, l'ensemble des valeurs calculées et lues est passé à une dernière fonction qui sert – selon la configuration de l'utilisateur – soit à les enregistrer dans un fichier de type `csv`, soit à les transférer en temps réel à une machine externe pour le traitement.

II.4.2.2 L'interface graphique pour l'analyse

L'interface graphique de l'application est une interface de commande et d'affichage. Elle permet de lancer et d'arrêter le `RecordingService`, ainsi que d'afficher les variables, les valeurs et les courbes des variables mesurées et estimées en temps réel.

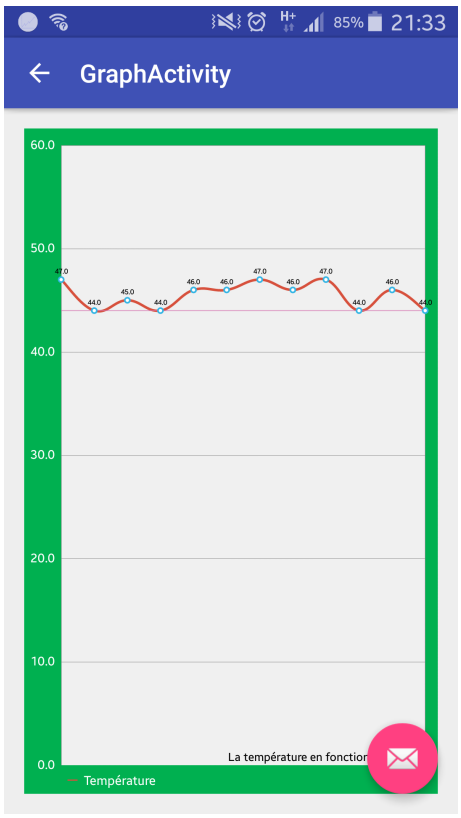


FIGURE II.8 – Une Capture d'écran de l'application *CommunicatingProfiler* affichant la courbe de la température du CPU.

II.4.3 Analyses des données acquises

Pendant l'acquisition des données, nous avons rencontré deux problèmes qui impactaient leur qualité. Le premier est lié à l'ordre des priorités de l'exécution des programmes. Le système d'exploitation planifie et exécute les tâches des programmes Java suivant la charge du système et un ordre de

priorité prédéfinies [168]. Une opération d'affichage par exemple, a plus de priorité qu'un processus exécuté en arrière plan. Par conséquent, dans les périodes de forte charge de travail, le service peut être retardé dans l'ordre des priorités, et voir sa période d'échantillonnage non respectée, donnant lieu à des périodes d'acquisition très grandes.

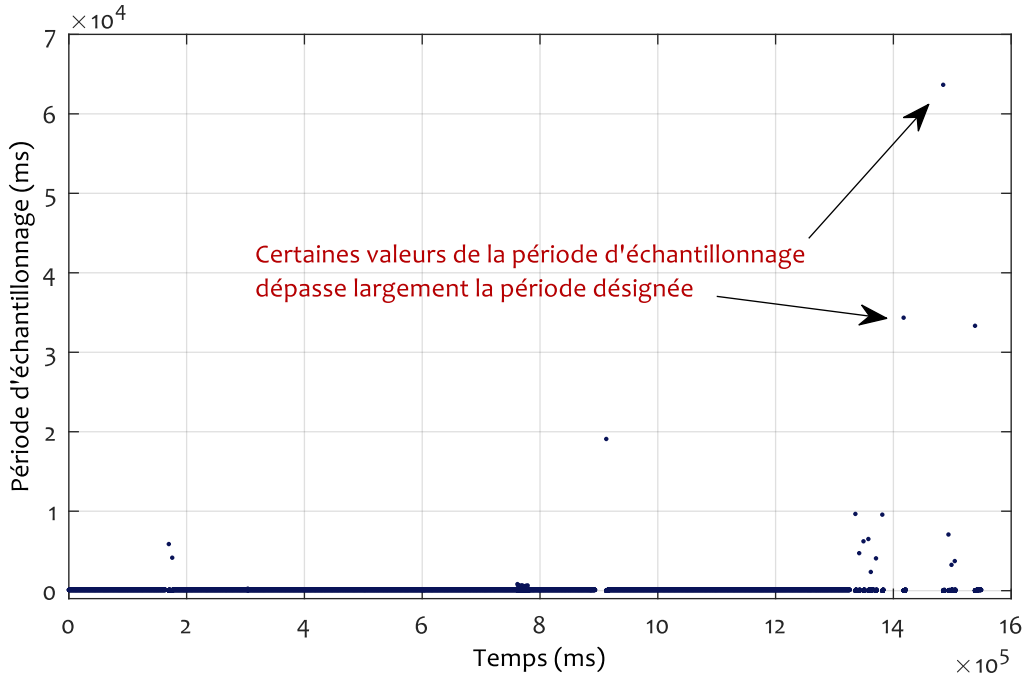


FIGURE II.9 – La variation des valeurs de la période d'échantillonnage pendant une expérience d'acquisition de données.

La Figure II.9 montre les périodes entre deux acquisitions lors d'une expérience de lecture où la période d'échantillonnage spécifiée est égale à $T_s = 20$ms. On remarque que les pas de lecture de certaines valeurs dépassent largement la période définie T_s, et atteignent même des dizaines de secondes.

La seconde difficulté dans la lecture des variables est liée à la gestions des files d'exécution par le système d'exploitation. Pour comprendre ce problème, il faut d'abord comprendre comment le système d'exploitation – Linux – gère les applications.

Pour le système d'exploitation, les applications correspondent à des files d'exécution, qu'ils laissent passer à tour de rôle selon un ordre de priorité prédéfini. Par exemple, si un utilisateur écoute de la musique et navigue sur internet en même temps (supposant qu'aucune autre tâche n'est en exécution), le système d'exploitation va laisser passer les instructions de l'application de la musique pendant un certain temps, pour que le processeur les exécute, puis il changera de contexte vers le navigateur et l'affichage. Tout cela se fait d'une manière qui, pour l'utilisateur, paraît continue (44 kHz pour la musique et 60 images par seconde pour l'affichage, par exemple). Si une application déborde, ou dépasse la période de temps qui lui est attribuée sans finir son exécution, elle doit attendre son prochain tour pour pouvoir la terminer.

Pendant les périodes où la charge du CPU est assez forte, nous avons remarqué que la file d'acquisition est parfois très longue et donc entrecoupée. en plus pendant ces périodes, le système essaiera quand même de respecter le temps d'échantillonage et lancer de nouvelles lectures, même si les précédentes ne sont pas finie, rendant ainsi les mesures prises lors de cette lecture fausses (désynchronisées, mesures en retards et trous de données). La Figure II.10 montre sur quelques échantillons, des temps de lecture allant jusqu'à 7 secondes. Ceci explique la contrainte que nous avons mis pour les donnée, où le temps d'une lecture $TS4 - TS1$ doit être inférieure au temps d'échantillonnage T_s (Équation II.5).

Étant donné qu'une synchronisation ou réparation de ces lectures est impossible, vu le changement rapide des variable, les données acquises pendant ces intervalles de temps de lecture, seront automatiquement supprimées de la base de données que nous utiliserons pour l'apprentissage et la modélisation. Nous notons aussi que nous n'avons rencontré ces cas de figure extrême que pendant des scénario d'exécution de benchmarks pendant une période d'étranglement thermique. Un scénario que nous avons choisi pour tester les limite de notre application.

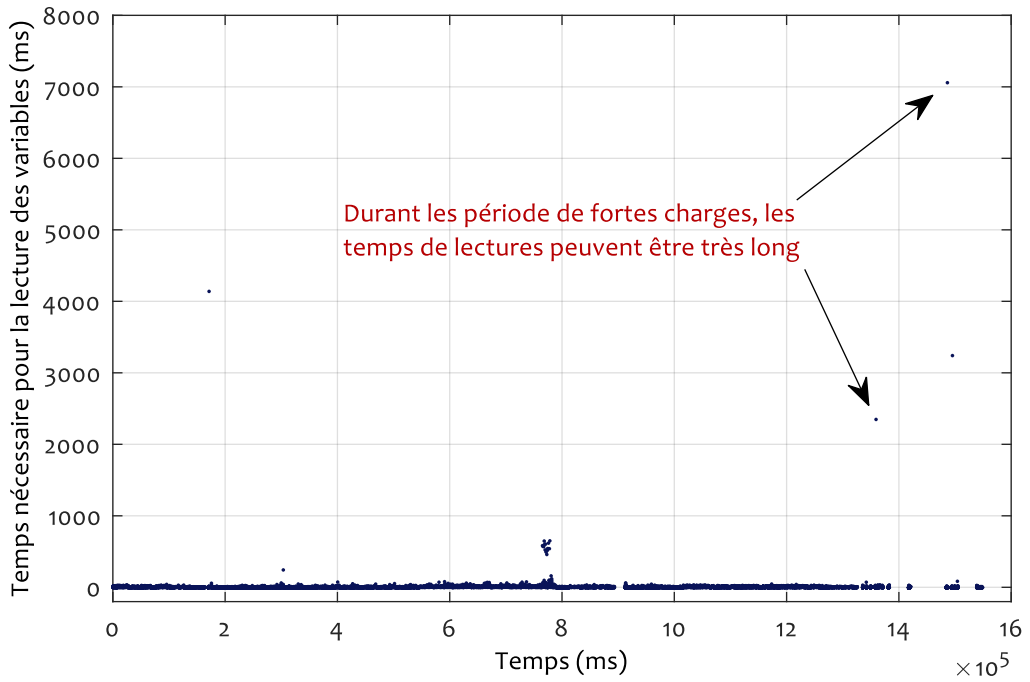


FIGURE II.10 – Les différentes entre le temps de début et le temps de fin des lectures en milli-secondes (ms).

II.5 Conclusion

Dans ce chapitre, nous avons entamé la construction d'un profiler basé sur un modèle, et présenté un outil d'acquisition et de traitements des données issues des SoC. L'organisation et la construction du profiler ont été menés afin garantir sa portabilité sur la majorité des SoC existants (après reconfiguration des chemins des traces), en particulier les SoC proposés dans le cadre du projet MMCD. Son objectif a été bien défini comme étant l'estimation ou le calcul des variables caractérisant l'état de fonctionnement du système. Ces variables ont été identifiées au préalable, après une analyse fine du fonctionnement du SoC.

Dans l'étape d'analyse et de traitement des données enregistrées, deux problématiques ont été identifiées et corrigées. La première concerne l'ordre de priorité des tâches, qui peut engendrer, dans les périodes de forte charge, un dépassement de la période d'échantillonnage prédéfinie. La seconde problématique est liée à la gestion des files d'exécution par le système d'exploitation, qui provoque des coupures, et donc une désynchronisation des données enregistrées. Ces contraintes ont été levé grâce à une procédure de détection et de suppression automatique des échantillons désynchronisés.

Ce traitement permet d'obtenir des séries temporelles dont les données synchronisées sont facilement exploitables et serviront à l'apprentissage, à la validation et aux tests des sous-systèmes constituant le modèle incrémental.

CHAPITRE III

MODÉLISATION INCRÉMENTALE DES SOC EMBRAQUÉS À CPU-GPU

III.1 Introduction

Les SoC hétérogènes sont des systèmes très complexes. Ils contiennent un grand nombre de transistors, plusieurs sous-modules spécialisés, et combinent des aspects logiciels et matériels. De plus, ces circuits peuvent être décrits par plusieurs dynamiques et caractérisés par des variables discrètes (telles que la charge du processeur), ou des variables non-linéaires continues (telles que la température et la consommation de la puissance). La surveillance d'un tel système à travers un modèle de référence nécessite la modélisation de plusieurs de ces variables (charge, puissance, température, etc.) en même temps. Néanmoins, la construction d'un modèle global pour toutes ces variables peut s'avérer ardu, voire impossible. Par conséquent, afin de modéliser toutes ces dynamiques, nous avons créé une nouvelle structure de modélisation. Cette structure – tout comme le système – est variable, et s'adapte à ce dernier en fonction de ses modes de fonctionnement (économie d'énergie, performances, etc.) et les variables à surveiller.

Ce chapitre met l'accent sur l'organisation et la simplification de la modélisation des SoC hétérogènes embarqués, en présentant une structure de modélisation gérable, adaptable et surtout extensible. Il reprend le processus de construction du profiler N^3 que nous avons entrepris dans le Chapitre II et décrit la seconde partie (voir Figure II.1).

La modélisation incrémentale présentée dans ce chapitre a été proposée dans [9]. Par la suite, une première version du modèle de puissance a été publiée dans [6]. Ce modèle a été amélioré et a également fait l'objet d'un article [4]. Enfin, les modèles de température peuvent être trouvés dans [1] et [2].

Après un rappel des variables caractérisant l'état de fonctionnement des SoC hétérogènes, nous explorons les relations causales entre ces variables. Puis, nous présentons la structure modulaire du modèle incrémental construite à partir de ces relations. Nous détaillons ensuite la modélisation de chacun des sous-systèmes constituants le modèle global. Pour cela, nous commençons par décrire la dynamique de chaque sous-système avant de motiver notre choix d'outils de modélisation pour chaque dynamique. Pour finir, le modèle incrémental est validé expérimentalement, hors-ligne puis en-ligne, sur deux systèmes différents.

III.2 Structure de modélisation interconnectée et incrémentale

Dans la Section II.3, nous avons défini la surveillance de l'état de fonctionnement du SoC comme étant l'objectif de N^3. Nous avons également souligné un ensemble de variables décrivant les aspects logiciels et matériels (voir II.3.1). Pour un système avec processeur doté de n cœur, ces variables sont :

— Les charges de processeurs (CPU et GPU) : $\text{Charge}_{\text{CPU1}}$, ... , $\text{Charge}_{\text{CPU}n}$ et $\text{Charge}_{\text{GPU}}$

— Les fréquences par cœur pour le CPU et la fréquence du GPU : f_1, ... , f_n, et f_{GPU}

— Le taux d'occupation de la mémoire : MOR

— La tensions par cœur pour le CPU et La tension du GPU : V_1, ... , V_n, et V_{GPU}

— La température du SoC ; T_{SoC}

— La puissance consommée par le SoC et le système : P_{SoC}

En étudiant ces variables, nous trouvons qu'elles sont bien liées par des relations causales claires. Lors du fonctionnement normal des systèmes étudiés, la variation de fréquence dans les processeurs (f_1, ... , f_n) est une fonction de la charge du processeur ($\text{Charge}_{\text{CPU1}}$, ... , $\text{Charge}_{\text{CPU}n}$) [167]. Par conséquent, c'est la charge qui impose la fréquence de fonctionnement des cœurs du CPU, et il en est de même pour le GPU. Les changements des fréquences (f_i) entraînent des changements des tensions (V_i) dans les cœurs du processeur, car cette dernière varie en fonction de la fréquence choisie grâce au DVFS [56].

Ensuite, les ajustements de la tension et de la fréquence des processeurs entraînent des variations de la puissance consommée par le SoC (P_{SoC}), étant donné que la consommation d'énergie dans les processeurs est une fonction des tensions et des fréquences de tous les coeurs [146]. En outre, l'augmentation des fréquences provoque toujours une augmentation de la température T_{SoC} indiquant un lien clair entre la fréquence, la puissance consommée et l'échauffement [56].

Pour contrôler l'échauffement du SoC, les processeurs modernes incluent toujours un régulateur thermique qui, chaque fois que la température d'un cœur franchit un certain seuil, réduit la fréquence de fonctionnement en dessous d'un seuil prédéfini par le constructeur, ou arrête complètement le processeur ou le cœur affecté [56].

Par conséquent, nous avons adopté une approche graduelle et une structure modulaire pour la modélisation du SoC comme illustré dans la Figure III.1. Le schéma de modélisation est composé d'un ensemble de sous-systèmes, conçus pour estimer chacun une seule variable.

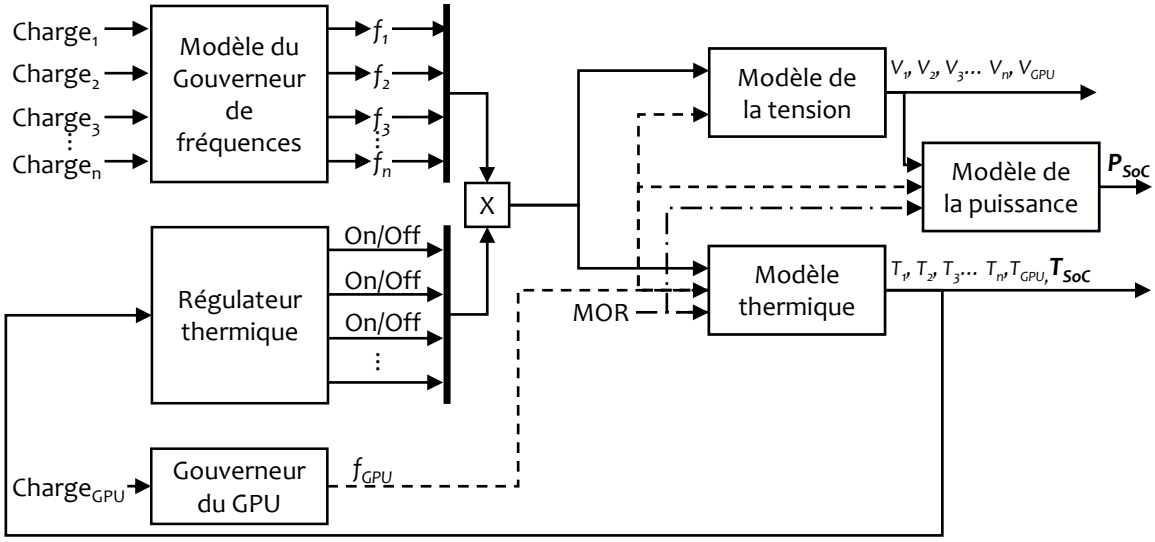


FIGURE III.1 – Schéma de blocs de la structure de modélisation incrémentale et interconnectée proposé pour les SoC hétérogènes avec un CPU contenant n cœur.

En exploitant les relations de causalité susmentionnées, et en connectant les sous-systèmes en conséquence, le résultat est un modèle capable d'estimer toutes les variables que nous avons définies avec des relations causales dont le sens physique est clair. Ainsi, la structure du modèle tient compte des différentes dynamiques présente dans le système, par conception. En outre, elle permet d'analyser progressivement et d'une façon incrémentale chacune des variables caractéristiques du SoC (Figure III.1). En premier lieu, les charges des processeurs ($\text{Charge}_{\text{CPU1}}$, ... , $\text{Charge}_{\text{CPU}n}$ et $\text{Charge}_{\text{GPU}}$) sont prises comme entrées. Ces entrées sont utilisées pour le calcul des fréquences f_1, ... , f_n, et f_{GPU}, qui à leur tour dictent les tensions des processeurs V_1, ... , V_n, et V_{GPU}, et permettent – en plus du MOR – de d'estimer la puissance P_{SoC} et la température T_{SoC}.

L'originalité de cette approche réside dans la flexibilité et l'aspect évolutif du modèle. Le fait que ce dernier soit constitué de sous-systèmes, facilite l'intégration ou le remplacement de ces sous-systèmes en cas de modifications ou de mises à jour. Elle offre ainsi à l'utilisateur, la possibilité d'utiliser le sous-système qu'il juge le plus approprié. Ces systèmes peuvent être aussi présentés dans une bibliothèque de modèles d'éléments (constituants) interchangeables (Figure III.2). Par exemple, au cours de ce travail, nous avons développé trois modèles de puissance :

— un réseau de neurones NARX (il sera présenté dans la suite du chapitre),

— un modèle de régression linéaire, similaire à celui développé par Kim et al. [92],

— un arbre de régression.

Tous ces modèles sont facilement interchangeables. En plus, la bibliothèque peut être complétée en permanence par des modèles tirés de la littérature, comme le modèle proposé par Kim et al. [13] dans la Figure III.2. Celle-ci montre l'aisance avec laquelle les sous-systèmes peuvent être connectés et interchangés.

En résumé, la structure interconnectée et incrémentale offre ainsi les avantages suivants :

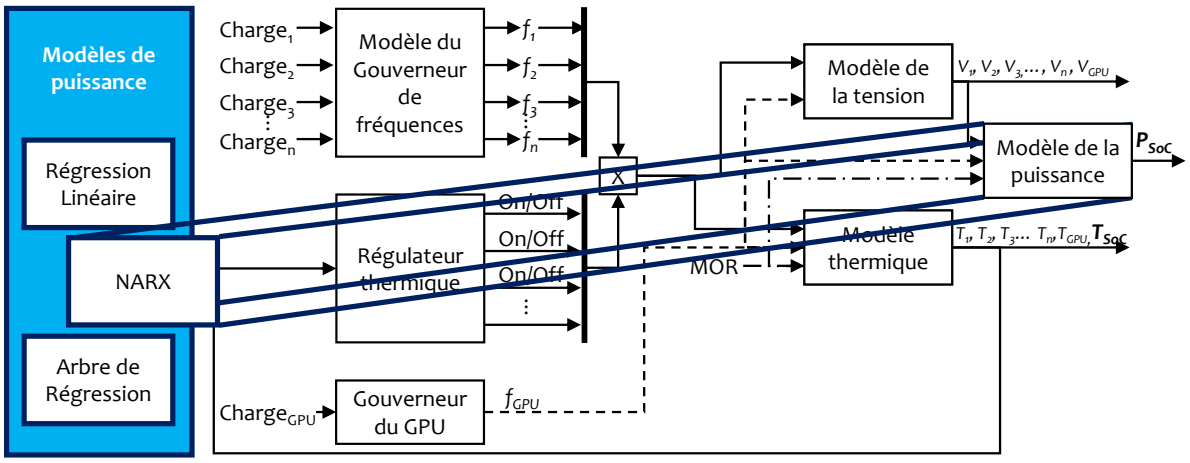


FIGURE III.2 – La structure incrémentale du modèle avec des sous-systèmes évolutifs et réutilisables, disposés dans une bibliothèque de modèles, à l'exemple des modèles de puissance NARX [5], la régression linéaire [13], et l'arbre de régression [9].

— L'intégration ou le remplacement aisé de sous-systèmes, dans le cas de changements de composants ou de mises à jour

— La capitalisation des connaissances, qui permet un gain de temps dans la phase de conception et de perfectionnement des systèmes embarqués à CPU et GPU

— La facilité de déploiement dans les systèmes électroniques embarqués

— Aucune instrumentation supplémentaire n'est nécessaire, le modèle n'utilise que des lectures et des capteurs disponibles sur le système

Dans ce qui suit, nous détaillons l'approche de modélisation de chacun des sous-systèmes. Tout d'abord, nous commençons par la fréquence et la tension, puis du régulateur thermique. Ensuite, nous passons à la modélisation de la puissance et de la température.

III.3 Modélisation par analyse

Parmi les variables caractéristiques du SoC, nous notons deux types de variables, les variables à évènement discrets et les variables continues échantillonnées. Dans notre cas, les variables à évènements discrets sont les variables dont la variation est régie par des algorithmes. Par conséquent, le modèle du sous-système correspondant sera représenté aussi par un algorithme. Ces sous-systèmes sont : le gouverneur de fréquence, le modèle de tension, et le régulateur thermique. Afin de modéliser ces sous-systèmes, nous procéderons à l'analyse de leurs algorithmes originaux.

III.3.1 Modélisation du gouverneur de fréquence

Sous Android, l'ajustement des fréquences se fait par des programmes appelés gouverneurs (*Frequency Governors*). Les processeurs ARM fonctionnent sous plusieurs fréquences définies dans une table de fréquences (*Frequency Table*). Le choix – ou l'ajustement – des fréquences, se fait par le gouverneur en fonction de la charge et d'autres critères pour lesquels il est conçu (performances, économies d'énergies, etc.). Le programme de ces gouverneurs calcule la valeur de la charge du CPU à des périodes prédéfinies, puis détermine la fréquence optimale correspondant à cette charge.

L'un des gouverneurs les plus célèbres est le gouverneur que les smartphones utilisent : le *Interactive Governor*. Ainsi, nous l'avons utilisé comme cas d'application dans notre modélisation. Toutefois, grâce à la structure modulaire du modèle, tout autre gouverneur peut le remplacer.

Sur le système, ce gouverneur est une procédure qui calcule la charge du processeur chaque 20 ms, et détermine la fréquence optimale – selon son algorithme – en fonction de cette charge [167]. Étant donné que la valeur de la charge est déjà acquise par le profiler (voir I.4.1), pour la modélisation du gouverneur, nous avons développé une fonction qui prends les charges du processeur comme entrées et donne les estimations des fréquences du processeur en sortie (Figure III.3).

Le fonctionnement simplifié de la fonction est décrit dans l'algorithme 2. Il est inspiré du code source disponible dans les dépôts de codes du fabricant de l'appareil mobile n° 1 [167]. Il commence

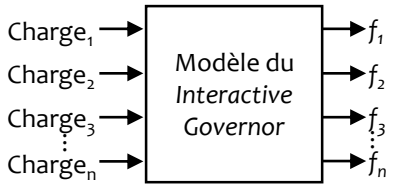


FIGURE III.3 – Les entrée et les sorties du modèle du gouverneur de fréquences.

par la définition des constantes de temps, et la constante de la valeur de la charge optimale à viser en changeant la fréquence (`target_load`). En termes simples, ce gouverneur augmente ou diminue la fréquence de chaque coeur en fonction de la charge, des constantes (`goHispeedLoad`, `targetLoad`, `hispeedFreq`, etc.) et des compteurs spécifiques de temps (`timerRate`, `downTimer`, etc. Ces compteurs sont déclarés dans l'algorithme 2). Par exemple, lorsque le CPU redeviens actif, le gouverneur démarre un compte à rebours avec la valeur `timerRate`. A la fin du compte à rebours, si la charge dépasse la valeur `goHispeedLoad`, le gouverneur sélectionne une nouvelle fréquence pour laquelle la charge sera égale ou proche de la valeur `targetLoad` prédéfinie. Le choix de la fréquence est fait par la fonction (*ChooseFreq*) détaillée dans l'algorithme 1. Cette fonction calcule la fréquence théorique pour laquelle la charge sera égale à `target_load`, puis choisi la fréquence la plus proche de cette valeur dans la table des fréquences. Le gouverneur prend également en compte les charges soudaines et importantes en augmentant directement la fréquence à `hispeedFreq`, si la fréquence en cours est inférieure à celle-ci.

Le GPU est aussi géré par des gouverneurs de fréquences qui ont le même principe de fonctionnement, c'est-à-dire augmenter ou diminuer la fréquence du GPU en fonction de la charge et le profil d'utilisation . Ses gouverneurs sont modélisés de la même façon que les gouverneurs du CPU.

Algorithm 1 Pseudo code de la Procedure de Choix de la fréquence : *ChooseFreq*

procedure CHOOSEFREQ($Load, currentFrequency$)
  Define `target_load`
  Define `FREQUENCY_TABLE`
  $frequency = currentFrequency \times (Load/$`target_load`$)$
  $TableSize =$`FREQUENCY_TABLE.size()`
  $Minimum = frequency$
  $BestIndex = 0$
  while $(index < TableSize)$ **do**
    if $frequency - TABLE_{FREQUENCY}(index) < Minimum$ **then**
      $BestIndex = index$
      $Break$
    else
      $index \leftarrow index + 1$
  return `FREQUENCY_TABLE`$(BestIndex)$

Algorithm 2 Pseudo code simplifié du modèle du Gouverneur Interactive de l'appareil mobile n° 1

procedure INTERACTIVE($Load, Time, currentFrequency$) ▷ la fonction prends la la charge et le temps actuelle en milli-secondes, et la fréquence actuelle comme entrées
▷ Décalration des constantes
Define `above_hispeed_delay` ▷ Temps d'attente avant d'augmenter la fréquence
Define `hispeed_freq` ▷ Fréquence auquels il faut augmenter directement si ▷ `Load` > `go_hispeed_load`
Define `go_hispeed_load` ▷ La charge à partir de laquelle il faut augmenter la fréquence
Define `min_sample_time` ▷ Temps d'attente avant de réduire la fréquence
Define `target_load` ▷ La valeur de la charge optimale
Define `timer_slack` ▷ Temps d'attente supplémentaire avant de d'éteindre le cœur
Define `timer_rate` ▷ Le temps entre deux calculs de la charge
if $Initialisation = 0$ **then** ▷ Initialisation des comptes à rebours et les fréquence
$currentHiSpeedTimer \leftarrow 0$ ▷ Compteur pour vérifier `above_hispeed_delay`
$currentTimers \leftarrow 0$ ▷ Compteur pour vérifier `timer_rate`
$currentDownTimers \leftarrow 0$ ▷ Compteur pour vérifier `min_sample_time` ▷ et sampling_down_factor
$currentTimersSlack \leftarrow 0$ ▷ Compteur pour vérifier `timer_slack`
$oldTime \leftarrow Time$
$Initialisation \leftarrow 1$ ▷ Pour ne plus revenir dans cette partie
else ▷ Mis à ajour des compteur
$currentHiSpeedTimer \leftarrow currentHiSpeedTimer + (Time - oldTime)$
$currentTimers \leftarrow currentTimers + (Time - oldTime)$
$currentDownTimers \leftarrow currentDownTimers + (Time - oldTime)$
$currentTimersSlack \leftarrow currentTimersSlack + (Time - oldTime)$
if ($currentTimers \geq$ `timer_rate`) **then** ▷ Varifie si la période d'échantillonnage est arrivée
if ($Load \geq$ `go_hispeed_load`) **then** ▷ La condition pour augmenter la fréquence
if ($currentFrequency <$ `hispeed_freq` $\wedge\ ChooseFreq(Load, currentFrequency) <$ `hispeed_freq`) **then**
$newFrequency \leftarrow$ `hispeed_freq`
else if ($ChooseFreq(Load, currentFrequency) >$ `hispeed_freq`) $\wedge$ ($currentHiSpeedTimer \geq$ `above_hispeed_delay`) **then**
$newFrequency \leftarrow ChooseFreq(Load, currentFrequency)$
else if ($ChooseFreq(Load, currentFrequency) < currentFrequency$) $\wedge$ ($currentDownTimers \geq$ `min_sample_time`) **then**
$newFrequency \leftarrow ChooseFreq(Load, currentFrequency)$
else if ($Load = 0$) $\wedge$ ($currentTimersSlack \geq$ `timer_slack`) **then**
$newFrequency \leftarrow 0$
$currentHiSpeedTimer \leftarrow 0$
$currentDownTimers \leftarrow 0$
else
$newFrequency \leftarrow currentFrequency$
return $newFrequency$

III.3.2 Modèle de la tension

Les lectures récupérées des tensions des cœurs du CPU montrent que, tout comme la fréquence, la tension est discrète et varie dans un ensemble de valeurs bien définies. Afin de mieux étudier la relation entre ces deux variables, les lectures de la tension sont tracées par rapport aux fréquences. La Figure III.4 montre ce tracé pour l'appareil mobile n° 1. Elle montre aussi une tendance quasi-linéaire (en rouge) où les valeurs sont concentrées, indiquant que chaque fréquence est associée à une valeur de tension fixe.

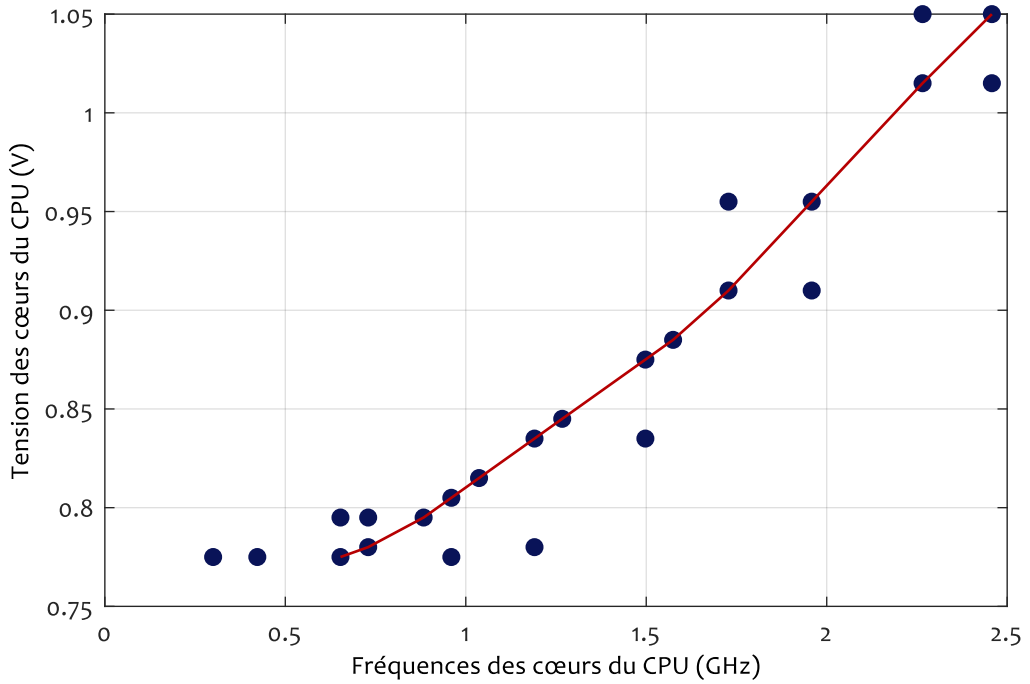


FIGURE III.4 – Les tensions des cœurs du CPU en fonction des fréquences de l'appareil mobile n° 1 : Les valeurs des tensions des cœurs du CPU tracées par rapport aux valeurs des fréquences du CPU montrent une tendance quasi-linéaire (Rouge).

En analysant les mesures de la table des fréquences de l'appareil mobile n° 1, nous trouvons 15 valeurs différentes de fréquences (plus la fréquence nulle qui correspond à un cœur éteint) contre 14 valeurs de la tension, indiquant que certaines fréquences partagent la même valeur de tension. Ensuite, nous traçons un histogramme des valeurs de tension pour chaque fréquence. Un exemple est présenté dans la Figure III.5 pour la fréquence $f = 1.49\,\text{GHz}$, où l'histogramme indique clairement que la tension associée à cette fréquence est $V_{1.49} = 0.875\,\text{V}$. Toutes les autres valeurs de la tension des cœurs du CPU ainsi que celles du GPU sont obtenues de la même manière. Les fréquences du CPU de l'appareil mobile n° 1 ainsi que les valeurs de tensions associés à ces fréquences sont données dans la Table III.1. L'algorithme de la fonction du modèle de tension est décrit dans l'algorithme 3. Cette fonction pend la fréquence actuelle du coeur du processeur comme entrée et donne la tension en sortie.

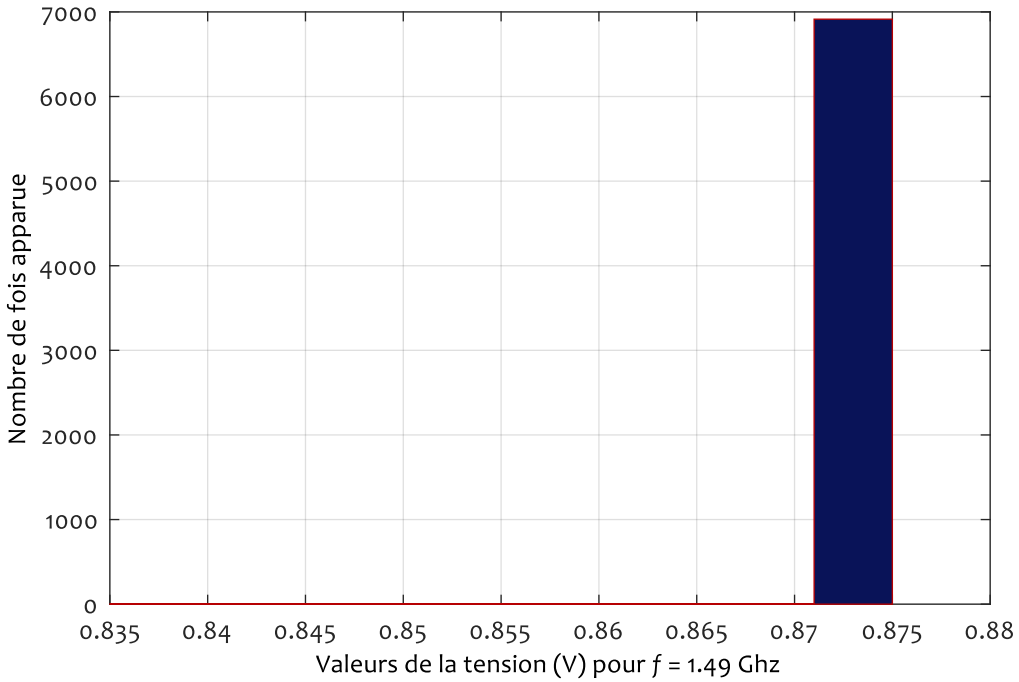


FIGURE III.5 – L'histogramme des valeurs de la tension des cœurs du CPU pour $f = 1.49\,\text{GHz}$.

TABLE III.1 – La table de fréquences des cœurs du CPU et les valeurs de tension correspondantes pour l'appareil mobile n° 1.

Fréquence du coeur (kHz)	Tension du coeur (V)
Éteint	0.500
300000	0.775
422400	0.775
652800	0.775
729600	0.780
883200	0.795
960000	0.805
1036800	0.815
1190400	0.835
1267200	0.845
1497600	0.875
1574400	0.885
1728000	0.910
1958400	0.955
2265600	1.015
2457600	1.050

Algorithm 3 Pseudo code de la procédure de la modélisation de la tension.

procedure FREQUENCYTOVOLTAGE($frequency$) ▷ la fonction prends la fréquence comme entrée
  Define $TABLE_{FREQUENCY}$ ▷ Initialisation de la table des fréquence (constante)
  Define $TABLE_{VOLTAGE}$ ▷ Initialisation de la table des tensions (constante)
  $index = TABLE_{FREQUENCY}(= frequency)$;
  return $TABLE_{VOLTAGE}(index)$

III.3.3 Modélisation du régulateur thermique

Les processeurs fonctionnent avec des hautes fréquences qui provoquent des augmentations importantes de leur température. Afin de protéger leurs équipements des dégâts causés par les températures excessives [169, 170], les constructeurs intègrent des dispositifs qui servent à baisser la température du processeur, soit en limitant la fréquence de fonctionnement maximale ou en éteignant le processeur ou le coeur concerné. Ces dispositifs sont appelés régulateurs thermiques (*Thermal regulators*).

Dans les appareils que nous étudions, les fabricants du SoC ont implémenté trois types de régulateurs thermiques, avec lesquels le programmeur peut réguler les températures du SoC. Les régulateurs mis en œuvre sont : proportionnel, intégral, dérivé (*Proportional Integral Derivative*, PID), le Single-Step (SS) et le *Monitor*, que nous avons pris comme support d'illustration dans ce travail.

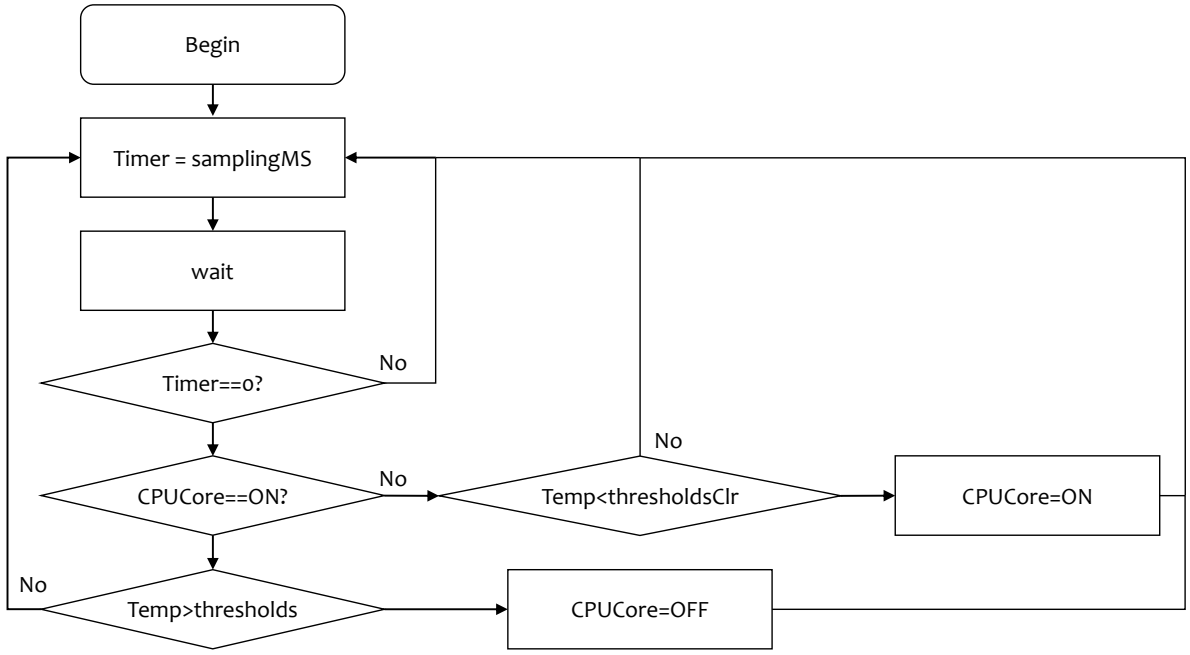


FIGURE III.6 – Un organigramme représentatif de l'algorithme du *Monitor*.

La Figure III.6 décrit l'organigramme de l'algorithme la fonction de la modélisation du *Monitor*. Le *Monitor* échantillonne les températures à une fréquence prédéfinie (`samplingMs`). Lorsque la valeur de la température d'un cœur dépasse un seuil prédéfini (`thresholds`), le cœur est éteint. Une fois la

température de ce cœur redescendue en dessous d'un second seuil (`thresholdsClr`), alors, ce cœur est à nouveau activé.

III.4 Modélisation orientée données

Dans la section précédente, nous avons détaillé la modélisation des variables caractéristiques régies par des algorithmes. Dans cette section, nous nous focaliserons sur les variables caractéristiques restantes, c'est à dire la puissance consommée et la température du SoC.

La modélisation boîte blanche de la température et de la puissance dans un SoC nécessite une connaissance physique fine de tous les composants internes de la puce [18]. Cette connaissance est rarement disponible, et même lorsqu'elle l'est, le processus de modélisation nécessiterait également la construction de machines à états finis [82] ou la simulation d'équations différentielles [38]. Ces modèles sont difficiles à mettre en oeuvre en pratique, surtout pour des puces contenant une multitude de sous-modules et un nombre immense de transistors.

En revanche, les techniques de régression entraînent le modèle à adapter ses sorties aux observations à l'aide des statistiques [82]. Bien que la bonne connaissance des facteurs influant sur la sortie à estimer est très appréciée, sauf qu'elle n'est pas nécessaire, pas plus que la preuve théorique formelle de la relation entre les entrées et les sorties [18]. Néanmoins, ces méthodes nécessitent de grandes quantités de données, de temps et de ressources informatiques pour l'entraînement et la validation [82].

Dans ce travail, nous avons donc opté pour les techniques d'apprentissage automatique guidées par les données.

III.4.1 Modélisation de la puissance

La disponibilité des traces fournies par le système d'exploitation, couplée avec la complexité du système, nous ont conduit à utiliser des techniques de modélisation boîte noire. Dans ce type de modélisation, il est possible d'utiliser des techniques de régression ou de classification. Les modèles de puissance basés sur la régression linéaire ont été bien étudiés dans la littérature, en particulier pour les smartphones [18, 87, 92, 13, 94, 103]. Leur précision est compromise par l'utilisation de modèles linéaires, ce qui entraîne des erreurs d'estimation car la dynamique de la puissance n'est pas linéaire.

Les modèles non linéaires tels que les réseaux de neurones artificiels [9, 105, 171] ou la régression à l'aide de *Support Vector Regression* (SVR) [172] permettent de surmonter ces limites [82, 104]. Cependant, les réseaux de neurones classiques ne tiennent pas compte du temps d'échantillonnage et des différents horodatages associés aux données. Ils ne sont pas non plus conçus pour prendre en compte les boucles de rétroaction, ou l'historique des valeurs de la sortie. Une alternative a été récemment explorée par Romansky et al. [65], qui ont utilisé les séries temporelles (*timeseries*) et l'apprentissage approfondi pour prédire la consommation d'énergie dans leur modèle *Deep Green*.

III.4.1.1 Choix des entrées du modèle de la puissance

Dans le Paragraphe II.3.1, nous avons listé les variables caractéristiques du SoC. Outre ces variables, nous avons également listé les variables nécessaires pour l'estimation de la consommation énergétique de l'appareil dans son ensemble et du SoC en particulier. La puissance consommée par chacun des composants de l'appareil peut être corrélée à l'une de ses variables.

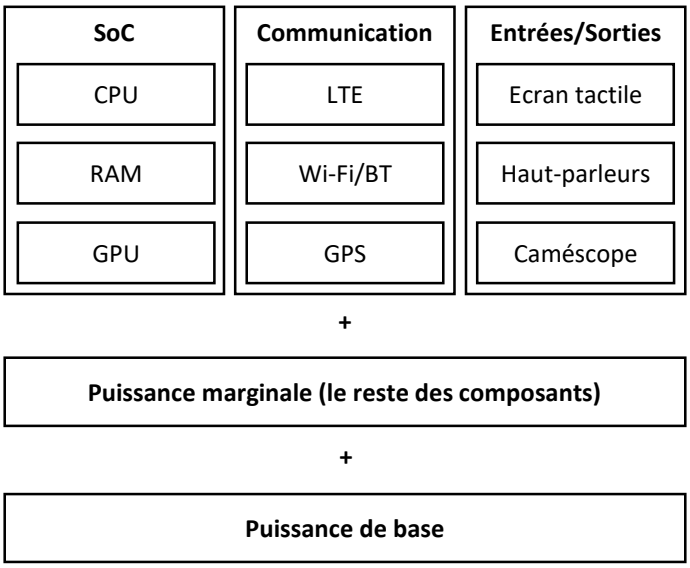


FIGURE III.7 – La répartition de la consommation électrique dans un smartphone typique.

En se référant à la Figure III.7, il y a trois types de composants à prendre en compte dans l'estimation de la puissance.

Le SoC : Dans ce dernier, la puissance consommée par le CPU est une fonction de sa fréquence et de sa tension [146, 173, 174]. Toutefois, comme nous avons vu dans la modélisation de la tension (voir III.3.2), la tension dans les CPU à DVFS est également une fonction de la fréquence [9]. Ainsi, la variable – et donc l'entrée – utilisée pour estimer la puissance consommée par le CPU est la fréquence (f) de chaque cœur.

Dans l'appareil mobile n° 2, le CPU a été configuré pour utiliser une fréquence pour chacun des bloc quadricœur dans sa configuration *big.Little* [159]. Par conséquent, il n'y aura que deux lectures de fréquence [175, 159]. La même entrée – la fréquence (f_{GPU}) est utilisée pour le GPU, car – comme pour le CPU – elle est directement corrélée à sa consommation d'énergie [13, 5, 155].

La RAM consomme environ 10% de la puissance totale du système [50]. Pour tenir compte de la puissance utilisée par la RAM, lors des premiers essais, nous avons opté pour l'utilisation de la valeur de la RAM occupée. Cependant, cette valeur est exprimée en absolue et ne tient pas en compte les valeurs maximales et minimales possibles utilisées par le système. Nous optons désormais pour l'utilisation du MOR (voir II.3.1).

Les modules de communication : Les modules de communication du smartphone sont : la radio cellulaire GSM/HSPA/LTE du téléphone, le module sans fil (Wi-Fi et Bluetooth) et le GPS. La consommation électrique de ces périphériques est principalement caractérisée par leur état de connexion et de transfert de données [132, 157]. Pour le module radio cellulaire, les entrées sont l'état de connexion (désactivé, connecté 3G, LTE, ...), l'état d'appel, la force du signal et le mode de transfert des données ainsi que la quantité de données transférées [132]. De même, pour le Wi-Fi et le Bluetooth, leur consommation d'énergie étant fortement influencée par leur état d'activité et par la bande passante [176], les entrées sont l'état (On/Off) et le transfert de données. Enfin, pour le GPS, nous avons seulement utilisé son statut (On/Off).

Les périphériques d'entrée et de sortie : Pour prendre en compte la consommation d'énergie de l'écran tactile, nous avons utilisé l'état de l'écran (On/Off) et sa luminosité comme entrées [92, 165], car pour les écrans AMOLED modernes, la puissance consommée est une fonction quadratique de la luminosité [165, 132]. La puissance consommée par l'écran dépend également de la résolution utilisée et de la mise à l'échelle de cette dernière [51]. Cependant, dans notre cas, la résolution est fixée à la valeur maximale possible sans aucune mise à l'échelle, cette dernière sera une constante qui n'affectera pas notre modèle.

Pour le cas des haut-parleurs, l'état (On/Off) et le volume sont utilisés comme entrées pour le modèle [129]. Et pour le microphone, seul l'état (On/Off) a été utilisé. Enfin, pour les appareils photo et le flash, nous avons utilisé l'état de l'appareil photo et du flash ainsi que son état d'enregistrement (enregistrement ou non par l'appareil photo).

En comptant toutes les entrées nécessaires pour la modélisation de tous les composants, le vecteur d'entrée u devient pour un smartphone avec un processeur à n cœurs, m haut-parleurs, et i appareils photo :

$$\begin{aligned} u(k) = [&f_1(k), \cdots, f_n(k), f_{GPU}(k), MOR(k), EtatDeConnexion(k), ForceDuSignal(k), \\ &ModeDeTransfert(k), WiFi_{On/OFF}(k), WiFi_{Transfert}(k), Bluetooth_{On/Off}(k), \\ &Bluetooth_{Transfert}(k), GPS_{On/Off}(k), Ecran_{On/OFF}(k), Ecran_{Luminosit}(k), \\ &HautParleur_{1,On/Off}(k), \cdots, HautParleur_{m,Volume}(k), AppareilP_{1,On/OFF}(k), \cdots, \\ &AppareilP_{i,Enregistrement}(k), Flash_{On/off}(k), Microphone_{i,Enregistrement}(k)] \end{aligned} \quad \text{(III.1)}$$

III.4.1.2 Description du modèle de puissance

Pour mettre en œuvre une solution compatible avec les systèmes non-linéaires et les séries temporelles, nous proposons les réseaux de neurones autorégressifs non-linéaires à entrées exogènes (*Nonlinear AutoRegressive with eXogenous inputs* (NARX) *Neural Networks*). Ces réseaux de neurones récursifs sont conçus pour la prédiction de séries temporelles [177]. Leur représentation mathématique

est donnée par l'équation III.2 :

$$y(k) = \varphi\left[y(k-1), \cdots, y(k-d_y), u(k-n), u(k-n-1), \cdots, u(k-n-d_u)\right] \tag{III.2}$$

où $y(k)$ et $u(k)$ représentent, respectivement la sortie à prédire et l'entrée du modèle au pas de temps k, n le retard à la réponse (le nombre d'échantillons nécessaires avant la première prédiction), et φ la fonction caractéristiques du modèle. Tandis que d_y et d_u sont les ordres de retard pour l'entrée et la sortie (également appelés la mémoire d'entrée et de sortie). La configuration donnant les meilleurs résultat dans ce travail est obtenue avec d_y, d_u et n tous mis à la valeur de 1, car la puissance réagit assez rapidement aux variations des entrées. Donnant ainsi :

$$y(k) = \varphi\left[y(k-1), y(k-2), u(k-1), u(k-2)\right] \tag{III.3}$$

Le modèle est composé de deux couches : une couche cachée et une couche de sortie. Les fonctions d'activation des neurones de la couche cachée sont des sigmoïdes (ς). La couche de sortie contient un seul neurone, représentant la puissance totale consommée $P_{Système}$, avec une fonction de transfert linéaire. La Figure III.8 montre la structure générale du modèle NARX avec le retour de sortie et la ligne de temporisation (*Time Line Delay*, TDL). Comparé à un réseaux de neurones artificiel classique, le NARX présente deux différences majeurs en terme d'architecture, qui sont la récursivité de la sortie et la ligne de temporisation (Figure III.8).

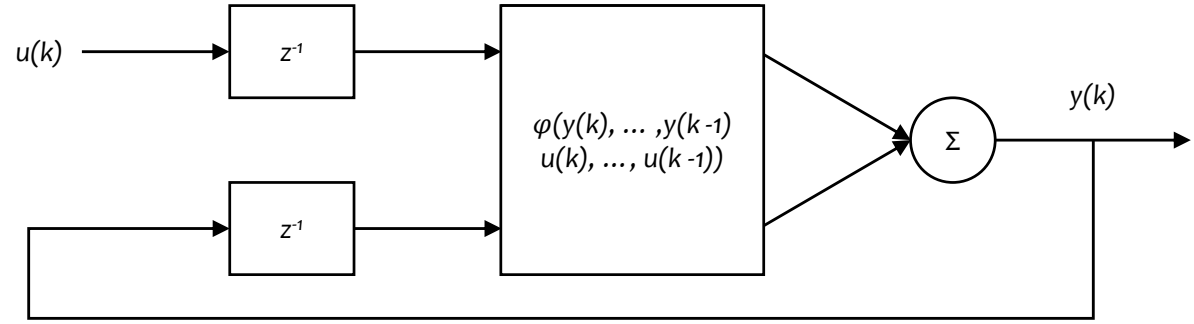


FIGURE III.8 – La structure du réseau NARX utilisée pour la modélisation de la puissance. z^{-1} : retard des entrées et sorties (d_y et d_u égale à 1).

Le premier avantage de l'utilisation des réseaux de neurones NARX réside dans leur capacité à estimer la dynamique non linéaire de la puissance tout en conservant une forme relativement simple et légère en termes de calcul, grâce au nombre relativement réduit des neurones. Le deuxième avantage est leur capacité à satisfaire le niveau de granularité choisi, qui est le niveau composants (voir II.3).

Pour illustrer ce niveau de granularité, la Figure III.7 décrit la répartition de la puissance totale consommée par l'appareil comme la somme des puissances consommées par chaque composant individuel (CPU, Wi-Fi, etc.), plus une quantité minimale de base toujours consommée par l'appareil, et une quantité négligeable consommée par le reste des composants dont la consommation est très

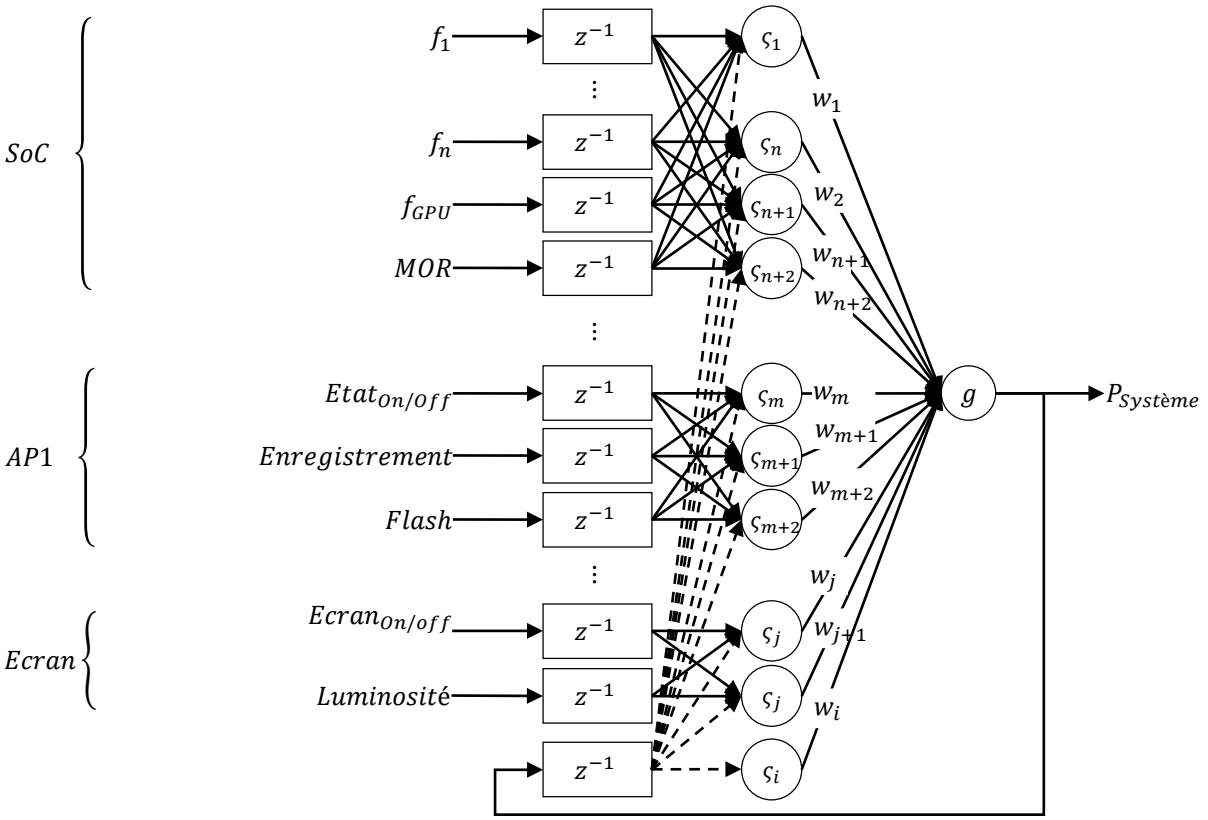


FIGURE III.9 – La configuration des interconnexions des neurones dans la couche d'entrée dans le modèle de puissance pour assurer le niveau granularité composants. AP1 : Appareil photo 1.

faible (voir II.3). En considérant par hypothèse, que la puissance consommée par les composants négligés et la puissance de base représentent une seule puissance statique, la puissance consommée par l'appareil devient :

$$P_{Système} = P_{Statique} + \sum_{i=1}^{n} P_i \tag{III.4}$$

où i indique le numéro du composant (CPU, GPU, RAM, etc.), et n correspond au nombre total de ces composants. Le niveau de granularité peut être assuré en configurant les neurones de la couche cachée de sorte à réaliser un découplage par composant. Ceci peut être obtenu en désactivant l'apprentissage des pondérations w reliant les neurones des composants différents. La Figure III.9 montre la configuration du modèle de puissance pour un système embarqué à n cœur, en prenant l'exemple de trois composants : le SoC, l'appareil photo 1 et l'écran. Étant donné que la couche de sortie est un neurone linéaire, la puissance consommée par le système devient :

$$\begin{aligned} P_{Système} = {} & w_1\varsigma_1 + \cdots + w_n\varsigma_n + w_{n+1}\varsigma_{n+1} + w_{n+2}\varsigma_{n+2} + \cdots + \\ & w_m\varsigma_m + w_{m+1}\varsigma_{m+1} + w_{m+2}\varsigma_{m+2} + \cdots + w_j\varsigma_j + w_{j+1}\varsigma_{j+1} + w_i\varsigma_i \end{aligned} \tag{III.5}$$

en désactivant tous les composants et en gardant uniquement le SoC, car c'est le composant nécessaire au fonctionnement du système, et en entraînant le modèle avec des données issues de cette configuration, la consommation de puissance du système (équation III.4) devient :

$$P_{Système} = P_{Statique} + P_{SoC} \tag{III.6}$$

et l'expression du modèle devient :

$$P_{Système} = \underbrace{w_1\varsigma_1 + \cdots + w_n\varsigma_n + w_{n+1}\varsigma_{n+1} + w_{n+2}\varsigma_{n+2} + w_i\varsigma_i}_{P_{Statique} + P_{SoC}} \tag{III.7}$$

pour quantifier la contribution individuelle de chaque composant, l'ensemble d'entraînement est construit comme suit. Dans un premier temps, seul le SoC est activé, pendant un intervalle de temps. Ensuite, les composants sont activé l'un après l'autre. Chaque composant est maintenu actif pendant un intervalle de temps, puis désactivé pendant une période de temps, avant d'activer le prochain composant. Pour finir, les composants sont remis à leur état de fonctionnement automatique.

Ainsi, le modèle *NARX* donne des estimations de la puissance consommée par le système dans son ensemble, mais aussi les puissances consommées par les composants individuellement.

$$\begin{aligned} P_{Système} = & \underbrace{w_1\varsigma_1 + \cdots + w_n\varsigma_n + w_{n+1}\varsigma_{n+1} + w_{n+2}\varsigma_{n+2} + w_i\varsigma_i}_{P_{Statique} + P_{SoC}} \\ & \underbrace{w_m\varsigma_m + w_{m+1}\varsigma_{m+1} + w_{m+2}\varsigma_{m+2}}_{P_{AppareilPhoto1}} + \cdots + \underbrace{w_j\varsigma_j + w_{j+1}\varsigma_{j+1}}_{P_{Écrans}} \end{aligned} \tag{III.8}$$

III.4.1.3 Synthèse du modèle

Le nombre de neurones dans le modèle est égale au nombre des variable d'entrée pour la couche cachée, plus un neurone pour la sortie en rétroaction, et un neurone dans la couche de sortie. Le modèle de la puissance devient ce smartphone :

$$P_{Système}(k) = \varphi\left[P_{SoC}(k-1), P_{SoC}(k-2), f_1(k-1), \cdots, Microphone_{i,Enregistrement}(k-2)\right] \tag{III.9}$$

Dans la Table III.2, tous les points décrits et détaillés précédemment sont résumés.

TABLE III.2 – Une description résumée du modèle de puissance NARX.

Système	**Appareil mobile n° 1**	**Appareil mobile n° 2**
Approche de modélisation	Boite noire	
Type du modèle	Réseau de neurones NARX	
Nombre de couches	2	
Couche cachée	• 28 neurones	• 26 neurones
	• Fonction d'activation : Sigmoïde (ς)	
Couche de sortie	• 1 neurone • Fonction d'activation : Linéaire (g)	
Entrées	**SoC**	
	• $f_1, \cdots, f_4$	• f_1 : CPU_{0-3} • f_2 : CPU_{4-7}
	• f_{GPU}	• f_{GPU}
	• MOR	• MOR
	Radio cellulaire • État de connexion : Off/2G/3G/LTE • État d'appel : On/Off • Force du signal (%) • Mode de transfert (LTE : idle/DRX/... , 3G : DCH/FACH...) • Taux de transfert (Octets émis et octets reçus)	
	Wi-Fi • État : On/Off • Transfert de données : 0/1	
	Bluetooth • État : On/Off • Transfert de données : 0/1	
	GPS • État : On/Off	
	Écran • État : On/Off • Luminosité : (%)	
	Haut-parleurs (×2) • État : On/Off • Volume : (%)	
	Microphone • Enregistrement : 0/1	
	Appareil photo (×2) • État : On/Off • Enregistrement : 0/1 • (×1) flash : On/Off	
Temps d'échantillonnage	20 ms	

III.4.2 Modélisation de la température du SoC

La dynamique de la température est différente de celles des autres variables étudiées, car ses variations ne dépendent pas uniquement des entrées, mais aussi de sa dynamique (ses propres valeurs précédentes). Notre objectif étant de surveiller l'état de fonctionnement du SoC, nous ne nous intéresserons pas aux mécanismes de la génération et du transfert de chaleur. Nous avons donc concentré notre attention sur la recherche de variables affectant la température.

III.4.2.1 Choix des entrées du modèle de température

Dans les processeurs, la température est une fonction directe de la fréquence [38]. De plus, Elle est aussi une fonction des valeurs de la consommation d'énergie [34, 37, 178, 179]. Par conséquent, les entrées du modèle de température ne sont que les fréquences de la CPU et du GPU, ainsi que le MOR et la puissance électrique (Équation III.10).

$$u(k) = [f_1(k), ..., f_n(k), f_{GPU}(k), MOR(k), P_{SoC}(k)] \tag{III.10}$$

n représente le nombre total des fréquences du CPU.

III.4.2.2 Description du modèle du comportement thermique

Pour bien représenter la dynamique de la température, nous avons choisi un modèle autorégressif à moyenne-mobile et à entrées exogènes (*Autoregressive –Moving-Average with Exogenous Inputs*, ARMAX). Les modèles ARMAX sont des modèles utilisés pour estimer ou prédire la valeur d'une variable en fonction de ses valeurs précédentes et celles du vecteur d'entrée [180]. Nous avons choisi un modèle ARMAX car il présente deux avantages essentiels à notre modélisation. D'abord, Ils est suffisamment rapide pour générer une estimation en ligne à des fréquences allant jusqu'à 50 Hz (Voir III.7.3), satisfaisant ainsi nos contraintes de performance (voir II.4.2.1). De plus, ces modèles étant dynamiques, ils peuvent également être utilisés pour la simulation du comportement du système, même hors ligne, et prédire la température de fonctionnement du SoC.

Un processus ARMAX discret à plusieurs entrées et une sortie, peut être décrit par l'équation aux récurrences suivante [180, 181] :

$$\begin{aligned} y(k) = a_1 y(k-1) + \cdots + a_{n_a} y(k-n_a) \\ + b_{1,1} u_1(k-d_u) + \cdots + b_{m,n_b} u_m(k-d_u-n_b) \\ + e(k) + c_1 e(k-1) + \cdots + c_{n_c} e(t-n_c) \end{aligned} \tag{III.11}$$

où $y(k)$ représente la sortie à prédire, $u(k)$ le vecteur d'entrée de taille m (Équation III.10), et $e(t)$ est la variable de la moyenne mobile. Les paramètres sont :

— Les paramètres de l'autorégression : $[a_1, \cdots, a_{n_a}]$,

— Les paramètres des entrées : $[b_{1,1}, \cdots, b_{1,n_b}], \cdots, [b_{m,1}, \cdots, b_{m,n_b}]$,

— Les paramètres de la moyenne mobile : $[c_1, \cdots, c_{n_c}]$.

Enfin, n_a, n_b et n_c sont les ordres du modèle et d_u est le retard de la réponse à l'entrée. Empiriquement, les meilleurs résultats que nous avons obtenus sont pour $d_u = 0$ (voir III.7.3).

Dans notre cas, la sortie $y(k)$ à estimer est la température T_{SoC}, et les entrées sont désignées dans le vecteur $u(k)$ (Équation III.10).

L'équation III.11 peut être réécrite sous la forme :

$$A(q^{-1})y(k) = B(q^{-1})u(k) + C(q^{-1})e(k) \tag{III.12}$$

où q^{-1} est l'opérateur de retard, $A(q^{-1})$ est le vecteur contenant les paramètres de l'auto-régression, $B(q^{-1})$ est la matrice des paramètres des entrées, et $C(q^{-1})$ contient les paramètres de la moyenne mobile [181].

Le processus d'identification consiste à trouver les meilleurs paramètres $A(q^{-1})$, $B(q^{-1})$ et $C(q^{-1})$ de sorte à ce que la sortie estimée du modèle $\hat{y}(k)$ correspond aux valeurs des mesures $y(k)$. En d'autres termes, minimiser l'erreur de d'estimation $\varepsilon(k)$:

$$\varepsilon(k) = y(k) - \hat{y}(k) \tag{III.13}$$

L'approche classique consiste à utiliser la méthode des moindre carrés pour minimiser l'erreur de prédiction [180]. D'après l'équation III.12, l'équation du modèle ARMAX peut-être réécrite sous la forme :

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})}u(k-\tau) + \frac{C(q^{-1})}{A(q^{-1})}e(k) \tag{III.14}$$

Puis sous une forme compacte :

$$y(k) = \theta\varphi^{\intercal}(k-1) + \varepsilon(k) \tag{III.15}$$

avec : $\theta = [-a_1, \cdots, c_{n_c}]$ le vecteur des paramètres, et $\varphi = [y(k-1), \cdots, y(k-n_a), u_1(k), \cdots, u_m(k-n_b), e(k-1), \cdots, e(k-n_c)]^{\intercal}$. Le vecteur des paramètres θ est le vecteur à identifier. L'estimation de $\hat{y}(k)$ est égale à :

$$\hat{y}(k) = \theta\varphi^{\intercal}(k-1) \tag{III.16}$$

Le vecteur des paramètres θ optimal pour générer les meilleure estimations du modèle ARMAX_{LS} peut être obtenu en utilisant et la méthode des moindres carrés étendus (*Least Squares*, LS) [182]. Il est égale à pour n échantillons :

$$\theta^* = \left[\sum_{k=1}^{n} \varphi^{\intercal}(k-1)\varphi(k-1)\right] \sum_{k=1}^{n} \varphi^{\intercal}(k-1)y(k) \tag{III.17}$$

où n est la longueur du vecteur de données de la sortie mesurée. Ainsi, le modèle de la température ARMAX_{LS} devient :

$$\begin{aligned} T_{SoC}(k) = a_1 T_{SoC}(k-1) + \cdots + a_{n_a} T_{SoC}(k-n_a) \\ + b_{1,1} f_1(k) + \cdots + b_{m,n_b} P_{SoC}(k-n_b) \\ + e(k) + c_1 e(k-1) + \cdots + c_{n_c} e(t-n_c) \end{aligned} \quad \text{(III.18)}$$

Cette approche d'identification nécessite la collection de données au préalable pour l'estimation des paramètres du modèle, nous forçant ainsi à réaliser une identification du modèle hors ligne. Donc, nous avons aussi implémenté une seconde approche avec un algorithme récursif : Moindre Carrée Récursive (*Recursive Least Squares*, RLS) [183], pour l'estimation du vecteur de paramètres θ à chaque échantillon. Cette méthode nous permettra de construire et d'identifier les paramètre du modèle en ligne. Cette algorithme est basée sur la minimisation de l'erreur d'estimation (Équation III.13) à chaque pas d'échantillonnage. L'équation III.14 du modèle devient :

$$y(k) = \frac{B(q^{-1})}{A(q^{-1})} u(k) + \frac{C(q^{-1})}{A(q^{-1})} \varepsilon(k) \quad \text{(III.19)}$$

Le listing présenté dans l'algorithme 4 est un pseudo-code décrivant comment les paramètres du modèle $\text{ARMAX}_{\text{RLS}}$ sont calculés à chaque itération pour tout le vecteur d'identification $y(k)$ [183]. Le modèle de température $\text{ARMAX}_{\text{RLS}}$ est représenté par l'équation :

$$\begin{aligned} T_{SoC}(k) = a_1 T_{SoC}(k-1) + \cdots + a_{n_a} T_{SoC}(k-n_a) \\ + b_{1,1} f_1(k) + \cdots + b_{m,n_b} P_{SoC}(k-n_b) \\ + \varepsilon(k) + c_1 \varepsilon(k-1) + \cdots + c_{n_c} \varepsilon(t-n_c) \end{aligned} \quad \text{(III.20)}$$

Enfin, la Table III.3 résume la configuration du modèle ARMAX de la température. Les ordres du modèle n_a, n_b et n_c sont choisi empiriquement à l'aide des erreurs moyenne MAPE et MSE obtenues pour chaque combinaisons allant de [2,2,2]∼[9,9,9].

TABLE III.3 – Une description résumée du modèle ARMAX de la température.

Système	**Appareil mobile n° 1**	**Appareil mobile n° 2**
Approche de modélisation	Boite noire	
Type du modèle	ARMAX	
Ordres du model	[2,2,2]∼[9,9,9]	
Retard des entrées (d_u)	0	
Entrées	**SoC** • $f_1, \cdots, f_8$ • f_{GPU} • MOR • P_{Soc}	• f_1 : CPU_{0-3} • f_2 : CPU_{4-7}
Temps d'échantillonnage	20 ms	

Algorithm 4 L'algorithme des Moindre Carrée Récursive (*Recursive Least Squares*).

Begin
Define orders : n_a, n_b, n_c
▷ Initialization of $\varphi_{k-1}^{\intercal}$
Initialize $\varphi_{y(k-1)}$
Initialize $\varphi_{u(k-1)}$
Initialize $\varphi_{\varepsilon(k-1)}$
$\varphi_{k-1} \leftarrow \left[\varphi_y; \varphi_u; \varphi_\varepsilon\right]$
$\hat{\theta}_{k-1} \leftarrow$ zeros(φ_{k-1}.length())
$F \leftarrow 100 \times$ eye(φ_{k-1}.length()) ▷ Initial gain
$y_k \leftarrow$ Read(Output)
while $y_k \neq null$ **do** ▷ Start the loop
    $\hat{y}_k \leftarrow \hat{\theta}_{k-1}^{\intercal} \cdot \varphi_{k-1}$
    $\varepsilon_k \leftarrow y_k - \hat{y}_k$
    $G \leftarrow F \cdot \varphi_{k-1}$ ▷ Recalculation of parameters
    norm $\leftarrow 1 + \varphi_{k-1} \cdot G$
    $F \leftarrow F - \dfrac{G \cdot G^{\intercal}}{\text{norm}}$
    $\hat{\theta}_{k-1} \leftarrow \hat{\theta}_{k-1} + G \cdot \varepsilon_k$
    ▷ Updating φ_{k-1}
    φ_y.addFirst($-y_k$)
    φ_y.poll($n_a + 1$)
    φ_u.addFirst(Read(Input))
    φ_u.poll($n_b + 1$)
    φ_ε.addFirst(ε_k)
    φ_ε.poll($n_c + 1$)
    $\varphi_{k-1} \leftarrow \left[\varphi_y; \varphi_u; \varphi_\varepsilon\right]$
    $y_k \leftarrow$ Read(Output) ▷ Reading the next output
End

III.5 Construction et implémentation du Modèle

Les réseaux de neurones NARX nécessitent une quantité importante de données et des ressources en calcul pour un l'apprentissage. Les processeurs mobiles ne sont pas encore capables de faire un tel entraînement de modèle. En effet, au cours de nos expérimentations, il a fallu environ 45 minutes pour entraîner le modèle NARX sur l'appareil mobile n° 2 avec 8×10^4 points de données,. Lorsque le nombre de points de données dépasse environ $1,2 \times 10^5$ points, l'application commence à présenter des ralentissements pour raison de RAM insuffisante.

La solution évidente à ces limitations est de réaliser l'apprentissage et l'identification sur un appareil séparé. Par conséquent, le processus de construction du modèle commence par la collecte de données. Les données sont ensuite transférées sur un appareil séparé pour l'apprentissage. Une fois le processus d'apprentissage terminé, les modèles sont installés sur un appareil associé aux appareils mobiles pour générer des estimations en ligne.

III.6 Synthèse des choix de conception du profiler N^3

La Table III.4 résume les choix de conception majeurs faits pendant la construction de N^3 en suivant les étapes décrit dans les chapitres II et III.

TABLE III.4 – Une description résumée des choix de conception et modélisation du profiler N^3 pour les smartphones.

Nom du profiler	N^3	
Objectif du profiler	Monitoring du SoC	
Niveau de granularité	Composants	
Type de profiling	Logiciel	
Sources des mesures	Traces du système	
Systèmes	Appareil mobile n° 1	Appareil mobile n° 2
Approche de modélisation	• Rétro-ingénierie • Boite noire	
Types des modèles	• Algorithmes • Tout ou rien • Réseau de neurones NARX • ARMAX	
Disponibilité des estimations	En ligne	
Apprentissage	Hors-appareil	
Implémentation	Mixte	
Temps d'échantillonnage	20 ms	

III.7 Résultats et discussion

Les résultats expérimentaux présentés dans cette partie ont été obtenus en considérant un scénario d'utilisation du système composé de trois étapes :

— Lors de la première étape, le système effectue une série de benchmarks standards. Ces benchmarks sont : PCMark [184], 3Dmark [185], AnTuTu Benchmark [186] et Geekbench4 [187]. Ils servent à tester les performances maximales des systèmes via de multiples tests de calcul, des rendus 3D, etc. Le lancement successif de ces tests génère des données décrivant le fonctionnement système sous des charges de travail fortes (100% à la fréquence maximale pendant 15 minutes). En plus, cela stressera également le système physiquement (augmentation de température) et décrira son comportement sous l'étranglement thermique (si présent).

— Durant la deuxième étape, le système est soumis à des tâches plus courantes : appel téléphonique, appel vidéo Internet, lecture vidéo locale, lecture vidéo en streaming, lecture audio, téléchargement de fichiers sous le réseau cellulaire, puis par Wi-Fi, prendre une photo sans flash, avec flash, et enregistrer une vidéo de 30 s.

— Lors de la troisième étape, l'appareil est laissé inactif pendant un certain temps.

Cette expérience de collecte de données est répétée plusieurs fois avec des variations dans l'ordre des points de repère et des tâches.

III.7.1 Validation des modèles de la fréquence et de la tension

La Figure III.10 montre les variations des fréquences mesurées et celles estimées par le modèle de fréquence de l'un des cœurs du CPU de l'appareil n° 1. Elle montre des résultats presque identiques avec un léger retard aux instants des changements de la fréquence. Le retard est expliqué par le temps nécessaire au modèle pour évaluer le changement. Le retard maximum noté est de $\tau_{f_{ref1}} = 0.1\,\text{s}$, pendant une période de fortes charges, en raison du retard de planification [188]. Cette valeur sera utilisée dans l'étape d'évaluation des indices de dérives des caractéristiques, dans le prochain chapitre. En analysant la Figure III.11, nous arrivons à la même conclusion pour le modèle de tension, la seule différence est le retard maximum enregistré $\tau_{Vref1} = 0.12\,\text{s}$.

Les figures III.12 et III.13 montrent les estimations la fréquence et de celui de la tension, respectivement pour l'appareil mobile n° 2. Celles-ci offrent les mêmes conclusions que pour l'appareil mobile n° 1 à la différence des retards maximums notés. Pour le cas la fréquence, le retard maximum est $\tau_{f_{ref1}} = 0.07\,\text{s}$, tandis que pour la tension il est de $\tau_{V_{ref2}} = 0.1\,\text{s}$. Enfin, les modèles des gouverneurs de fréquences et de tensions du GPU des deux appareils offrent pratiquement les mêmes résultats que ceux des CPU.

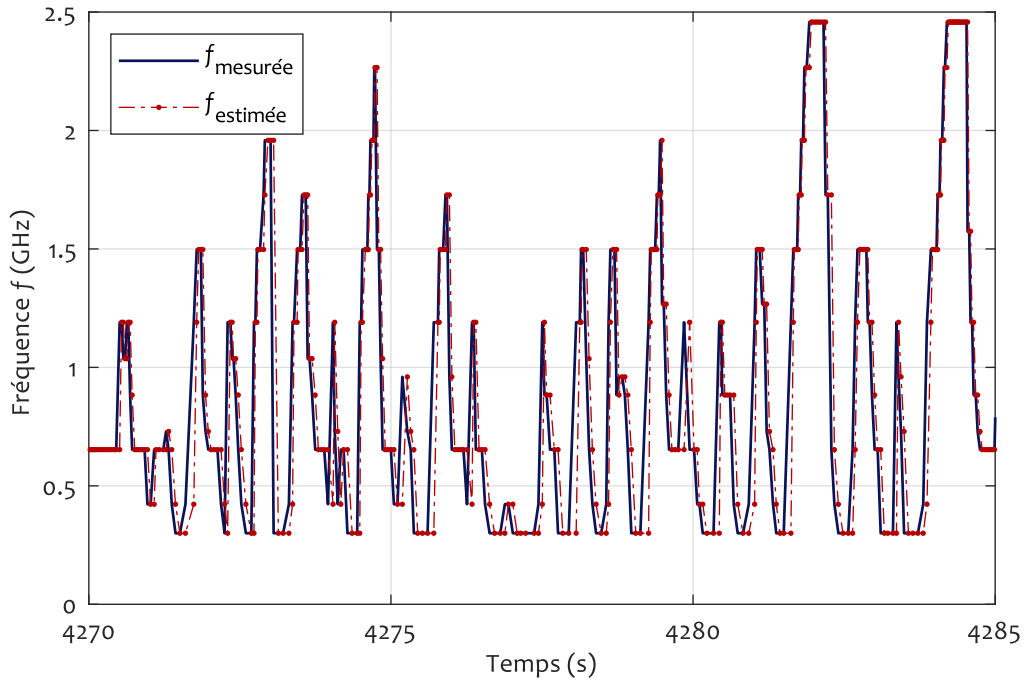


FIGURE III.10 – Les valeurs des fréquences acquises contres les estimations du modèle de fréquence d'un cœur du CPU de l'appareil mobile nº 1.

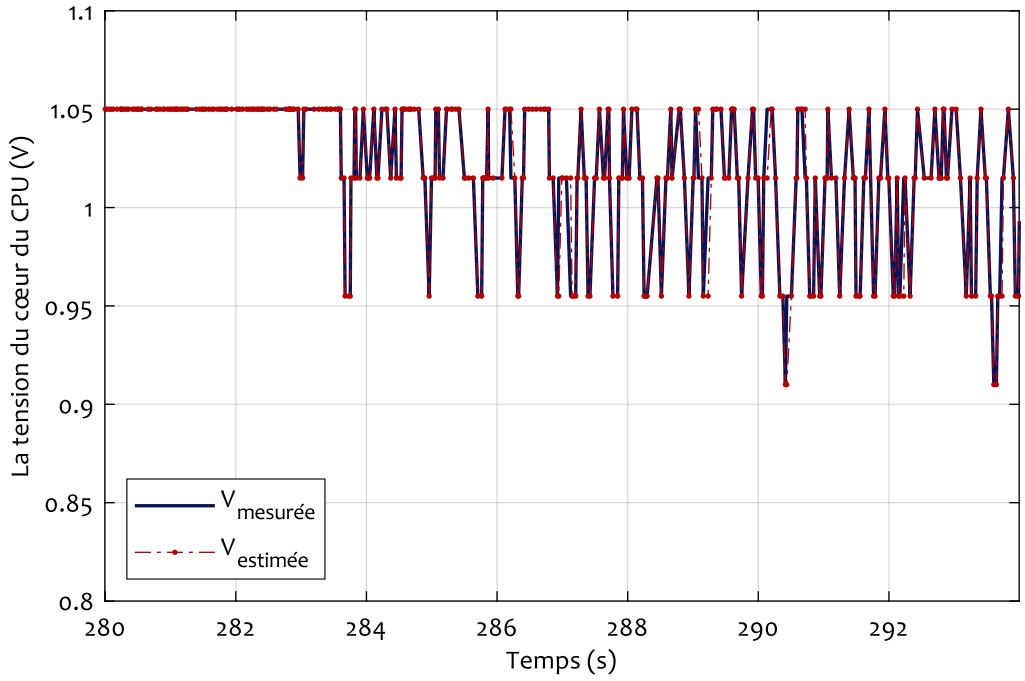


FIGURE III.11 – Les estimation du modèle de tension contre les lectures du système pour un cœur du CPU de l'appareil mobile nº 1.

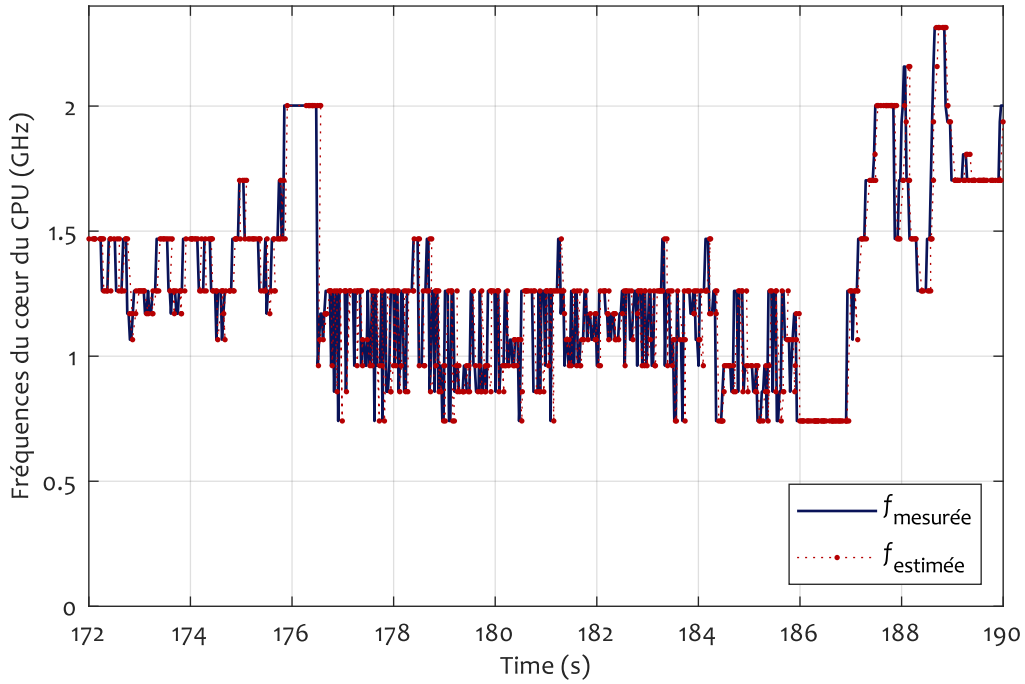


FIGURE III.12 – Les estimation du modèle de fréquence contre les lectures du système pour un cœur du CPU de l'appareil mobile n° 2.

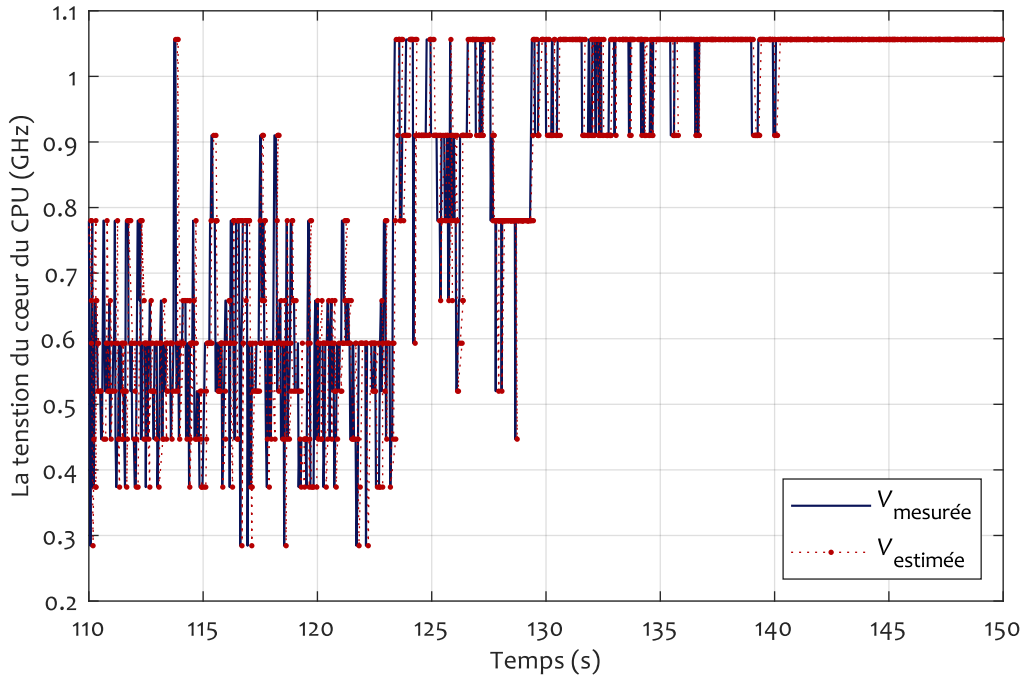


FIGURE III.13 – Les estimation du modèle de tension contre les lectures du système pour un cœur du CPU de l'appareil mobile n° 2.

Les retards maximum enregistrés durant l'expérimentation de la validation des modèles de fréquence et de tensions sont affichés dans la Table III.5. Ces résultats – les courbes d'enregistrement ainsi que les retards enregistrés – valident les modèles de fréquences et de tensions sur les deux appareils.

TABLE III.5 – Les retard maximum $\tau_{f_{ref}}$ et $\tau_{V_{ref}}$ enregistrés sur les appareils mobiles n° 1 et n° 2.

Système	**Appareil Mobile n° 1**		**Appareil Mobile n° 2**	
Retard maximum (s)	$\tau_{f_{ref1}}$	$\tau_{V_{ref1}}$	$\tau_{f_{ref2}}$	$\tau_{V_{ref2}}$
CPU	0.1	0.12	0.07	0.1
GPU	0.12	0.13	0.09	0.11

III.7.2 Validation du modèle de puissance

Dans les prochains paragraphes, nous détaillerons les résultats de la modélisation de la puissance, en commençant par la présentation des résultats obtenues avec des données enregistrées au préalables, c'est à dire un apprentissage et un test hors lignes.

III.7.2.1 Apprentissage, validation, et test hors-ligne du modèle de puissance

Au début de la Section III.7, nous avons décrit le scénario de collection des données. Une fois ces données sont collectées, elles sont ensuite divisées en trois groupes : un dataset pour l'apprentissage (*training set*, 60%), un deuxième pour la validation (*Validation set*, 20%), et un dernier dataset pour les tests (*test set*, 20%) et l'évaluation de la performance. La construction du dataset d'apprentissage est décrite dans le Paragraphe III.4.1.

Les résultats des dataset de tests pour différents nombres d'échantillons d'apprentissage sont présentés dans la Table III.6. Ces résultats montrent la corrélation entre les mesures et les sorties du modèle ($R \approx 1$). De plus, les prédictions s'adaptent à la sortie, avec des erreurs très faibles (MAPE = 2,2% pour l'appareil mobile n° 2), indiquant la précision du modèle. Ainsi,le modèle NARX est validé avec les résultats du meilleur dataset présentés dans la Table III.7.

Néanmoins, la Table III.6 montre également les limites de l'utilisation des réseaux de neurones. En effet, le MAPE est passé de 5% à 2,2% en ajoutant davantage d'échantillons, mais au prix d'un temps d'entraînement plus longs. De plus, après un certain nombre d'échantillons ($1,8 \times 10^6$ pour l'appareil mobile n° 2, par exemple), le MAPE ne s'est plus amélioré.

TABLE III.6 – Le temps d'entraînement nécessaire pour différents nombres d'échantillons ainsi que la MAPE, et la MSE et R des dataset de tests.

	Appareil Mobile n° 2				Appareil Mobile n° 1			
Nombre total d'échantillons	temps d'entraînement (min)	MAPE (%)	MSE	R	temps d'entraînement (min)	MAPE (%)	MSE	R
$\sim 8 \times 10^4$	7	5	7.56×10^{-3}	0.986	12	2.8	4.72×10^{-3}	0.953
$\sim 2 \times 10^5$	12	3	7.23×10^{-3}	0.986	15	2.8	4.68×10^{-3}	0.953
$\sim 4 \times 10^5$	14	2.9	7.18×10^{-3}	0.986	18	2.6	3.67×10^{-3}	0.953
$\sim 8 \times 10^5$	20	2.4	$7.60 \times 10^{-}4$	0.987	29	2.6	3.56×10^{-3}	0.962
$\sim 1.2 \times 10^6$	32	2.2	$4.48 \times 10^{-}4$	0.987	35	2.6	3.61×10^{-3}	0.967
$\sim 1.8 \times 10^6$	57	2.2	$3.17 \times 10^{-}4$	0.987	57	2.6	3.54×10^{-3}	0.967

TABLE III.7 – Les meilleurs résultats de l'apprentissage et de la validation du modèle NARX pour l'apprail mobile n° 2.

Dataset	Entraînement	Validation	Test	**Test en ligne**
MSE	2.32×10^{-4}	2.68×10^{-4}	3.17×10^{-4}	$\mathbf{7.04 \times 10^{-4}}$
MAPE (%)	2.13	2.19	2.20	**2.82**

III.7.2.2 Validation de l'utilisation en ligne

Après avoir validé le modèle grâce aux datasets de test, nous l'intégrons maintenant au fonctionnement en ligne, en le déployant sur un ordinateur relié à l'appareil mobile n° 2 via le protocole TCP/IP incorporé dans l'application *CommunicatingProfiler*. Les entrées du modèle sont les fréquences estimées par le modèle de la fréquence avec les autres autres entrées nécessaire acquise du SoC.

Ensuite, nous testons sa précision et ses performances via une utilisation normale aléatoire et non scriptée de l'appareil. Ce test, qui est configuré pour imiter une utilisation normale de l'appareil, comprend les activités quotidiennes : Appels cellulaires et vidéo via des applications, l'envoi de messages via SMS et les d'applications, la lecture et le streaming des vidéo, la prise et l'envoi d'images, navigation du Web, et enfin des jouer à deux jeux différents (*Dots : A Game About Connecting* [189] et *Monument Valley* [190]).

La Table III.8 met en évidence les performances en ligne du modèle sur l'appareil mobile n° 2 De plus, dans la Figure III.14, la consommation d'énergie mesurée du smartphone est comparée à celle estimées par N^3. Elle montre clairement que le modèle NARX prédit avec précision la consommation d'énergie du téléphone avec un minimum d'erreurs d'estimation. Il a une MAPE de seulement 2,82%, et une MAE enregistrée de 0.0168 W, alors que le MSE est de $7,04 \times 10^{-4}$. De plus, 97% des estimations – en termes d'échantillons – réalisées par le modèle, ont une valeur d'erreur nulle ou inférieure à la précision des mesures, qui est de 10^{-3} W).

TABLE III.8 – Les résultats des tests en ligne du modèle NARX sur l'appareil mobile n° 2.

	Appareil mobile n° 2
Durée du test	2.25 h
Nombre d'échantillons	$\sim 403 \times 10^3$
MAE (W)	0.0168
MAPE (%)	2.82
MSE	7.04×10^{-4}

La Figure III.16 illustre la distribution des erreurs d'estimation du modèle NARX pour le test en ligne. Les erreurs ont une valeur moyenne de $1,178 \times 10^{-4}$W et un écart type de 0,0265, avec 99,4 % des valeurs des erreurs comprise dans $\mu \pm 3 \times \sigma$. De plus, le test de Kolmogorov – Smirnov (KS) confirme que cette distribution correspond au profil d'une distribution normale standard, et le test ARCH d'Engle confirme leur homoscédasticité.

III.7.2.3 Évaluation et comparaison des performances

En termes de performances, la durée moyenne d'échantillonnage de N^3 est de 19.8 ms, ce qui satisfait notre contrainte de conception ($T_s = 20$ ms). De plus, N^3 n'a causé que 3% de charge

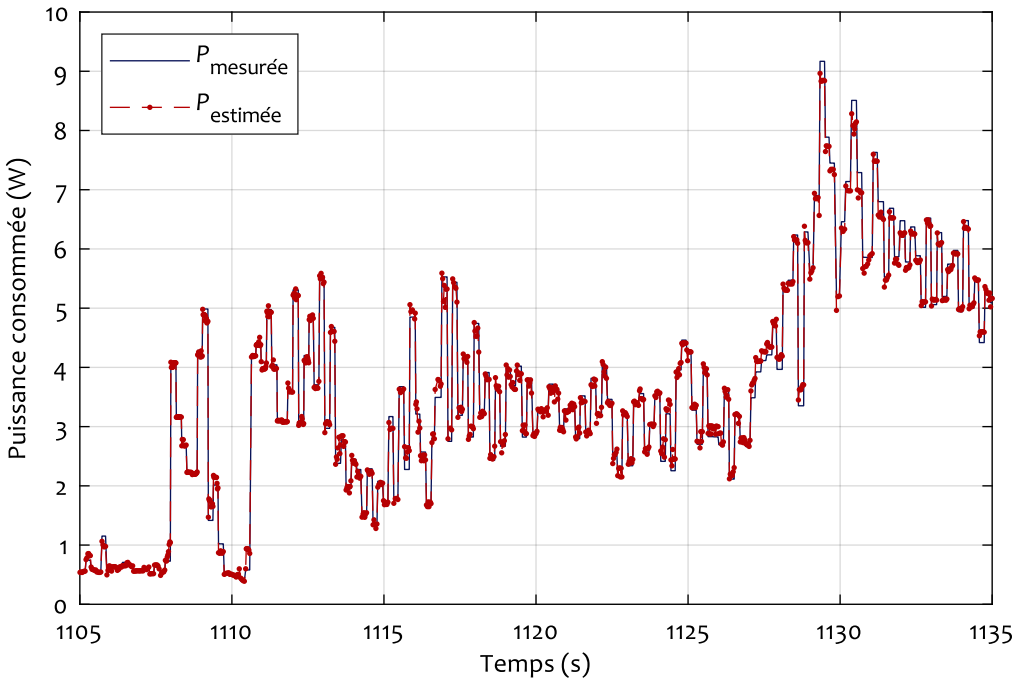


FIGURE III.14 – Les mesures de puissance de l'appareil mobile n° 2 comparées aux estimations en ligne du modèle NARX.

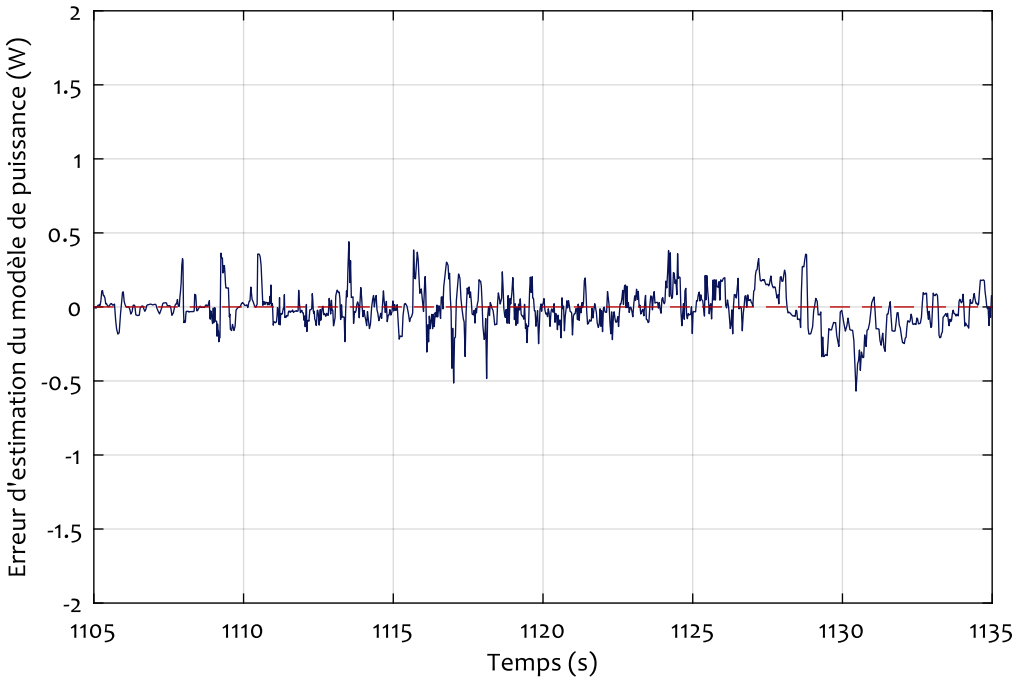


FIGURE III.15 – Les érreurs d'estimation du modèle de puissance $\varepsilon_P(k) = P_{mesurée}(k) - P_{estimée}(k)$ pour le test en ligne de l'appareil mobile n° 2.

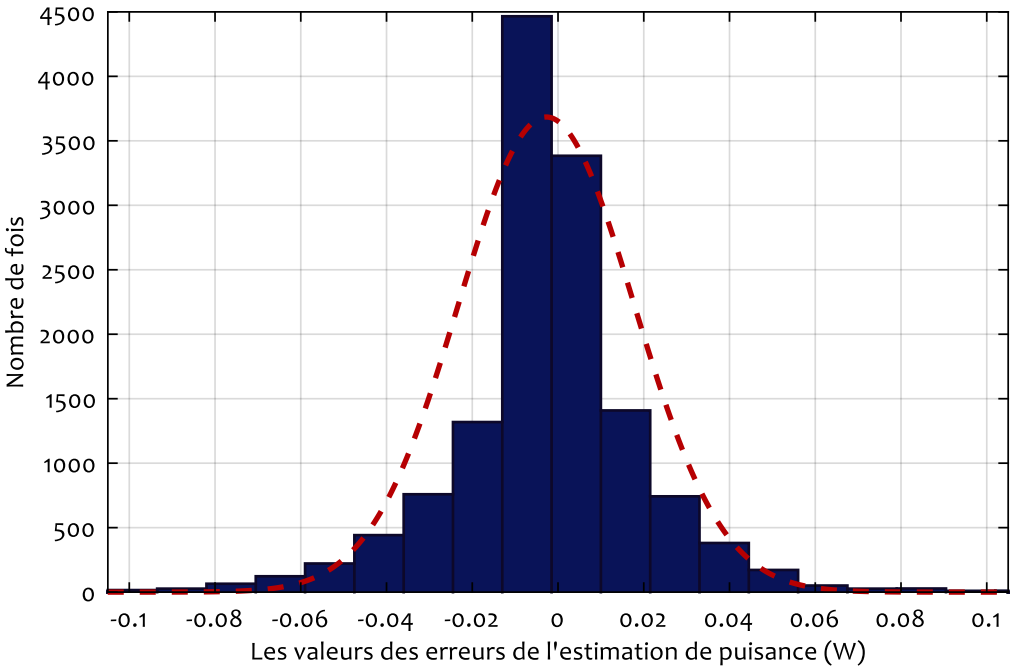


FIGURE III.16 – L'histogramme des erreurs relatives d'estimation du modèle de puissance NARX en bleu, avec le profil de la distribution normale correspondante en rouge.

supplémentaire pendant le test. Quant à la surcharge de consommation en puissance, elle ne présentait qu'une augmentation de 5% dans le même test.

Enfin, nous avons comparé nos résultats avec les modèles de puissance dont les résultats sont disponibles dans la littérature. La Table III.9 montre la précision de tous ces modèles comparées à celle du modèle de puissance présenté dans ce travail.

Bien que le tableau montre que notre modèle est comparable, voir meilleurs que certains modèles établis, il convient également de noter que certains de ces profilers ont des niveaux de granularité supérieurs à celui de notre modèle. De plus, certaines œuvres, comme *PowerBooter*, n'ont pas été mises à jour depuis plusieurs années, et *Trepn* a été conçu à l'aide des SoC de la même marque, les SoC, et aurait donc probablement obtenu une meilleure précision avec cette marque de SoC [43].

Ces résultats sont très encourageant et valident ainsi notre modélisation de puissance à l'aide des réseaux de neurones NARX, avec des performances très satisfaisantes.

III.7.3 Validation du modèle de température

Au cours de la description du modèle de température dans le paragaphe III.4.2.2, nous avons décrit deux méthodes pour le calcul des paramètres du modèle ARMAX : Les moindres carrés étendus et les moindres carrés récursifs. Nous avons donc fait l'identification de deux modèles à l'aide de ces deux méthodes : ARMAX_{LS} pour les moindres carrés étendus et $\text{ARMAX}_{\text{RLS}}$ pour les moindres carrés récursifs. Dans le reste de ce paragraphe, nous explorons les résultats obtenus par les modèles établis par ces deux méthodes.

TABLE III.9 – Une comparaison de la précision de plusieurs modèles et profilers de puissance avec les résultats du modèle NARX.

Profiler/Model	MAPE (%)	Source	Année
***N*³**	**2.88**	**Mesures**	2019
Snapdragon Profiler	4.8	Mesures	2018
Trepn	5.5	Mesures	2018
Yoon et al. [99]	5.1	[99]	2017
PETrA	4	[93]	2017
Walker et al. [100]	3.8	[100]	2017
Djedidi et al. [9]	**4.4**	**[9]**	**2017**
Kim et al. [13]	2.9	[13]	2015
eLens	~ 10	[153]	2013
Kim et al. [92]	$3.8 \sim 7.2$	[92]	2012
Power Doctor	10	[150]	2012
eProf	10	[150]	2011
Sesame	5	[87]	2011
PowerBooter	4.1 / 24	[18]/Mesures	2010/2018

Pour l'identification, seuls deux datasets sont nécessaires. Le premier dataset est utilisé pour l'estimation des paramètres du modèle, tandis que le second est un dataset utilisé pour la validation. Pour le modèle de température, les résultats présentés pour valider et mettre en valeur la précision du modèle, sont ceux obtenus en utilisant un dataset unique (pour chaque appareil) contenant environ 4×10^4 échantillons (environ 15 minutes de test). Puis enfin, comme dans le cas du modèle de puissance, le modèle a aussi été testé en ligne.

La première série d'essais a été lancée avec différents ensembles d'ordres. Le meilleur ensemble est choisi en fonction de deux critères : MAPE et MSE. La Table III.10 montre un échantillon des résultats obtenus avec plusieurs configurations d'ordres $[n_a, n_b, n_c]$. Ces résultats montrent que le modèle $\text{ARMAX}_{\text{RLS}}$ donne des résultats très encourageants, avec des MAPE aussi bas que $0.7 \sim 1.2\%$, et une MAE de l'ordre des 0.5 °C. La Figure III.17 montre comment les estimations du modèle $\text{ARMAX}_{\text{RLS}}$ suivent bien les mesures du système pendant les phases de réchauffement et les phases de refroidissement. Les erreurs d'estimation du modèle sont affichées dans la Figure III.18. Ces erreurs ont une faible valeur (MAE = 0.58 °C) et affirment la précision des estimations du modèle.

La Table III.10 montre aussi que l'ordre du modèle dépend de l'appareil. La combinaison d'ordre donnant les meilleurs résultats pour l'appareil mobile n° 1 a été trouvée rapidement, avec des ordres assez bas ([2,4,2]). Alors, que pour l'appareil mobile n° 2, la meilleure combinaison est égale à [4,4,2], qui a aussi générée des résultats très satisfaisant même pour l'appareil mobile n° 1.

TABLE III.10 – L'évolution de la précision de l'estimation générée par le modèle $ARMAX_{RLS}$ en fonction de ses ordres pour les deux appareils mobiles.

Appareil mobile n° 1						**Appareil mobile n° 2**					
Ordres du modèle			MAE °C	MAPE (%)	MSE	Ordres du modèle			MAE °C	MAPE (%)	MSE
n_a	n_b	n_c				n_a	n_b	n_c			
2	4	1	0.572628	1.186267	1.202266	2	4	1	2.634785	4.432009	27.717705
2	**4**	**2**	**0.571418**	**1.180702**	**1.453916**	2	4	2	1.481039	2.328012	3.128547
2	6	1	0.622204	1.297458	1.240013	4	1	1	1.468235	2.645218	2.912354
2	6	2	0.689443	1.445995	1.386370	4	2	1	1.392638	2.223086	2.753949
3	4	2	0.585938	1.207557	1.314500	4	2	2	1.899935	3.232330	12.409634
3	5	1	0.627544	1.292716	1.569117	6	6	1	2.221837	3.826583	22.454720
3	5	2	0.676534	1.394789	2.157853	8	8	6	2.735453	4.137845	22.875134
4	**4**	**1**	**0.589068**	**1.213134**	**1.339650**	**4**	**4**	**2**	**0.474134**	**0.731232**	**0.357114**
5	6	2	0.596366	1.234902	1.242312	9	9	2	0.4793154	0.762315	0.382348

TABLE III.11 – Les résultats de la validation et du test en ligne du modèle $ARMAX_{LS}$ comparé à ceux du modèle $ARMAX_{RLS}$ sur l'appareil mobile n° 2.

Modèle	Validation		**Test en ligne**	
	MAE(°C)	MAPE (%)	MAE(°C)	MAPE (%)
$ARMAX_{LS}$	0.38	0.65	**0.83**	**1.48**
$ARMAX_{RLS}$	0.47	0.73	**0.78**	**1.43**

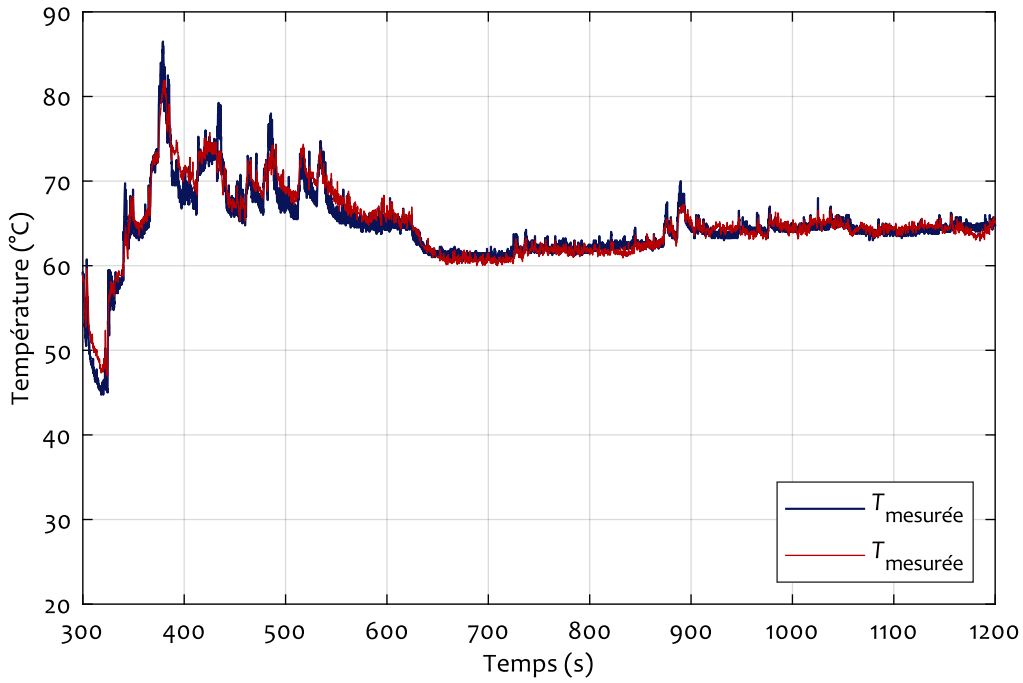


FIGURE III.17 – Les estimations de la température du modèle $\text{ARMAX}_{\text{RLS}}$ par rapport aux lectures du système dans l'appareil mobile n° 1.

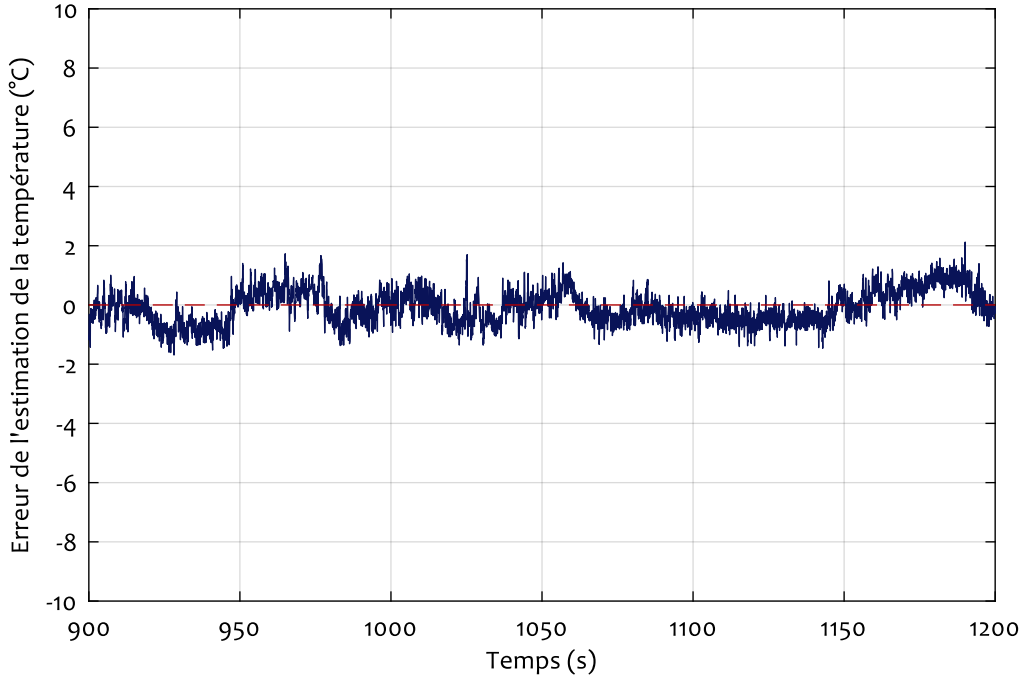


FIGURE III.18 – Les erreurs d'estimation du modèle $\text{ARMAX}_{\text{RLS}}$ $\varepsilon_T = T_{mesurée} - T_{estimée}$ pour l'appareil mobile n° 1.

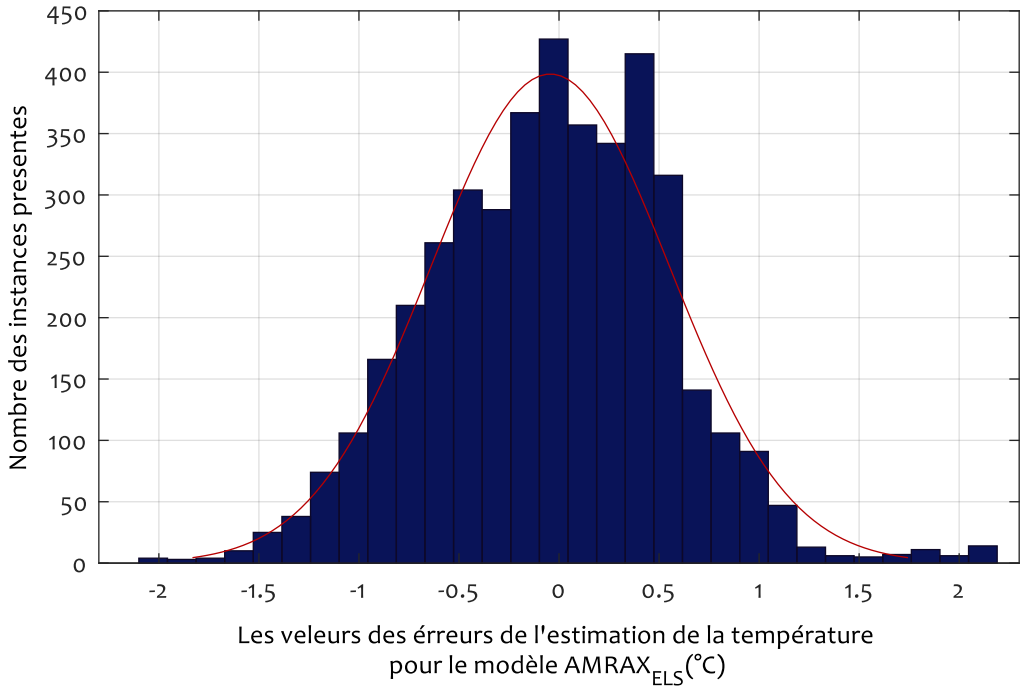


FIGURE III.19 – La distribution des erreurs du modèle ARMAX$_{RLS}$ pour l'appareil mobile n° 1.

Pendant la deuxième série d'expérimentations, nous avons comparé les résultats donnés par le modèle ARMAX$_{LS}$ et ceux donnés par ARMAX$_{RLS}$. La Table III.11 affiche les MAPE et les MAE des modèles ARMAX$_{LS}$ et ARMAX$_{RLS}$ pour le dataset de validation, ainsi que pour le test en ligne. Bien que le modèle ARMAX$_{LS}$ présente un léger avantage dans ses résultats de validation hors ligne avec une MAE de 0.38 °C (1,48 %), l'estimation en ligne (notre cas d'utilisation) montre encore un léger avantage pour le modèle ARMAX$_{RLS}$.

Les erreurs d'estimation des deux modèles ARMAX sont normalement distribuées (test de K-S), et ne présentent pas d'hétéroscédasticité (test ARCH d'Engle). Comme dans le cas du modèle de puissance, avec 99,2% des valeurs comprises entre $\mu \pm 3 \times \sigma$, pour ARMAX$_{LS}$ et 99.4% pour ARMAX$_{RLS}$. La Figure III.19 affiche la distribution de ces erreurs le modèle ARMAX$_{RLS}$ pendant le test en ligne.

La Figure III.20 affiche l'évolution de la température estimée sur l'appareil mobile n° 2 par rapport à la température mesurée du SoC. Les estimations fournies par notre modèle suivent bien la dynamique de la température mesurée pendant les phases de grandes variations (réchauffement et refroidissement), ainsi que pendant les phases avec de faibles variations de température. La figure montre également que le modèle prend en compte les conditions initiales. La modélisation de la température est donc validée.

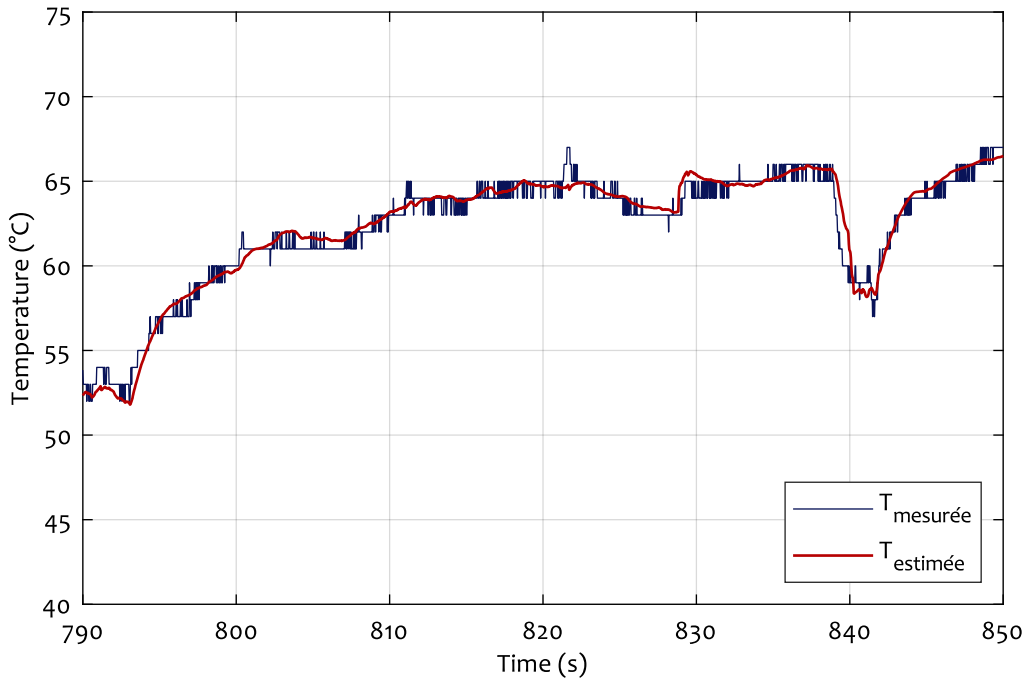


FIGURE III.20 – l'évolution des estimations de la température du modèle $\text{ARMAX}_{\text{RLS}}$ par rapport aux lectures du système dans l'appareil mobile n° 2, pendant le test en ligne.

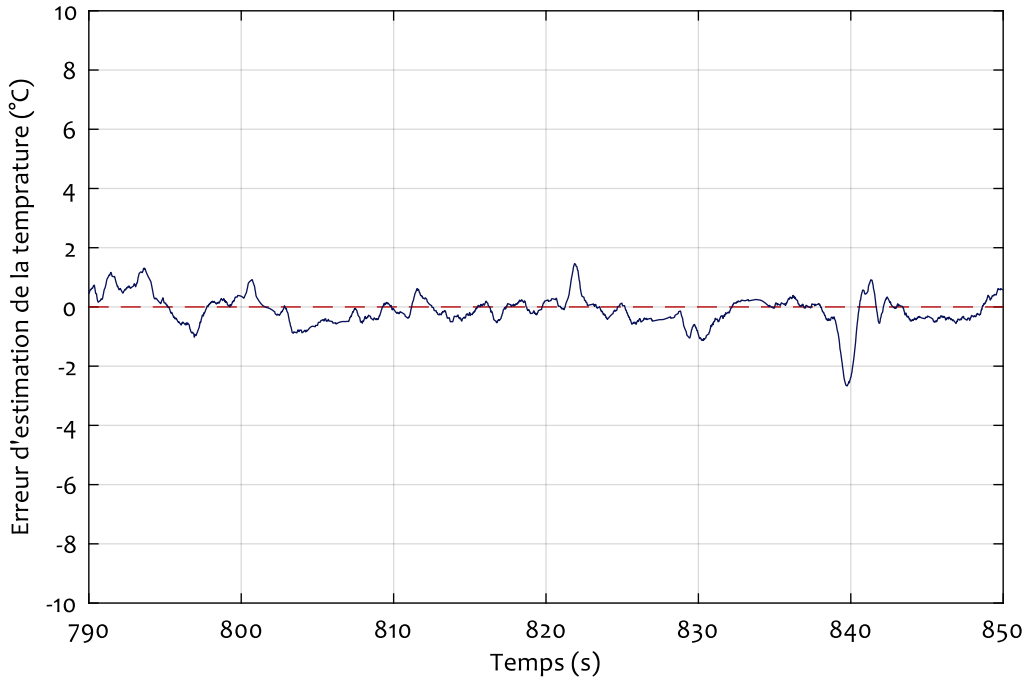


FIGURE III.21 – Les erreurs d'estimation du modèle $\text{ARMAX}_{\text{RLS}}$ $\varepsilon_T = T_{mesurée} - T_{estimée}$ pour l'appareil mobile n° 2, pendant le test en lignes.

III.8 Conclusion

Nous avons développé dans ce chapitre une approche de modélisation incrémentale des SoC hétérogènes, qui présente le modèle global du système sous forme de modules (sous-systèmes) interconnectés par des relations de cause à effet. Cette structure modulaire est également ouverte et évolutive grâce à la possibilité de construction et d'intégration de nouveaux modules dans le modèle global, permettant ainsi au modèle de s'adapter aux différents SoC existants sur le marché, et aussi de suivre les évolutions technologiques permanentes dans ce domaine. L'implémentation et le test de cette structure effectués sur deux SoC de générations différentes et configurations distinctes montrent la portabilité de cette structure et son adaptabilité.

Les entrées du modèle sont identifiées afin de représenter les aspects matériels et les aspects logiciels à travers les charges des processeurs et le taux d'occupation mémoire. Le modèle que nous avons présenté dans ce chapitre est capable d'estimer des variables discrètes et des variables continues échantillonnées, il a été validé expérimentalement hors ligne, puis en ligne en utilisant le système d'acquisition et de traitement des données. Les résultats obtenus montrent l'efficacité du modèle et sa capacité à estimer différentes dynamiques.

Les sorties estimées par ce modèle incrémental sont les variables qui caractérisent l'état de fonctionnement du système. Elles dérivent de leurs valeurs nominales lorsque le système se dégrade, en termes de performance (lenteur dans l'exécution des instructions, saturation des mémoires, ...), ou en termes de dynamique (surchauffe, surconsommation de puissance).

Dans le chapitre qui suit, nous proposons un algorithme de détection de la dérive de ces caractéristiques qui se base sur le modèle développé dans ce chapitre.

CHAPITRE IV

SURVEILLANCE DES SOC DES SYSTÈMES EMBARQUÉS

IV.1 Introduction

Les études de diagnostic ou de surveillance dans les systèmes avioniques ou même automobiles sont principalement concentrées sur les pièces mobiles des appareils. Néanmoins, avec l'intégration des nouvelles technologies intelligentes, ces études devaient être adaptées pour prendre en compte les nouveaux systèmes de commande, qui dans la plupart des cas intègrent des microprocesseurs. Ces SoC sont généralement intégrés dans un environnement complexe, avec des cycles de réchauffement et de refroidissement liés au fonctionnement des moteurs et l'altitude, ainsi que des conditions vibratoires à forte variabilité, susceptibles de provoquer un vieillissement accéléré de ces dispositifs. Ces derniers auraient alors une durée de vie inférieure à la moyenne annoncée par les fabricants.

Dans ce chapitre, nous proposons une méthode de diagnostic capable de surveiller l'état de fonctionnement des SoC grâce à la détection précoce des dérives de leurs caractéristiques. Pour répondre aux contraintes imposées dans le cadre du projet MMCD, à savoir la généricité, l'efficacité et la simplicité de déploiement des algorithmes, nous avons développé une approche basée sur le principe de la redondance analytique, où la redondance n'est pas créée par des modèles physiques mais en utilisant le modèle incrémental développé dans le Chapitre III. Ainsi, l'approche proposée ne nécessite généralement pas d'instrumentation supplémentaire et permet de surveiller toutes les variables qui caractérisent l'état de fonctionnement du SoC. Pour l'évaluation des indices des dérives et la prise de décision, nous avons opté pour des méthodes faciles à mettre en œuvre et à paramétrer pour une large gamme de SoC. Les travaux réalisés dans ce chapitre ont été publié dans [4].

Le reste de ce chapitre est organisé en 5 parties. La prochaine section comprend une présentation des travaux réalisée dans littérature abordant la surveillance et le diagnostic des systèmes à microprocesseur avec un œil sur les SoC embarqués. Ensuite, nous aborderons notre algorithme de surveillance, son implémentation, suivi par l'approche de génération et traitement des résidus. Enfin, nous validerons l'algorithme de surveillance à l'aide d'une une expérience de fonctionnement normale des appareils mobiles, puis nous le testerons grâce à des scénarios simulant différents fonctionnements.

IV.2 La surveillance des systèmes embarqués dans la littérature

Dans le domaine de la surveillance, la plupart des travaux sur le diagnostic et la détection des dérives sont résumés dans une revue en deux parties par Gao et al. [191, 192]. Dans cette revue, les auteurs ont réparti ces travaux en quatre catégories : les approches basées sur des modèles, les approches basées sur le signal, les approches basées sur la connaissance, et les approches hybrides/actives. Ce travail a également mis en évidence les applications de diagnostic des pannes, ainsi que les méthodes de tolérance aux pannes et leurs applications [191, 192].

Notre travail entre dans la seconde catégorie (approches basées sur les signaux [191]). Cependant, la plupart des travaux existants n'étudient pas spécifiquement les SoC de manière indépendante, mais plutôt les cartes électroniques dans l'ensemble. Ces cartes sont souvent considérées comme

des systèmes discrets dont le fonctionnement dépend du bon fonctionnement de tous les sous-composants et font l'hypothèse qu'il existe une forte dépendance de fonctionnement entre chacun de ces composants [193]. En conséquence, les méthodes de diagnostic développées sont basées sur des modèles causaux tels que le graphe à flux de signaux multiples (*multi-signal flow graph*) [194], les modèles à flux d'informations (information flow) [195], les graphes orientés (Direct graphs) [196] et les arbres de défaillances (fault tree) [197].

Cui et al. [198] ont introduit un modèle de dépendance amélioré, puis l'ont utilisé pour la détection et l'isolation des défauts sur une carte électronique hébergeant une puce CPU-GPU utilisée dans le domaine de l'avionique. Ce nouveau modèle offre des améliorations par rapports au travaux existants dans la littérature en trois points principaux. Premièrement, il permet de ne pas prendre en compte et d'éliminer plusieurs pannes en incluant des commutateurs pour déconnecter les parties du corps principal du modèle dans le modèle graphique de dépendance (*Dependency graphical model*, DGM). Deuxièmement, les auteurs ont implémenté dans leurs modèles les concepts de reconfiguration dynamique. Enfin, ce modèle est le premier à prendre en compte les tests de dysfonctionnements [198].

Dans la revue réalisée par Gizopoulos et al. [199], les auteurs ont fourni une classification et une étude détaillée des techniques existantes de détection d'erreurs en ligne, appliquées aux architectures de processeurs multicœurs. Ces méthodes sont classées en quatre catégories principales : exécution en redondance (*redundant execution*) [200–202], les approches d'auto-test périodique intégré (*periodic Built-In Self-Test*, BIST) [203], les approches de vérification dynamique (*dynamic verification*) [204, 205], et les approches de détection des anomalies (anomaly detection approaches) [206, 207]. Les résultats de cette étude comparative mettent en évidence l'efficacité des méthodes de vérification dynamique dans l'identification des défauts transitoires et permanents, ainsi que les défauts de conception. Ce dernier point était au centre des travaux récents de Sinha et al. [208] qui ont étudié la fiabilité des systèmes matériels-logiciels au stade de la conception. Les auteurs ont proposé un modèle fonctionnel unifié de simulation pour détecter les potentielles défaillances [208].

Alternativement, Mercati et al. [34] ont considéré la fiabilité comme un problème d'optimisation convexe, et ont proposé le *Workload-Aware Reliability Management* (Gestion de la fiabilité en fonction de la charge de travail, WARM) ; une stratégie optimale pour gérer la fiabilité des systèmes multicœurs. En 2014, la norme IEEE 1687-2014 a été publiée. Cette norme décrit une méthodologie permettant d'accéder à l'instrumentation des dispositifs à base de semi-conducteurs [209]. Zadegan et al. [210] ont utilisé les fonctionnalités réseau de cette norme, pour la surveillance sur le terrain des systèmes embarqués, ainsi que la localisation rapide des défauts.

Enfin, Löfwenmark and Nadjm-Tehrani [211] ont publié une revue qui regroupait des travaux portant sur les systèmes multicœurs en avionique. Dans leur revue, les auteurs ont souligné le fait que la sensibilité des systèmes aux défauts ne cesse d'augmenter à cause de la réduction de la taille des transistors, et ont mis en évidence les domaines dans lesquels des recherches sont encore nécessaires [211].

Tous les travaux mentionnés ci-dessus décrivent des méthodes qui se concentre sur un aspect ou un composant spécifique du système embarqué ou du SoC (fiabilité, composants matériels ou logiciels, etc.), ou sont applicable hors ligne. Ce travail de thèse, en revanche, présente un cadre de surveillance capable de suivre toutes les variables caractéristiques du SoC en ligne, en faisant le lien entre les composants logiciels et matériels.

IV.3 Approche de surveillance

La méthode de surveillance que nous proposons est complémentaire de celles mentionnées dans l'état de l'art, car elle vise la détection en ligne et précoce des dérives des caractéristiques du système, et notifier l'utilisateur. Les causes de ces dérives peuvent être liés aux mécanismes de vieillissement de la carte tels que la rupture diélectrique dépendante du temps (*Backend Time-Dependent Dielectric Breakdown*, BTDDB) [111] ou le cyclage thermique (*Thermal Cycling*) [170]. Ces dérives peuvent être aussi lié à une utilisation non-optimale de la carte ou une sous-dimensionnement du système informatique le poussant à fonctionner à des haute température. Comme cette algorithme est destiné à être déployé dans des systèmes et des environnements critiques pour la sécurité (avions), sa tâche principale sera d'assurer que le SoC fonctionne correctement et dans des conditions optimales. Pour ce faire, nous proposons une méthode qui repose sur la redondance analytique, créée par la construction d'un modèle de référence du système. Cette technique compare les sorties réelles du système aux références issues du modèle d'estimation, générant ainsi des indicateurs de défauts. Ces derniers, communément appelés résidus, sont évaluées par des méthodes probabilistes afin de prendre en compte les incertitudes de modélisation dans le processus de prise de décision.

La redondance analytique a déjà été utilisée pour le diagnostic des défaillances dans la partie matérielle des systèmes électroniques. Toutefois, à notre connaissance, ce travail constitue la première utilisation de cette technique pour surveiller à la fois les composantes matérielles et logicielles.

La Figure IV.1 présente les étapes de notre approche de surveillance des SoC avec un CPU à n coœrs et un GPU intégrés. Dans cette approche, le SoC à surveiller tourne et génère ses variables caractéristique en fonction de la charge de travail (Charge_1, ... ,Charge_n, Charge_{GPU}) et le MOR. En parallèle, le modèle incrémental de ce SoC – dit modèle de référence – génère des estimations à l'aide cette charge de travail et ce MOR comme entrées (voir III.2). Ainsi, nous obtenons deux vecteurs de variables caractéristiques ($f_1, \cdots, f_n, f_{GPU}, V_1, \cdots, V_n, V_{GPU}, P_{SoC}, T_{SoC}$) ; un vecteur de mesures et un vecteurs d'estimations. Ensuite, ces deux vecteurs sont utilisés pour détecter les dérives dans trois étapes :

— **La génération des résidus** par la comparaison de chaque variable caractéristique mesurée avec son homologue estimée pendant un fonctionnement normal,

— **L'évaluation des résidus et le calcul des seuils** qui enveloppent les erreurs d'estimation,

— **La génération d'alarme** si les résidus traités dépassent les seuils.

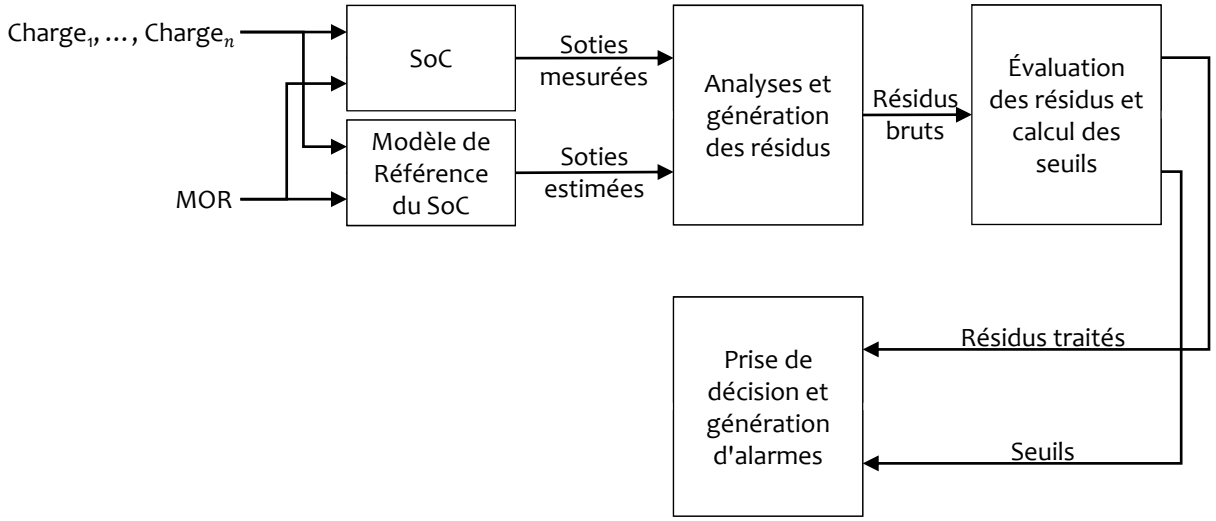


FIGURE IV.1 – Schéma synoptique de la méthode de surveillance proposée.

Ces étapes seront détaillées dans les prochains paragraphes.

Enfin, grâce à la structure modulaire du modèle d'estimation proposé, chaque résidu généré est associé à des modules spécifiques clairement identifiés, permettant ainsi – en présence d'un défaut – de localiser sa source et d'isoler facilement les sous-systèmes défectueux.

IV.4 Lieu de l'implémentation de l'algorithme de surveillance

Il existe deux manières possibles de mettre en œuvre un algorithme de surveillance, le déployer directement sur l'appareil à surveiller, ou utiliser un autre processeur pour le faire. Comme nous l'expliquons dans le prochain chapitre, nous avons choisi d'implémenter l'algorithme de surveillance sur un équipement externe pour ne pas subir de limitation de calcul.

Du point de vue de la fiabilité, l'état de fonctionnement d'une carte électronique devrait être surveillé de manière externe afin de réduire les calculs sur la carte même, et pour que l'état de la carte n'affecte pas le processus de surveillance, et ne subisse pas de surcharge. Si la carte se bloque ou tombe en panne, des avertissements et des indications de défaillance peuvent toujours être générés. Ainsi, nous avons utilisé un PC et l'avons connecté aux appareils surveillés via une connexion TCP/IP. Le profiler N^3 transfère les données de l'appareil au PC où le modèle est implémenté. Le PC génère ainsi les variables caractéristiques de référence estimées, pour les comparer aux lectures envoyées par N^3. Néanmoins, durant la partie expérimentation du présent chapitre, nous avons été ramenés à réaliser des essaies hors ligne, étant donné la natures des expérimentation.

IV.5 Génération et évaluation des résidus des variables caractéristiques

La conception d'un système de surveillance basé sur les résidus, passe par deux étapes principales : La première est la génération des résidus, la seconde est leur évaluation. En théorie, les résidus générés ont une valeur nulle dans les conditions de fonctionnement normales. Toutefois, en pratique, en raison des incertitudes, il est rare que les résidus soient égaux à zéro. Nous notons deux deux types d'approches trouvées dans la littérature spécialisée pour leur évaluation :

— **Les approches basées sur des modèles** utilisent l'expression analytique des résidus pour générer des seuils, qui prennent en compte les incertitudes des paramètres et des modèles, ainsi que les bruits de mesure [212–216].

— **Les approches basées sur les signaux** considèrent les résidus comme des signaux et ne tiennent pas compte de leurs origines physiques. L'idée consiste à extraire les propriétés statistiques et probabilistes du signal résiduel en fonctionnement normal et à les utiliser comme références pour prendre une décision (générer une alarme, par exemple) [217–220].

Dans le cadre de ce travail, nous utilisons une approche basée sur les signaux que nous détaillerons dans la suite de cette section.

IV.5.1 Génération des résidus

Comme nous l'avons déjà mentionné dans la Section IV.3, dans ce travail, les résidus bruts sont générés en comparant les sorties actuelles du système avec celles estimées par le modèle de référence (voir Figure IV.1). Nous les exprimons comme suit pour chacun des variables caractéristiques :

$$\begin{cases} r_{f_i}(k) = \dfrac{f_{i_{estimée}}(k) - f_{i_{mesurée}}(k)}{f_{i_{estimée}}(k)} & \text{(IV.1)} \\ r_{V_i}(k) = \dfrac{V_{i_{estimée}}(k) - V_{i_{mesurée}}(k)}{V_{i_{estimée}}(k)} & \text{(IV.2)} \\ r_P(k) = \dfrac{P_{estimée}(k) - P_{mesurée}(k)}{P_{estimée}(k)} & \text{(IV.3)} \\ r_T(k) = \dfrac{\vartheta_{estimée}(k) - \vartheta_{mesurée}(k)}{\vartheta_{estimée}(k)} & \text{(IV.4)} \end{cases}$$

où $i = \{1, ..., n, GPU\}$ indique soit le GPU, soit le numéro du cœur du CPU, l'indice $_{estimée}$ indique les estimations du modèle et l'indice $_{mesurée}$ dénote les valeurs mesurées (ou lectures). ϑ est la valeur de la température en degrés Kelvin. Nous avons utilisé cette valeur pour éviter la division par 0 dans le cas – improbable – de $T_{SoC} = 0\,°C$. Le cas restant est le cas du GPU ou d'un cœur du CPU éteint. Dans ce cas, le SoC ne génère pas de traces pour la fréquence. Nous la représentons donc comme une fréquence nulle ($f = 0$) dans les mesures et les estimations. Toutefois, lors du calcul des résidus bruts sa valeurs est mise à $f = -1\text{Hz}$ pour éviter la division par 0.

Cette approche de génération des résidus, est bien adaptée à la détection de dérives progressives dans les caractéristiques du système, et permet ainsi une détection précoce de la dégradation. L'interprétation de ces dérives de caractéristiques est réalisée à l'aide des relations de causalité entre les variables, en se basant sur l'expertise des fabricants et des opérateurs du système. Par conséquent, cette méthode utilise une grande partie des informations disponibles (connaissances physiques, expertise et mesures).

IV.5.2 Évaluation des résidus

Cette étape nous permet de déterminer la présence ou l'absence d'un défaut. Le but de cette évaluation est de générer un résidu normalisé $R = 0,1$. Ce résidu sert d'indicateur, pour les utilisateurs non spécialistes, de la présence (ou non) d'un défaut. Ainsi, les résidus bruts générés dans le paragraphe précédent, sont évalués chacun avec des méthodes adaptées à son type, pour tenir compte des erreurs et des retards d'estimation.

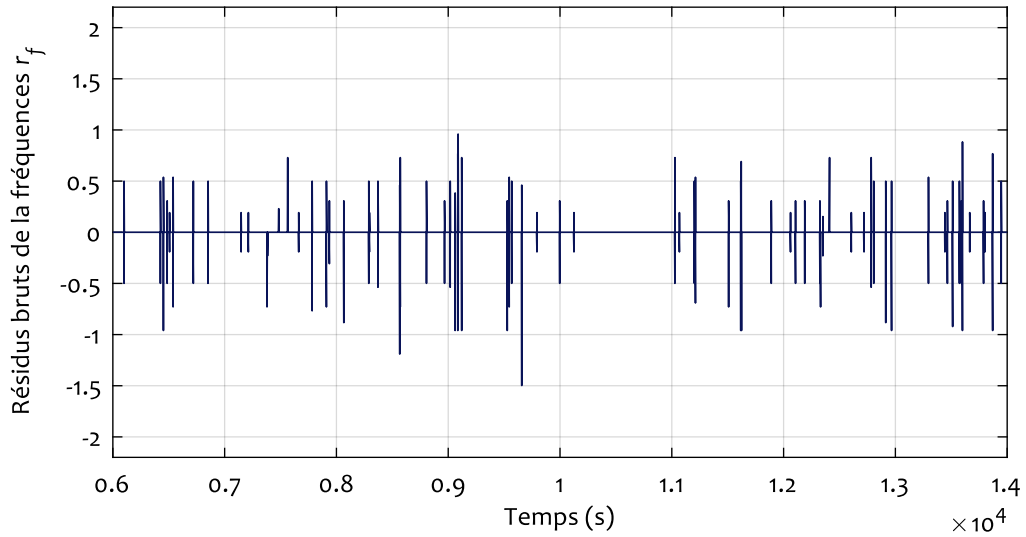
IV.5.2.1 Évaluation des résidus de la fréquence et de la tension

L'analyse des résidus bruts de la fréquence $r_f(k)$ et de la tension $r_V(k)$ montre que ces signaux sont égaux à zéro, mais incluent les différences momentanées – vus comme des pics dans les figures IV.2a et IV.3a respectivement – pendant les moments où la fréquence ou la tension changent de valeurs. Ces pics sont causés par un retard dans le calcul des valeurs estimées par rapport aux valeurs mesurées. Pour prendre en compte ce délai dans le processus de décision, nous avons identifié les valeurs maximales des retards, τ_{fref} et τ_{fref} pour la fréquence et la tension respectivement. Ces valeurs sont rapportées dans la Table III.5. Ensuite, nous les avons utilisées pour calculer des résidus normalisés $R_f(k)$ et $R_V(k)$ comme suit :

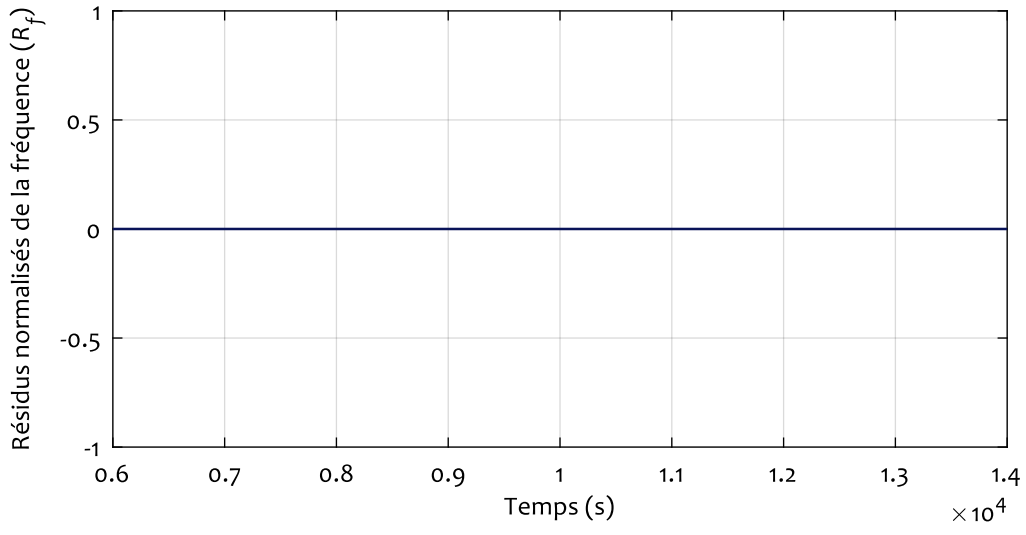
$$R_x(k) = \begin{cases} 1 & \text{Si } r_x \neq 0 \quad \text{ET} \quad \tau_{x_d}(k) > \tau_{x_{ref}}, \\ 0 & \text{Sinon} \end{cases} \tag{IV.5}$$

où $x = \{f, V\}$ indique la fréquence ou la tension, et $\tau_{x_d}(k)$ est la valeur du retard calculé pour chacune de ces deux variables pour l'échantillon k.

Les profils des résidus bruts $r_f(k)$ et des résidus normalisés $R_f(k)$ pris lors de nos tests en ligne sur l'appareil mobile n° 1, sont présentés dans la Figure IV.2. Cette dernière montre comment les résidus bruts sont sensibles aux retards d'estimation du modèle aux instants des changements des fréquences, entraînant ainsi l'apparition d'un certain nombre de pics. Cependant, les résidus normalisés $R_f(k)$ sont égaux à zéro sur la durée du test. Par conséquent, aucune fausse alarme n'est enregistrée. La Figure IV.3, nous permet de tirer la même conclusion pour les résidus bruts de tension $r_v(k)$ et les résidus normalisés $R_v(k)$, qui ne présentent également aucune fausse alarme.



(a)



(b)

FIGURE IV.2 – Les profils des résidus $r_f(k)$ et $R_f(k)$ en fonctionnement normal : (a) : résidus bruts $r_f(k)$, (b) : résidus normalisés $R_f(k)$.

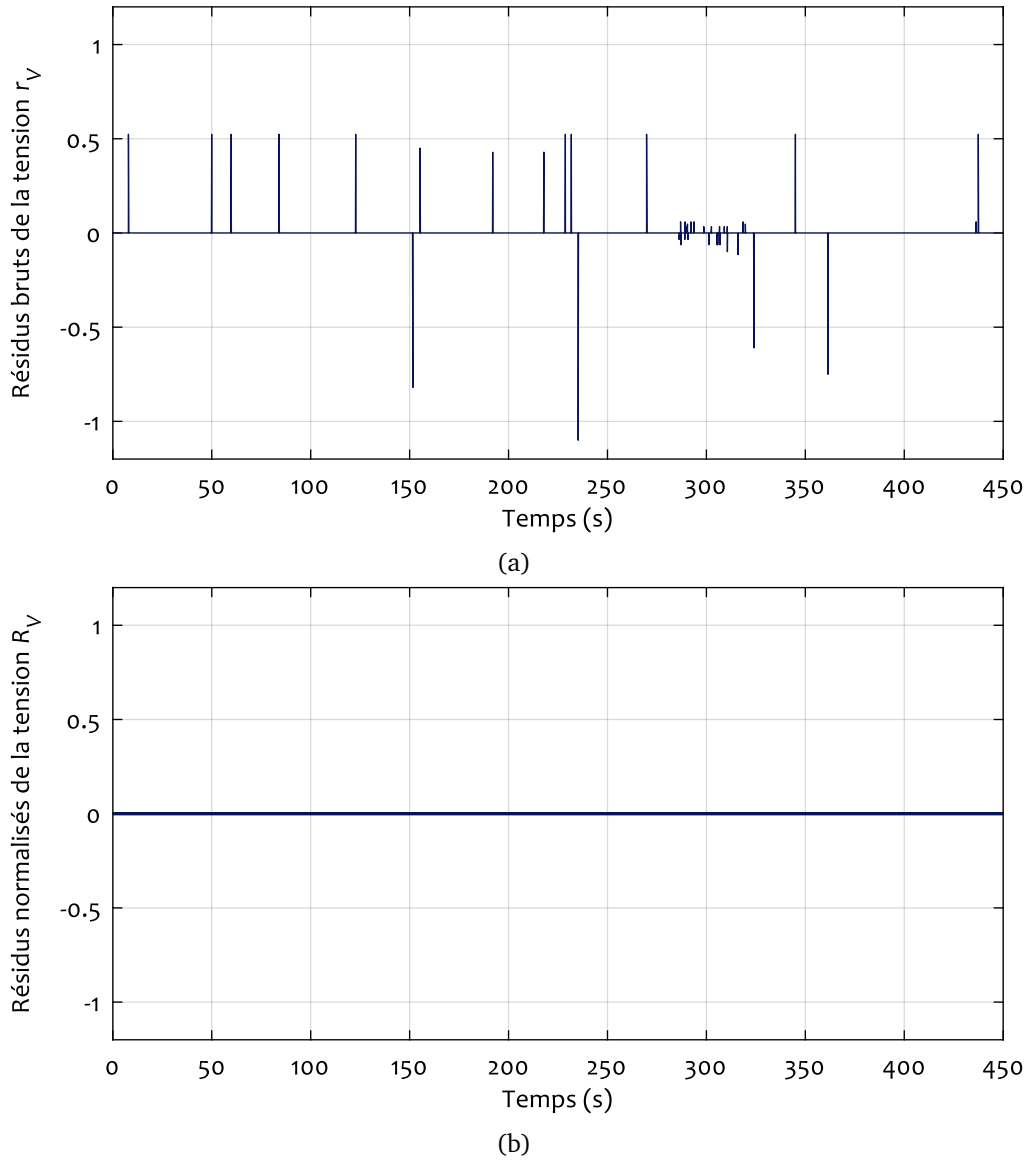


FIGURE IV.3 – Les profils des résidus $r_V(k)$ et $R_V(k)$ en fonctionnement normal : (a) : résidus bruts $r_V(k)$, (b) : résidus normalisés $R_V(k)$.

IV.5.2.2 Évaluation des résidus de puissance et de température

L'analyse des résidus bruts de la puissance $r_P(k)$ et de la température $r_T(k)$ (Figure IV.4a et Figure IV.5a) montre que ces signaux ont un bruit à haute fréquence, avec une moyenne proche de zéro, dû au erreurs d'estimation (voir III.7.2.2 et III.7.3). Le bruit est également normalement distribué autour de cette moyenne (Figures III.16 et Figure III.19). Cette propriété implique que 99,7 % des résidus seront présents entre les valeurs de la moyenne μ plus ou moins trois fois l'écart type σ (Table IV.1).

La dérive est une variation lente du signal (à basses fréquences). Elle est donc portée par la moyenne [221]. Alors pour minimiser l'effet du bruit et d'éviter les fausses alarmes provenant d'erreurs d'estimation et de synchronisation (les hautes fréquence), nous utilisons la méthode de la

TABLE IV.1 – Les taux des valeurs des résidus $r_P(k)$ et $r_T(k)$ comprises entre $\mu \pm 3 \times \sigma$ Pour les deux appareils mobile.

Système	**Appareil Mobile n° 1**		**Appareil Mobile n° 2**	
Résidus bruts	$r_P(k)$	$r_T(k)$	$r_P(k)$	$r_T(k)$
$r_x \in [\mu \pm 3 \times \sigma]$ **(%)**	99.3	99.4	99.4	99.7

moyenne glissante pour calculer des résidus moyennés de la puissance et de la température, r_{m_p} et r_{m_T}, respectivement, à partir des résidus bruts $r_P(k)$ et $r_T(k)$. La moyenne mobile est la somme non pondérée des données sur une fenêtre de n échantillons divisée par n :

$$r_{m_x}(k) = \frac{1}{n} \sum_{i=k-(n-1)}^{k} r_x(i) \tag{IV.6}$$

où $x = \{P, T\}$ désigne la puissance ou la température. Ensuite, nous utilisons μ et σ des signaux moyennés pour calculer deux seuils formant une enveloppe autour de chacun des résidus (Équation IV.7). Ainsi, les seuils du fonctionnement normal sont générés de la manière suivante pour la puissance (les résidus et les seuils moyennés de la température sont obtenus à l'aide de la même formule) :

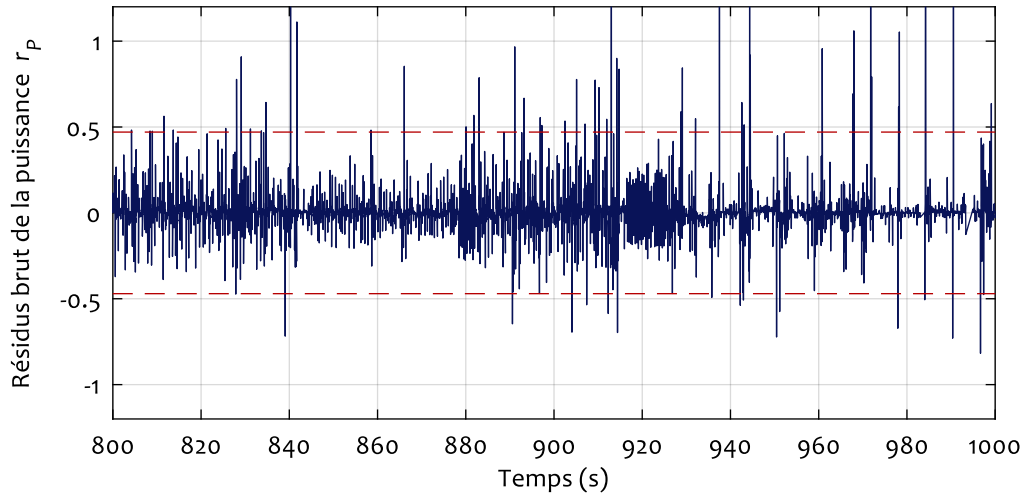
$$\begin{cases} r_{m_p}(k) &= \dfrac{1}{n} \displaystyle\sum_{i=k-(n-1)}^{k} r_P(i) \quad , \\ th^+_{r_p} &= \mu_{r_{m_P}} + 3 \times \sigma_{r_{m_P}} , \\ th^-_{r_p} &= \mu_{r_{m_P}} - 3 \times \sigma_{r_{m_P}} \end{cases} \tag{IV.7}$$

Pour simplifier d'avantage le processus de prise de décision, nous générons les résidus normalisés $R_P(k)$ et $R_T(k)$ à partir des résidus moyennés $r_{m_p}(k)$ et $r_{m_T}(k)$ comme suit :

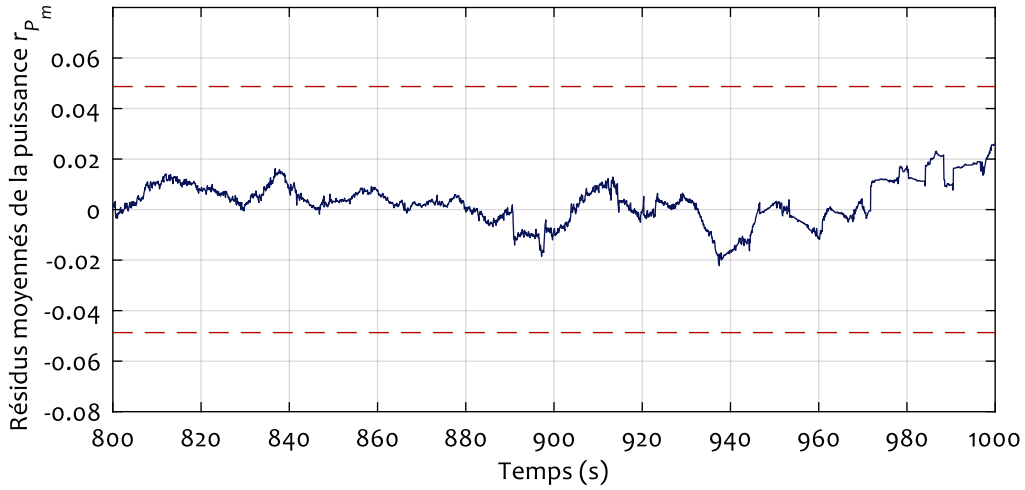
$$\begin{cases} R_x(k) = 1 & \text{Si} \quad \|r_{m_x}(k)\| > \|th_{r_x}\|, \\ R_x(k) = 0 & \text{Sinon} \end{cases} \tag{IV.8}$$

$x = \{P, T\}$ pour la puissance ou la température.

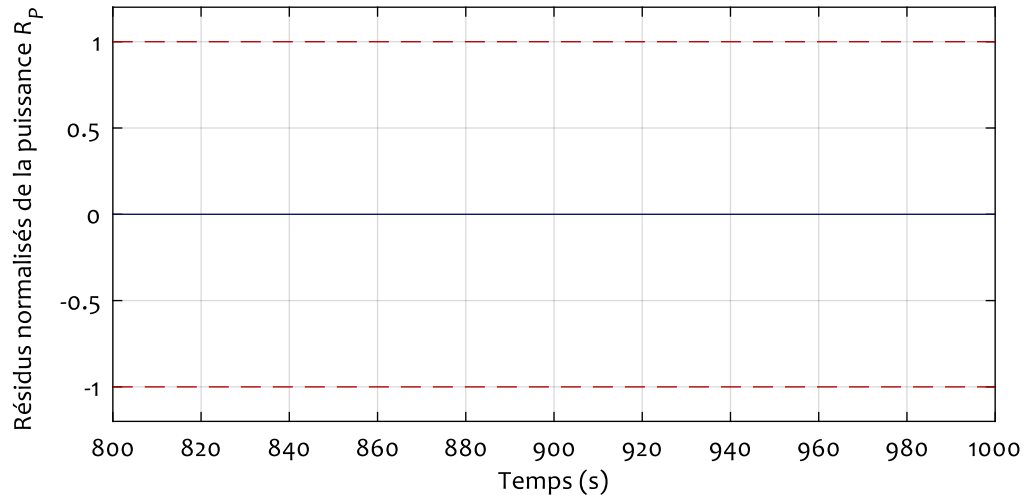
Les résidus bruts $r_P(k)$, moyennés $r_{m_p}(k)$ et normalisés $R_P(k)$ de la puissance, issus des essais en ligne sur l'appareil mobile n° 2 sont donnés dans la Figure IV.4 avec les seuils calculés (en rouge). Dans les résidus bruts, $r_P(k)$, 0.06% des données ne se situent pas dans la plage de fonctionnement normale, générant des fausses alarmes. Le nombre de fausses alarmes est réduit à zéro en utilisant les résidus moyennés $r_{m_p}(k)$. Ainsi, les résidus moyennés améliorent considérablement la performance des résidus normalisés en puissance. La Figure IV.5 montre tous les résidus de température $r_T(k)$, $r_{m_T}(k)$ et $R_T(k)$ ainsi que les seuils (lignes rouges), obtenus par des tests sur l'appareil mobile n° 1, et permet de tirer la même conclusion que pour les résidus de la température.



(a)

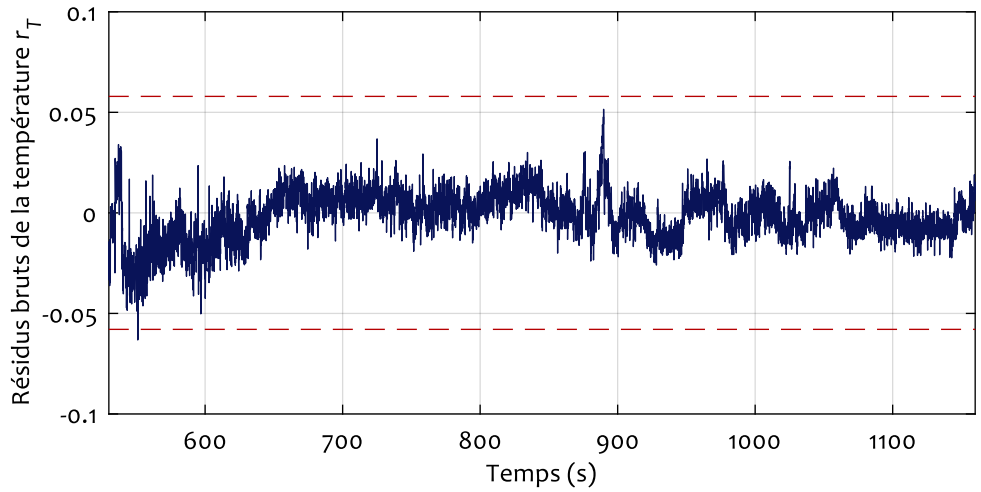


(b)

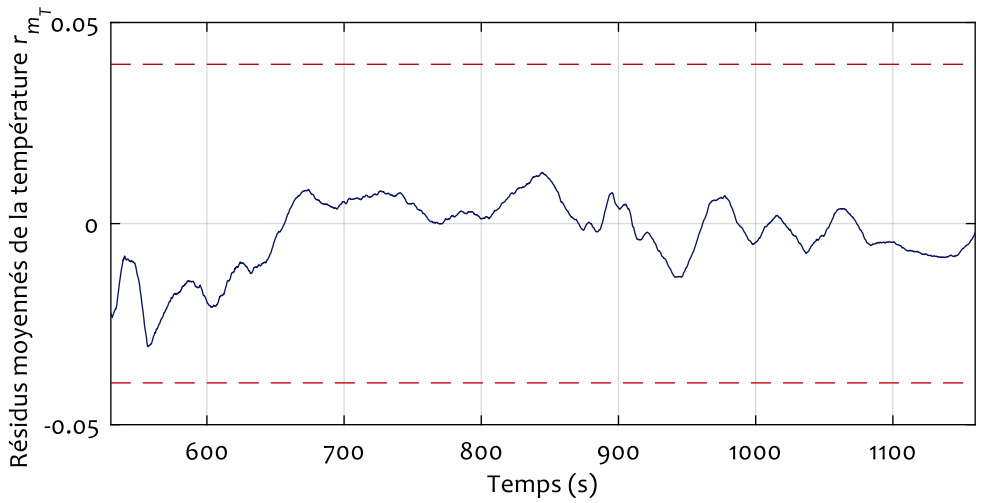


(c)

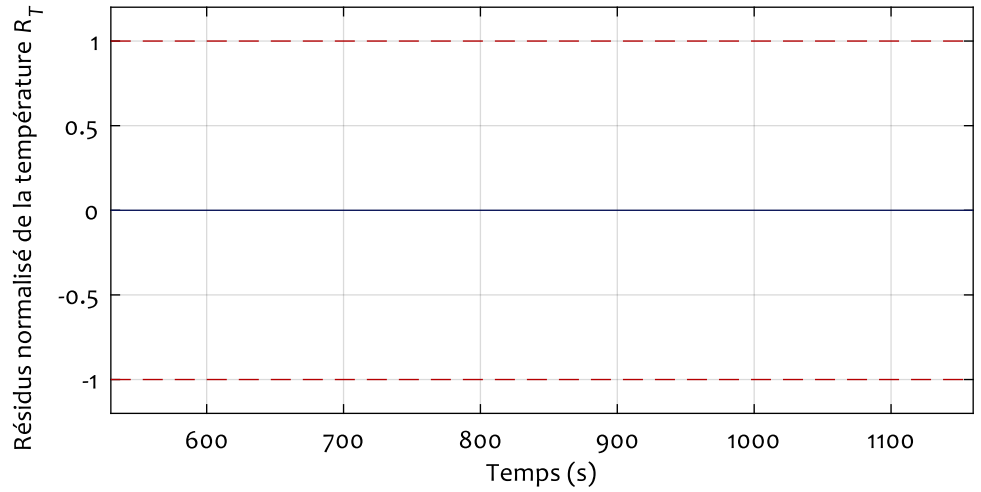
FIGURE IV.4 – Les profils des résidus $r_P(k)$, $r_{m_p}(k)$ et $R_P(k)$ en fonctionnement normal : (a) : résidus bruts $r_P(k)$, (b) : résidus moyennés $r_{m_p}(k)$, (c) : résidus normalisés $R_P(k)$.



(a)



(b)



(c)

FIGURE IV.5 – Les profils des résidus $r_T(k)$, $r_{m_T}(k)$ et $R_T(k)$ en fonctionnement normal : (a) : résidus bruts $r_T(k)$, (b) : résidus moyennés $r_{m_T}(k)$, (c) : résidus normalisés R_T(k).

IV.5.3 Isolation des défauts

Après la détection des défauts vient l'isolation des sous-systèmes défectueux, qui reste un domaine de recherche en cours et en développement. Dans ce travail, l'isolation des défaillances est obtenue grâce à la structure incrémentale et interconnectée du modèle (Figure III.1). En effet, le modèle est constitué de modules interconnectés, et les entrées de chaque module sont les variables de référence estimées par le modèle du sous-système qui le précède. En présence de défauts, lorsqu'un seul résidu réagi, le défaut est donc localisé. Lorsque plusieurs résidus régissent, on remplace en commençant par le module le plus en amont, les entrées de chaque module concerné (entrées qui sont initialement les valeurs de référence générés par le module qui le précède) par les valeurs réelles mesurées par N^3. Si c'est le sous-système considéré qui est en défaut, le résidu va continuer à générer une alarme. S'il n'est pas en défaut, le résidu va revenir à zéro. Cela veut dire que la déviation de la variable à la sortie de ce module est causée par un défaut au niveau des modules qui le précèdent, et qui s'est propagée par lien de causalité vers la sortie du sous-système considéré. On procède ainsi par élimination pour localiser les sous-systèmes en défaut.

IV.6 Test et validation de l'algorithme de surveillance

Afin de valider notre algorithme de surveillance, et tester la sensibilité des résidus aux pannes et aux dégradations, nous proposons trois scénarios de pannes différents avec des défaillances provenant de l'environnement, ou du coté logiciel, ou du coté matériel de l'appareil. Au cours de ce processus d'expérimentation, l'appareil est toujours soumis aux mêmes conditions de validation que dans le chapitre précédent (voir III.7), mais au milieu de ces utilisations et prises de données, l'un des défauts décrits ci-dessous sera introduit.

IV.6.1 Défauts de contrôle

Dans le premier scénario de défaillance, nous introduisons un défaut de fréquence, qui peut correspondre à un bug du gouverneur des fréquences ou à un dysfonctionnement de l'horloge du système dont la fréquence ne correspondrait plus à la charge de travail. Dans ce scénario, le processeur fonctionne normalement au départ, mais à partir d'un instant t_0 bien défini, il opérera sous une fréquence constante qui ne correspondra pas à celle calculée par le gouverneur.

La Figure IV.6 montre l'évolution des fréquences mesurées et estimées sur l'appareil mobile n° 1. À l'instant $t_0 = 46\,\text{s}$, la fréquence mesurée est verrouillée à une fréquence constante qui ne correspond plus à la charge d'entrée. Cette erreur est immédiatement détectée par le résidu brut $r_f(k)$, comme indiqué dans la Figure IV.7a. Par conséquent, le résidu normalisé $R_f(k)$ passe de 0 à 1, générant une alarme (Figure IV.7b).

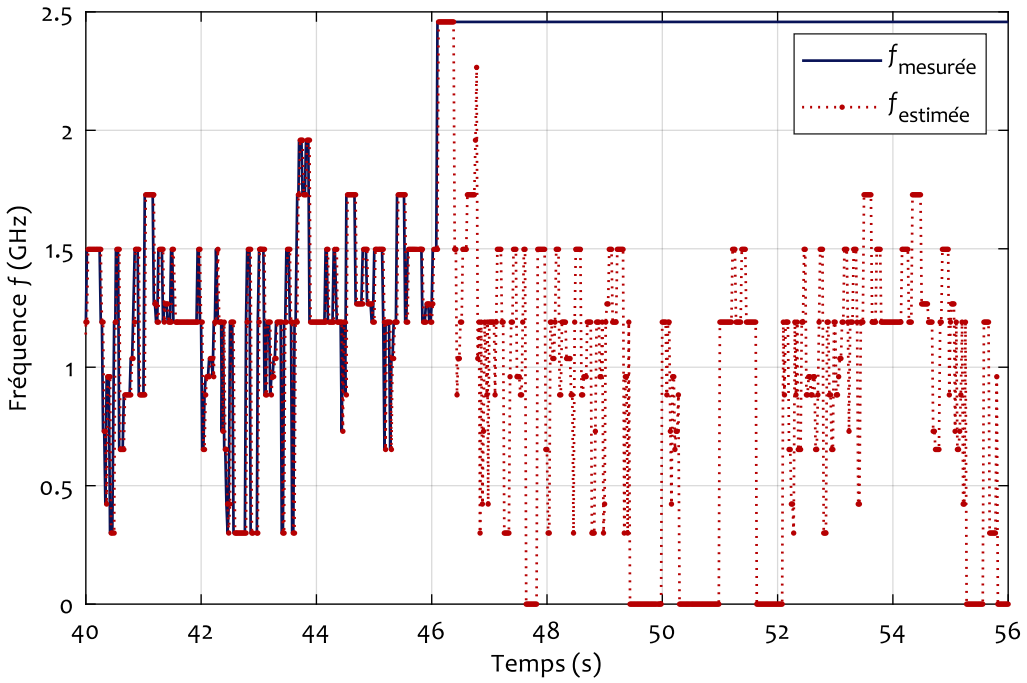


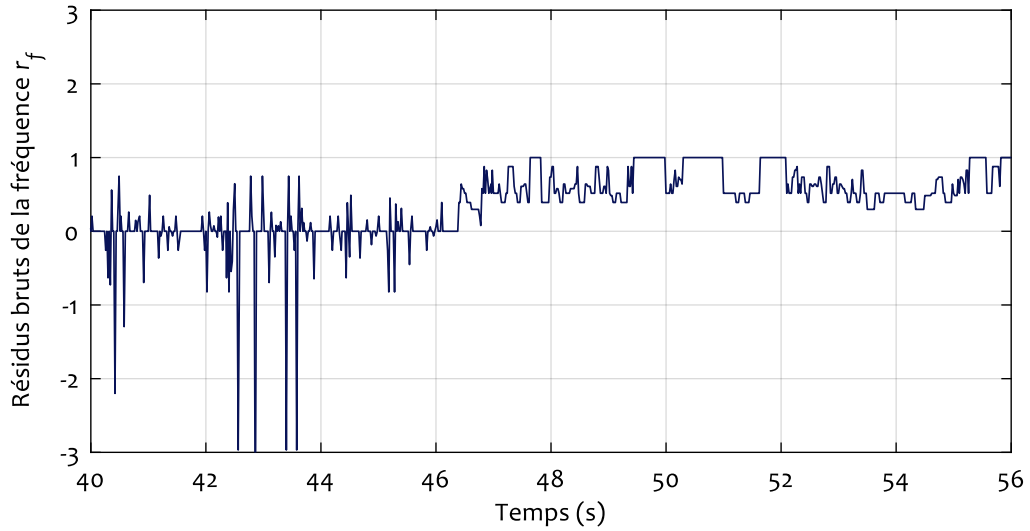
FIGURE IV.6 – La fréquence estimée à partir des charges à l'aide du modèle de fréquences par rapport à la fréquence mesurée bloquée à la valeur maximale autour de $t_0 = 46\,\text{s}$, pour l'appareil mobile n° 1.

IV.6.2 Défauts matériels ou composants

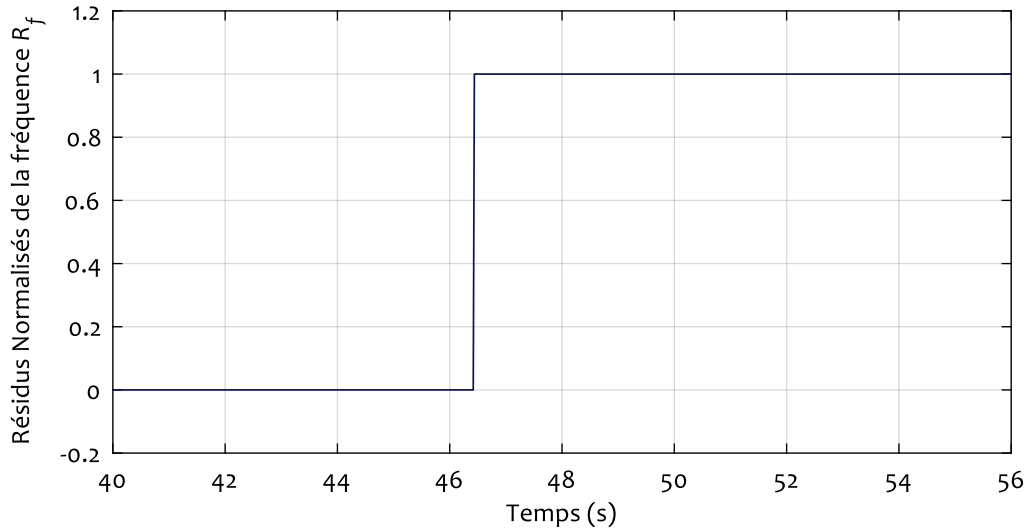
Ce scénario est destiné à simuler un composant défectueux, la présence de corps étrangers sur la puce (comme une accumulation de poussière) ou la modification des caractéristiques de la puce due à l'usure. De tels défauts et dérives provoquent généralement une consommation énergétique accrue.

Comme déjà décrit dans le Paragraphe III.4.1, le modèle NARX est uniquement entraîné pour surveiller la puissance consommée par l'appareil et le SoC. Ainsi, pour simuler la surconsommation provoquée par l'une des raisons susmentionnées, nous avons branché un périphérique externe – une lampe USB à LED sur le système en cours de fonctionnement.

Dans la Figure IV.8, les valeurs mesurées de la puissance consommées de l'appareil mobile n° 2, sont affichées avec les valeurs estimées par le modèle NARX. Cette figure montre une différence notable entre les deux puisssances par rapport aux résultats de la modélisation de puissance (la Figure III.14). Lors du calcul et de l'évaluation des résidus bruts (Figure IV.9a), nous constatons que la plupart des valeurs se situent dans l'enveloppe des seuils. Néanmoins, en utilisant les résidus moyennés (Figure IV.9b), nous constatons que la moyenne des résidus se situe en dehors de l'enveloppe des seuils et, ainsi, une alarme est générée.



(a)



(b)

FIGURE IV.7 – Les profils des résidus r_f (a) et R_f (b) en fonctionnement défectueux. La fréquence est bloquée à la valeur maximale autour de $t = 46\,\text{s}$, le temps auquel une alarme est générée $R_f = 1$.

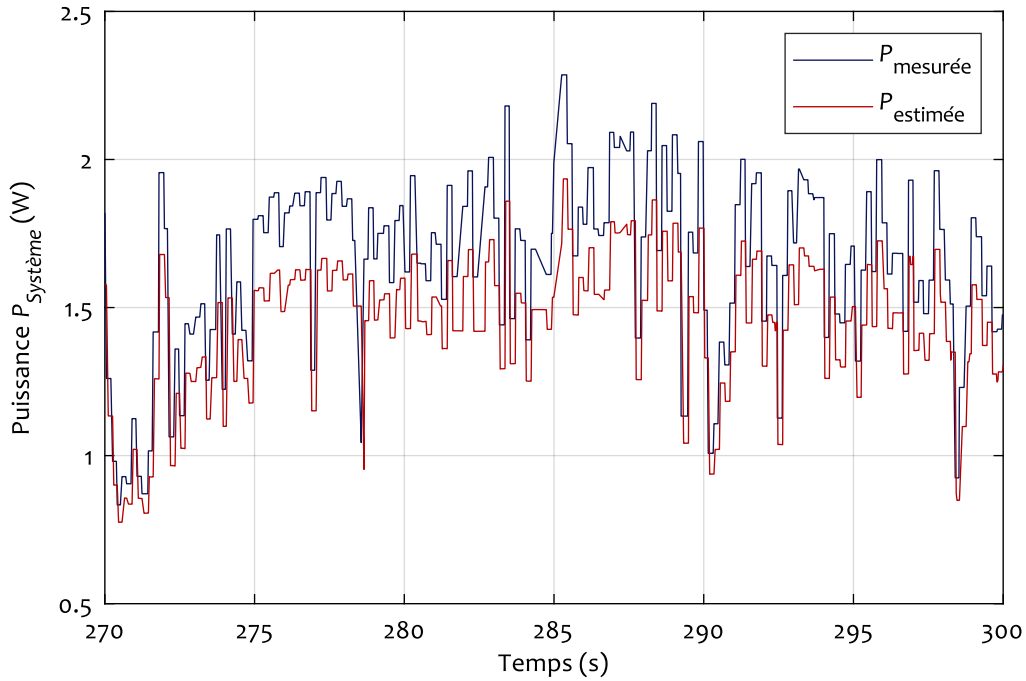
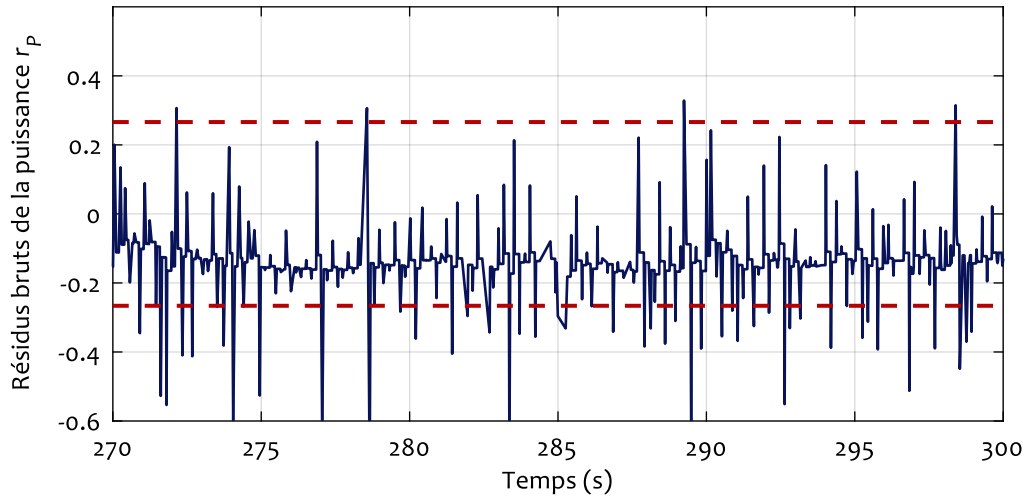
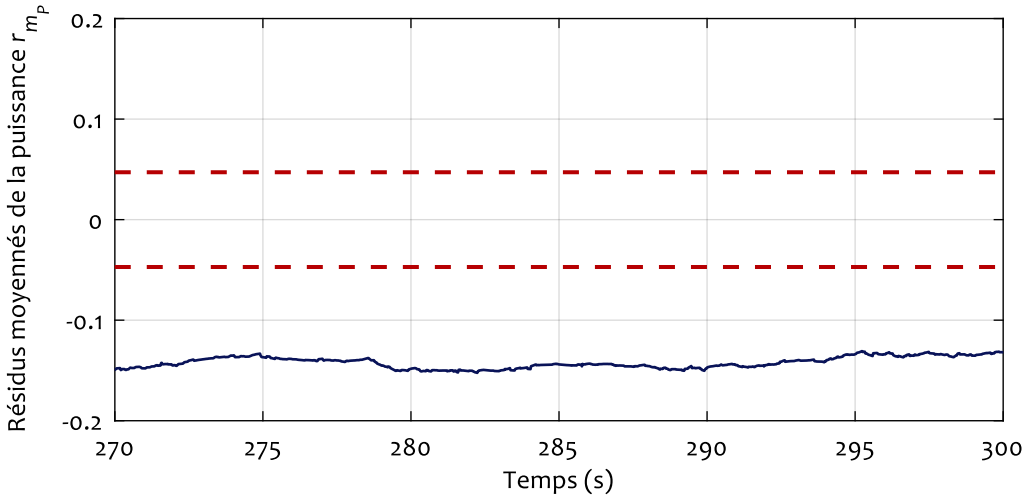


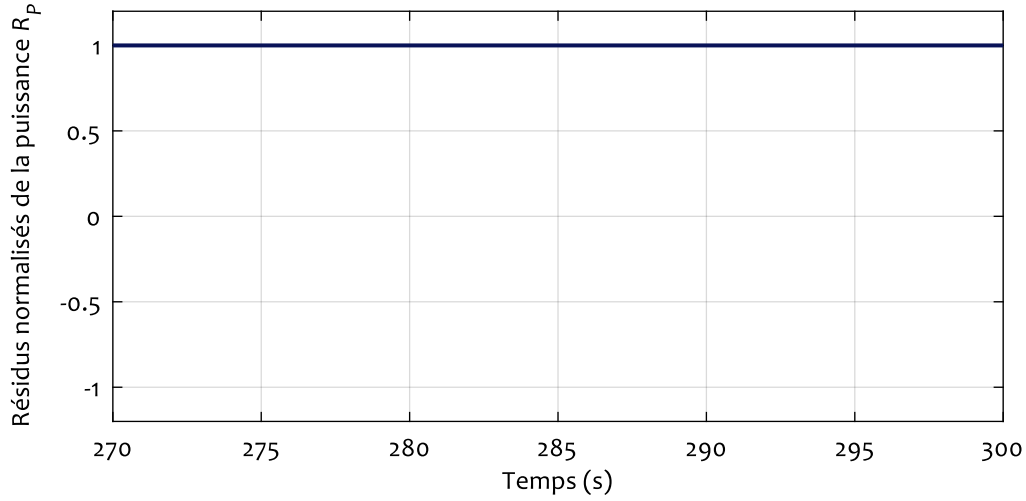
FIGURE IV.8 – Les valeurs de la puissance mesurées comparées aux valeurs estimées pour l'appareil mobile n° 2, au cours de l'expérience de l'introduction d'une lampe LED.



(a)



(b)



(c)

FIGURE IV.9 – L'évolution des résidus $r_P(k)$, $r_{m_p}(k)$ et $R_P(k)$ de l'appareil mobile n° 2 au cours de l'expérience de surveillance de la puissance consommée avec la lampe LED.

IV.6.3 Défauts causés par l'environnement

Les défauts décrits dans les deux paragraphes précédents proviennent de la puce elle-même ou du logiciel utilisé. En revanche, le défaut que nous introduisons dans ce scénario de défaillance est généralement dû à l'environnement du système. Dans cette partie du test, la puce est chauffée, ce qui permet de tester la capacité de l'algorithme de détection de défauts à détecter un échauffement anormal du SoC, pouvant être provoqué par une défaillance du système de refroidissement ou des surtensions ou même des radiations.

Pour simuler de tels défauts, Il fallait chauffer les appareils à une température supérieure à leur température de fonctionnement, sans endommager leurs composants ou compromettre leur intégrité structurelle. Par conséquent, l'appareil mobile n° 1 a été scellé dans un sac étanche et placé dans un bain-marie à une température de 80 °C, tandis que l'appareil mobile n° 2 a été exposée à une lampe de projecteur à 1000 W.

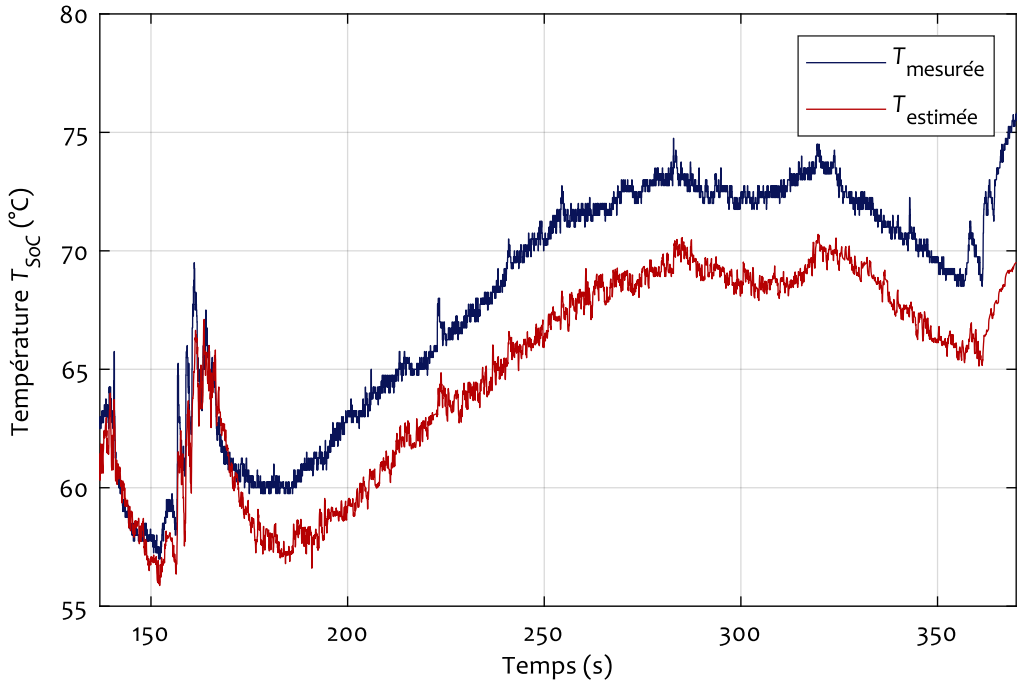
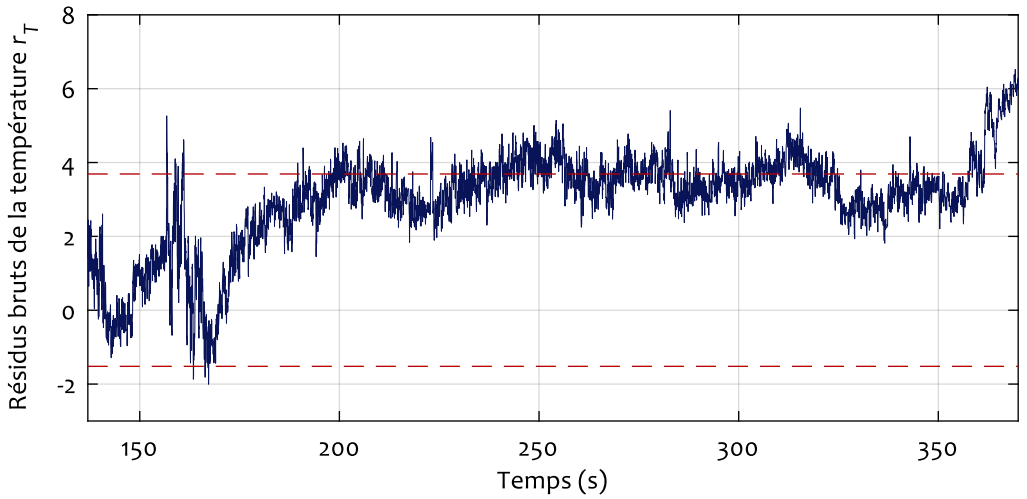
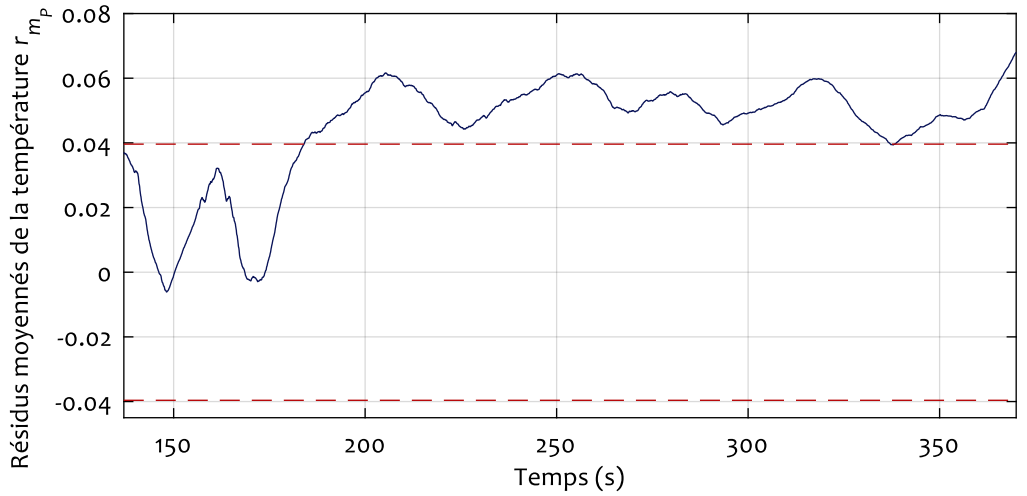


FIGURE IV.10 – Les températures mesurées et estimées du smartphone lorsque celui-ci est mis dans un environnement surchauffé. La divergence entre les deux courbes commence autour de $t = 190$s.

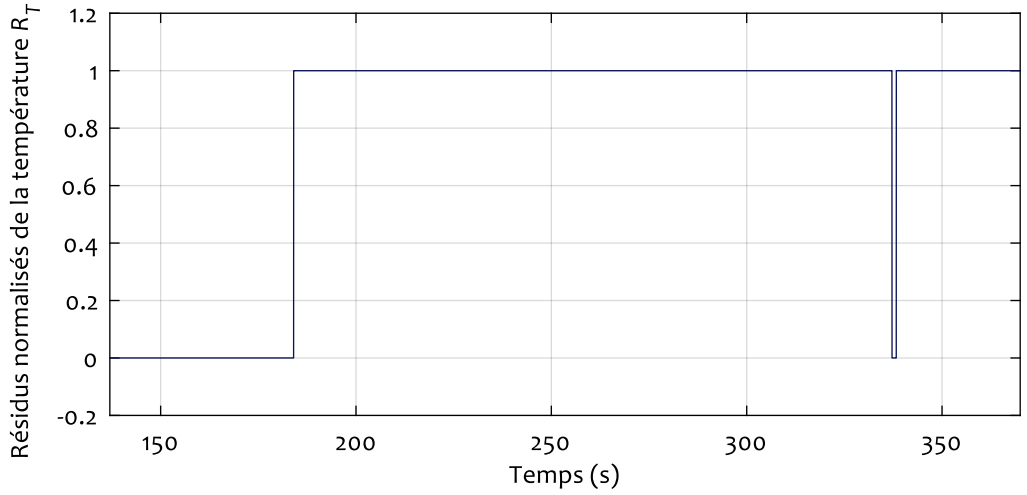
La Figure IV.10 montre l'évolution des températures mesurées et estimées du SoC de l'appareil mobile n° 1 (vu la nature ce test, les résultats reportés ici sont générés hors lignes). La différence entre les deux courbes devient évidente à partir de $t = 180$ s, environ 20 s après l'immersion de l'appareil dans l'eau. La Figure IV.11 montre la réaction des résidus $r_T(k)$, $r_{m_T}(k)$ et $R_T(k)$. La dérive est détectés à l'instant $t = 190$ s, où le résidu $r_{m_T}(k)$ dépasse l'enveloppe de fonctionnement normal. Ensuite, une alarme est générée ($R_T(k)$ passe de 0 à 1).



(a)



(b)



(c)

FIGURE IV.11 – l'évolution des résidus $r_T(k)$, $r_{m_T}(k)$ and $R_T(k)$ de l'appareil mobile n° 1, au cours de l'expérience du bain-marie.

IV.7 Conclusion

Dans ce chapitre, nous avons présenté un algorithme de détection des dérives des caractéristiques des SoC embarqués. L'algorithme est constitué de trois étapes : la génération des indices de dérive, leur évaluation, et enfin la génération d'alarmes.

Dans la première étape, nous avons utilisé le modèle incrémental développé dans le Chapitre III, afin de générer des références, puis les comparer aux mesures en ligne issues du système d'acquisition et calculer des résidus.

Dans la deuxième étape, compte tenu du fait que les mesures sont bruitées et que le modèle présente des incertitudes, notamment des retards d'estimation dûs aux temps de calcul et les bruit de mesures des entrées du modèle, nous avons proposé une méthode d'évaluation des résidus. La méthode proposée considère le résidu comme un signal dont les propriétés (distribution normale des valeurs et homoscédasticité) sont utilisées pour générer des seuils de fonctionnements normale sous la forme d'une enveloppe. Cette évaluation donne lieu à des résidus normalisés. Ces derniers sont des résidus facilement interprétables. Les résultats expérimentaux, obtenus dans différents modes de fonctionnements et sur différents systèmes, montrent l'efficacité de l'approche proposée, et sa capacité à détecter et identifier de façon précoce les dérives des caractéristiques des SoC.

CHAPITRE V

PROTOTYPAGE D'UN SYSTÈME MÉCATRONIQUE POUR LA MODÉLISATION ET LA SURVEILLANCE

V.1 Introduction

Dans les chapitres précédents, nous avons présenté une structure de surveillance des systèmes embarqués basés sur l'architecture ARM. Nous avons commencé par la création du profiler N^3, qui collecte les données caractéristiques des appareils à surveiller (Chapitre II). Ces données sont ensuite envoyées à l'équipement de supervision (Figure V.1). Elles sont utilisées, dans un premier, temps pour générer des estimations de référence pour les variables caractérisant l'état de fonctionnement du SoC grâce au modèle incrémental (voir Chapitre III). Dans un second temps, les lectures et mesures envoyées par le profiler sont comparées avec les références générées par le modèle incrémental, générant des indices de dérives (Chapitre IV). Ces derniers sont utilisés pour détecter et signaler toute dégradation dans l'état de fonctionnement du système.

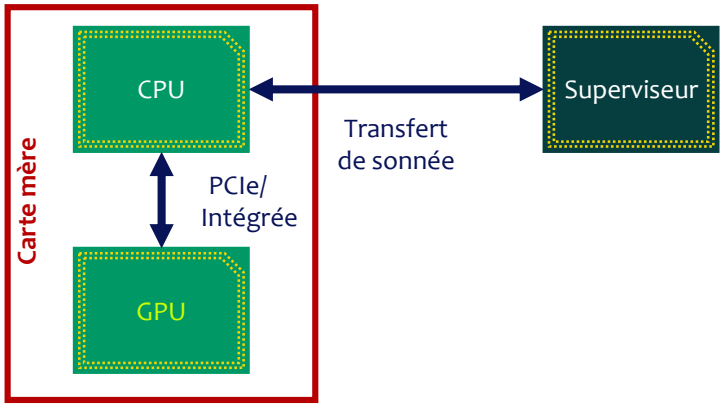


FIGURE V.1 – Le schéma synoptique de la structure de supervision.

L'un des plus grands avantages de cette méthode de modélisation et de surveillance est la facilité relative avec laquelle elle peut être implémentée sur les systèmes embarqués. En effet, la généricité de cette structure nous permet l'adapter à tous les systèmes embarqués ayant l'architecture ARM (et même d'autres systèmes à microprocesseurs). De plus, en choisissant une modélisation basée sur les données, nous n'avons plus qu'à adapter et entraîner les sous-systèmes du modèle incrémental en fonction des systèmes choisis.

Cette facilité de portabilité est une propriété primordiale pour une méthode destinée à une implémentation industrielle, où elle doit s'adapter à des contraintes d'instrumentation et d'utilisation différentes, surtout dans des systèmes critiques pour la sécurité. Dans ce chapitre, nous construisons un prototype pour valider l'approche de modélisation et de surveillance proposée dans ce manuscrit, en testant ses performances sur une carte de développement (certifiée pour une utilisation dans le domaine de l'avionique). La construction du prototype nous permet également de tester la portabilité de nos algorithmes sur un système à caractère industriel, qui servira ainsi de preuve de concept pour le projet MMCD. Un partie des résultats abordés dans ce chapitre a fait l'objet de publication dans [1], [4], et [5].

V.2 Mise en œuvre du prototype

Au cours des précédents chapitres, nous avons utilisé des smartphones pour le test et la validation de notre structure de modélisation et de supervision. L'utilisation de ces appareils est motivée par le fait qu'ils sont dotés de SoC basés sur l'architecture ARM, la même architecture envisagée par les partenaires industriels du projet MMCD. De plus, au cours des échanges préliminaires avec ces partenaires, deux parties différentes ont été distinguées : le système embarqué et le superviseur. La structure du système de supervision donnée dans le Figure V.1 montre bien que le système de supervision sera déployé sur un support indépendant du systèmes embarqué à surveiller. Le support qui hébergera et exécutera les algorithmes développés dans ce travail de thèse, peut être un système à microprocesseur (microcontrôleur, ordinateur, etc.) ou une autre type de système programmable tel qu'un réseau de portes programmables (*Field-Programmable Gate Array*, FPGA) ou un *Application-Specific Integrated Circuit* (ASIC).

V.2.1 Architecture du prototype

En se basant sur ces indications, nous avons établis un prototype de démonstration. Dans ce démonstrateur, nous avons utilisé une carte de développement recommandée par les partenaires industriels en tant que système embarqué à superviser, tandis que pour le superviseur, nous avons opté pour un système à microprocesseur (ordinateur). Une vue d'ensemble de l'architecture du démonstrateur est donnée dans la Figure V.2, où trois parties se distinguent, notamment, la carte de développement, l'afficheur émulant le fonctionnement d'un écran dans un cockpit d'avion, et le superviseur.

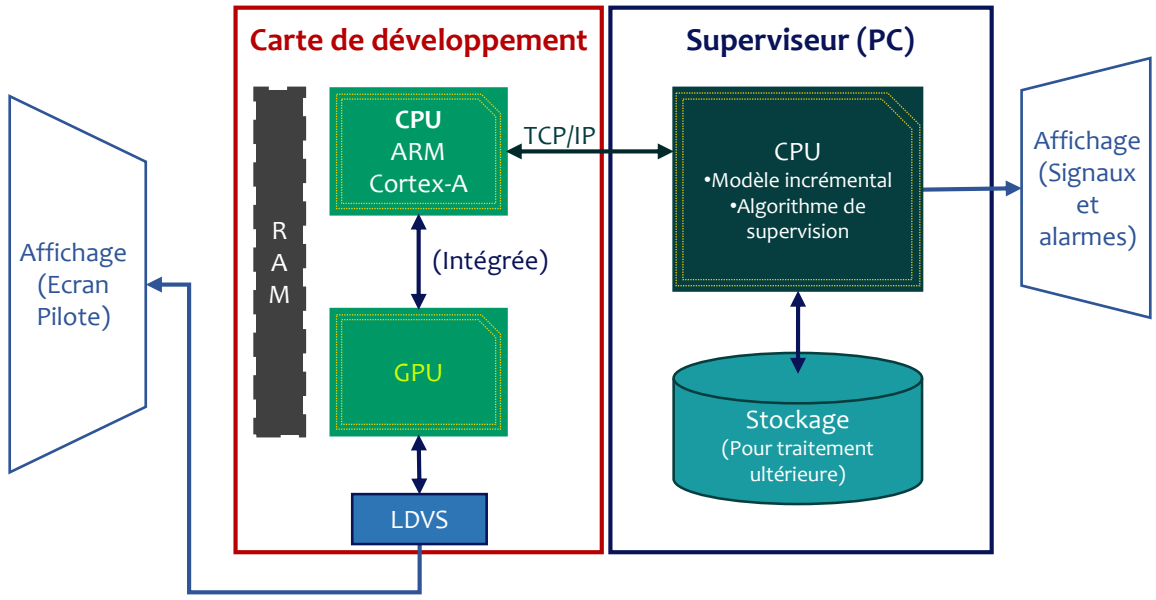


FIGURE V.2 – l'architecture générale du prototype proposé pour le projet MMCD.

La Figure V.2 montre les différents transferts et tâches effectués pendant le fonctionnement. En résumé, dans ce prototype, le profiler N^3 est mis en place sur les deux systèmes (la carte de développement et le PC superviseur), et exécute les tâches suivantes :

1. Sur la carte de développement :
   — l'acquisition des variables caractéristique $(f, V \cdots, T_{SoC})$ chaque $T_s = 20\,\text{ms}$
   — l'envoie des vecteurs de données acquises au superviseur via une connexion TCP/IP
2. Sur le superviseur :
   — la récupération des vecteurs de données envoyés par N^3 sur la carte,
   — La récupération de la mesure de la puissance consommée par la carte mesurée par un multimètre (le système embarqué n'est pas équipé de capteurs de puissance ou des mesures électriques),
   — la vérification des horodatages des mesures,
   — la génération des estimations des variables caractéristiques grâce au modèle de référence,
   — l'envoie des vecteurs des mesures et des estimations pour l'algorithme de surveillance
   — le stockages des mesures et des estimations pour tout traitement ultérieur sous format `csv` sur le périphérique de stockage du PC superviseur.

Ensuite, l'algorithme de surveillance récupère les vecteurs des mesures et ceux des estimations puis procède au traitement des résidus, à la génération des alarmes, et à leur affichage. Une vue d'ensemble du démonstrateur en marche est donnée dans la Figure V.3 avec la carte de développement, l'afficheur, le multimètre et le superviseur. Ils sont distincts et séparés.

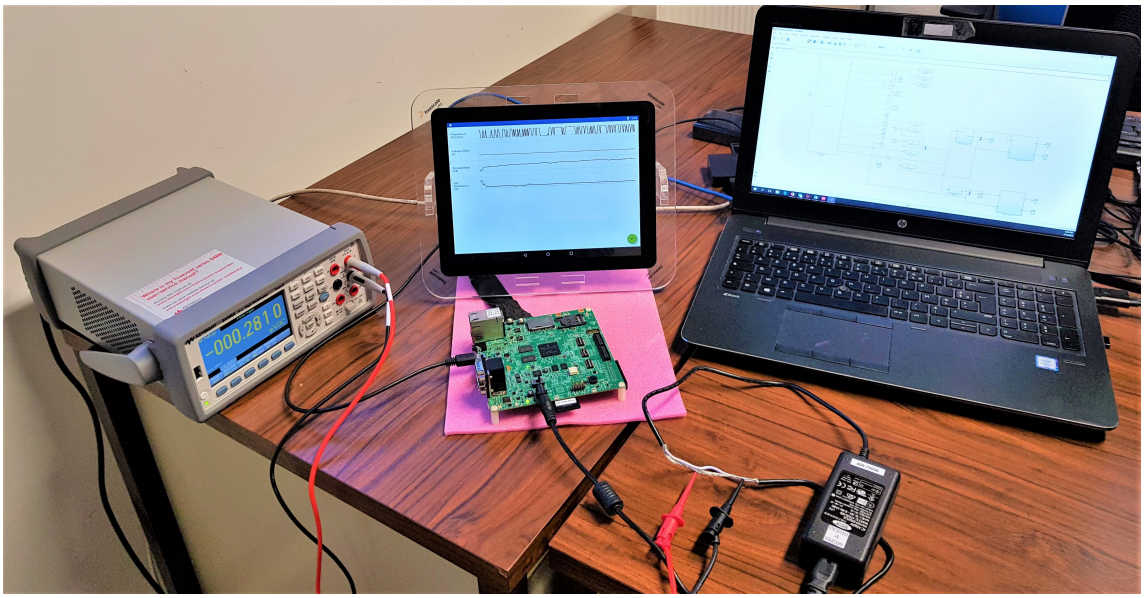


FIGURE V.3 – Une vue d'ensemble sur le prototype avec – de droite à gauche – le PC superviseur, la carte de développement et sont afficheur, et enfin le multimètre pour la mesure de la puissance.

V.2.2 La carte de développement

La carte de développement est la carte *Freescale* [1] *i.XM6 SoloX* [11]. Cette carte est utilisée, en général, pour la conception des systèmes d'exécution et d'affichage de contenus multimédia et d'*infotainment*, notamment pour les véhicules, car elle est certifiée pour des utilisations critiques en termes de sécurité.

La carte fonctionne sous Linux et elle est compatible avec Android 6.0.1. Elle est dotée d'un processeur ARM Cortex-A9 et de 1 GB de RAM [11]. Cette carte est aussi connectée à un écran tactile utilisé à la fois pour simuler un affichage et pour donner ou des commandes. La Table V.1, affiche les caractéristiques pertinentes de la carte par rapport à ce prototype. Cependant, la carte n'est pas équipée de capteurs pour la mesure de la puissance consommée (ni de la tension à ses bornes ou du courant). Mais, puisqu'il s'agit d'un prototype, nous avons donc utilisé un multimètre externe pour la mesure de cette variable.

TABLE V.1 – Les caractéristiques principales de la carte de développement Freescale i.MX6 SoloX [11].

	Carte de développement
OS	Android 6.0.1 (Marshmallow)
SoC • CPU • GPU • RAM	MCIMX6SX 1 cœur CPU + 1 microcontrôleur — 1 GHz ARM Cortex-A9 — 0.2 GHz ARM Cortex-M4 3D : Vivante GC400T et 2D : Vivante GC320 [2] 1 GB
Communication	—
I/O	Touchscreen : HD LCD
Source d'énergie	Branchée en continu

V.3 Modélisation du prototype

Après avoir mis en place le prototype, nous passons, dans cette section, à la construction et la validation de la partie modélisation pour l'estimation des variables caractéristiques du système embarqué, c'est à dire : $f(k), V(k), P_{SoC}(k)$, et $T_{SoC}(k)$. Pour ce faire, nous utilisons le modèle incrémental développé dans le Chapitre III, et nous l'adaptons à la carte de développement.

1. Devenue *NXP*.
2. Les modules Vivantes GC400T et GC320 sont des module de traitement des graphiques 3D et 2D, respectivement. Cependant, ils n'entrent pas sous la définition du GPU établie par Nvidia [222, 223].

V.3.1 Adaptation du modèle incrémental à la carte développement

L'*i.MX SoloX* est un système avec un processeur mono-cœur et un microcontrôleur. Pour notre cas d'utilisation, le microcontrôleur n'était pas nécessaire. D'ailleurs, au cours de nos expérimentation, il n'était pas sollicité et restait éteint. En outre, le module Vivante GC400T/Vivante GC320 n'est pas un GPU au sens moderne du terme mais plutôt un moteur graphique [224], et fonctionne sous des fréquences fixes. Par conséquent, ces deux composants du SoC ne sont pas incorporés dans le modèle, et leur consommation en puissance fera partie de la puissance statique consommée par la carte (voir III.4.1).

Ces hypothèses résultent en un modèle incrémental du SoC plus simple par rapport au cas des deux appareil mobiles. La Figure V.4 montre le schéma synoptique du modèle incrémental pour la carte de développement. Le modèle a toujours la même structure que celle présentée dans le Chapitre III. Toutefois, les dimensions des variables surveillées $f(k)$ et $V(k)$ sont réduite à $n = 1$.

En plus de la structure générale du modèle, la carte de développement et les appareils mobiles partagent aussi le même gouverneur de fréquences (Interactive), le même modèle de tension (Table de correspondance), et le même régulateur thermique (`Monitor`). Comme nous avons déjà présenté et validé ces algorithmes en détails dans le Chapitre III, et pour éviter toute répétition, nous ne détaillerons pas ces algorithmes à nouveau dans ce paragraphe, et nous référons le lecteur à la Section III.3 et au Paragraphe III.7.1 pour plus de détails.

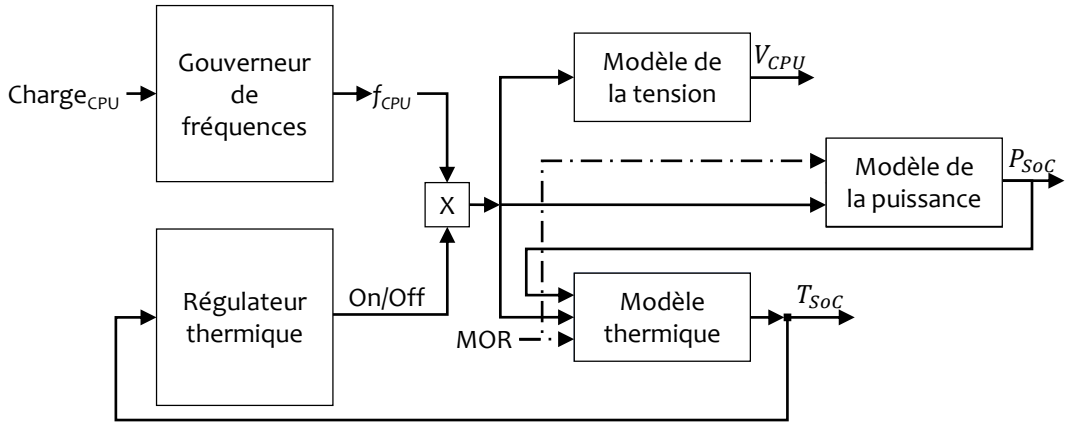


FIGURE V.4 – Le schéma du modèle incrémental adapté au prototype basé sur la carte de développement.

Les deux sous-systèmes restant à adapter dans le modèle incrémental sont les modèles basés sur les données : le modèle de puissance et celui de température. Pour le modèle de puissance, nous utilisons toujours un modèle NARX. Donc, il est similaire à ceux des appareils mobiles, mais il se différencie – naturellement – dans le nombre et la dimension des entrées. En effet, la carte de développement n'a pas le même nombre de composants de communication, d'entrées et de sorties. De plus, elle n'a qu'un processeur mono-cœur. Par conséquent, le nombre d'entrées – et donc de neurones – nécessaires pour

la modélisation de la puissance consommée par le système est considérablement réduit (4 entrées) :

$$u(k) = [f_{CPU}(k), MOR(k), \acute{E}cran_{On/Off}(k), Luminosité(k)] \tag{V.1}$$

La seconde différence que nous notons dans le modèle de la carte de développement, par rapport aux modèles des deux appareils mobiles, est l'ajout d'un neurone connecté à une entrée constante (C=1). Cette entrée constante a été ajoutée empiriquement, car durant le processus de validation, nous avons constaté que l'ajout de cette entrée – et le neurone correspondant – a amélioré la précision du modèle comparé à son absence. La Figure V.5 montre la structure du modèle NARX conçue pour la carte de développement. Le modèle est toujours composé de deux couches, avec 5 neurones dont la fonction d'activation est une sigmoïde dans les couches cachées, et un neurone avec une fonction d'activation linéaire dans la couche de sortie.

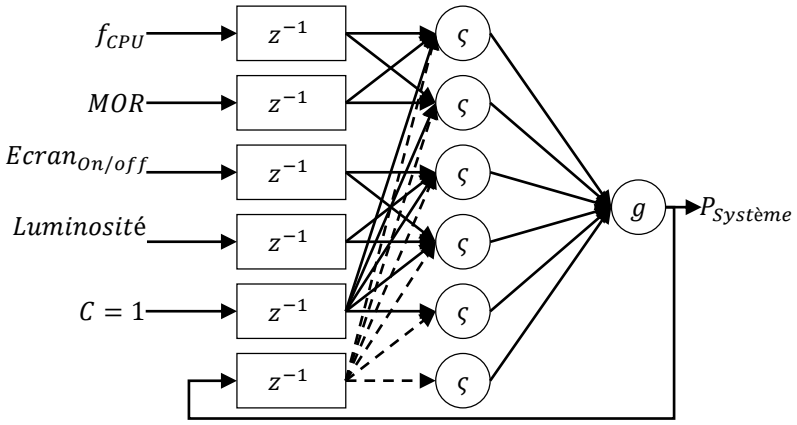


FIGURE V.5 – La structure du modèle NARX de puissance pour la carte de développement.

Le modèle de température reçoit le même traitement que le modèle de puissance. Le modèle est toujours un modèle ARMAX qui a la même structure que les modèles des appareils mobiles. Toutefois, il est adapté à la carte de développement avec la réduction du nombre d'entrées et leurs dimensions : $u(k) = [f_{CPU}(k), MOR(k), P_{SoC}(k)]$ (Figure V.4).

Enfin, la Table V.2 résume les principaux détails de modélisation implémentés dans le modèle incrémental, pour l'adapter à la carte de développement. Parmi ces détails, nous notons surtout le passage des sources de mesure purement logicielles à une combinaisons des traces du système et mesures délivrées par le multilmètre.

TABLE V.2 – Une synthèse de la modélisation adaptée à la carte de développement.

Nom du profiler	N^3	
Niveau de granularité	SoC	
Type de profiling	Logiciel	
Sources des mesures	Mixte • Traces du système • Multimètre	
Modélisation par analyse		
Modélisation de fréquence	• Type • Entrée	• Procédure (Interactive) • $\text{Charge}_{\text{CPU}}$
Modélisation de tension	• Type • Entrée	• Table de correspondance • f_{CPU}
Modélisation du régulateur thermique	• Type • Entrée	• Tout ou rien (`Monitor`) • T_{CPU}
Modélisation en boite noire		
Modélisation de puissance	• Type • Nombre de couche • Nombre de neurones • Retard (TDL) • Entrées	• NARX • 2 • 6 • 1 4 — f_{CPU} — MOR — Statut de l'écran — Luminosité de l'écran
Modélisation de la Température	• Type • Ordre • Retard des entrée (d_u) • Entrées	• ARMAX $[2,2,2] \sim [6,6,6]$ 0 3 — f_{CPU} — MOR — P_{SoC}

V.3.2 Résultats de la modélisation de la carte de développement

Après avoir décrit le modèle incrémental adapté à la carte de développement dans le paragraphe précédent, nous passons maintenant à sa validation. Pour ce faire, nous suivons les même étapes décrites dans le Chapitre III. Ainsi, les résultats de modélisation que nous présentons dans ce paragraphe sont obtenus avec des données enregistrées dans le même scénario d'utilisation décrit dans la Section III.7, dans les mêmes conditions de fonctionnement normal (sans introduction de défauts), pour une première validation hors ligne.

V.3.2.1 Validation des modèles de fréquence et de tension

Les premiers résultats à analyser sont ceux de l'estimation de la fréquence du CPU. Ces derniers sont présentés dans la Figure V.6, qui montre que les estimations suivent parfaitement les lectures du système. Le retard d'estimation (τ), noté pendant la modélisation des appareils mobiles est présent aussi sur la carte de développement. Dans ces expérimentations, le retard maximum est $\tau_{f_{ref}} = 0.149\,\text{s}$. Nous notons aussi que la fréquence ne présente pas autant de changements que dans le cas des appareils mobiles, ceci est dû aux limitations de la carte en termes de puissance de calcul et de performances, car la carte est mono-cœur avec 1 GB de RAM. Par conséquent, pendant 97% de la durée de l'expérience, la carte fonctionnait sous la fréquence maximale du processeur $f_{CPU_{max}} = 996\,\text{MHz}$.

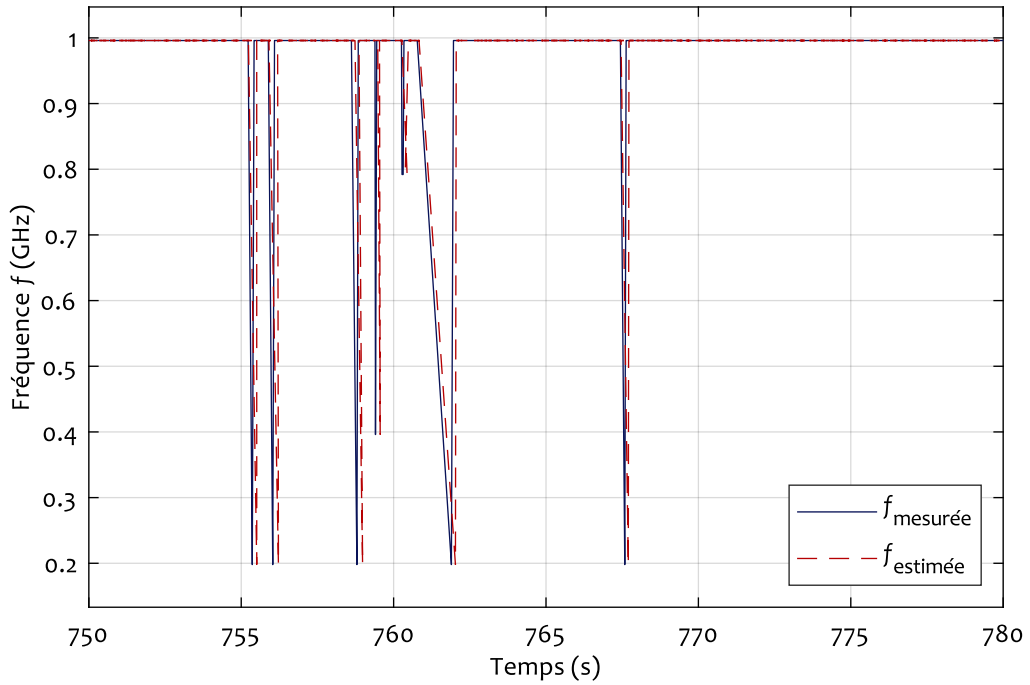


FIGURE V.6 – Les estimations de la fréquence du CPU de la carte de développement comparées aux lectures.

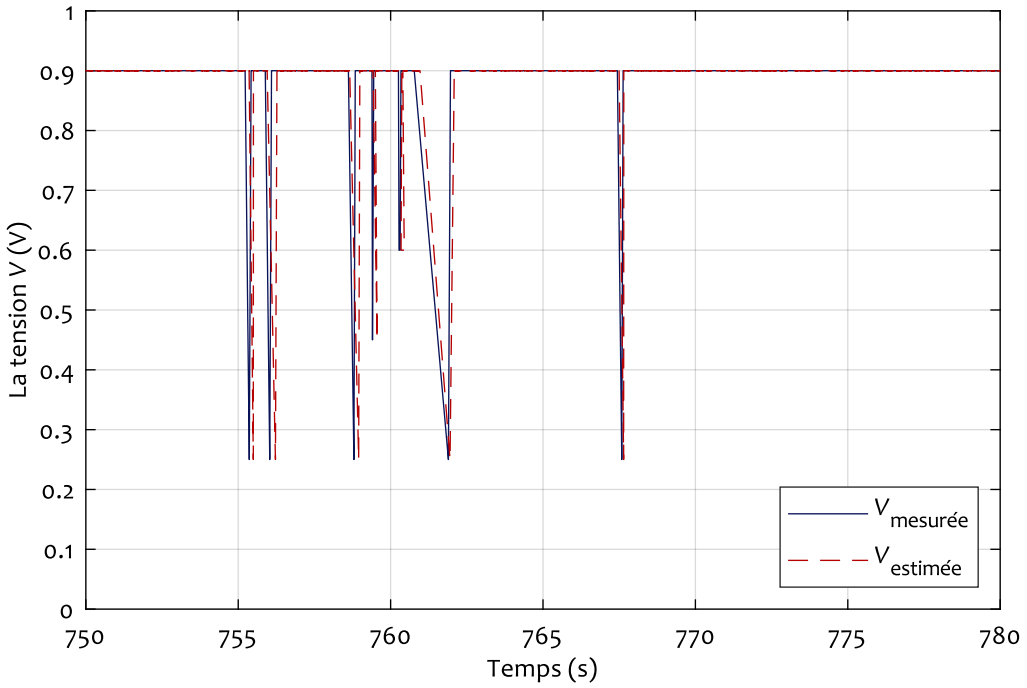


FIGURE V.7 – Les estimations de la tension du CPU de la carte de développement comparées aux lectures.

Les résultats de la modélisation de la tension sont pratiquement identiques à ceux de la fréquence (Figure V.7). Les estimations de la tension du CPU suivent bien les lectures prises sur la carte, et le retard maximum enregistré est $\tau_{V_{ref}} = 0.193\,\text{s}$.

V.3.2.2 Validation du modèle de puissance

Pour la validation du modèle de la puissance sur la carte de développement, les données enregistrées sont divisées en trois datasets sous la même configuration présentée dans le sous-paragraphe III.7.2.1 : 60% pour l'apprentissage, 20% pour la validation, et 20% pour les test (La construction du dataset d'apprentissage est décrite dans le Paragraphe III.4.1). La tables V.3 affiche les résultats obtenus avec le datasets d'entraînement pour différents nombres d'échantillons. Ces résultats valident encore le choix des réseaux de neurones NARX pour la modélisation de la puissance consommée par les systèmes embarqués, et montrent les performances de ce modèle avec des erreurs très faibles (MAPE = 2.8%), et une forte corrélation entre les mesures et les estimations ($R \approx 1$). Comme pour les apareils mobiles, les erreurs relatives d'estimation du modèle NARX sont normalement distribuées (KS-test) et centrées autour de $\mu = 0.0116\,\text{W}$, avec 99.3% de ces erreurs dans l'intervalle $[\mu \pm 3 \times \sigma]$. Le test de ARCH d'Engle confirme l'homoscédasticité de ces erreurs.

La Figure V.8 présente les estimations du modèle NARX comparées aux mesures prise par le multimètre. Elle montre que les estimations du modèle suivent bien les mesures, malgré le bruit que ces dernières présentent. Par conséquent, la modélisation de la puissance pour la carte de développement est validée.

TABLE V.3 – Le temps d'entraînement nécessaire pour différentes taille de dataset, ainsi les résultats obtenus en termes de MAPE, MSE, et R.

Nombre total des échantillons	Temps d'apprentissage (min)	MAPE (%)	MSE	R
$\sim 8 \times 10^4$	4	2.9	4.72×10^{-3}	0.942
$\sim 2 \times 10^5$	6	2.9	4.68×10^{-3}	0.942
$\sim 4 \times 10^5$	10	2.8	3.67×10^{-3}	0.942
$\sim 8 \times 10^5$	12	2.8	3.56×10^{-3}	0.963
$\sim 1.2 \times 10^6$	15	2.8	3.61×10^{-3}	0.961
$\sim 1.8 \times 10^6$	19	2.8	3.54×10^{-3}	0.961

Nous notons aussi qu'en raison du manque de composants de la carte de développement, le nombre d'échantillons nécessaires à l'entraînement du modèle et à l'amélioration des résultats est largement inférieur à celui requis dans le cas d'appareils mobiles (Tables V.3).

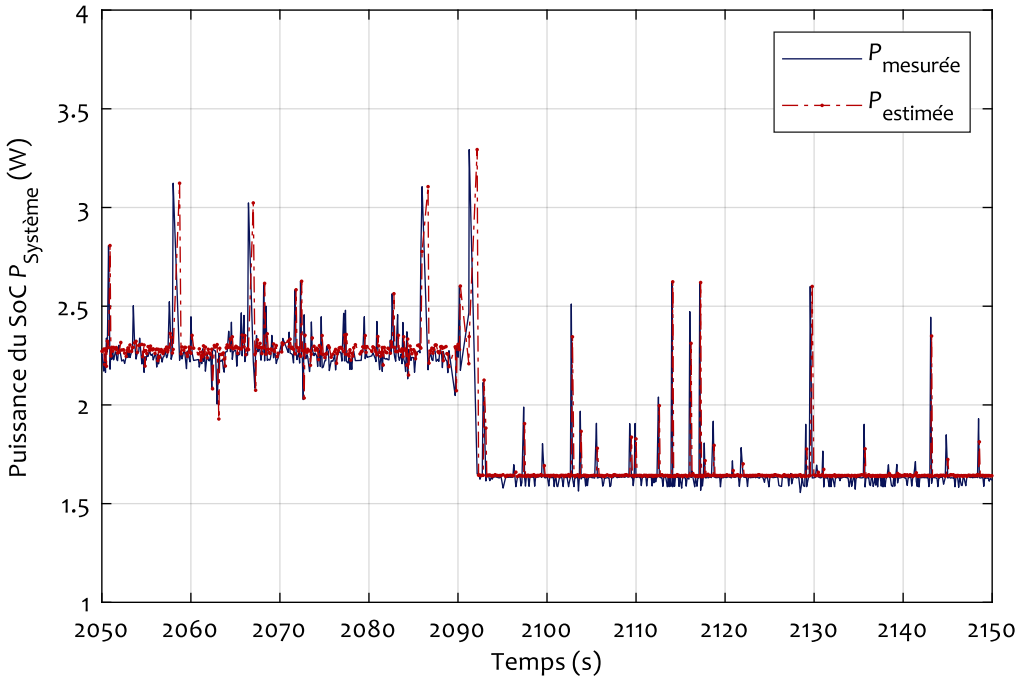


FIGURE V.8 – Les estimations de la puissance comparées aux mesures du multimètre de la carte de développement.

V.3.2.3 Validation du modèle de la Température

Pour la validation des modèles $ARMAX_{RLS}$ et $ARMAX_{LS}$, nous commençons par la répartition des données en deux datasets : Un pour l'identification et un deuxième pour la validation (voir III.7.3). En suite, nous passons au choix des ordres du modèle $[n_a, n_b, n_c]$. Pour ce faire, nous avons testé des modèles $ARMAX_{RLS}$ avec des ordres allant de $[2, 2, 2]$ jusqu'à $[6, 6, 6]$. La Table V.4 affiche quelques

TABLE V.4 – L'évolution de la précision et le temps moyens pour la génération des estimations du modèle $\text{ARMAX}_{\text{RLS}}$ en fonction de ses ordres pour la carte de développement.

Ordres du modèle n_a	n_b	n_c	MAPE (%)	Temps moyen de génération (s)
2	2	2	11.1853	18×10^{-3}
3	3	3	7.9672	18×10^{-3}
4	**4**	**2**	**0.8377**	$\mathbf{\sim 25 \times 10^{-3}}$
4	4	4	1.3757	$\sim 30 \times 10^{-3}$
5	5	2	0.7215	$\sim 65 \times 10^{-3}$
6	6	2	1.3667	$\sim 135 \times 10^{-3}$

des configurations testées, et montre que la précision (en terme de MAPE) augmente – relativement, jusqu'à un certain point – en augmentant les ordres du modèle. Néanmoins, le choix des ordres ne repose pas que sur la précision du modèle, mais aussi sur le temps nécessaire au modèle pour la génération d'une estimation sur la combinaison système-superviseur. Par conséquent, l'ensemble des ordres qui satisfait le mieux nos contraintes de précision et de performances est $[4, 4, 2]$.

Une fois les ordres optimaux pour le modèle choisis, l'étape suivante pour la validation du modèle $\text{ARMAX}_{\text{RLS}}$ est le choix du nombre d'échantillons pour l'identification des paramètres. En théorie, l'ensemble des données d'entraînement idéal, devrait contenir des données représentant toutes les informations possibles sur le système. Toutefois, dans la pratique, les informations contenues dans le dataset d'entraînement sont limitées par la taille de ce dernier et les informations contenues dans ces échantillons. La Table V.5 montre comment la précision des estimations du modèle – en termes de MAPE – augmente avec le nombre d'échantillons. Cependant, cela montre également que cette augmentation des précisions n'est pas absolue et que la précision peut diminuer à partir un certain ensemble d'ordre, même en augmentant du nombre d'échantillons, à cause des irrégularités pouvant survenir dans les échantillons. Ceci est également montré dans les comparaisons présentées des estimations par rapport au mesures dans la Figure V.9 et les erreurs relatives d'estimation dans la Figure V.10. Par conséquent, pour choisir le meilleur modèle pendant l'identification (dans ce cas, $n = 3000$), nous calculons le MSE de validation à chaque m échantillon ($m = 500$ dans ce cas d'étude). Si le MSE est amélioré, les paramètres du modèle sont mis à jours, sinon l'ancien modèle du système est retenu.

Enfin, nous comparons les résultats des deux d'approches de calcul des paramètres : les moindres carrés étendus et les moindres carrés récursives, avec les modèles ARMAX_{LS} et $\text{ARMAX}_{\text{RLS}}$. La Figure V.11 montre une comparaison entre les mesures du système et les estimations des deux modèles ARMAX_{LS} et $\text{ARMAX}_{\text{RLS}}$, qui sont proches des mesures. Les résultats des performances des deux modèles sont affichés dans la Table V.6. Ces résultats montrent un léger avantage pour le modèle $\text{ARMAX}_{\text{RLS}}$ avec un meilleur MAE. Toutefois, le temps nécessaire pour que le modèle $\text{ARMAX}_{\text{RLS}}$ génère une estimation est en moyenne égale à 25×10^{-3} s même hors-ligne. Ceci est dû au capacité

TABLE V.5 – L'influence du nombre d'échantillons sur la précision du modèle $ARMAX_{RLS}$.

Nombre d'échantillons	MAE (°C)	MAPE (%)
500	64.34	169.1200
1000	57.77	151.7027
1500	52.46	137.6220
2000	0.92	2.3688
2500	0.64	1.5077
3000	**0.34**	**0.8377**
3500	0.65	1.5100
4000	0.57	1.5005

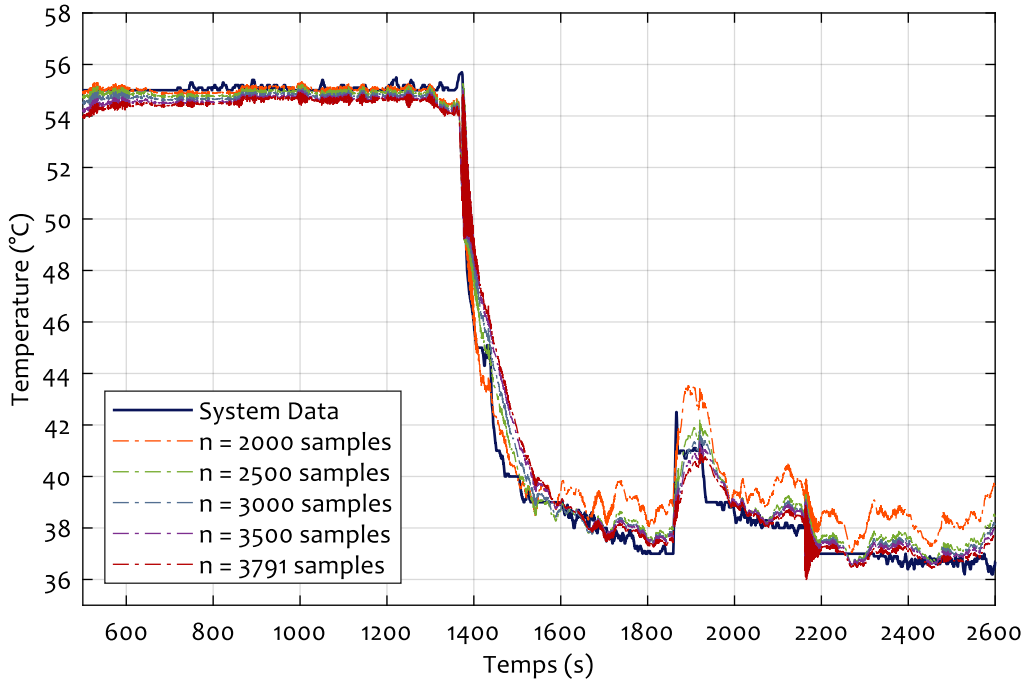


FIGURE V.9 – Les éstimations générées par plusieurs modèles $ARMAX_{RLS}$ avec différentes tailles de dataset d'entraînement comparées aux mesures du système.

de calcul limité de la carte de développement, comparée à celles des appareils mobiles où nous avons fini par utilisé $ARMAX_{RLS}$. Ainsi, le temps d'échantillonage devient le critère pertinent pour le choix.

Le modèle $ARMAX_{LS}$, en revanche, satisfait le temps d'échantillonage nécessaire pour le fonctionnement du modèle incrémental, tout en ayant une excellente précision (MAE = 0.56 °C). Par conséquent, c'est le modèle que nous adoptons pour la surveillance en ligne dans le cas de la carte du développement, et donc la suite de ce travail.

Les valeurs des erreurs relatives du modèle $ARMAX_{LS}$ (et aussi $ARMAX_{RLS}$) sont affichées sur la Figure V.12. Le test de ARCH d'Engle affirme l'homoscédasticité des erreurs, et le test KS confirme qu'elles sont normalement distribuées autour de la moyenne $\mu = -1.9057 \times 10^{-4}$ (Figure V.13). De

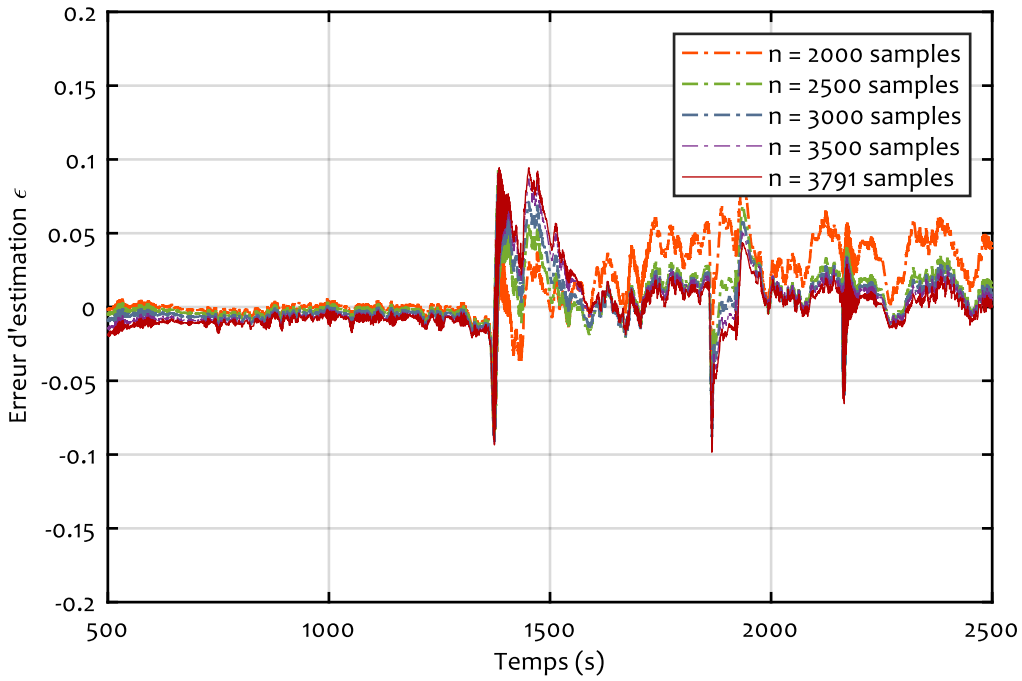


FIGURE V.10 – Les erreurs relatives d'estimations de la température $\varepsilon_T(k) = \dfrac{\vartheta(k) - \vartheta(\hat{k})}{\vartheta(k)}$ générées par plusieurs modèles $\text{ARMAX}_{\text{RLS}}$ avec différentes tailles de dataset d'entraînement comparées aux mesures du système.

TABLE V.6 – Une comparaison entre la précision et le temps nécessaire pour la génération des estimation des modèles $\text{ARMAX}_{\text{RLS}}$ et ARMAX_{LS}.

Modèle	Taille du dataset d'entraînement	MAE (°C)	MAPE (%)	Temps moyen de génération (s)
$\text{ARMAX}_{\text{RLS}}$ (Hors-ligne)	3000	0.34	0.8377	$\sim 25 \times 10^{-3}$
ARMAX_{LS} (Hors-ligne)	4000	0.52	1.1658	1×10^{-3}
ARMAX_{LS} (En-ligne)	**4000**	**0.56**	**1.3757**	$\mathbf{\sim 18 \times 10^{-3}}$

plus, 99.43% des valeurs de ces erreurs sont dans l'intervalle $[\mu \pm 3 \times \sigma]$. Ainsi, la modélisation de la température pour le prototype est validée.

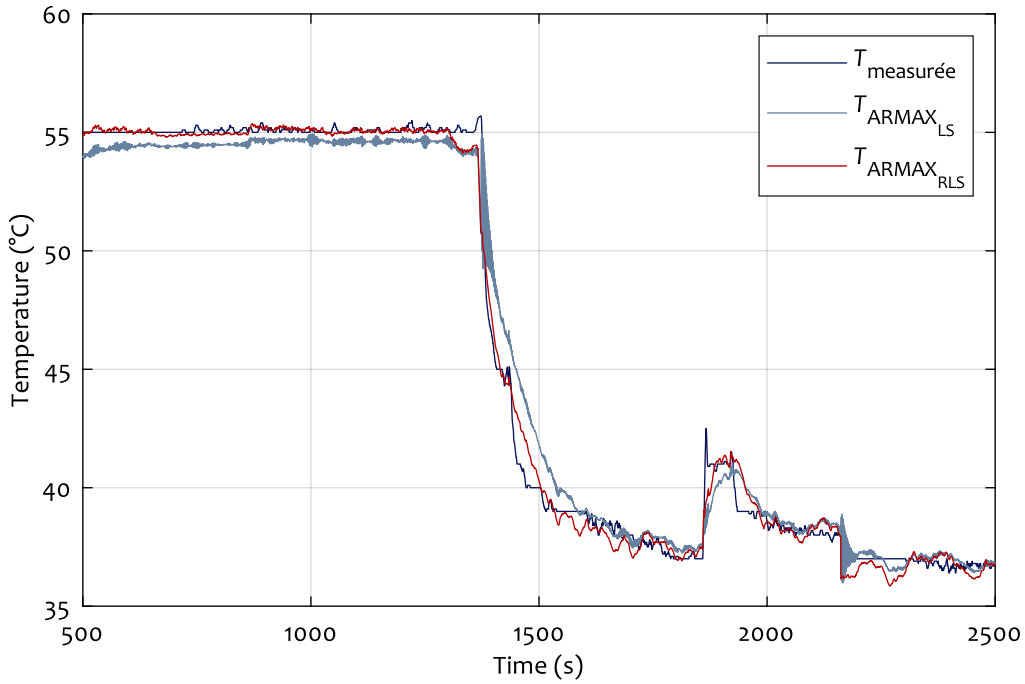


FIGURE V.11 – Les estimations générées par les modèles $\text{ARMAX}_{\text{RLS}}$ et ARMAX_{LS} par rapport aux mesures du système.

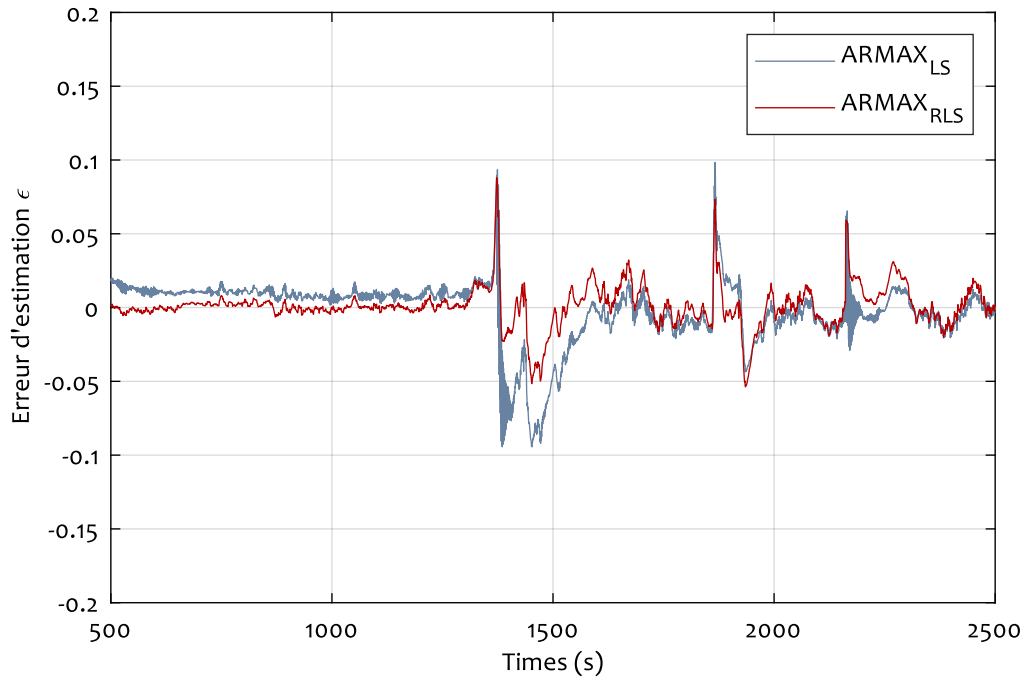


FIGURE V.12 – Les erreurs relatives d'estimations de la température $\varepsilon_T(k) = \dfrac{\vartheta(k) - \vartheta(\hat{k})}{\vartheta(k)}$ générées par les modèles $\text{ARMAX}_{\text{RLS}}$ et ARMAX_{LS}.

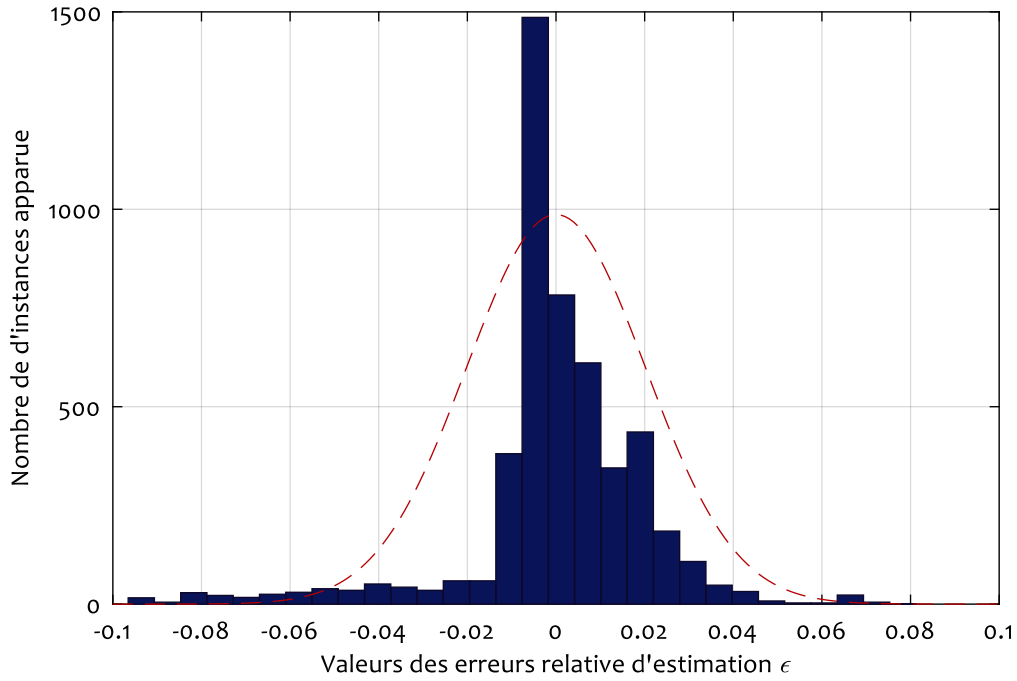


FIGURE V.13 – L'histogramme des erreurs relatives d'estimation du modèle de $ARMAX_{LS}$ en bleu, avec le profil de la distribution normale correspondante en rouge.

V.4 Test du prototype

Dans le Chapitre IV, nous avons introduit et validé notre algorithme de surveillance pour les systèmes embarqués. Ensuite, dans la dernière partie du même chapitre, nous avons testé la sensibilité de cet algorithme face à des défauts dans multiples scénarios (voir IV.6). Ces tests étaient indicatifs sur la validité de notre approche et à sa capacité pour la détection de plusieurs types de défauts. Toutefois, elle a été réalisé sur des systèmes commercials et déjà pré-emballés avec tout l'équipement nécessaire pour l'évacuation de la chaleur et les circuits de régulation de puissance nécessaires. Plus important encore, ces systèmes ne sont pas certifiés pour des applications critiques à la sécurité. Par conséquent, nous avons mis en place un prototype avec un système certifié pour les applications critiques à la sécurité, et dans cette section, nous procédons à la validation de l'algorithme de surveillance, puis le test de ce dernier avec la reproduction de deux scénarios défaillants.

Une vidéo de démonstration du processus de validation et du test de l'algorithme de surveillance a été filmée et mise ligne sur les archives ouvertes *HAL* de l'université d'Aix-Marseille [7]. Cette vidéo et à la fois un résumée de cette partie et aussi une validation visuelle du prototype utilisé dans ce travail.

V.4.1 Validation sous le fonctionnement normal

Les résultats et la validation des modèles présentés dans la Section V.3 sont obtenus hors-ligne. Ceci est nécessaire pour l'entraînement, la validation, éventuellement la comparaison des modèles,

mais ne garanti pas la capacité de toute la structure à générer des estimations précises, détecter les défauts, ou faire face au difficultés qui pourraient survenir en raison des performances limité de la carte de développement. Seulement un test en ligne simulant des conditions de travail normale pourrait confirmer la validité de cette preuve de concept. Ainsi, nous avons mis le prototype en marche, et lancé le processus de surveillance tout en simulant une utilisation normale (Figure V.14).

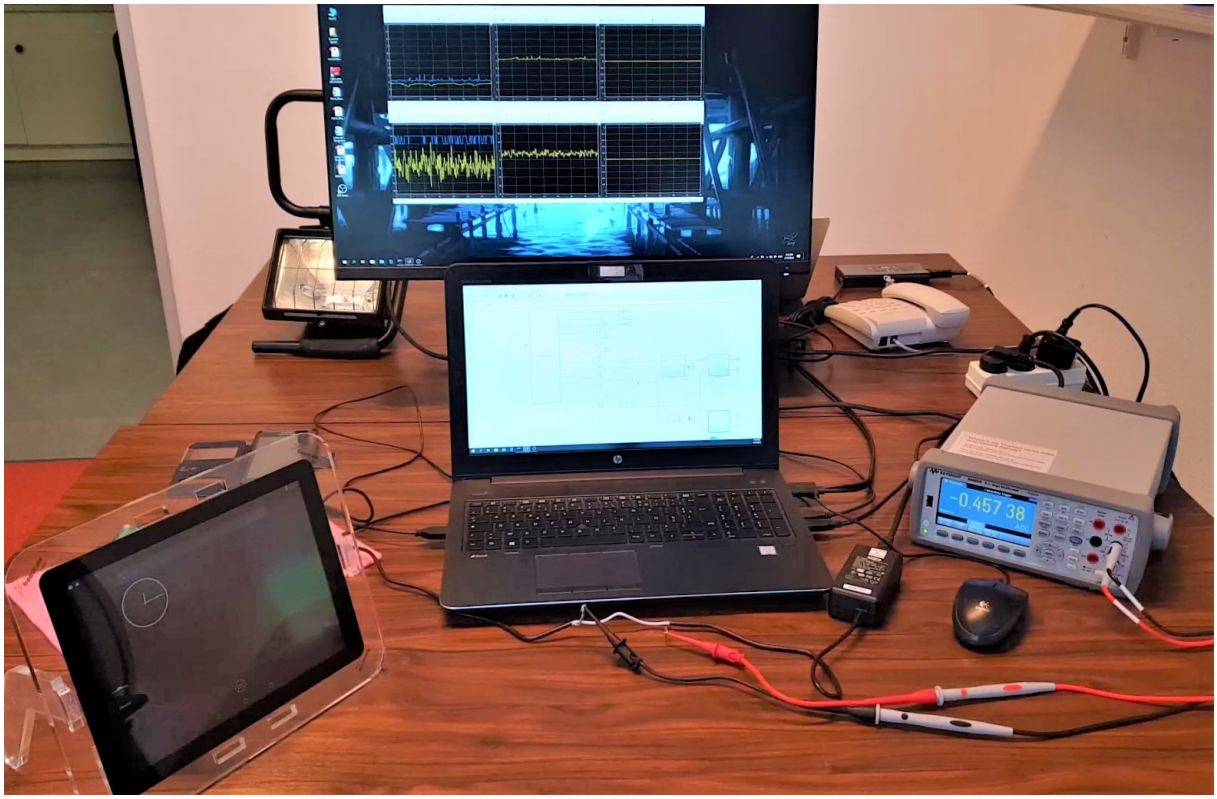
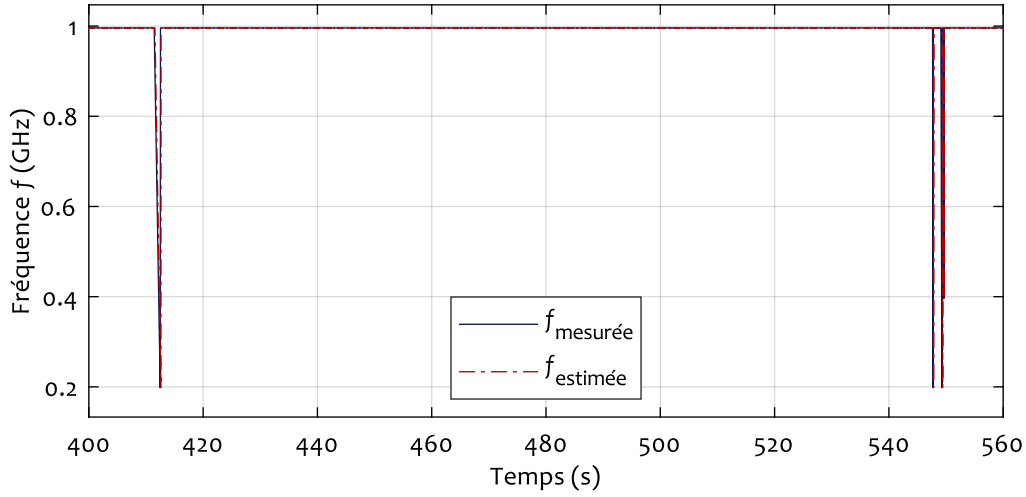


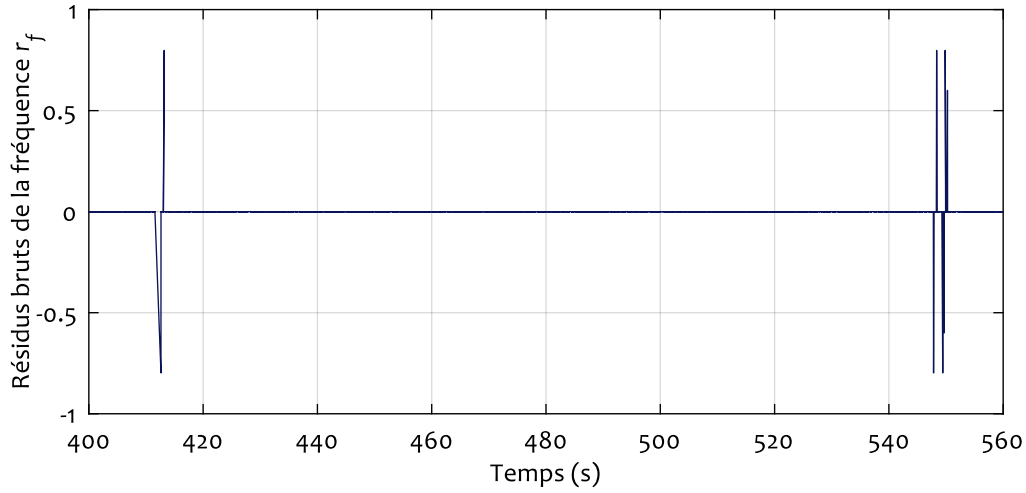
FIGURE V.14 – Une vue d'ensemble du prototype en marche.

Les figures V.15a, V.16, et V.18 montrent l'évolution de trois des variables caractéristiques (f_{CPU}, P_{SoC}, et T_{SoC}), pendant une utilisation normale de 15 minutes de la carte de développement. En premier lieu, la Figure V.15 affiche une comparaison entre les fréquences estimées et celle mesurées (V.15a), les résidus bruts r_f (V.15b) et les résidus normalisés R_f (V.15c). Comme pour le cas hors-ligne, pendant le test en ligne, les estimations en-ligne suivent bien les mesures avec un retard maximal $\tau_{f_{max}} = 0.201\,\text{s}$. Ceci a donné lieu a 131 fausses alarmes de durée maximale égale à $0.114\,\text{s}$ sur environs 45000 échantillons ($\sim 0.003\%$). Une fois $\tau_{f_{ref}}$ a été adapté à la valeur $\tau_{f_{max}}$, ces fausses alarmes ont été réduites à 0 (Figure IV.2b). La modélisation de la tension donne des résultats identiques. Ainsi, les deux sous-systèmes sont validés pendant l'utilisation en ligne.

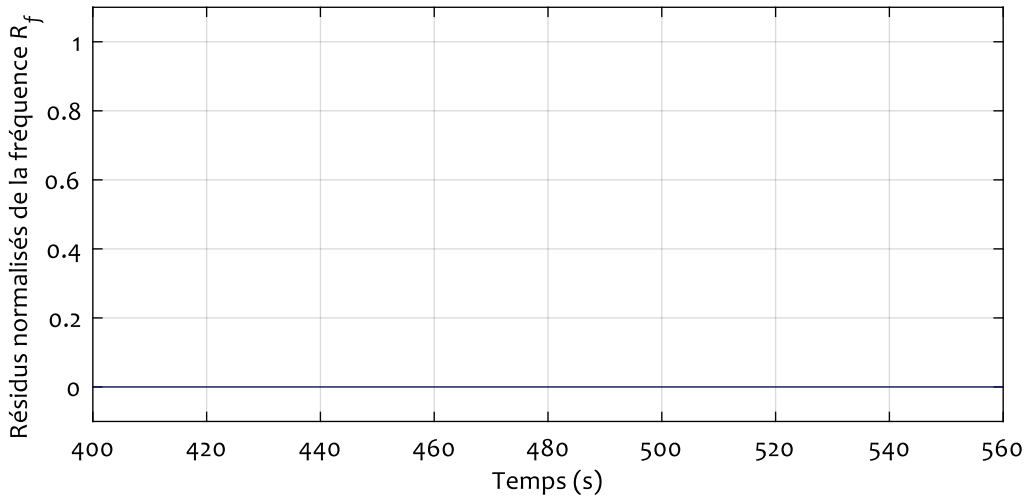
Ensuite, les estimations et les mesures de la puissance consommées par le SoC pendant le test d'utilisation normale en ligne sont affichées dans la Figure V.16. Les résultats obtenus pendant ce test sont très satisfaisants avec un MAPE égale à 2.88%, est très proche de la validation hors-ligne. En outre, malgré les performances limités de la carte du prototype, le temps d'échantillonnage est respecté. De plus, la Figure V.17 affiches les trois résidus de puissance et montre l'efficacité de l'algorithme



(a)



(b)



(c)

FIGURE V.15 – Les valeurs des fréquences estimées et mesurées avec les évolutions des résidus $r_f(k)$, $R_f(k)$ du prototype, au cours du fonctionnement normal. (a) : Les fréquences mesurées contre les fréquences estimées. (b) : Les résidus bruts de la fréquence. (c) : Les résidus normalisés de la fréquence.

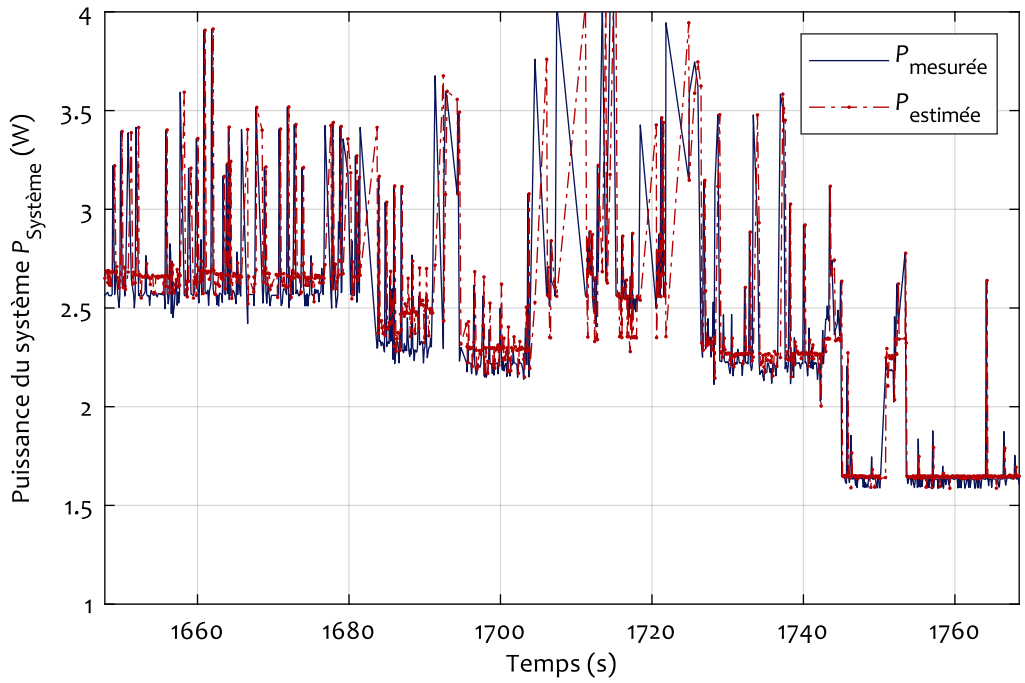
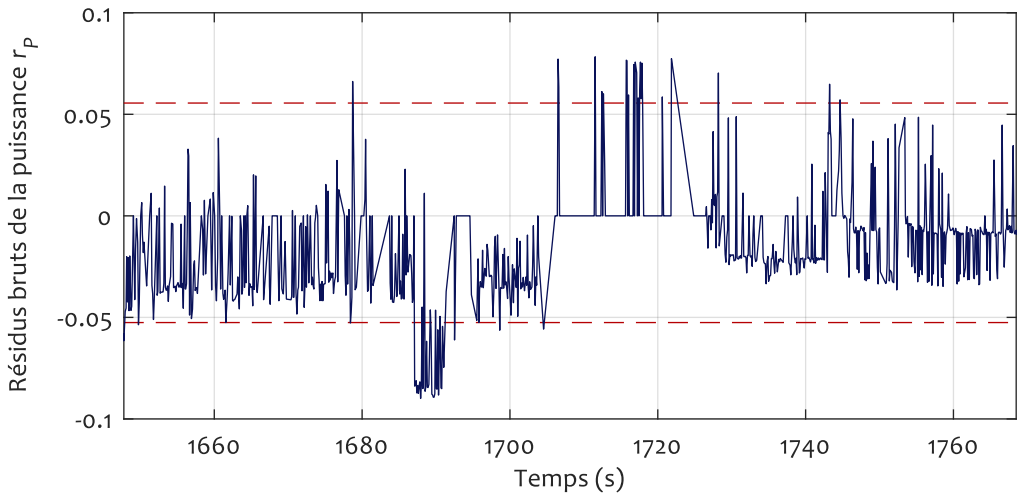


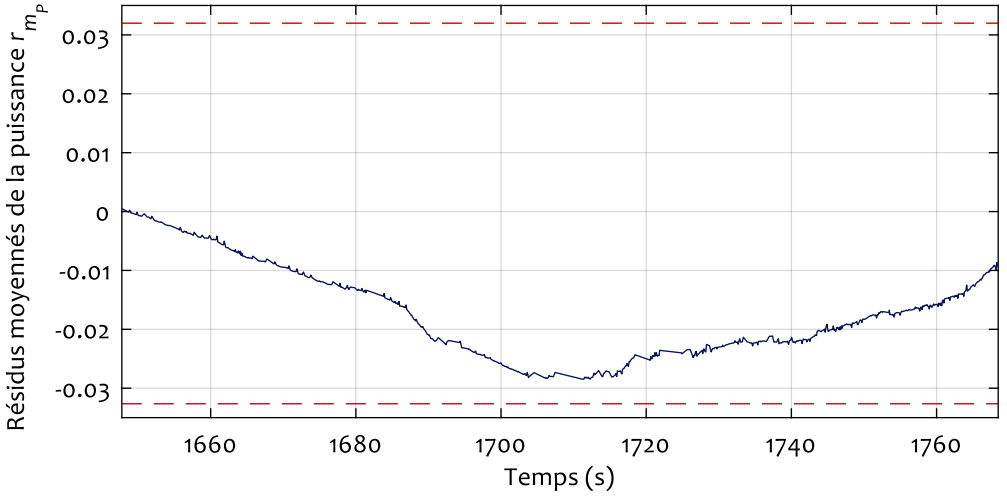
FIGURE V.16 – Les estimations de la puissance comparées aux mesures du multimètre pour la carte de développement, pendant le fonctionnement normal.

de surveillance à éviter les fausses alarmes (Figure V.17c) en filtrant les résidus bruts (Figure V.17a), qui sont assez bruités. Par conséquent, la modélisation est la surveillance de la puissance sont aussi validés pendant l'utilisation en ligne.

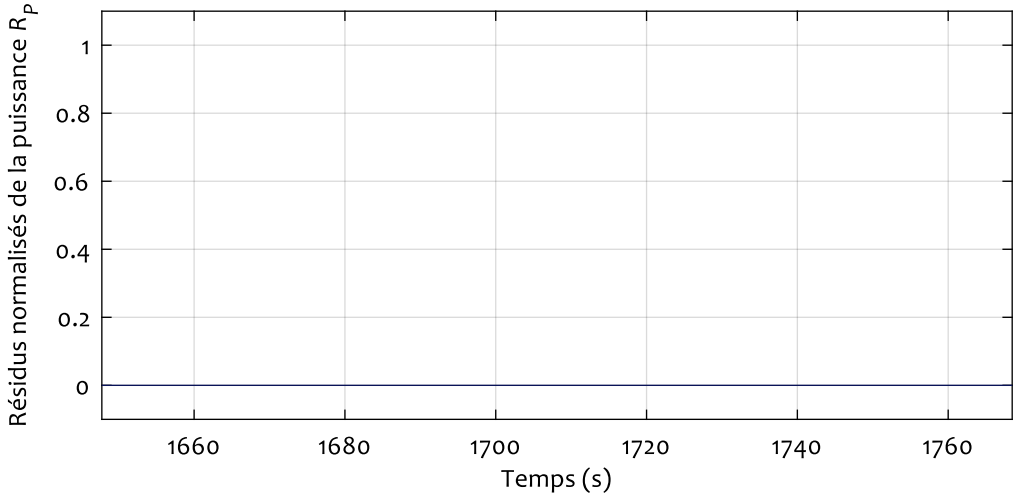
La dernière modélisation à valider en ligne est la validation de la température. Les estimations et les mesures de cette dernière sont affichées sur la Figure V.18. Sur cette figure, les estimation de la température mettent du temps pour converger aux mesures. Pendant ce temps de convergence, la valeur des résidus moyennés sont en de hors de l'enveloppe crée par les seuils (Figure V.19b), et ainsi une alarmes est générée (Figure V.19c). Toutefois, une fois les estimations et les mesures convergent, elle est éliminée. Après cette convergence, les estimations données par le modèle ARMAX_{LS} sont très satisfaisante avec un MAE $= 0.56\,°\text{C}$ (MAPE $= 1.3757\%$) et un temps d'échantillonnage respecté (Table V.6). Donc, la modélisation et la surveillance de la température sont validées pendant l'utilisation normale en-ligne.



(a)



(b)



(c)

FIGURE V.17 – Les profils des résidus $r_P(k)$, $r_{m_P}(k)$ et $R_P(k)$ de la carte de développement, en fonctionnement normal : (a) : résidus bruts $r_P(k)$, (b) : résidus moyennés $r_{m_P}(k)$, (c) : résidus normalisés $R_P(k)$.

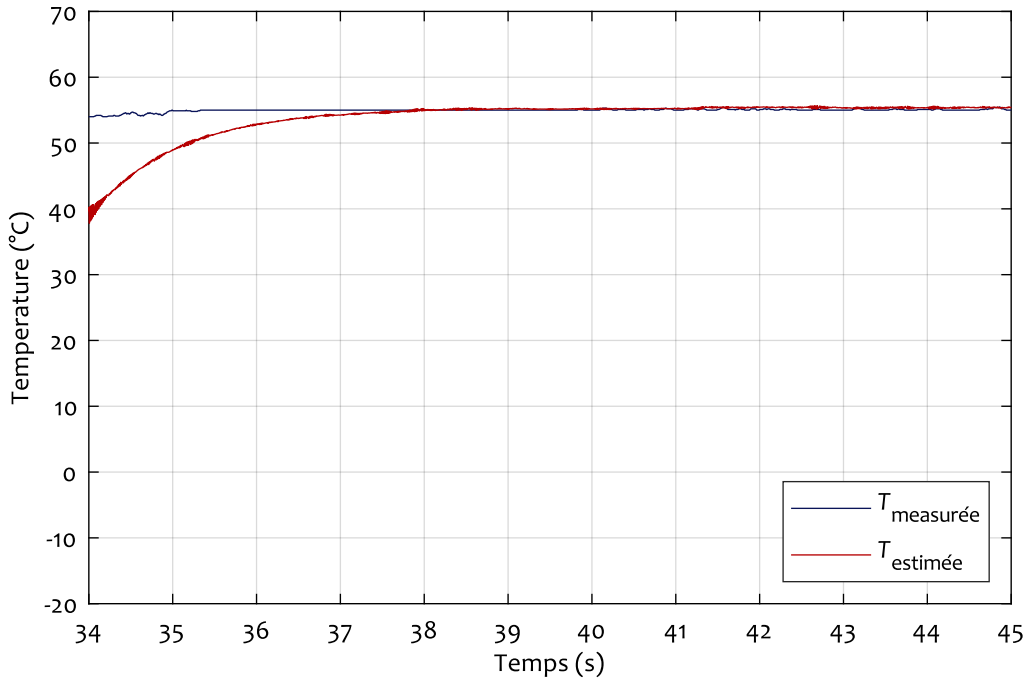
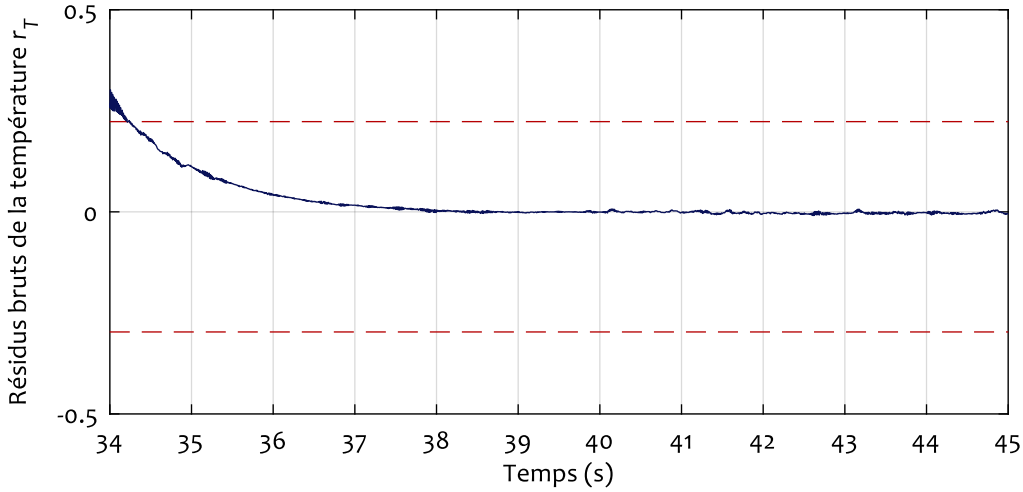
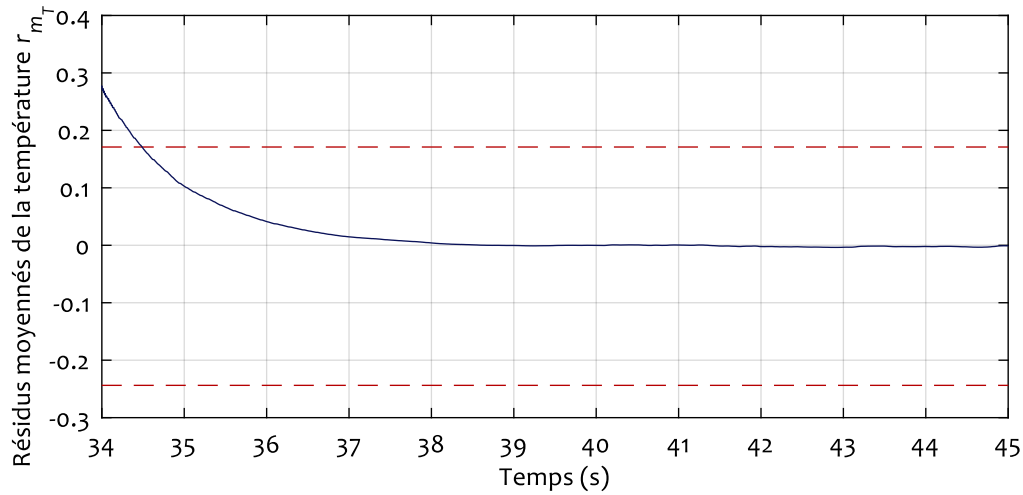


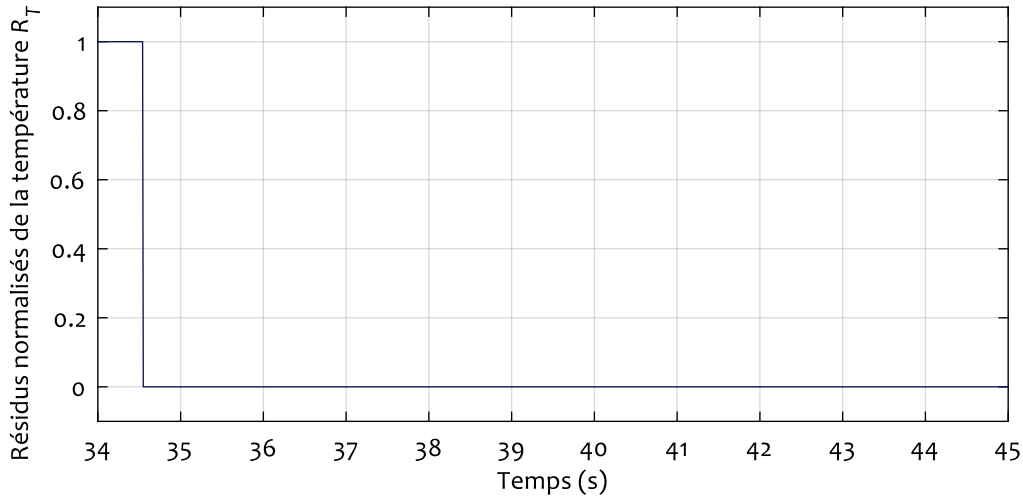
FIGURE V.18 – Les estimations générées par le modèle de température par rapport aux mesures du système, pendant le fonctionnement normal.



(a)



(b)



(c)

FIGURE V.19 – Les profils des résidus $r_T(k)$, $r_{m_T}(k)$ et $R_T(k)$ de la carte de développement, en fonctionnement normal : (a) : résidus bruts $r_T(k)$, (b) : résidus moyennés $r_{m_T}(k)$, (c) : résidus normalisés $R_T(k)$.

V.4.2 Branchement d'un périphérique

Dans ce scénario, nous reproduisons le défaut décrit dans le Paragraphe IV.6.2. Au cours d'une expérience de fonctionnement normal, nous connectons un périphérique étranger au système. Sur la carte de développement, nous branchons un smartphone sur l'un ses port USB. Les résultats des estimations du modèle NARX et les mesures du multimètre sont affichés dans la Figure V.20. Le smartphone est branché aux alentours de $t = 1840\,\text{s}$. À ce moment, un écart commence a se produire entre les estimations et les mesures. Cet écart est instantanément détecté par les résidus (Figure V.21), notamment les résidus moyennés $r_{m_P}(k)$ qui dépassent l'enveloppes des seuils (Figure V.21b), et causent la génération d'une alarme (Figure V.21c). Cette dernière automatiquement disparaît quand le smartphone est débranché aux alentours de $t = 1970\,\text{s}$, prouvant ainsi la capacité dans notre algorithme de surveillance à détecter la présence des anomalies et des défauts de puissance en ligne.

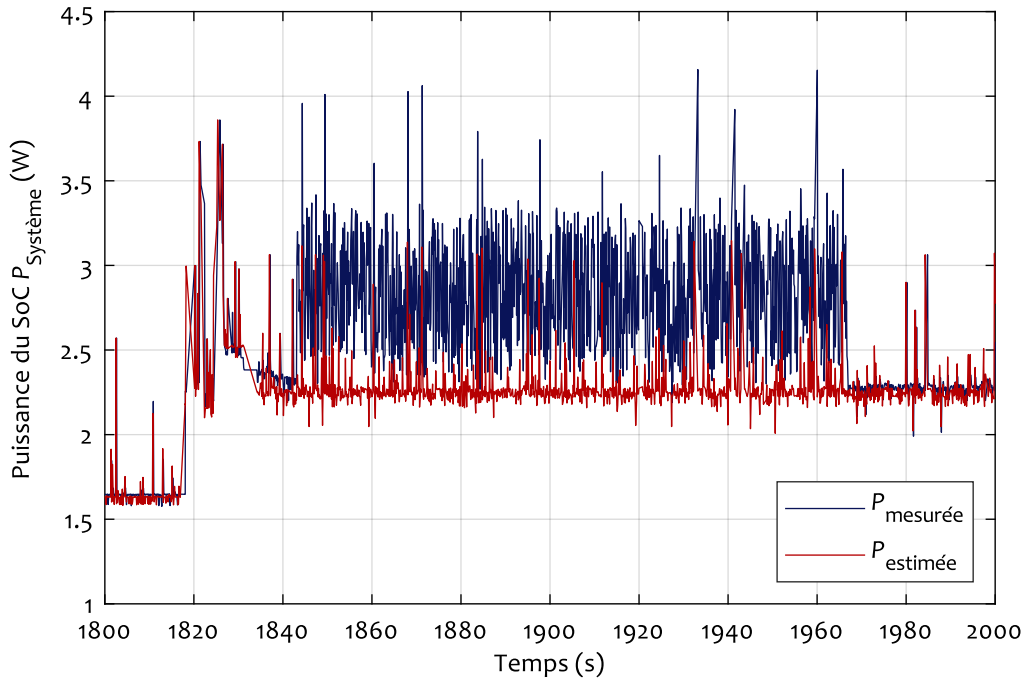
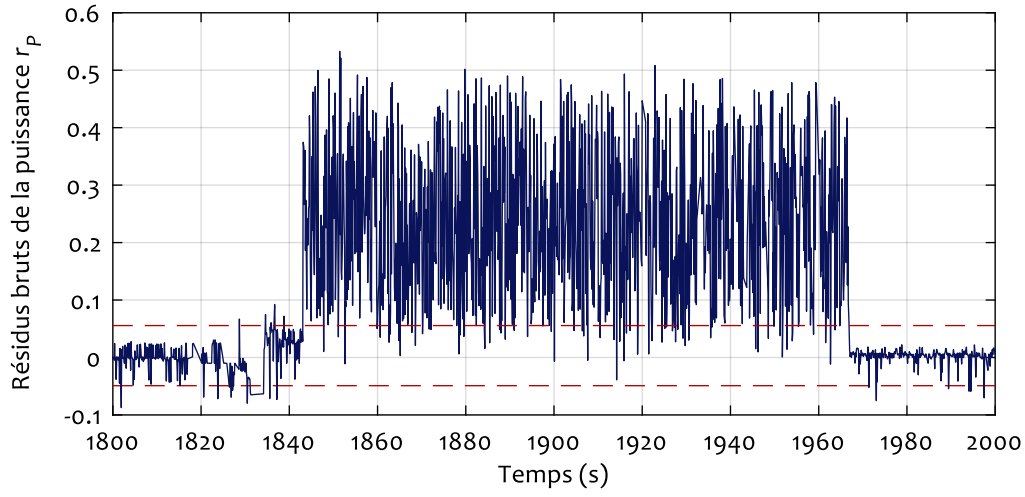


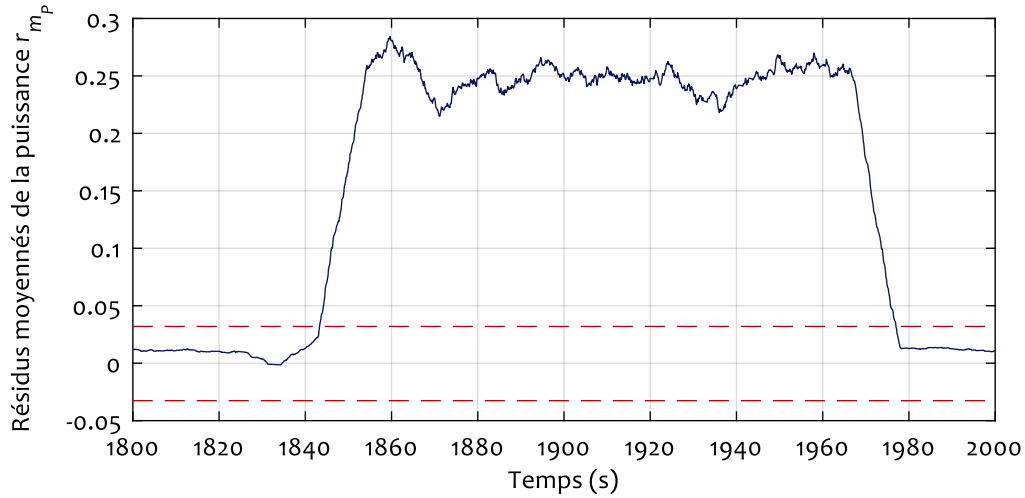
FIGURE V.20 – Les estimations de la puissance comparées aux mesures du multimètre pour la carte de développement, pendant le branchement d'un périphérique inconnu.

V.4.3 Environnement surchauffé

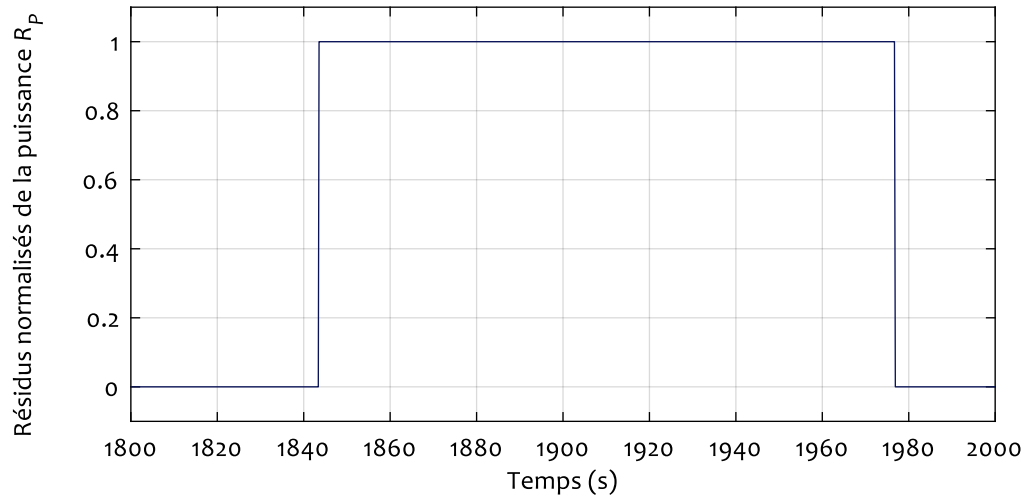
Le dernier scénario d'expérimentation avec les défaillances présenté dans ce chapitre, concerne la détection des défaut provenant de l'environnement. Donc, à l'image du défaut présenté dans le Paragraphe IV.6.3, au cours d'une expérience de surveillance la carte sera chauffé avec un projecteur ayant une lampe de puissance de 1000 W. Les résultats de cette expérience sont présentés dans les figures V.22 et V.23. L'échauffement commence aux alentours de $t = 380\,\text{s}$, les mesures de température sur la carte réagissent en présentant un bruit, puis commencent à diverger des estimations



(a)



(b)



(c)

FIGURE V.21 – Les profils des résidus $r_P(k)$, $r_{m_p}(k)$ et $R_P(k)$ de la carte de développement, pendant le branchement d'un périphérique inconnu : (a) : résidus bruts $r_P(k)$, (b) : résidus moyennés $r_{m_p}(k)$, (c) : résidus normalisés $R_P(k)$.

du modèle $ARMAX_{LS}$. Cette divergence cause les résidus moyennés de sortir de l'enveloppe des seuils (Figure V.23b) et par conséquent la génération d'une alarme (Figure V.23c). L'alarme disparaît éventuellement après le refroidissement de la carte (après 300 s).

Ces résultat nous permettent, ainsi, de conclure positivement sur la sensibilité de notre algorithme de surveillance aux défauts qui se manifestent sous forme d'une augmentation de température.

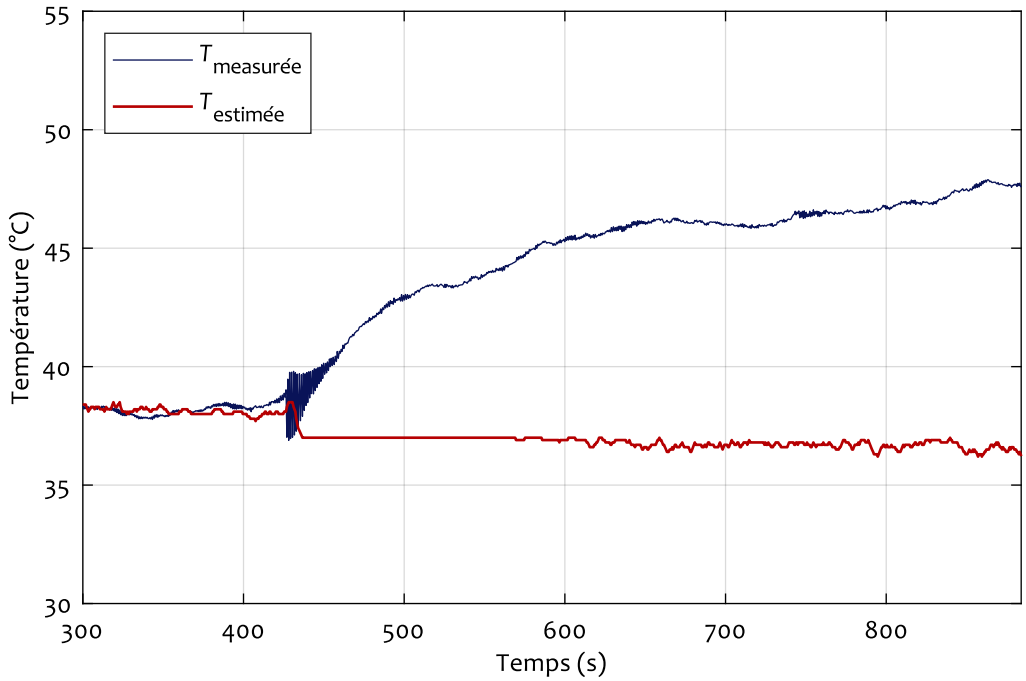


FIGURE V.22 – Les estimations générées par le modèle de température par rapport aux mesures du système, sous la chaleur générée par le projecteur.

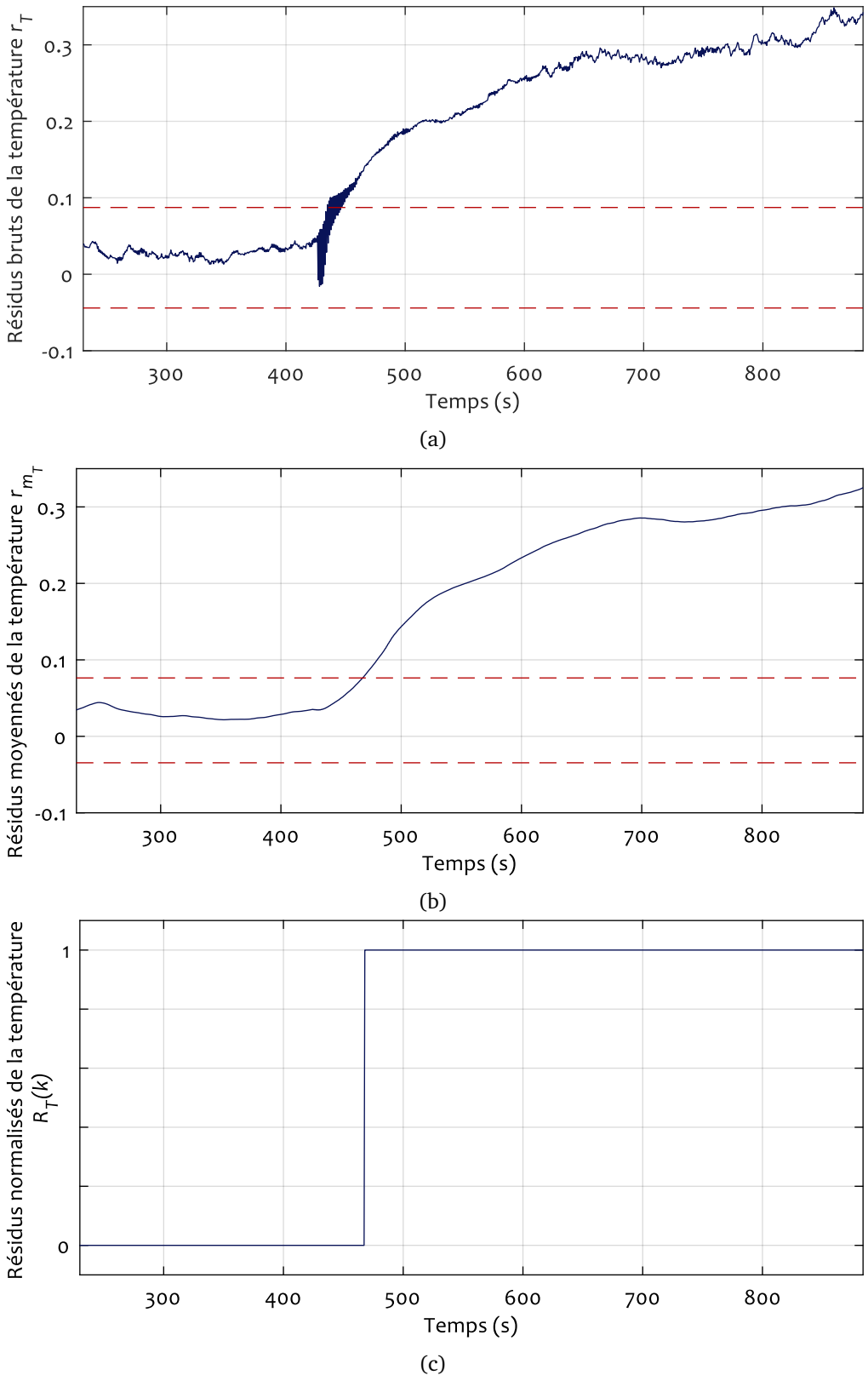


FIGURE V.23 – Les profils des résidus $r_T(k)$, $r_{m_T}(k)$ et $R_T(k)$ de la carte de développement, sous la chaleur générée par le projecteur : (a) : résidus bruts $r_T(k)$, (b) : résidus moyennés $r_{m_T}(k)$, (c) : résidus normalisés $R_T(k)$.

V.5 Conclusion

Dans ce chapitre, nous avons mis en place un prototype basé sur une carte de développement certifiée pour les systèmes critiques pour la sécurité. Ce prototype sert de preuve de concept pour la validation du modèle incrémental, et de l'algorithme de surveillance sur un matériel à caractère industriel, prêt pour déploiement sur le terrain.

Une fois le prototype mis en place, nous avons adapté le modèle incrémental aux spécificités de la carte de développement. La structure flexible du modèle a rendu ce processus assez simple, en le réduisant à la mise à jour des composants à modéliser, et l'entraînement des modèles avec des données provenant de la carte. Ensuite, nous avons validé les sous-systèmes du modèle incrémental, avec des résultats très encourageants. Nous avons également calculé les seuils nécessaires pour le fonctionnement l'algorithme de surveillance.

La dernière section du chapitre a été consacrée au test et à la validation de l'algorithme de surveillance. Les résultats de la validation en fonctionnement en ligne, mettent en lumière la précision de notre modélisation et la robustesse de l'algorithme de surveillance face aux incertitudes d'estimation en éliminant les fausses alarmes. Tandis que les tests avec deux scénario de défaillance ont démontré la sensibilité de l'algorithme face à des dérives de différents types.

Enfin, les résultats obtenus dans ce chapitre témoignent de l'adaptabilité de la structure de modélisation et de surveillance aux systèmes embarqués basés sur l'architecture ARM, et de sa capacité à satisfaire des contraintes de performance sur une carte de développement certifiée pour une utilisation sur des systèmes à risques.

Conclusion Générale

Le projet FUI 19, *Multi Modular Cockpit Display* (MMCD), avait pour objectif le remplacement par des écrans tactiles, de tous les outils d'affichage et de contrôle analogiques dans les cockpits des avions. En effet, ces dispositifs offrent une grande versatilité, et permettent de remplacer par des algorithmes plusieurs tâches initialement réalisées par des composants mécaniques, des circuits électroniques analogiques ou par des opérateurs humains. En revanche, les systèmes avioniques étant des systèmes à risques, la sûreté de fonctionnement des SoC embarqués associés aux écrans tactiles, devient l'une des problématiques principales à résoudre dans le cadre de ce projet.

Afin de répondre à cette demande, nous avons développé une approche de modélisation incrémentale et de surveillance en ligne des SoC embarqués. En plus des verrous scientifiques liés à la modélisation et au diagnostic des SoC, les aspects pratiques ont également été pris en compte par la proposition d'un système de modélisation et de surveillance respectant les contraintes d'instrumentation des SoC ainsi que les contraintes d'adaptabilité à une large gamme de SoC embarqués.

Le premier chapitre de cette thèse a débuté par une analyse des systèmes embarqués hétérogènes et leurs constituants. Grâce à la compréhension du fonctionnement de ces systèmes complexes, les variables caractérisant leur état de fonctionnement ont été identifiées par la suite et les relations causales qui régissent la dynamique de ces variables ont été déterminées.

Un tour d'horizon de la littérature sur les approches de modélisation des SoC a été proposé dans la deuxième partie de ce chapitre. L'étude des avantages et des limites des approches déjà existantes, ainsi que des applications proposées, a permis d'apporter la première contribution par la définition d'une méthodologie générique de construction de profilers et modèles, applicable à une large gamme de SoC. Cette méthodologie nous a permis de proposer par la suite un modèle incrémental des SoC embarqués.

Les dimensions miniatures de ces systèmes et l'interaction entre parties physiques (matérielles) et logicielles accroît leur complexité par rapport à celle de systèmes industriels. En effet, l'instrumentation des SoC n'est pas constituée uniquement de capteurs réels, tels que les capteurs de température, mais également de capteurs *« logiciels »*, capables de suivre et de mesurer l'activité logicielle des SoC, comme la charge des processeurs et la RAM utilisée. Nous avons donc consacré le deuxième chapitre de cette thèse à la construction d'un profiler – que nous nommons N^3 – capable de lire, d'enregistrer et de traiter les mesures physiques et logicielles des variables caractéristiques de ces systèmes. Notre principale contribution y est le développement et la mise en place d'un algorithme d'identification

dans le profiler N^3 ainsi que le traitement automatique des problèmes de synchronisation des données, liées essentiellement à la priorisation des tâches dans les périodes de forte charge des processeurs.

Le travail d'analyse et de synchronisation des données, effectué dans le deuxième chapitre, a permis de préparer des vecteurs de données, sous forme de séries temporelles complètes et synchronisées. L'existence de cette base de données a orienté notre choix vers une modélisation guidée par les données des SoC. Nous avons dans un premier temps proposé une structure incrémentale, où le SoC est défini comme un ensemble de sous-systèmes causalement interconnectés et présentés dans une bibliothèque de modèles ouverts, évolutifs et interchangeables. Cette structure assure la généricité du modèle et constitue une des contributions principales de ce travail de recherche. La modélisation de chaque sous-système est réalisée en analysant la dynamique de chaque variable à estimer par rapport aux dynamiques des variables d'entrée du modèle. Ainsi, pour l'estimation de la fréquence et des tensions des processeurs, nous avons opté pour des algorithmes construits grâce à l'analyse des algorithmes de DVFS des SoC étudiés. La température suit un modèle ARMAX tandis que la puissance est modélisé par un réseau de neurones NARX.

Après les étapes d'apprentissage et de test, chaque modèle est validé expérimentalement. Les résultats obtenus sur différents systèmes ont été analysés pour mettre en évidence les performances obtenues, avec des scénarios de fonctionnement représentatifs de celui du SoC dans des conditions réelles. Les incertitudes de modélisations pouvant être liées au retard de l'estimation sur les mesures ou aux bruits de mesures ont été quantifiées dans ce chapitre puis utilisées dans le quatrième chapitre consacré au développement d'une méthode de détection de dérives des caractéristiques des SoC.

Le modèle incrémental n'est pas développé uniquement pour l'estimation des variables qui caractérisent l'état de fonctionnement des SoC embarqués, mais également pour surveiller leur dynamique, et détecter des dérives relatives à la dégradation de l'état de fonctionnement de ces systèmes. La méthode que nous avons proposé dans le Chapitre IV combine l'efficacité et la simplicité de paramétrage et de mise en œuvre. Elle est basée sur le principe de la redondance analytique obtenue grâce au modèle incrémental.

Les indicateurs de dérives sont considérés dans ce chapitre comme des signaux, évalués en utilisant des méthodes statistiques. La méthode d'évaluation de chaque résidu est déterminée en fonction de ses propriétés (présence de retards, présence de bruits, etc.) afin de générer des résidus normalisés facilement interprétables. L'algorithme de détection de dérive a été validé expérimentalement grâce à des scénarios de dégradation comparables à ceux pouvant se produire en conditions réelles. Les résultats expérimentaux obtenus montrent la capacité de notre algorithme à détecter aussi bien des défauts matériels que des défauts issus de la partie logicielle du système ou de l'environnement. L'évaluation des résidus étant facilement paramétrable, l'utilisateur peut, selon ses besoins, gérer le compromis entre fausses alarmes et non-détections avec une grande flexibilité.

Dans le cinquième chapitre, nous avons mis en place un démonstrateur qui a servi de preuve de concept pour le projet MMCD. Nous avons détaillé toutes les étapes de mise en œuvre : la structure du démonstrateur, le choix de ces constituants, l'adaptation de N^3 et du modèle incrémental

aux composants matériels et logiciels de la carte de développement certifiée, et le paramétrage de l'algorithme de détection des dérives. Les scénarios de fonctionnement normal et les scénarios de dégradation ont été étudiés afin de se rapprocher des conditions d'utilisation dans le domaine de l'avionique, en terme de charge de calcul mais également en terme de phénomènes de dégradations. Ces dernières peuvent être dues à des court-circuits provoqués par les vibrations et les surchauffes provoquées par des défauts de refroidissement, des charges électrostatiques ou encore par des radiations. Les résultats expérimentaux montrent à la fois l'aisance dans le déploiement des algorithmes et leur efficacité, aussi bien dans l'estimation en ligne des caractéristiques que dans la surveillance de leurs dérives.

La précocité de la détection des dérives des caractéristiques des SoC a toujours été un impératif dans le développement de l'approche proposée dans cette thèse. Les résultats obtenus sur les différents systèmes utilisés dans ce travail de recherche en fonctionnement dégradé montrent clairement que lorsque les dérives sont détectées, les systèmes sont encore en fonctionnement, et les tâches attendues sont encore réalisées correctement.

Nous avons donc pour perspective l'utilisation des résultats présentés dans ce manuscrit comme support de développement d'algorithmes de pronostic de défaillance des SoC. La première étape consiste en une étude préliminaire d'analyse de la pertinence de la modélisation de la tendance de dérive de certaines des caractéristiques estimées par le modèle incrémental, pour l'estimation du temps de vie résiduel des SoC. Dans [2], nous avons analysé la corrélation entre la dérive de la température des SoC et leur processus de vieillissement sur le long terme afin d'estimer le temps de vie résiduel (*Remaining Useful Life*, RUL) des SoC. La mise en place d'un algorithme de pronostic de défaillance sur la base des résultats obtenus dans cette thèse, permettra de remplacer les stratégies de maintenance préventive et corrective adoptées jusque-là dans le domaine de l'avionique, pour la maintenance de leurs composants électroniques embarqués.

BIBLIOGRAPHIE

[1] Oussama Djedidi, Nacer M'Sirdi, and Aziz Naamane. Adaptive Estimation of the Thermal Behavior of CPU-GPU SoCs for Prediction and Diagnosis. In *IMAACA 2019 – Proceedings of the International Conference on Integrated Modeling and Analysis in Applied Control and Automation, 2019*, volume 1, pages 93–98, Lisbon, Portugal, September 2019. Bruzzone, Dauphin-Tanguy and Junco Eds. URL https://hal-amu.archives-ouvertes.fr/hal-02311475.

[2] Oussama Djedidi, Mohand DJEZIRI, and Samir Benmoussa. Failure Prognosis of Embedded Systems Based on Temperature Drift Assessment. In *IMAACA 2019 – Proceedings of the International Conference on Integrated Modeling and Analysis in Applied Control and Automation, 2019*, volume 1, pages 11–16, Lisbon, Portugal, September 2019. Bruzzone, Dauphin-Tanguy and Junco Eds. URL https://hal-amu.archives-ouvertes.fr/hal-02311476.

[3] Oussama Djedidi, Mohand DJEZIRI, and Kouider nacer M'SIRDI. Data-driven monitoring of Systems On Chip for Multifunction Modular Cockpit Display. 5th Summer School on INtelligent signal processing for FrontIEr Research and Industry (INFIERI 2019), May 2019. URL https://hal.archives-ouvertes.fr/hal-02144441. Poster.

[4] Oussama Djedidi, Mohand A. Djeziri, and Nacer K. M'Sirdi. Data-Driven Approach for Feature Drift Detection in Embedded Electronic Devices. *IFAC-PapersOnLine*, 51(24) :1024–1029, jan 2018. ISSN 24058963. doi : 10.1016/j.ifacol.2018.09.714.

[5] Oussama Djedidi, Mohand A. Djeziri, Nacer K. M'Sirdi, and Aziz Naamane. Constructing an Accurate and a High-Performance Power Profiler for Embedded Systems and Smartphones. In *Proceedings of the 21st ACM International Conference on Modelling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM '18)*, volume 18, pages 79–82, Montréal, Canada, 2018. ACM Press. ISBN 9781450359603. doi : 10.1145/3242102.3242139.

[6] Oussama Djedidi, Mohand A. Djeziri, Nacer K. M'Sirdi, and Aziz Naamane. A Novel Easy-to-construct Power Model for Embedded and Mobile Systems Using Recursive Neural Nets to Estimate Power Consumption of ARM-based Embedded Systems and Mobile Devices. In *Proceedings of the 15th International Conference on Informatics in Control, Automation and Robotics (ICINCO 2018)*, volume 1, pages 541–545, Porto, Portugal, aug 2018. INSTICC, SciTePress. ISBN 978-989-758-321-6. doi : 10.5220/0006915805410545.

[7] Oussama Djedidi, Mohand Arab Djeziri, and Kouider nacer M'SIRDI. Prototyping of a Data-driven monitoring of Systems On Chip for Multifunction Modular Cockpit Display (MMCD) Project, June 2018. URL https://hal-amu.archives-ouvertes.fr/hal-02293986.

[8] Oussama Djedidi, Mohand Arab Djeziri, and Nk M'Sirdi. Modélisation orientée diagnostic des composants électroniques embarqués. In *Congrès National de la Recherche des IUT (CNRIUT'2018)*, Aix en Provence, France, June 2018. URL https://hal-amu.archives-ouvertes.fr/hal-01902905.

[9] Oussama Djedidi, Mohand A. Djeziri, Nacer K. M'Sirdi, and Aziz Naamane. Modular Modelling of an Embedded Mobile CPU-GPU Chip for Feature Estimation. In *Proceedings of the 14th International Conference on Informatics in Control, Automation and Robotics*, volume 1, pages 338–345, Mardrid, Spain, 2017. SciTePress. ISBN 978-989-758-263-9. doi : 10.5220/0006470803380345.

[10] George Kyriazis. Heterogeneous System Architecture : A Technical Review. Technical report, AMD, 2012.

[11] NXP Inc. i.MX 6SoloX Applications Processors | Arm® Cortex®-A9, Cortex-M4 |NXP, 2019. URL https://www.nxp.com/products/processors-and-microcontrollers/applications-processors/i.mx-applications-processors/i.mx-6-processors/i.mx-6solox-processors-heterogeneous-processing-with-arm-cortex-a9-and-cortex-m4-cores:i.MX6SX.

[12] IFixit. Samsung Galaxy S5 Teardown - iFixit. URL https://www.ifixit.com/Teardown/Samsung+Galaxy+S5+Teardown/24016.

[13] Young Geun Kim, Minyong Kim, Jae Min Kim, Minyoung Sung, and Sung Woo Chung. A novel GPU power model for accurate smartphone power breakdown. *ETRI Journal*, 37(1) :157–164, 2015. ISSN 22337326. doi : 10.4218/etrij.15.0113.1411.

[14] GSM Arena. Samsung Galaxy S5 - Full phone specifications, 2016. URL https://www.gsmarena.com/samsung_galaxy_s5-6033.php.

[15] GSM Arena. Samsung Galaxy S8+ - Full phone specifications, 2016. URL https://www.gsmarena.com/samsung_galaxy_s8+-8523.php.

[16] AMD. What is Heterogeneous Computing? - AMD, 2014. URL https://developer.amd.com/heterogeneous-computing-2/what-is-heterogeneous-computing/http://developer.amd.com/resources/heterogeneous-computing/what-is-heterogeneous-computing/.

[17] Deniz Akdur, Vahid Garousi, and Onur Demirörs. A survey on modeling and model-driven engineering practices in the embedded software industry. *Journal of Systems Architecture*, 91 : 62–82, nov 2018. ISSN 13837621. doi : 10.1016/j.sysarc.2018.09.007.

[18] Lide Zhang, Birjodh Tiwana, Zhiyun Qian, Zhaoguang Wang, Robert P. Dick, Zhuoqing Morley Mao, and Lei Yang. Accurate online power estimation and automatic battery behavior based power model generation for smartphones. In *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES/ISSS '10*, page 105, New York, New York, USA, 2010. ACM Press. ISBN 9781605589053. doi : 10.1145/1878961.1878982.

[19] Federico Faggin, Marcian E. Hoff, Stanley Mazor, and Masatoshi Shima. History of the 4004. *IEEE Micro*, 16(6) :10–20, 1996. ISSN 02721732. doi : 10.1109/40.546561. URL http://ieeexplore.ieee.org/document/546561/.

[20] Qualcomm Inc. Snapdragon 855 Mobile Platform | Qualcomm. URL https://www.qualcomm.com/products/snapdragon-855-mobile-platform.

[21] Peter Marwedel. *Embedded System Design : Embedded Systems, Foundations of Cyber-Physical Systems, and the Internet of Things*. 2017. ISBN 331956045X.

[22] Tammy. Noergaard. Embedded Systems Architecture : A Comprehensive Guide for Engineers and Programmers. In *Embedded Systems Architecture*, pages 261–293. Elsevier/Newnes, 2013. ISBN 0750677929.

[23] David A. Patterson and John L. Hennessy. *Computer Organization and Design, Fifth Edition : The Hardware/Software Interface*. Morgan Kaufmann, San Francisco, CA, USA, 5th edition, 2013. ISBN 0-12-407726-9 978-0-12-407726-3.

[24] John L. Hennessy and David A. Patterson. *Computer Architecture, Fifth Edition : A Quantitative Approach*. The Morgan Kaufmann Series in Computer Architecture and Design. Morgan Kaufmann, Waltham, MA, 5 edition edition, sep 2011. ISBN 978-0-12-383872-8.

[25] HSA Foundation. HSA Foundation ARM, AMD, Imagination, MediaTek, Qualcomm, Samsung, TI, 2017. URL http://www.hsafoundation.com/.

[26] David Blythe. Rise of the graphics processor. *Proceedings of the IEEE*, 96(5) :761–778, may 2008. ISSN 00189219. doi : 10.1109/JPROC.2008.917718.

[27] Nicholas Wilt. *CUDA Handbook : A Comprehensive Guide to GPU Programming, The*. Addison-Wesley Professional, Upper Saddle River, NJ, 1 edition edition, jun 2013. ISBN 978-0-321-80946-9.

[28] Kevin Krewell. What's the Difference Between a CPU and a GPU? | The Official NVIDIA Blog, 2009. URL https://blogs.nvidia.com/blog/2009/12/16/whats-the-difference-between-a-cpu-and-a-gpu/.

[29] Mali-T860 & Mali-T880 - ARM, . URL https://www.arm.com/products/multimedia/mali-gpu/high-performance/mali-t860-t880.php.

[30] Mali GPU - ARM, . URL https://www.arm.com/products/multimedia/mali-gpu.

[31] Snapdragon 855 mobile platform, Aug 2019. URL https://www.qualcomm.com/products/snapdragon-855-plus-mobile-platform.

[32] NVIDIA. The World's Fastest Mobile Processors | NVIDIA Tegra|NVIDIA. URL http://www.nvidia.com/object/tegra.html.

[33] NVIDIA. Tegra K1 Whitepaper, jan 2014. URL https://developer.nvidia.com/content/tegra-k1-whitepaper.

[34] Pietro Mercati, Francesco Paterna, Andrea Bartolini, Luca Benini, and Tajana Šimunić Rosing. WARM : Workload-Aware Reliability Management in Linux/Android. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(9) :1557–1570, sep 2017. ISSN 02780070. doi : 10.1109/TCAD.2016.2611501.

[35] Pietro Mercati, Vinay Hanumaiah, Jitendra Kulkarni, Simon Bloch, and Tajana Simunic Rosing. User-centric Joint Power and Thermal Management for Smartphones. In *Proceedings of the 6th International Conference on Mobile Computing, Applications and Services*. ICST, 2014. ISBN 978-1-63190-024-2. doi : 10.4108/icst.mobicase.2014.257788.

[36] Tiantian Li, Ge Yu, and Jie Song. Minimizing energy by thermal-aware task assignment and speed scaling in heterogeneous MPSoC systems. *Journal of Systems Architecture*, 89 :118–130, sep 2018. ISSN 13837621. doi : 10.1016/j.sysarc.2018.08.003.

[37] Junlong Zhou, Jianming Yan, Kun Cao, Yanchao Tan, Tongquan Wei, Mingsong Chen, Gongxuan Zhang, Xiaodao Chen, and Shiyan Hu. Thermal-aware correlated two-level scheduling of real-time tasks with reduced processor energy on heterogeneous MPSoCs. *Journal of Systems Architecture*, 82 :1–11, jan 2018. ISSN 1383-7621. doi : 10.1016/J.SYSARC.2017.09.007.

[38] Mohammad Javad Dousti, Majid Ghasemi-Gol, Mahdi Nazemi, and Massoud Pedram. ThermTap : An online power analyzer and thermal simulator for Android devices. In *Proceedings of the International Symposium on Low Power Electronics and Design*, volume 2015-Septe, pages 341–346. IEEE, jul 2015. ISBN 9781467380096. doi : 10.1109/ISLPED.2015.7273537.

[39] Joon-Myung Kang, Sin-Seok Seo, and James Won-Ki Hong. Personalized Battery Lifetime Prediction for Mobile Devices based on Usage Patterns. *Journal of Computing Science and Engineering*, 5(4) :338–345, dec 2011. ISSN 1976-4677. doi : 10.5626/JCSE.2011.5.4.338.

[40] Joon-Myung Kang, Chang-Keun Park, Sin-Seok Seo, Mi-Jung Choi, and James Won-Ki Hong. User-Centric Prediction for Battery Lifetime of Mobile Devices. In *CHALLENGES FOR NEXT GENERATION NETWORK OPERATIONS AND SERVICE MANAGEMENT, PROCEEDINGS*, volume 5297, pages 531–534. Springer, Berlin, Heidelberg, oct 2008. ISBN 978-3-540-88622-8. doi : 10.1007/978-3-540-88623-5_69.

[41] Dongwon Kim, Yohan Chon, Wonwoo Jung, Yungeun Kim, and Hojung Cha. Accurate Prediction of Available Battery Time for Mobile Applications. *ACM Transactions on Embedded Computing Systems*, 15(3) :1–17, may 2016. ISSN 15399087. doi : 10.1145/2875423.

[42] Swapnili P. Karmore and Anjali R. Mahajan. New Approach for Testing and providing Security Mechanism for Embedded Systems. In *Procedia Computer Science*, volume 78, pages 851–858. Elsevier, jan 2016. doi : 10.1016/j.procs.2016.02.073. URL https://www-sciencedirect-com.lama.univ-amu.fr/science/article/pii/S1877050916000752?via%3Dihub.

[43] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Feng Xia, and Muhammad Shiraz. A Review on mobile application energy profiling : Taxonomy, state-of-the-art, and open research issues. *Journal of Network and Computer Applications*, 58 :42–59, dec 2015. ISSN 10958592. doi : 10.1016/j.jnca.2015.09.002.

[44] Mahmoud Bokhari and Markus Wagner. Optimising Energy Consumption Heuristically on Android Mobile Phones. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion - GECCO '16 Companion*, pages 1139–1140, New York, New York, USA, 2016. ACM Press. ISBN 9781450343237. doi : 10.1145/2908961.2931691.

[45] Arun Tomy, P. I. Liji, and B. S. Manoj. On reducing energy consumption as a function of space and time in mobile devices. In *2016 IEEE International Conference on Advanced Networks and Telecommunications Systems (ANTS)*, pages 1–6. IEEE, nov 2016. ISBN 978-1-5090-2193-2. doi : 10.1109/ANTS.2016.7947875.

[46] Matthew Travers, Rishad Shafik, and Fei Xia. Power-Normalized Performance Optimization of Concurrent Many-Core Applications. In *Proceedings - International Conference on Application of Concurrency to System Design, ACSD*, pages 94–103. IEEE, jun 2017. ISBN 9781509025893. doi : 10.1109/ACSD.2016.14.

[47] Jing Huang, Renfa Li, Jiyao An, Derrick Ntalasha, Fan Yang, and Keqin Li. Energy-Efficient Resource Utilization for Heterogeneous Embedded Computing Systems. *IEEE Transactions on Computers*, 66(9) :1518–1531, sep 2017. ISSN 00189340. doi : 10.1109/TC.2017.2693186.

[48] Mohammed A.N. Al-Hayanni, Ashur Rafiev, Rishad Shafik, and Fei Xia. Power and Energy Normalized Speedup Models for Heterogeneous Many Core Computing. In *Proceedings - International Conference on Application of Concurrency to System Design, ACSD*, pages 84–93. IEEE, jun 2017. ISBN 9781509025893. doi : 10.1109/ACSD.2016.16.

[49] Anil Kanduri, Mohammad Hashem Haghbayan, Amir M. Rahmani, Pasi Liljeberg, Axel Jantsch, Hannu Tenhunen, and Nikil Dutt. Accuracy-Aware Power Management for Many-Core Systems Running Error-Resilient Applications. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(10) :2749–2762, oct 2017. ISSN 10638210. doi : 10.1109/TVLSI.2017.2694388.

[50] Fatimah Abdualaziz Almusalli, Noor Zaman, and Raihan Rasool. Energy efficient middleware : Design and development for mobile applications. In *2017 19th International Conference on Advanced Communication Technology (ICACT)*, pages 541–549. IEEE, 2017. ISBN 978-89-968650-9-4. doi : 10.23919/ICACT.2017.7890149.

[51] Imjae Hwang, Hyuck-Joo Kwon, Ji-Hye Chang, Yeongkyu Lim, Cheong Ghil Kim, and Woo-Chan Park. An Effective Viewport Resolution Scaling Technique to Reduce the Power Consumption in Mobile GPUs. *KSII Transactions on Internet and Information Systems*, 11(8), aug 2017. ISSN 19767277. doi : 10.3837/tiis.2017.08.009.

[52] Peramanathan Sathyamoorthy, Edith C.-H. Ngai, Xiping Hu, and Victor C. M. Leung. Energy Efficiency as an Orchestration Service for Mobile Internet of Things. In *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 155–162. IEEE, nov 2015. ISBN 978-1-4673-9560-1. doi : 10.1109/CloudCom.2015.90.

[53] Mahmoud A. Bokhari, Brad Alexander, and Markus Wagner. In-vivo and offline optimisation of energy use in the presence of small energy signals. In *Proceedings of the 15th EAI International Conference on Mobile and Ubiquitous Systems : Computing, Networking and Services - MobiQuitous '18*, pages 207–215, New York, New York, USA, 2018. ACM Press. ISBN 9781450360937. doi : 10.1145/3286978.3287014.

[54] Edip Demirbilek, J.-C. Grgoire, Ahmad Vakili, and Lionel Reyero. Modelling and improving the battery performance of a mobile phone application : A methodology. In *5th International Conference on Energy Aware Computing Systems and Applications, ICEAC 2015*, pages 1–4. IEEE, mar 2015. ISBN 9781479917716. doi : 10.1109/ICEAC.2015.7352165.

[55] Mona A Abou-of, Ahmed H. Taha, and Amr A. Sedky. Trade-off between Low Power and Energy Efficiency in Benchmarking. In *2016 7th International Conference on Information and Communication Systems (ICICS)*, pages 322–326. IEEE, apr 2016. ISBN 9781467386142. doi : 10.1109/IACS.2016.7476072.

[56] Jae Min Kim, Young Geun Kim, and Sung Woo Chung. Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices. *IEEE Transactions on Computers*, 64(1) : 286–292, jan 2015. ISSN 00189340. doi : 10.1109/TC.2013.188.

[57] Asieh Salehi Fathabadi, Michael J. Butler, Sheng Yang, Luis Alfonso Maeda-Nunez, James Bantock, Bashir M. Al-Hashimi, and Geoff V. Merrett. A model-based framework for software portability and verification in embedded power management systems. *Journal of Systems Architecture*, 82 :12–23, jan 2018. ISSN 1383-7621. doi : 10.1016/J.SYSARC.2017.12.001.

[58] Thorsten Zitterell and Christoph Scholl. Improving energy-efficient real-time scheduling by exploiting code instrumentation. *Proceedings of the International Multiconference on Computer Science and Information Technology, IMCSIT 2008*, 3 :763–771, 2008. ISSN 1896-7094. doi : 10.1109/IMCSIT.2008.4747329.

[59] Daecheol You and K.-S. Chung. Dynamic voltage and frequency scaling framework for low-power embedded GPUs. *Electronics Letters*, 48(21) :1333, 2012. ISSN 00135194. doi : 10.1049/el.2012.2624.

[60] Jurn Gyu Park, Chen Ying Hsieh, Nikil Dutt, and Sung Soo Lim. Quality-aware mobile graphics workload characterization for energy-efficient DVFS design. In *2014 IEEE 12th Symposium on Embedded Systems for Real-Time Multimedia, ESTIMedia 2014*, pages 70–79. IEEE, oct 2014. ISBN 9781479963072. doi : 10.1109/ESTIMedia.2014.6962347.

[61] Shaosong Li and Shivakant Mishra. Optimizing power consumption in multicore smartphones. *Journal of Parallel and Distributed Computing*, 95 :124–137, sep 2015. ISSN 07437315. doi : 10.1016/j.jpdc.2016.02.004.

[62] Elliott Wen, Winston K.G. Seah, Bryan Ng, Xuefeng Liu, Jiannong Cao, and Xuefeng Liu. GBooster : Towards Acceleration of GPU-Intensive Mobile Applications. In *Proceedings - International Conference on Distributed Computing Systems*, pages 1408–1418. IEEE, jun 2017. ISBN 9781538617915. doi : 10.1109/ICDCS.2017.73.

[63] Stephen Marz, Brad Vander Zanden, and Brad Vander Zanden. Reducing Power Consumption and Latency in Mobile Devices Using an Event Stream Model. *ACM Transactions on Embedded Computing Systems*, 16(11) :42–59, dec 2016. ISSN 15583465. doi : 10.1145/2964203.

[64] Ziling Lu, Chun Cao, and Xianping Tao. Improving screen power usage model on android smartphones. In *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, volume 2016-May, pages 167–173. IEEE, dec 2016. ISBN 9781467396448. doi : 10.1109/APSEC.2015.45.

[65] Stephen Romansky, Neil C. Borle, Shaiful Chowdhury, Abram Hindle, and Russ Greiner. Deep Green : Modelling Time-Series of Software Energy Consumption. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 273–283. IEEE, sep 2017. ISBN 978-1-5386-0992-7. doi : 10.1109/ICSME.2017.79.

[66] Matteo Ciman and Ombretta Gaggi. An empirical analysis of energy consumption of cross-platform frameworks for mobile development. *Pervasive and Mobile Computing*, 39 :214–230, aug 2017. ISSN 15741192. doi : 10.1016/j.pmcj.2016.10.004.

[67] Radhika Mittal, Aman Kansal, and Ranveer Chandra. Empowering developers to estimate app energy consumption. In *Proceedings of the 18th annual international conference on Mobile computing and networking - Mobicom '12*, page 317, New York, New York, USA, 2012. ACM Press. ISBN 9781450311595. doi : 10.1145/2348543.2348583.

[68] Hiroki Furusho, Kenji Hisazumi, Takeshi Kamiyama, Hiroshi Inamura, Tsuneo Nakanishi, and Akira Fukuda. Poster : an energy profiler for android applications used in the real world. In *Proceedings of the 10th international conference on Mobile systems, applications, and services - MobiSys '12*, page 517, New York, New York, USA, 2012. ACM Press. ISBN 9781450313018. doi : 10.1145/2307636.2307712.

[69] Ella Peltonen, Eemil Lagerspetz, Petteri Nurmi, and Sasu Tarkoma. Energy modeling of system settings : A crowdsourced approach. In *2015 IEEE International Conference on Pervasive Computing and Communications, PerCom 2015*, pages 37–45. IEEE, mar 2015. ISBN 9781479980338. doi : 10.1109/PERCOM.2015.7146507. URL http://ieeexplore.ieee.org/document/7146507/.

[70] Ella Peltonen, Eemil Lagerspetz, Petteri Nurmi, and Sasu Tarkoma. Constella : Crowdsourced system setting recommendations for mobile devices. *Pervasive and Mobile Computing*, 26 : 71–90, feb 2016. ISSN 15741192. doi : 10.1016/j.pmcj.2015.10.011.

[71] Jaeseong Lee, Yohan Chon, and Hojung Cha. Evaluating battery aging on mobile devices. In *Proceedings of the 52nd Annual Design Automation Conference on - DAC '15*, pages 1–6, New York, New York, USA, 2015. IEEE. ISBN 9781450335201. doi : 10.1145/2744769.2744838.

[72] Albert Potsch, Achim Berger, and Andreas Springer. Efficient analysis of power consumption behaviour of embedded wireless IoT systems. In *I2MTC 2017 - 2017 IEEE International Instrumentation and Measurement Technology Conference, Proceedings*, pages 1–6. IEEE, may 2017. ISBN 9781509035960. doi : 10.1109/I2MTC.2017.7969658.

[73] Karan Nair, Janhavi Kulkarni, Mansi Warde, Zalak Dave, Vedashree Rawalgaonkar, Ganesh Gore, and Jonathan Joshi. Optimizing power consumption in iot based wireless sensor networks using Bluetooth Low Energy. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT)*, pages 589–593. IEEE, oct 2015. ISBN 978-1-4673-7910-6. doi : 10.1109/ICGCIoT.2015.7380533.

[74] Borja Martinez, Màrius Montón, Ignasi Vilajosana, and Joan Daniel Prades. The Power of Models : Modeling Power Consumption for IoT Devices. *IEEE Sensors Journal*, 15(10) :5777–5789, oct 2015. ISSN 1530437X. doi : 10.1109/JSEN.2015.2445094.

[75] Alessio Merlo, Mauro Migliardi, and Paolo Fontanelli. On energy-based profiling of malware in Android. In *Proceedings of the 2014 International Conference on High Performance Computing and Simulation, HPCS 2014*, pages 535–542. IEEE, jul 2014. ISBN 9781479953127. doi : 10.1109/HPCSim.2014.6903732. URL http://ieeexplore.ieee.org/document/6903732/.

[76] Alessio Merlo, Mauro Migliardi, and Paolo Fontanelli. Measuring and estimating power consumption in Android to support energy-based intrusion detection. *Journal of Computer Security*, 23(5) :611–637, aug 2015. ISSN 0926227X. doi : 10.3233/JCS-150530.

[77] A. M. Abbasi, M. Al-tekreeti, Y. Ali, K. Naik, A. Nayak, N. Goel, and B. Plourde. A framework for detecting energy bugs in smartphones. In *2015 6th International Conference on the Network of the Future (NOF)*, pages 1–3. IEEE, sep 2015. ISBN 978-1-4673-8386-8. doi : 10.1109/NOF.2015.7333297.

[78] Jonghoe Koo, Kitaek Lee, Wonbo Lee, Yongseok Park, and Sunghyun Choi. BattTracker : Enabling energy awareness for smartphone using Li-ion battery characteristics. In *Proceedings - IEEE INFOCOM*, volume 2016-July, pages 1–9. IEEE, apr 2016. ISBN 9781467399531. doi : 10.1109/INFOCOM.2016.7524422.

[79] Guillermo Suarez-Tangil, Juan E. Tapiador, Pedro Peris-Lopez, and Sergio Pastrana. Power-aware anomaly detection in smartphones : An analysis of on-platform versus externalized operation. *Pervasive and Mobile Computing*, 18 :137–151, apr 2015. ISSN 15741192. doi : 10.1016/j.pmcj.2014.10.007.

[80] Luca Caviglione, Mauro Gaggero, Jean François Lalande, Wojciech Mazurczyk, and Marcin Urbański. Seeing the unseen : Revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Transactions on Information Forensics and Security*, 11(4) :799–810, apr 2016. ISSN 15566013. doi : 10.1109/TIFS.2015.2510825.

[81] Luca Ardito, Giuseppe Procaccianti, Marco Torchiano, and G. Migliore. Profiling Power Consumption on Mobile Devices. In *Proceedings of The Third International Conference on Smart Grids, Green Communications and IT Energy-aware Technologies*, pages 101–106, Lisbon, 2013. IARIA.

[82] Mohammad Ashraful Hoque, Matti Siekkinen, Kashif Nizam Khan, Yu Xiao, and Sasu Tarkoma. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *ACM Computing Surveys*, 48(3) :1–40, 2015. ISSN 03600300. doi : 10.1145/2840723.

[83] Elhadj Benkhelifa, Tom Welsh, Loai Tawalbeh, Yaser Jararweh, and Anas Basalamah. Energy Optimisation for Mobile Device Power Consumption : A Survey and a Unified View of Modelling for a Comprehensive Network Simulation. *Mobile Networks and Applications*, 21(4) :575–588, aug 2016. ISSN 15728153. doi : 10.1007/s11036-016-0756-y.

[84] Google Inc. Android 5.0 APIs | Android Developers, 2017.

[85] Google Inc. Measuring Power Values | Android Open Source Project, 2017.

[86] Inc. Qualcomm Innovation Center. Trepn Profiler - Android Apps on Google Play, 2019. URL https://play.google.com/store/apps/details?id=com.quicinc.trepn&hl=fr_CH.

[87] Mian Dong and Lin Zhong. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *MobiSys*, page 335, New York, New York, USA, 2011. ACM Press. ISBN 9781450306430. doi : 10.1145/1999995.2000027.

[88] Wonwoo Jung, Chulkoo Kang, Chanmin Yoon, Dongwon Donwon Kim, and Hojung Cha. DevScope : A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. *Proceedings of the eighth IEEE/ACM/IFIP international conference on Hardware/software codesign and system synthesis - CODES+ISSS '12*, page 353, 2012. doi : 10.1145/2380445.2380502.

[89] Abhinav Pathak, Y Charlie Hu, and Ming Zhang. Where is the Energy Spent Inside My App ? : Fine Grained Energy Accounting on Smartphones with Eprof. In *Proceedings of the 7th ACM European Conference on Computer Systems*, EuroSys '12, pages 29–42, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1223-3. doi : 10.1145/2168836.2168841.

[90] Abhijeet Banerjee, Lee Kee Chong, Sudipta Chattopadhyay, and Abhik Roychoudhury. Detecting energy bugs and hotspots in mobile apps. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2014*, volume 3, pages 588–598, New York, New York, USA, 2014. ACM Press. ISBN 9781450330565. doi : 10.1145/2635868.2635871.

[91] Alex Shye, Benjamin Scholbrock, and Gokhan Memik. Into the wild : Studying real user activity patterns to guide power optimizations for mobile architectures. *Micro*, pages 168–178, 2009. ISSN 1072-4451. doi : 10.1145/1669112.1669135.

[92] Minyong Kim, Joonho Kong, and Sung Woo Chung. Enhancing online power estimation accuracy for smartphones. *IEEE Transactions on Consumer Electronics*, 58(2) :333–339, 2012. ISSN 00983063. doi : 10.1109/TCE.2012.6227431.

[93] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. PETrA : A Software-Based Tool for Estimating the Energy Profile of Android Applications. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 3–6. IEEE, may 2017. ISBN 978-1-5386-1589-8. doi : 10.1109/ICSE-C.2017.18.

[94] Narendra Kumar Shukla, Rosarium Pila, and Saurabh Rawat. Utilization-based power consumption profiling in smartphones. In *Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics, IC3I 2016*, pages 881–886. IEEE, dec 2016. ISBN 9781509052554. doi : 10.1109/IC3I.2016.7919046.

[95] Shaiful Alam Chowdhury and Abram Hindle. GreenOracle : Estimating Software Energy Consumption with Energy Measurement Corpora. In *Proceedings of the 13th International Workshop on Mining Software Repositories - MSR '16*, pages 49–60, New York, New York, USA, 2016. ACM Press. ISBN 9781450341868. doi : 10.1145/2901739.2901763.

[96] Shaiful Alam Chowdhury, Luke N Kumar, Md. Toukir Imam, Mohomed Shazan Mohomed Jabbar, Varun Sapra, Karan Aggarwal, Abram Hindle, and Russell Greiner. A system-call based model of software energy consumption without hardware instrumentation. In *2015 6th International Green and Sustainable Computing Conference*, pages 1–6. IEEE, dec 2016. ISBN 9781509001729. doi : 10.1109/IGCC.2015.7393719.

[97] Ekarat Rattagan, Edward T.H. Chu, Ying Dar Lin, and Yuan Cheng Lai. Semi-online power estimation for smartphone hardware components. In *2015 10th IEEE International Symposium on Industrial Embedded Systems, SIES 2015 - Proceedings*, pages 174–177. IEEE, jun 2015. ISBN 9781467377119. doi : 10.1109/SIES.2015.7185058.

[98] Kitae Kim, Donghwa Shin, Qing Xie, Yanzhi Wang, Massoud Pedram, and Naehyuck Chang. FEPMA : Fine-Grained Event-Driven Power Meter for Android Smartphones Based on Device Driver Layer Event Monitoring. In *Design, Automation I& Test in Europe Conference and Exhibition (DATE)*, pages 1–6, New Jersey, 2014. IEEE Conference Publications. ISBN 9783981537024. doi : 10.7873/DATE.2014.380.

[99] Chanmin Yoon, Seokjun Lee, Yonghun Choi, Rhan Ha, and Hojung Cha. Accurate power modeling of modern mobile application processors. *Journal of Systems Architecture*, 81 :17–31, nov 2017. ISSN 13837621. doi : 10.1016/j.sysarc.2017.10.001.

[100] Matthew J. Walker, Stephan Diestelhorst, Andreas Hansson, Anup K. Das, Sheng Yang, Bashir M. Al-Hashimi, and Geoff V. Merrett. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(1) :106–119, jan 2017. ISSN 02780070. doi : 10.1109/TCAD.2016.2562920.

[101] Xiang Chen, Yiran Chen, Mian Dong, and Charlie Zhang. Demystifying Energy Usage in Smartphones. *Design Automation Conference (DAC)*, pages 1–5, 2014. ISSN 0738100X. doi : 10.1109/DAC.2014.6881397.

[102] Soumya Kanti Datta, Christian Bonnet, and Navid Nikaein. Personalized power saving profiles generation analyzing smart device usage patterns. In *2014 7th IFIP Wireless and Mobile Networking Conference, WMNC 2014*, pages 1–8. IEEE, may 2014. ISBN 9781479930609. doi : 10.1109/WMNC.2014.6878858.

[103] Fengyuan Xu, Yunxin Liu, Qun Li, and Yongguang Zhang. V-edge : fast self-constructive power modeling of smartphones based on battery voltage dynamics. In *nsdi'13 Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, pages 43–55, Lombard, IL, 2013. USENIX. ISBN 978-1-931971-00-3.

[104] Sidartha A.L. Carvalho, Daniel C. Cunha, and Abel G. Silva-Filho. On the use of nonlinear methods for low-power CPU frequency prediction based on Android context variables. In *Proceedings - 2016 IEEE 15th International Symposium on Network Computing and Applications, NCA 2016*, pages 250–253. IEEE, oct 2016. ISBN 9781509032167. doi : 10.1109/NCA.2016.7778627.

[105] Sameer Alawnah and Assim Sagahyroon. Modeling of smartphones' power using neural networks. *Eurasip Journal on Embedded Systems*, 2017(1) :22, dec 2017. ISSN 16873963. doi : 10.1186/s13639-017-0070-1.

[106] Y Zhang and D Parikh. Hotleakage : A temperature-aware model of subthreshold and gate leakage for architects. *University of Virginia . . .* , 2003.

[107] Diary R. Sulaiman. Microprocessors thermal challenges for portable and embedded systems using thermal throttling technique. *Procedia Computer Science*, 3 :1023–1032, jan 2011. ISSN 1877-0509. doi : 10.1016/J.PROCS.2010.12.168.

[108] Kang Jia Wang, Hong Chang Sun, and Zhong Liang Pan. An analytical thermal model for Three-Dimensional integrated Circuits with integrated micro-channel cooling. *Thermal Science*, 21(4) :1601–1606, 2017. ISSN 03549836. doi : 10.2298/TSCI160716041W.

[109] Matteo Ferroni, Alessandro Antonio Nacci, Matteo Turri, Marco Domenico Santambrogio, and Donatella Sciuto. Experimental Evaluation and Modeling of Thermal Phenomena on Mobile Devices. In *Euromicro Conference on Digital System Design (DSD)*, pages 306–313. IEEE, aug 2015. ISBN 978-1-4673-8035-5. doi : 10.1109/DSD.2015.20. URL http://ieeexplore.ieee.org/document/7302290/.

[110] Temperature-dependent Electromigration Reliability. In Yi-Kan Cheng, Ching-Han Tsai, Chin-Chi Teng, and Sung-Mo Steve Kang, editors, *Electrothermal Analysis of VLSI Systems*, pages 121–155. Springer US, Boston, MA, 2005. ISBN 978-0-306-47024-0. doi : 10.1007/0-306-47024-1_6.

[111] Chang Chih Chen and Linda Milor. Microprocessor Aging Analysis and Reliability Modeling Due to Back-End Wearout Mechanisms. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 23(10) :2065–2076, oct 2015. ISSN 10638210. doi : 10.1109/TVLSI.2014.2357756.

[112] Mauricio Altieri, Suzanne Lesecq, Diego Puschini, Olivier Heron, Edith Beigne, and Jorge Rodas. Evaluation and mitigation of aging effects on a digital on-chip voltage and temperature sensor. In *Proceedings - 2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS 2015*, pages 111–117. IEEE, sep 2015. ISBN 9781467394192. doi : 10.1109/PATMOS.2015.7347595.

[113] D. Marculescu. Mitigating lifetime underestimation : A system-level approach considering temperature variations and correlations between failure mechanisms. In *2012 Design, Automation I& Test in Europe Conference I& Exhibition (DATE)*, pages 1269–1274. IEEE, mar 2012. ISBN 978-1-4577-2145-8. doi : 10.1109/DATE.2012.6176687. URL http://ieeexplore.ieee.org/document/6176687/http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6176687.

[114] Amir M. Rahmani, Mohammad Hashem Haghbayan, Antonio Miele, Pasi Liljeberg, Axel Jantsch, and Hannu Tenhunen. Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 25(2) :427–440, feb 2017. ISSN 10638210. doi : 10.1109/TVLSI.2016.2591798. URL http://ieeexplore.ieee.org/document/7530840/.

[115] Rz Ayoub and Ts Rosing. Predict and act : dynamic thermal management for multi-core processors. *Proceedings of the 14th ACM/IEEE international . . .*, pages 99–104, 2009. ISSN 15334678. doi : 10.1145/1594233.1594256.

[116] Linwei Niu and Dakai Zhu. Reliability-aware scheduling for reducing system-wide energy consumption for weakly hard real-time systems. *Journal of Systems Architecture*, 78 :30–54, aug 2017. ISSN 1383-7621. doi : 10.1016/J.SYSARC.2017.06.004.

[117] Thidapat Chantem, X. Sharon Hu, and Robert P. Dick. Temperature-aware scheduling and assignment for hard real-time applications on MPSoCs. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(10) :1884–1897, oct 2011. ISSN 10638210. doi : 10.1109/TVLSI.2010.2058873. URL http://ieeexplore.ieee.org/document/5551266/.

[118] David R. Bild, Sanchit Misra, Thidapat Chantem, Prabhat Kumar, Robert P. Dick, X. Sharon Hu, Li Shang, and Alok Choudhary. Temperature-aware test scheduling for multiprocessor systems-on-chip. In *IEEE/ACM International Conference on Computer-Aided Design, Digest of Technical Papers, ICCAD*, pages 59–66. IEEE, nov 2008. ISBN 9781424428205. doi : 10.1109/ICCAD.2008.4681552. URL http://ieeexplore.ieee.org/document/4681552/.

[119] Qing Xie, Mohammad Javad Dousti, and Massoud Pedram. Therminator : A Thermal Simulator for Smartphones Producing Accurate Chip and Skin Temperature Maps. *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, pages 117–122, 2014. ISSN 15334678. doi : 10.1145/2627369.2627641.

[120] Yuxiang Fu, Li Li, Kun Wang, and Chuan Zhang. Kalman Predictor-Based Proactive Dynamic Thermal Management for 3-D NoC Systems with Noisy Thermal Sensors. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(11) :1869–1882, nov 2017. ISSN 02780070. doi : 10.1109/TCAD..2661808.

[121] Shaosong Li and Shivakant Mishra. A Middleware for Power Aware Scheduling on Multicore Smartphones. In *Proceedings of the 12th EAI International Conference on Mobile and Ubiquitous Systems : Computing, Networking and Services*, pages 110–119. ACM, 2015. ISBN 978-1-63190-072-3. doi : 10.4108/eai.22-7-2015.2260049.

[122] Armen Dzhagaryan, Aleksandar Milenković, Mladen Milosevic, and Emil Jovanov. An Environment for Automated Measuring of Energy Consumed by Android Mobile Devices. In *PECCS 2016 - Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems*, pages 28–39, Lisbon, Portugal, feb 2016. SCITEPRESS - Science and Technology Publications. ISBN 978-989-758-195-3. doi : 10.5220/0005950800280039.

[123] Li Sun, Ramanujan K. Sheshadri, Wei Zheng, and Dimitrios Koutsonikolas. Modeling WiFi active power/energy consumption in smartphones. In *Proceedings - International Conference on Distributed Computing Systems*, pages 41–51. IEEE, jun 2014. ISBN 9781479951680. doi : 10.1109/ICDCS.2014.13.

[124] Sidartha A. L. Carvalho, Lucas M. F. Harada, Rafael N. Lima, Carolina M. A. Barbosa, Daniel C. Cunha, and Abel G. Silva-Filho. Identifying power consumption signatures in LTE conformance tests using machine learning. In *2018 IEEE 9th Latin American Symposium on Circuits and Systems (LASCAS)*, pages 1–4. IEEE, feb 2018. ISBN 978-1-5386-2311-4. doi : 10.1109/LASCAS.2018.8399980.

[125] Dario Di Nucci, Fabio Palomba, Antonio Prota, Annibale Panichella, Andy Zaidman, and Andrea De Lucia. Software-based energy profiling of Android apps : Simple, efficient and reliable ? In *SANER 2017 - 24th IEEE International Conference on Software Analysis, Evolution, and Reengineering*, volume 15, pages 103–114. IEEE, feb 2017. ISBN 9781509055012. doi : 10.1109/SANER.2017.7884613.

[126] Paul Holleis, Marko Luther, Gregor Broll, and Bertrand Souville. A DIY Power Monitor to Compare Mobile Energy Consumption in Situ. *Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services, ACM*, pages 416–421, 2013. doi : 10.1145/2493190.2494087.

[127] Abhinav Pathak, Y Charlie Hu, Ming Zhang, Paramvir Bahl, and Yi-Min Wang. Fine-Grained Power Modeling for Smartphones Using System Call Tracing. In *Proceedings of the sixth conference on Computer systems EuroSys 11*, page 153, 2011. ISBN 9781450306348. doi : 10.1145/1966445.1966460.

[128] Francesco Beneventi, Andrea Bartolini, Andrea Tilli, and Luca Benini. An effective gray-box identification procedure for multicore thermal modeling. *IEEE Transactions on Computers*, 63 (5) :1097–1110, may 2014. ISSN 00189340. doi : 10.1109/TC.2012.293.

[129] Rahul Murmuria, Jeffrey Medsger, Angelos Stavrou, and Jeffrey M. Voas. Mobile application and device power usage measurements. In *Proceedings of the 2012 IEEE 6th International Conference on Software Security and Reliability, SERE 2012*, pages 147–156. IEEE, jun 2012. ISBN 9780769547428. doi : 10.1109/SERE.2012.19.

[130] Nairan Zhang, Parameswaran Ramanathan, Kyu-Han Kim, and Sujata Banerjee. PowerVisor : A Battery Virtualization Scheme for Smartphones. In *Proceedings of the third ACM workshop on Mobile cloud computing and services - MCS '12*, page 37, New York, New York, USA, 2012. ACM Press. ISBN 9781450313193. doi : 10.1145/2307849.2307859.

[131] James Bornholt, Todd Mytkowicz, and Kathryn S. McKinley. The model is not enough : Understanding energy consumption in mobile devices. In *2012 IEEE Hot Chips 24 Symposium (HCS)*, pages 1–3. IEEE, aug 2012. ISBN 978-1-4673-8879-5. doi : 10.1109/HOTCHIPS.2012.7476509. URL http://ieeexplore.ieee.org/document/7476509/.

[132] Takeshi Kamiyama, Hiroshi Inamura, and Ken Ohta. A model-based energy profiler using online logging for Android applications. In *2014 7th International Conference on Mobile Computing and Ubiquitous Networking, ICMU 2014*, pages 7–13. IEEE, jan 2014. ISBN 978-1-4799-2231-4. doi : 10.1109/ICMU.2014.6799050.

[133] Yao Guo, Chengke Wang, and Xiangqun Chen. Understanding application-battery interactions on smartphones : A large-scale empirical study. *IEEE Access*, 5 :13387–13400, 2017. ISSN 21693536. doi : 10.1109/ACCESS.2017.2728620.

[134] Donghwa Shin, Kitae Kim, Naehyuck Chang, Woojoo Lee, Yanzhi Wang, Qing Xie, and Massoud Pedram. Online estimation of the remaining energy capacity in mobile systems considering system-wide power consumption and battery characteristics. In *Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC*, pages 59–64. IEEE, jan 2013. ISBN 9781467330299. doi : 10.1109/ASPDAC.2013.6509559.

[135] Qualcomm Technologies Inc. Snapdragon Profiler - Qualcomm Developer Network, 2019. URL https://developer.qualcomm.com/software/snapdragon-profiler.

[136] Pierre-André Cornillon and Eric Matzner-Løber. *Régression : Théorie et applications*. Statistique et probabilités appliquées. Springer, 2007. URL https://hal.archives-ouvertes.fr/hal-00955892.

[137] Frank J. Massey Jr. The kolmogorov-smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253) :68–78, 1951. doi : 10.1080/01621459.1951.10500769. URL https://www.tandfonline.com/doi/abs/10.1080/01621459.1951.10500769.

[138] Robert F. Engle. Autoregressive conditional heteroscedasticity with estimates of the variance of united kingdom inflation. *Econometrica*, 50(4) :987–1007, 1982. ISSN 00129682, 14680262. URL http://www.jstor.org/stable/1912773.

[139] Qualcomm Inc. Trepn Power Profiler - FAQs - Qualcomm Developer Network, 2019. URL https://developer.qualcomm.com/software/trepn-power-profiler/faq.

[140] Raja Wasim Ahmad, Abdullah Gani, Siti Hafizah Ab Hamid, Anjum Naveed, K. O. Kwangman, and Joel J.P.C. Rodrigues. A case and framework for code analysis–based smartphone application energy estimation. *International Journal of Communication Systems*, 30(10) :e3235, jul 2017. ISSN 10991131. doi : 10.1002/dac.3235.

[141] Mahmoud A. Bokhari, Bobby R. Bruce, Brad Alexander, and Markus Wagner. Deep parameter optimisation on Android smartphones for energy minimisation. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion on - GECCO '17*, pages 1501–1508, New York, New York, USA, 2017. ACM Press. ISBN 9781450349390. doi : 10.1145/3067695.3082519.

[142] Abram Hindle, Alex Wilson, Kent Rasmussen, E. Jed Barlow, Joshua Charles Campbell, and Stephen Romansky. GreenMiner : a hardware based mining software repositories software energy consumption framework. In *Proceedings of the 11th Working Conference on Mining Software Repositories - MSR 2014*, pages 12–21, Hyderabad, India, 2014. ACM Press. ISBN 9781450328630. doi : 10.1145/2597073.2597097.

[143] Chanmin Yoon, Dongwon Kim, Wonwoo Jung, Chulkoo Kang, and Hojung Cha. AppScope : Application Energy Metering Framework for Android Smartphone Using Kernel Activity Monitoring. *Presented as part of the 2012 USENIX Annual Technical Conference (USENIX ATC 12)*, pages 387–400, 2012. URL https://dl.acm.org/citation.cfm?id=2342857&CFID=208795690&CFTOKEN=39036451.

[144] Mark Gordon, Lide Zhang, and Birjodh Tiwana. PowerTutor, 2011. URL http://ziyang.eecs.umich.edu/projects/powertutor/index.html.

[145] Mario Linares-Vásquez, Gabriele Bavota, Carlos Eduardo Bernal Cárdenas, Rocco Oliveto, Massimiliano Di Penta, and Denys Poshyvanyk. Optimizing energy consumption of GUIs in Android apps : a multi-objective approach. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2015*, pages 143–154, New York, New York, USA, 2015. ACM Press. ISBN 9781450336758. doi : 10.1145/2786805.2786847.

[146] Majid L. Altamimi and Kshirasagar Naik. A Computing Profiling Procedure for Mobile Developers to Estimate Energy Cost. In *Proceedings of the 18th ACM International Conference on Modeling, Analysis and Simulation of Wireless and Mobile Systems - MSWiM '15*, pages 301–305, New York, New York, USA, 2015. ACM Press. ISBN 9781450337625. doi : 10.1145/2811587.2811627.

[147] Ding Li, Shuai Hao, Jiaping Gui, and William G.J. Halfond. An Empirical Study of the Energy Consumption of Android Applications. In *2014 IEEE International Conference on Software Maintenance and Evolution*, pages 121–130. IEEE, sep 2014. ISBN 978-1-4799-6146-7. doi : 10.1109/ICSME.2014.34.

[148] Tianxing Jin, Songtao He, and Yunxin Liu. Towards Accurate GPU Power Modeling for Smartphones. In *Proceedings of the 2nd Workshop on Mobile Gaming - MobiGames '15*, pages 7–11, New York, New York, USA, 2015. ACM Press. ISBN 9781450334990. doi : 10.1145/2751496.2751502.

[149] Jemin Lee, Hyunwoo Joe, and Hyungshin Kim. Automated power model generation method for smartphones. *IEEE Transactions on Consumer Electronics*, 60(2) :190–197, may 2014. ISSN 00983063. doi : 10.1109/TCE.2014.6851993. URL http://ieeexplore.ieee.org/document/6851993/.

[150] Lee Jaymin, Joe Hyunwoo, and Kim Hyungshin. Smart phone power model generation using use pattern analysis. In *2012 IEEE International Conference on Consumer Electronics (ICCE)*, pages 412–413. IEEE, jan 2012. ISBN 978-1-4577-0231-0. doi : 10.1109/ICCE.2012.6161925.

[151] Tero Arpinen, Erno Salminen, Timo D. Hämäläinen, and Marko Hännikäinen. MARTE profile extension for modeling dynamic power management of embedded systems. *Journal of Systems Architecture*, 58(5) :209–219, apr 2012. ISSN 13837621. doi : 10.1016/j.sysarc.2011.01.003. URL https://www-sciencedirect-com.lama.univ-amu.fr/science/article/pii/S1383762111000154.

[152] Immanuel König, Aq Memon, and Klaus David. Energy consumption of the sensors of Smartphones. In *10th International Symposium on Wireless Communications Systems (ISWCS)*, pages 723–727, aug 2013. ISBN 9783800735297.

[153] Shuai Hao, Ding Li, William G J Halfond, and Ramesh Govindan. Estimating Mobile Application Energy Consumption Using Program Analysis. In *Proceedings of the 2013 International Conference on Software Engineering*, ICSE '13, pages 92–101, Piscataway, NJ, USA, may 2013. IEEE Press. ISBN 978-1-4673-3076-3. doi : 10.1109/ICSE.2013.6606555.

[154] A. A. Nacci, F. Trovò, F. Maggi, M. Ferroni, A. Cazzola, D. Sciuto, and M. D. Santambrogio. Adaptive and flexible smartphone power modeling. *Mobile Networks and Applications*, 18 (5) :600–609, oct 2013. ISSN 1383469X. doi : 10.1007/s11036-013-0470-y. URL http://link.springer.com/10.1007/s11036-013-0470-y.

[155] Minyong Kim and Sung Woo Chung. Accurate GPU power estimation for mobile device power profiling. *Digest of Technical Papers - IEEE International Conference on Consumer Electronics*, pages 183–184, 2013. ISSN 0747668X. doi : 10.1109/ICCE.2013.6486850.

[156] Minyong Kim, Joonho Kong, and Sung Woo Chung. An online power estimation technique for multi-core smartphones with advanced display components. In *2012 IEEE International Conference on Consumer Electronics (ICCE)*, pages 666–667. IEEE, jan 2012. ISBN 978-1-4577-0231-0. doi : 10.1109/ICCE.2012.6162019.

[157] Kaneda Yusuke, Takumi Okuhira, Ishihara Tohru, Hisazumi Kenji, Kamiyama Takeshi, and Katagiri Masaji. A run-time power analysis method using OS-observable parameters for mobile terminals. In *Proc. of International Conference on Embedded Systems and Intelligent Technology(ICESIT)*, volume 2010, pages 1–6, 2010. doi : 10.1109/ICMU.2014.6799050.

[158] Selim Gurun and Chandra Krintz. A run-time, feedback-based energy estimation model For embedded devices. In *Proceedings of the 4th international conference on Hardware/software codesign and system synthesis - CODES+ISSS '06*, page 28, New York, New York, USA, 2006. ACM Press. ISBN 1595933700. doi : 10.1145/1176254.1176264.

[159] ARM Inc. Technologies | big.LITTLE – Arm Developer, 2018. URL https://www.arm.com/why-arm/technologies/big-little.

[160] Brendon Gregg. CPU Utilization is Wrong, 2017. URL http://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html.

[161] D.R.Saravanan. Understanding Processor Utilization on POWER Systems - AIX, 2010. URL https://www.ibm.com/developerworks/community/wikis/home?lang=en%5C#!/wiki/PowerSystems/page/UnderstandingCPUutilizationonAIX.

[162] Aaron Kili. Understand Linux Load Averages and Monitor Performance of Linux. URL https://www.tecmint.com/understand-linux-load-averages-and-monitor-performance/.

[163] Jingwen Leng, Tayler Hetherington, Ahmed ElTantawy, Syed Gilani, Nam Sung Kim, Tor M. Aamodt, and Vijay Janapa Reddi. GPUWattch : Enabling Energy Optimizations in GPGPUs. *Proceedings of the 40th Annual International Symposium on Computer Architecture - ISCA '13*, 41 :487, 2013. ISSN 0163-5964. doi : 10.1145/2485922.2485964.

[164] Sunpyo Hong and Hyesoon Kim. An integrated GPU power and performance model. In *ACM SIGARCH Computer Architecture News*, volume 38 of *ISCA '10*, page 280, New York, NY, USA, 2010. ACM. ISBN 9781450300537. doi : 10.1145/1816038.1815998.

[165] Ying Dar Lin, Ekarat Rattagan, Yuan Cheng Lai, Li Pin Chang, Yun Chien Yo, Cheng Yuan Ho, and Shun Lee Chang. Calibrating parameters and formulas for process-level energy consumption profiling in smartphones. *Journal of Network and Computer Applications*, 44 : 106–119, sep 2014. ISSN 10958592. doi : 10.1016/j.jnca.2014.04.014.

[166] Google. Services | Android Developers. URL https://developer.android.com/guide/components/services.html.

[167] Samsung. Samsung Opensource Release Center, 2019. URL http://opensource.samsung.com/.

[168] Scott Horn. Understanding application priority and process states - application development, Sep 2014. URL https://www.androidcookbook.info/application-development/understanding-application-priority-and-process-states.html.

[169] Temperature-dependent Electromigration Reliability. In *Electrothermal Analysis of VLSI Systems*, pages 121–155. Springer US, Boston, MA, 2002. ISBN 978-0-306-47024-0. doi : 10.1007/0-306-47024-1_6. URL https://doi.org/10.1007/0-306-47024-1_6.

[170] E Karl, D Blaauw, D Sylvester, and T Mudge. Reliability modeling and management in dynamic microprocessor-based systems. In *2006 43rd ACM/IEEE Design Automation Conference*, pages 1057–1060, 2006. ISBN 0738-100X VO -. doi : 10.1145/1146909.1147174.

[171] Sidartha Azevedo Lobo De Carvalho, Daniel Carvalho Da Cunha, and Abel Guilhermino Da Silva-Filho. Autonomous Power Management for Embedded Systems Using a Non-linear Power Predictor. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 22–29. IEEE, aug 2017. ISBN 978-1-5386-2146-2. doi : 10.1109/DSD.2017.68.

[172] Mel Stockman, Mariette Awad, Rahul Khanna, Christian Le, Howard David, Eugene Gorbatov, and Ulf Hanebutte. A novel approach to memory power estimation using machine learning. In *2010 International Conference on Energy Aware Computing*, pages 1–3, 2010. ISBN 978-1-4244-8274-0. doi : 10.1109/ICEAC.2010.5702284.

[173] Rance Rodrigues, Arunachalam Annamalai, Israel Koren, and Sandip Kundu. A study on the use of performance counters to estimate power in microprocessors. *IEEE Transactions on Circuits and Systems II : Express Briefs*, 60(12) :882–886, 2013. ISSN 15497747. doi : 10.1109/TCSII.2013.2285966.

[174] Adam Kerin. Power vs. Performance Management of the CPU | Qualcomm, 2013. URL https://www.qualcomm.com/news/onq/2013/10/25/power-vs-performance-management-cpu.

[175] Inc. Samsung. Exynos 9 Series 8895 Processor : Specs, Features | Samsung Exynos, 2017. URL http://www.samsung.com/semiconductor/minisite/exynos/products/mobileprocessor/exynos-9-series-8895/.

[176] Jian Li, Jin Xiao, James Won Ki Hong, and Raouf Boutaba. FSM-based Wi-Fi power estimation method for smart devices. In *Proceedings of the 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM 2015*, pages 147–155. IEEE, may 2015. ISBN 9783901882760. doi : 10.1109/INM.2015.7140287.

[177] Hang Xie, Hao Tang, and Yu He Liao. Time series prediction based on narx neural networks : An advanced approach. In *Proceedings of the 2009 International Conference on Machine Learning and Cybernetics*, volume 3, pages 1275–1279. IEEE, jul 2009. ISBN 9781424437030. doi : 10.1109/ICMLC.2009.5212326.

[178] Francesco Paterna and Tajana Simunic Rosing. Modeling and mitigation of extra-SoC thermal coupling effects and heat transfer variations in mobile devices. In *2015 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2015*, pages 831–838. IEEE, nov 2016. ISBN 9781467383882. doi : 10.1109/ICCAD.2015.7372657.

[179] Andrea Bartolini, Matteo Cacciari, Andrea Tilli, Luca Benini, and Matthias Gries. A Virtual Platform Environment for Exploring Power, Thermal and Reliability Management Control Strategies in High-performance Multicores. In *Proceedings of the 20th Symposium on Great Lakes Symposium on VLSI*, GLSVLSI '10, pages 311–316, New York, NY, USA, 2010. ACM. ISBN 978-1-4503-0012-4. doi : 10.1145/1785481.1785553.

[180] System Identification : The Bases BT - Digital Control Systems : Design, Identification and Implementation. pages 201–245. Springer London, London, 2006. ISBN 978-1-84628-056-6. doi : 10.1007/978-1-84628-056-6_5. URL https://doi.org/10.1007/978-1-84628-056-6_5.

[181] Eric H.K Fung, Y.K Wong, H.F Ho, and Marc P Mignolet. Modelling and prediction of machining errors using ARMAX and NARMAX structures. *Applied Mathematical Modelling*, 27(8) :611–627, 2003. doi : 10.1016/S0307-904X(03)00071-4.

[182] Lennart Ljung. *System Identification*. American Cancer Society, 1999. ISBN 9780471346081. doi : 10.1002/047134608X.W1046. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/047134608X.W1046.

[183] Ioan D Landau, N M'Sirdi, and M M'Saad. Techniques de modélisation récursive pour l'analyse spectrale paramétrique adaptative. *Revue de Traitement du Signal*, 3 :183–204, 1986.

[184] Futuremark Oy. PCMark for Android Benchmark - Android Apps on Google Play, 2019. URL https://play.google.com/store/apps/details?id=com.futuremark.pcmark.android.benchmark.

[185] Futuremark Oy. 3DMark - The Gamer's Benchmark - Android Apps on Google Play, 2019. URL https://play.google.com/store/apps/details?id=com.futuremark.dmandroid.application.

[186] AnTuTu. AnTuTu Benchmark - Android Apps on Google Play, 2019. URL https://play.google.com/store/apps/details?id=com.antutu.ABenchMark.

[187] Primate Labs Inc. Geekbench 4 - Android Apps on Google Play, 2019. URL https://play.google.com/store/apps/details?id=com.primatelabs.geekbench.

[188] Jinchao Chen, Chenglie Du, Fei Xie, and Bin Lin. Scheduling non-preemptive tasks with strict periods in multi-core real-time systems. *Journal of Systems Architecture*, 90 :72–84, oct 2018. ISSN 13837621. doi : 10.1016/j.sysarc.2018.09.002.

[189] Dots : A game about connecting - apps on google play. URL https://play.google.com/store/apps/details?id=com.nerdyoctopus.gamedots&hl=en.

[190] Monument valley - apps on google play. URL https://play.google.com/store/apps/details?id=com.ustwo.monumentvalley&hl=en.

[191] Zhiwei Gao, Carlo Cecati, and Steven X. Ding. A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part I : Fault Diagnosis. *IEEE Transactions On Industrial Electronics*, 62(6) :3768 – 3774, jun 2015. ISSN 0278-0046. doi : 10.1109/TIE.2015.2417501.

[192] Zhiwei Gao, Carlo Cecati, and Steven X. Ding. A survey of fault diagnosis and fault-tolerant techniques-part II : Fault diagnosis with knowledge-based and hybrid/active approaches. *IEEE Transactions on Industrial Electronics*, 62(6) :3768–3774, 2015. ISSN 02780046. doi : 10.1109/TIE.2015.2419013.

[193] Andreas Steininger. Testing and built-in self-test – A survey. *Journal of Systems Architecture*, 46(9) :721–747, 2000. ISSN 13837621. doi : 10.1016/S1383-7621(99)00041-7.

[194] Somnath Deb, Krishna R. Pattipati, Vijay Raghavan, Mojdeh Shakeri, and Roshan Shrestha. Multi-Signal Flow Graphs : A Novel Approach for System Testability Analysis and Fault Diagnosis. *IEEE Aerospace and Electronic Systems Magazine*, 10(5) :14–25, may 1995. ISSN 08858985. doi : 10.1109/62.373993.

[195] J.W. Sheppard. Maintaining diagnostic truth with information flow models. In *Conference Record. AUTOTESTCON '96*, pages 447–454. IEEE, 1996. ISBN 0-7803-3379-9. doi : 10.1109/AUTEST.1996.547773.

[196] Guangfan Zhang. *Optimum Sensor Localization/Selection In A Diagnostic/Prognostic Architecture*. Phd dissertation, University System of Georgia, 2005.

[197] Feng Wu Wang, Jun You Shi, and Lu Wang. Method of diagnostic tree design for system-level faults based on dependency matrix and fault tree. In *2011 IEEE 18th International Conference on Industrial Engineering and Engineering Management, IE and EM 2011*, number PART 2, pages 1113–1117. IEEE, sep 2011. ISBN 9781612844473. doi : 10.1109/IEEM.2011.6035351.

[198] Yiqian Cui, Junyou Shi, and Zili Wang. An analytical model of electronic fault diagnosis on extension of the dependency theory. *Reliability Engineering I& System Safety*, 133 :192–202, 2015. ISSN 09518320. doi : 10.1016/j.ress.2014.09.015.

[199] Dimitris Gizopoulos, Mihalis Psarakis, Sarita V Adve, Pradeep Ramachandran, Siva Kumar Sastry Hari, Daniel Sorin, Albert Meixner, Arijit Biswas, and Xavier Vera. Architectures for online error detection and recovery in multicore processors. *2011 Design, Automation I& Test in Europe*, (c) :1–6, mar 2011. ISSN 1530-1591. doi : 10.1109/DATE.2011.5763096.

[200] Nidhi Aggarwal and Parthasarathy Ranganathan. Configurable isolation : building high availability systems with commodity multi-core processors. In *Acm Sigarch . . .* , volume 35, pages 470–481, New York, New York, USA, 2007. ACM Press. ISBN 9781595937063. doi : 10.1145/1250662.1250720.

[201] Christopher LaFrieda, Engin Ipek, José F. Martínez, and Rajit Manohar. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In *Proceedings of the International Conference on Dependable Systems and Networks*, pages 317–326. IEEE, jun 2007. ISBN 0769528554. doi : 10.1109/DSN.2007.100.

[202] S S Mukherjee, M. Kontz, and S K Reinhardt. Detailed design and evaluation of redundant multithreading alternatives. In *29th Annual International Symposium on Computer Architecture, 2002. Proceedings*, pages 99–110. IEEE, 2002. ISBN 076951605X. doi : 10.1109/ISCA.2002.1003566.

[203] Smitha Shyam, Kypros Constantinides, Sujay Phadke, Valeria Bertacco, and Todd Austin. Ultra low-cost defect protection for microprocessor pipelines. In *Proceedings of the 12th international conference on Architectural support for programming languages and operating systems - ASPLOS-XII*, volume 41, page 73, New York, New York, USA, 2006. ACM Press. ISBN 1595934510. doi : 10.1145/1168857.1168868.

[204] T.M. M Austin. DIVA : A reliable substrate for deep submicron microarchitecture design. In *32nd Annual Int. Symp. on Microarchitecture, (MICRO-32)*, number November, pages 196–207. IEEE Comput. Soc, 1999. ISBN 0-7695-0437-X. doi : 10.1109/MICRO.1999.809458.

[205] Albert Meixner, Michael E. Bauer, and Daniel Sorin. Argus : Low-Cost, Comprehensive Error Detection in Simple Cores. In *40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007)*, pages 210–222. IEEE, 2007. ISBN 0-7695-3047-8. doi : 10.1109/MICRO.2007.18.

[206] Nicholas J. Wang and Sanjay J. Patel. ReStore : Symptom-based soft error detection in microprocessors. *IEEE Transactions on Dependable and Secure Computing*, 3(3) :188–201, jul 2006. ISSN 15455971. doi : 10.1109/TDSC.2006.40.

[207] Man-Lap Li, Pradeep Ramachandran, Swarup Kumar Sahoo, Sarita V Adve, Vikram S Adve, and Yuanyuan Zhou. Understanding the Propagation of Hard Errors to Software and Implications for Resilient System Design. *SIGOPS Oper. Syst. Rev.*, 42(2) :265–276, 2008. ISSN 0163-5980. doi : 10.1145/1353535.1346315.

[208] Sourav Sinha, Neeraj Kumar Goyal, and Rajib Mall. Early prediction of reliability and availability of combined hardware-software systems based on functional failures. *Journal of Systems Architecture*, 92 :23–38, jan 2019. ISSN 13837621. doi : 10.1016/j.sysarc.2018.10.007.

[209] IEEE Computer Society. Test Technology Standards Committee., Institute of Electrical and Electronics Engineers., and IEEE-SA Standards Board. *IEEE standard for access and control of instrumentation embedded within a semiconductor device - 1687*. 2014. ISBN 9780738194165. doi : 10.1109/IEEESTD.2014.6974961.

[210] Farrokh Ghani Zadegan, Dimitar Nikolov, and Erik Larsson. On-Chip Fault Monitoring Using Self-Reconfiguring IEEE 1687 Networks. *IEEE Transactions on Computers*, 67(2) :237–251, feb 2018. ISSN 0018-9340. doi : 10.1109/TC.2017.2731338.

[211] Andreas Löfwenmark and Simin Nadjm-Tehrani. Fault and timing analysis in critical multi-core systems : A survey with an avionics perspective. *Journal of Systems Architecture*, 87 :1–11, jun 2018. ISSN 13837621. doi : 10.1016/j.sysarc.2018.04.001.

[212] Samir Benmoussa and Mohand A. Djeziri. Remaining useful life estimation without needing for prior knowledge of the degradation features. *IET Science, Measurement I& Technology*, 11(8) :1071–1078, nov 2017. ISSN 1751-8822. doi : 10.1049/iet-smt.2017.0005. URL https://digital-library.theiet.org/content/journals/10.1049/iet-smt.2017.0005.

[213] Mohand A. Djeziri, B. Ould Bouamama, and R. Merzouki. Modelling and robust FDI of steam generator using uncertain bond graph model. *Journal of Process Control*, 19(1) :149–162, 2009. ISSN 09591524. doi : 10.1016/j.jprocont.2007.12.009.

[214] Olivier Adrot, Didier Maquin, and José Ragot. Fault detection with model parameter structured uncertainties. In *5th European Control Conference, ECC'99*, page CDROM, 1999.

[215] Joaquim Armengol, Luoise Travé-Massuyès, Josep Vehí, and Josep Lluís de la Rosa. A Survey on interval model simulators and their properties related to fault detection. *Annual Reviews in Control*, 24 :31–39, jan 2000. ISSN 13675788. doi : 10.1016/S1367-5788(00)00002-X.

[216] Linglai Li and Donghua Zhou. Fast and robust fault diagnosis for a class of nonlinear systems : Detectability analysis, 2004. ISSN 00981354.

[217] Mohand A. Djeziri, R. Merzouki, B. Ould Bouamama, and G. Dauphin-Tanguy. Fault detection of backlash phenomenon in mechatronic system with parameter uncertainties using bond graph approach. In *2006 IEEE International Conference on Mechatronics and Automation, ICMA 2006*, volume 2006, pages 600–605. IEEE, jun 2006. ISBN 1424404665. doi : 10.1109/ICMA.2006.257639.

[218] Michèle Basseville. On-board Component Fault Detection and Isolation Using the Statistical Local Approach. *Automatica*, 34(11) :1391–1415, nov 1998. ISSN 00051098. doi : 10.1016/S0005-1098(98)00086-7.

[219] Chuanhu Ge, Hongying Yang, Hao Ye, and Guizeng Wang. A Fast Leak Locating Method Based on Wavelet Transform. *Tsinghua Science and Technology*, 14(5) :551–555, 2009. ISSN 10070214. doi : 10.1016/S1007-0214(09)70116-6.

[220] S. Benmoussa, Mohand A. Djeziri, B. Ould Bouamama, and R. Merzouki. Empirical mode decomposition applied to fluid leak detection and isolation in process engineering. In *18th Mediterranean Conference on Control and Automation, MED'10*, pages 1537–1542. IEEE, jun 2010. ISBN 978-1-4244-8091-3. doi : 10.1109/MED.2010.5547829.

[221] Thi Bich Lien Nguyen. *Approche statistique pour le pronostic de défaillance : application à l'industrie du semi-conducteur*. PhD thesis, 2016. URL http://www.theses.fr/2016AIXM4310. Thèse de doctorat dirigée par Ouladsine, MustaphaDjeziri, Mohand Arab et Ananou, Bouchra Mathématiques et informatique Aix-Marseille 2016.

[222] Nvidia. Graphics Processing Unit (GPU)|NVIDIA, 1999. URL https://www.nvidia.com/object/gpu.html.

[223] Pcmag. GPU Definition from PC Magazine Encyclopedia, 2019. URL https://www.pcmag.com/encyclopedia/term/43886/gpu.

[224] NXP Semiconductors. i.MX 6SoloX Applications Processors for Consumer Products, 2018. URL https://www.nxp.com/docs/en/data-sheet/IMX6SXCEC.pdf.

Abstract

Systems on Chip are increasingly embedded in safety-critical systems, such as aeronautical systems and energy production equipment. Such technological evolution allows for significant improvements in performance but presents limits in terms of reliability and security. Therefore, the development of new tools for the monitoring and diagnosis of embedded electronic systems—Systems on Chip, in particular—is currently one of the scientific challenges to overcome, in order to ensure a broader and safer use of these systems in safety-critical equipment.

The work presented in this thesis aims to develop an approach for detecting and identifying drifts in embedded Systems of Chips characteristics and performance. The proposed approach is based on an incremental model built from reusable and exchangeable modules able to adapt and accommodate the broad range of Systems on Chips available on the market. This model is then used to estimate a set of characteristics relating to the state of operation of the SoC.

The diagnostic algorithm developed in this work consists of generating drift signals though the online comparison of the estimated characteristics to those measured. Then, the assessment of residuals and decision making are performed by statistical methods appropriate to the nature of each drift.

The developed approach has been experimentally validated on different Systems on Chip, as well as on a demonstrator developed as part of this work. The obtained experimental results validate and show the efficiency and robustness of the incremental model and the monitoring algorithm.

Resumé

Les Systèmes-sur-Puce (*Systems on Chip*, SoC) sont de plus en plus embarqués dans des systèmes à risque comme les systèmes aéronautiques et les équipements de production d'énergie. Cette évolution technologique permet un gain de temps et de performance, mais présente des limites en termes de fiabilité et de sécurité. Ainsi, le développement d'outils de surveillance et de diagnostic des systèmes électroniques embarqués, en particuliers les SoC, est devenu l'un des verrous scientifiques à lever pour assurer une large utilisation de ces systèmes dans les équipements à risque en toute sécurité.

Ce travail de thèse s'inscrit dans ce contexte, et a pour objectif le développement d'une approche de détection et identification des dérives des performances des SoC embarqués. L'approche proposée est basée sur un modèle incrémental, construit à partir de modules réutilisables et échangeables pour correspondre à la large gamme de SoC existants sur le marché. Le modèle est ensuite utilisé pour estimer un ensemble de caractéristiques relatives à l'état de fonctionnement du SoC.

L'algorithme de diagnostic développé dans ce travail consiste à générer des indices de dérives par la comparaison en ligne des caractéristiques estimées à celles mesurées. L'évaluation des résidus et la prise de décision sont réalisées par des méthodes statistiques appropriées à la nature de chaque indice de dérive.

L'approche développée a été validée expérimentalement sur des SoC différents, ainsi que sur un démonstrateur développé dans le cadre de ce travail. Les résultats expérimentaux obtenus, montrent l'efficacité et la robustesse de l'approche développée.