Incremental Modeling and Monitoring of Embedded CPU-GPU Chips

Oussama Djedidi, Mohand Djeziri

processes

Article

Incremental Modeling and Monitoring of Embedded CPU-GPU Chips

Oussama Djedidi * and Mohand Djeziri

Aix-Marseille University, Université de Toulon, CNRS, LIS, 13397 Marseille CEDEX 20, France; mohand.djeziri@lis-lab.fr

* Correspondence: oussama.djedidi@lis-lab.fr

Received: 25 April 2020; Accepted: 30 May 2020; Published: 9 June 2020

Abstract: This paper presents a monitoring framework to detect drifts and faults in the behavior of the central processing unit (CPU)-graphics processing unit (GPU) chips powering them. To construct the framework, an incremental model and a fault detection and isolation (FDI) algorithm are hereby proposed. The reference model is composed of a set of interconnected exchangeable subsystems that allows it to be adapted to changes in the structure of the system or operating modes, by replacing or extending its components. It estimates a set of variables characterizing the operating state of the chip from only two global inputs. Then, through analytical redundancy, the estimated variables are compared to the output of the system in the FDI module, which generates alarms in the presence of faults or drifts in the system. Furthermore, the interconnected nature of the model allows for the direct localization and isolation of any detected abnormalities. The implementation of the proposed framework requires no additional instrumentation as the used variables are measured by the system. Finally, we use multiple experimental setups for the validation of our approach and also proving that it can be applied to most of the existing embedded systems.

Keywords: analytical redundancy; embedded systems; modeling; monitoring; smartphones; system-on-chip

---

1. Introduction

Heterogeneous systems-on-chips (SoC) combine more than one type of processor, generally central processing units (CPU) and graphics processing units (GPU) [1]. They rose to popularity in the early 2000s, thanks to their suitability for most households and entertainment uses. Nowadays, heterogeneous SoCs are at the hearts of most modern computer systems and handheld devices, including those used in safety-critical systems (military, aerospace, automobile, etc.) [2]. In these increasingly embedded and mobile systems, manufacturers are evermore dealing with the challenge of designing chips offering high performance while maintaining minimal power consumption and manageable thermal output.

The goal of the project behind this study is to develop touchscreens to serve as both the primary displays and controls in the cockpits of the future. Reliability being a primordial aspect of these systems, in this particular study, we propose a monitoring framework for the surveillance of the embedded systems powering the said touchscreens.

To monitor such chips, we propose an incremental model with an incorporated fault detection and isolation (FDI) algorithm. Together, the pair allows for the early detection of faults and drifts in the system [3]. The idea is to construct an incremental and interconnected modeling structure that is composed of simpler subsystems, each written to estimate only one variable or characteristic [4]. This approach would result in separating the different—and mostly irreconcilable—dynamics, like discrete and continuous ones. This modeling methodology constitutes the first contribution of this work.

The FDI algorithm aims to detect errors or anomalies in the behavior of the system, mainly in power consumption, operating temperature, and possible software bugs affecting critical system programs or drivers such as the frequency scaling governor. The algorithm relies on analytical redundancy, and as will be described later, the interconnected structure of our model makes it easy to detect and isolate such anomalies, since every subsystem only estimates one variable.

The main advantage of our modeling framework and monitoring algorithm is that they only rely on data and sensors present on most modern chips and therefore can be deployed on most of the current and forthcoming SoCs with no additional development nor adaptation—apart from model adapting and training. Once the program is implemented, one can easily use its alarms and fault indicators to watch over the operating state of the device and isolate faulty subsystems. It can also be used to investigate the effect of these errors on the rest of the system, and even view wear-traits for diagnosis and prognosis purposes.

The remainder of this paper is organized into eight sections. We start by exploring and summarizing established works studying the subject of modeling and monitoring embedded chips, in Section 2. Then, the proposed modeling and monitoring approach is presented in Section 3, and the experimental setup in Section 4. In Section 5, we detail the modeling process of each subsystem and explain the modeling choices. Section 6 is used to present the FDI algorithm and describe the process of residual generation and evaluation (the decision-making process). In Sections 7 and 8, we proceed to present and discuss experimental results where The fault detection algorithm is validated by examining residuals in normal and faulty scenarios. Finally, we draw a conclusion from the collected data and obtained results.

2. Literature Overview

In the approach proposed hereafter, we select a set of variables that characterize the operating state of the SoC (functioning correctly, normal power consumption, etc.). In the first step, our monitoring framework estimates these variables. Then, it monitors them to detects faults and drifts.

To our knowledge, there is no work treating the problem of online monitoring the operating state of CPU-GPU chips as we do, which in itself shows the gap we are trying to address in the literature, and indicate novelty of this work. Nevertheless, some works study problems that are close to ours. In the remainder of this section, we explore the main research done in these areas, limiting our scope to works done on embedded or mobile SoCs.

2.1. Power Consumption Modeling in Embedded SoCs

Power consumption in embedded systems is a vivid and active field of research, particularly in the case of smartphones. These devices are abundantly available, and easy to program. More importantly, they come with a limited power supply, which incited developers to study the battery life [5], construct energy and power consumption models [6–8], and closely follow the influence of user actions and applications on power consumption [9]. Furthermore, studying the embedded systems' power profile has led to improving energy efficiency [10], reliability [11,12], and also monitoring and detecting anomalies [13–15] and energy hogs [16].

There exists three main literature reviews detailing power profiling and modeling in smartphones [17–19]. In the first review, Hoque et al. [17] gave an overview of the steps of building an energy profiler. Then, they went onto detailing the different energy measurement mechanisms (external instruments and self-metering), types of models (utilization-based, event-based, and code-based), modeling philosophies (white-box vs. black-box), and profiling schemes. Finally, the authors proposed a taxonomy of the studied profilers based on their deployment; either internally on the device, or externally, and for where the model is constructed, on or off the device.

Amongst the surveyed profilers and models, several fall within the scope of our work according to their granularity. The latter describes the level at which the profiler can estimate energy consumption [18]. Of these profilers, there are those built by system providers like Google's *Android Power Profiler* [20,21], and Qualcomm's *Trepn Profiler* [22]. We also cite works published in the literature such as *PowerBooter* [23], *Sesame* [24], and *DevScope* [25].

The second review mentioned essentially the same works, but it looked at them through the lens of their implementation (hardware and software-based) [18], whereas the third review focused on network simulation [19].

Apart from the works mentioned in the reviews, Kim et al. [26] proposed a polynomial model of which every term reflects the power consumption of one component of the smartphone, whereas a later version of this study improved upon the power estimation for the GPU [27]. More recently, several new models were also published, either for the system as a whole [28–32], or just the CPU [7,33]. Some of them even offered better accuracy [7,28,33] and proved that software profiling is as and reliable as the hardware counterpart [29,31].

Most of the models estimating power consumption use either finite state machines (FSM) like *Eprof* [34] and *DevScope* [25], or the more popular regression-based models as is the case for at least Kim et al. [26], Kim et al. [27], Shukla et al. [30], and Xu et al. [35], or a combination of both Jung et al. [25]. There is also a new trend of using nonlinear methods like neural networks [36–38].

Black-box techniques—like regression and data-fitting—are popular for these systems, mainly because of the latter's complexity and the interactions of physical phenomena underhand. Moreover, the inputs of the models constructed by machine learning or regression-based approaches are not subject to a formal proof of interaction with the estimated outputs [23]. The choice of inputs is generally the result of observations and data analysis, and experimental tests [17].

2.2. Temperature Modeling in Embeded SoCs

The thermal performance of the SoC is an important variable for its proper functioning and determining its life cycle. Thus, temperature modeling is usually achieved for optimization purposes. During the design phase, it is established to define the temperature thresholds [39] or the size of the heat evacuation apparatus (radiators, heat pipes, etc.). During the operating phase, certain models are built to optimize thermal behavior [12,40]. While, other models are used to study the reliability of the system by investigating the effects of internal and external temperatures on the chip [41], or on the lifetime of the system [42,43] to improve it [44].

In addition to the models designed for optimization, the literature contains models focusing on energy and thermal management. For instance, Mercati et al. [45] proposed a method to adapt the operating conditions (processors' frequencies, screen brightness, etc.) to the needs of the user (usage, applications running, etc.). Other works aim at the same goal by proposing new scheduling policies [11,46] or new Dynamic Voltage and Frequency Scaling (DVFS) algorithms [12].

The closest work to our framework—models used online estimation and monitoring—is the *Therminator* [47] simulator and its second version *ThermTap* [48]. These programs are developed for the online estimation of temperature in mobile devices for debugging purposes.

2.3. SoC Monitoring and Diagnosis

In the monitoring field, Gao et al. [49] summarized most of the work on fault diagnosis and detection in a two parts survey. In this survey, they categorized and divided these works into four categories: model-based, signal-based, knowledge-based, and hybrid/active approaches. The survey also went to highlight fault diagnosis application, and fault tolerance methods [50].

This study falls into the second category (signal-based approaches [49]). However, most of the existing work does not specifically study the SoC independently but rather studies electronic boards as a whole. These boards are often viewed as discrete systems whose functioning depends on the proper operation of all the components, with the assumption that there is a strong dependence of operation between each of these components [51]. Accordingly, the developed diagnosis methods are based on causal models such as multi-signal flow graph [52], information flow model [53], directed graph [54], and the fault tree [55].

Cui et al. [56] introduced an improved dependency model and used it for fault detection and isolation on an electronic board harboring a CPU-GPU chip. The chip, like our case, was used in the field of avionics. The new model is capable of eliminating multiple faults by disconnecting them from the main model via switches in the Dependency Graphical Model (DGM). The model also is the first implementation of dynamic reconfiguration concepts, and takes into consideration the malfunctions of tests.

Additionally, Gizopoulos et al. [57] provided a classification and a detailed study of existing online error detection techniques applied to multicore processor architectures. These approaches are classified into four main categories: redundant execution [58,59], periodic Built-In Self-Test (BIST) approaches [60], dynamic verification approaches [61,62], and anomaly detection approaches [63,64]. The results of this comparative study highlight the effectiveness of the dynamic verification approaches in targeting transient faults, permanent faults, and design bugs. The latter was the focus of the recent work by Sinha et al. [65] who studied the reliability of the hardware-software combined system at the design stage and proposed a unified functional model that can be simulated to detect potential failures.

In a different approach, Mercati et al. [45] viewed reliability as a convex optimization problem and proposed the Workload-Aware Reliability Management (WARM); an optimal policy for multicore systems, while Zadegan et al. [66] used the networking capabilities of the Institute of Electrical and Electronics Engineers Standard (IEEE Std) 1687–2014 standard for in-field monitoring of embedded systems and fast localization of the faults.

Finally, Löfwenmark and Nadjm-Tehrani [67] published a survey that regrouped works focusing on multicore systems in avionics. They suggested that there is an increased sensitivity to faults due to shrinking transistor sizes, and highlighted areas where research is still needed.

2.4. Contributions

All of the works mentioned above describe novel methods with great results in their respective fields. Each of them focuses on a specific aspect or a component of the embedded system or the SoC (reliability or power consumption modeling, hardware components or software components, etc.). This work, on the other hand, presents a new modeling framework that is capable of estimating and monitoring all of the SoC's characteristics variables online at once, and link of the software and hardware parts. The obvious obstacle to overcome, however, was creating a model capable of estimating all of these variables.

The dynamics of CPU-GPU chips are quite diverse and can be considered from different angles. They can be described with discrete variables (like the CPU load) or seen as discrete event subsystems (such as the frequency), not to mention the nonlinear continuous dynamics like temperature and power consumption. To model all of these dynamics, the chip is viewed as a system with a variable structure, in which the speed, power consumption, and the generated heat depend on both the software load and the operation mode (power saving, performance, etc.). This causal link between the variables allows for the creation of an incremental modeling structure that simplifies the modeling process. Hence, the emphasis in this work is on streamlining and organizing both the modeling and monitoring processes into a manageable and adaptable framework, rather than presenting novel modeling for all the individual dynamics. Moreover, the incremental model naturally accounts for the variability in the structure of the system and estimates different dynamics at once, and more importantly, it is easily adaptable to changes in components or operating modes. Additionally, it gives the model builder the freedom of choice between writing new novel models—as was done hereafter—or selecting ones from the library of existing models—like *PowerBooter* [23]—allowing for better monitoring results and greater control over the whole model.

3. Modeling and Monitoring Approach

The monitoring approach proposed in this paper is complementary to the methods mentioned in the previous paragraph because it aims to provide early detection of drifts in the system's characteristics caused by wear, or harsh conditions, or over-solicitation.

Since it is going to be deployed in safety-critical systems and environments (airplanes), the monitoring algorithm's main job will be to ensure that the SoC is behaving correctly and working under optimal conditions. To do so, we first describe the operating state that the algorithm has to monitor.

3.1. The Selection of Variables

The operating state of an SoC needs to be described by both software and physical aspects of the said SoC since both are intricately linked. Furthermore, the selected variables have to be either readable directly from the system or measurable to ease model validation. Thus, we selected the following variables to describe the operating state of The SoC:

- **Per-core CPU load ($\mathbf{Load_1}$, ..., $\mathbf{Load_n}$):** Sometimes also called CPU utilization [68]. The load is the sum of times the processor spends either busy or waiting (e.g., for I/O) during a sampling period, relative to that sampling period in percent [69].
- **GPU Load ($\mathbf{Load_{GPU}}$):** Same definition as for the CPU load, it is the sum of times the processor spends either busy or waiting during a sampling period, relative to that sampling period in percent [70].
- **Memory Occupation Rate (MOR):** Memory usage plays a huge role in power consumption [71] and thermal output of a SoC [72]. To characterize the influence of the Random Access Memory (RAM) on the temperature of the SoC and its power consumption, we needed to include its value as an input to those models. However, the value of the Random Access Memory (RAM) on its own is indicative of neither the base value used nor the maximum. Thus, we define MOR as the ratio of the occupied RAM (memory) relative to its full size.
- **Physical measurements of the SoC:**
  - Per-core CPU frequencies and the frequency of the GPU ($f_1, ..., f_n, f_{GPU}$)
  - Per-core CPU voltages and the voltage of the GPU ($V_1, ..., V_n, V_{GPU}$)
  - Temperature of the SoC (T_{SoC}), and the temperature per core if available ($T_1, ..., T_n, T_{GPU}$)
  - The power consumed by the board or the device (P)

3.2. Interconnected and Incremental Modeling

The set of aforementioned variables all follow different dynamics and are influenced by different factors, which makes establishing a global model for all of them arduous. However, investigating these variables, we find clear causal relationships between most of them. Firstly, the CPU load dictates the frequency on which the CPU should run, and it is the same for the GPU. Consequently, the voltage will change according to the chosen frequency through DVFS [40]. Secondly, power consumption in processors is a direct function of the voltage and the frequency [12]. Thirdly, higher frequencies always cause the temperature to rise indicating a clear link and correlation between these two variables [46]. Finally, modern CPU always include a thermal regulator that will either limit the highest frequency possible or shutdown the processor (or one of its core), whenever its temperature crosses a certain threshold it [40].

Therefore, instead of trying to build a global model for all these variables, we adopted a gradual approach and a modular structure for the modeling of the SoC in hand.

The modeling scheme is composed of a set of subsystems each written to estimate only one variable. In addition, by exploiting previously mentioned causal relationships and connecting the subsystems accordingly, the result is a model capable of estimating all the variables we defined and also accounting for the variable structure of the system. As shown in Figure 1, such a modular approach, indeed, allows to incrementally and progressively analyze each of the variables, and model it.

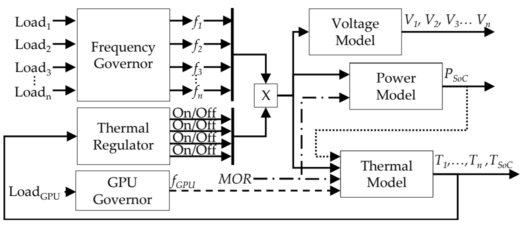


Figure 1. General diagram of the proposed incremental modeling of an embedded system-on-chip (SoC). MOR: Memory Occupation Rate.

Another clear advantage of this modeling scheme is that it allows for easy integration or replacement of subsystems in the case of changes or updates. Thus, giving the user the flexibility to use whichever subsystem he finds most suitable. For instance, during the course of this work, we developed three power models; a Nonlinear Autoregressive with eXogenous input (NARX) neural network—which will be presented later in the article, a regression-based model similar to the one developed by Kim et al. [26], and a regression-tree. All of these models were easily interchangeable and the library can be amended even by models from the literature. Figure 2 shows how easy it is to interchange subsystems and connect them into the general model.

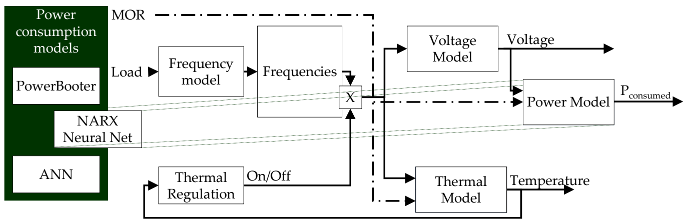


Figure 2. The incremental structure of the general model makes very simple to adapt it to changes in the structure and even exchange components at will. ANN: Artfical neural network. NARX: Nonlinear Autoregressive with eXogenous input.

The interconnected model is required to have the least overhead. Thus, it only uses readings and sensors available on the system. Additionally, variables like the frequency—and consequently voltage and power consumption—are reevaluated every 20 ms [73]. The model needs to read these variables and generates estimations at this sampling time. During the building of the monitoring framework, we have built and experimented with several types of models—notably for the temperature and power consumption. However, in this work, we present the models that delivered the best accuracy while satisfying the speed of estimation requirements.

The variables estimated by the model, characterize the operating state of the SoC (i.e., both the CPU and GPU) and are used as inputs to the monitoring algorithm.

3.3. Monitoring Approach

Most of the fault diagnosis techniques, presented in Section 2, are based on the dependency theory, hardware and software redundancy, and online verification tests. The method proposed in this work, on the other hand, is based on analytical redundancy, created by the construction of a reference model of the system. This technique compares the real outputs of the system to reference ones generated by the developed estimation model. In the presence of faults, the system's outputs deviate from their reference values, thus, generating fault indicators. The latter, commonly called residuals, are evaluated by a probabilistic method to account for modeling uncertainties in the decision-making process.

The use of analytical redundancy is fairly common for the diagnosis of fault in the hardware section of systems. Yet, to our knowledge, this is the first use of this technique to monitor both hardware and software components. Figure 3 presents an overview diagram of our approach to monitoring an SoC with an embedded CPU and GPU.

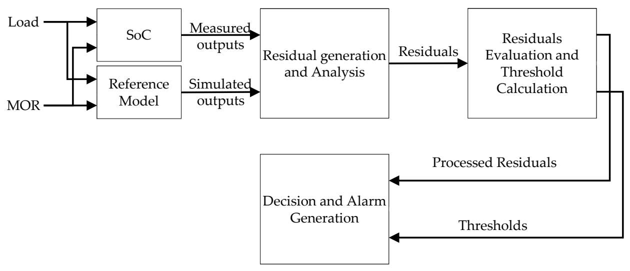

Load
MOR
SoC
Reference Model
Measured outputs
Simulated outputs
Residual generation and Analysis
Residuals
Residuals Evaluation and Threshold Calculation
Processed Residuals
Thresholds
Decision and Alarm Generation

Figure 3. Overview of the proposed monitoring method.

Finally, the modularity of the incremental estimation model allows for each of the generated residuals to be associated with an identified module. Hence, in the presence of faults, faulty subsystems can be easily isolated.

4. Experimental Setup

Two systems were used in this study for testing and experimental validation; an Android smartphone and an ARM-based development board.

4.1. The Android Smartphone

These devices are ubiquitous and developed application can be effortlessly transferred from one device to another. Moreover, the availability of the source code for their operating systems makes some of the needed parameters accessible for reading and even modification. Additionally, they are equipped with the all necessary sensors for the measurement of all the variables mentioned in Section 3.1.

The smartphone we used in this study is a Samsung Galaxy S5 [74]. It was equipped with an SoC harboring a quad-core processor with variable frequencies—through frequency and voltage scaling—ranging between 0.3–2.45 GHz. It also has a GPU with frequencies ranging between 200–578 MHz. The SoC is covered by the system's 2 GB low power RAM.

After much experimentation, this phone was no longer usable. For the last part of experimentation (see Section 8.3), we used a second device equipped with an octa-core processor arranged in a big.LITTLE configuration (Arm Holdings, Cambridge, United Kingdom, 2007) [75], running at frequencies up to 2.3 GHz, a state of the art GPU, and 4 GB of RAM.

4.2. The ARM-Based Development Board

For the actual prototype of the project, we use a safety-critical certified development board. The board runs on Linux and is Android capable. It has a one core ARM Cortex-A9 processor (ARM Holdings, Cambridge, United Kingdom, 2007) [76] and is equipped with 1 GB of RAM. Since it is only a prototype, the board is not equipped with sensors to measure consumed power (or drawn current for that matter), yet. We used an external ampere meter and an oscilloscope.

4.3. The Monitoring Equipment

There are two possible ways of implementing a monitoring algorithm; either implement it directly on the monitored board or use external equipment to do so. Even though we tested both ways, especially on the phone where it was feasible and had a utility that the phone monitors its own operating state. From a reliability standpoint, the operating state of an electronic board should be monitored externally, to reduce overhead on the board, and if the board freezes or fails, warnings and fail indications could still be generated. Hence, for this prototype, we used a PC and linked it to the monitored device through a TCP/IP connection. An application that we developed would send data from the device to the PC which would generate estimated reference variables and compares them to the readings.

Figure 4 shows the actual development board alongside its touch screen and the multimeter used to measure the power it draws. The monitoring PC with the monitoring program is alongside as well.

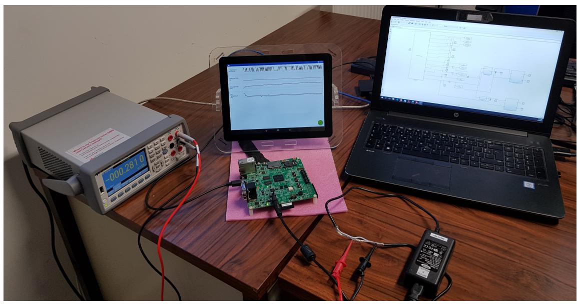


Figure 4. Setup with the development board, its touchscreen, and the ampere meter.

5. System Modeling

In the following paragraphs, we detail the modeling process of each of the subsystems. Firstly, we start by reverse engineering algorithms of the frequency governor, voltage scaling, and the thermal regulator. Then we move onto the black-box modeling of power and temperature.

5.1. Reverse Engineering the Algorithms

These algorithms are already present on the system. To generate similar outputs as they do, we studied and repurposed their code for simulation and monitoring.

5.1.1. Frequency Governor

Frequency scaling is carried out by programs called Governors. In the studied systems, several frequency governors exist; they calculate the frequency according to usage needs as well as several other factors (responsiveness, power consumption, etc.). Smartphones usually tend to use the Interactive Governor (Google Inc, Mountain View, CA, United States, 2015) [77], which is also compatible with the ARM-based board. Thus, it has been used in our case study. Nevertheless, thanks to the modular structure of the model, any other governor can replace it, if needed. Figure 5 displays a simplified flowchart of the governor's algorithm. It is inspired by the source code available in the code deposits of the manufacturer of the studied smartphone [73]. In a summary, it increases and decreases the frequency of each core as a function of the load, specific constants (*goHispeedLoad*, *targetLoad*, *hispeedFreq*, etc.), and timers (*timerRate*, *downTimer*...).

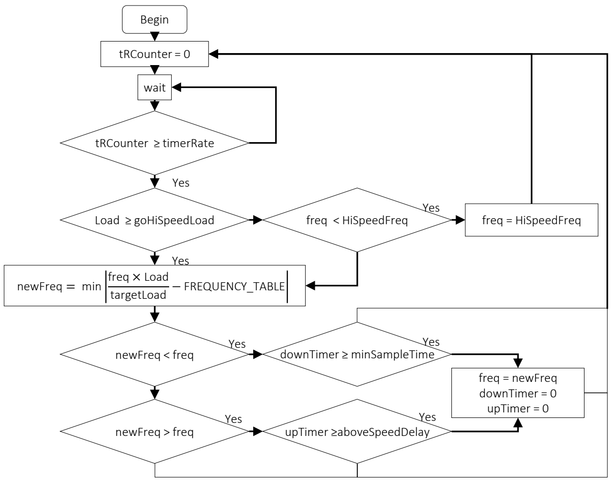


Figure 5. A simplified flowchart for modeling of the Interactive Governor.

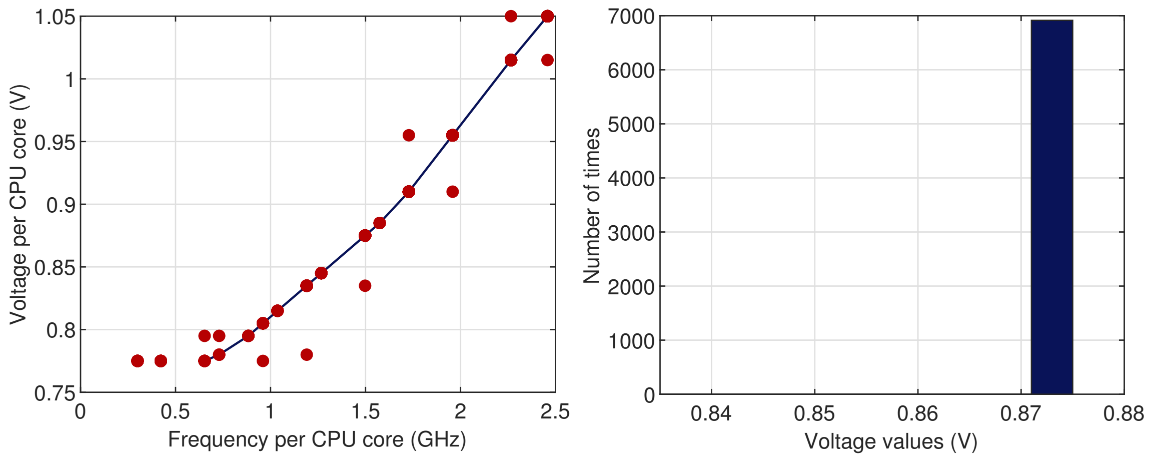
5.1.2. The Thermal Regulator

The manufacturers of the SoCs used in this study programmed three of thermal regulators that can be selected to control the temperatures of the SoC. These regulators are Proportional Integral Derivative (PID), Single Step (SS), and the one used in our systems; Monitor. This algorithm simply samples the temperature at a predefined rate (*samplingMs*), and if a core's temperature is higher than the predefined threshold, the core will be shut down, until it drops below a second threshold (*thresholdsClr*), when the core can be turned on again.

5.1.3. Voltage Scaling

Recovered voltage readings of the CPU cores show that, like the frequency, voltage is discreet and varies in a set of well-defined values. To see the relationship between these two variables, voltage readings are plotted against the frequencies (Figure 6a). It shows an almost linear trend (in red) where the values are concentrated, indicating that each frequency is associated with a fixed voltage value.

Furthermore, by analyzing the measurements, we find 15 frequency values (plus a zero frequency for a turned-off core) against 14 voltage values, leading to the belief that some frequencies share the same voltage value. Thus, to better investigate the relationship between the frequencies and voltages, we use the histogram of voltage values for each frequency. As shown in Figure 6b for $f = 1.49\,\text{GHz}$, the histogram clearly indicates that the core voltage for this frequency value is $V_{1.49} = 0.875\,\text{V}$. All of the other voltage values are obtained in the same manner.



(**a**) (**b**)

Figure 6. Frequency vs. voltage for the Samsung Galaxy S5 Central Processing Unit (CPU). (**a**) The voltage values drawn against the frequency values (in red), show a quasi-linear trend (in blue). (**b**) A histogram of voltage values for $f = 1.49\,\text{GHz}$.

5.2. Black-Box Modeling

White-box modeling of temperature and power in an SoC would require design level knowledge of all the inner components of the chip [23]. It would also require constructing finite state machines [17] or simulating differential equations [48]. Identification techniques, on the other hand, train the model to fit its outputs to observations using statistics [17]. Good expertise of the factors influencing the output is appreciated, but not required, nor is the formal theoretical proof of the relation between the inputs and outputs [23]. Still, these methods require large amounts of data, time, and computational resources for training and validation to account for all cases [17], unlike white-box models.

5.2.1. Power Model

Before starting the model construction and training processes, it is necessary to determine the inputs of the model. For microprocessors, the power consumption is often given by the [78]:

$$P_{CPU} = C \times f \times V^2 \tag{1}$$

where f is the frequency, V the voltage and C the capacitance of the microprocessor. C depends on the design, the internal wiring, and gate switching in the microprocessors [78]. While we cannot directly use Equation (1) to compute power consumption, we can use it as a guide to select the variables that correlate the most with power consumption in the microprocessor—in this case, the frequency and voltage.

It was also shown in the previous paragraph that the voltage and the frequency data are tied one to the other. Therefore, the frequencies of the cores, the frequency of the GPU, and the MOR are used as the inputs to our power model. Some of the models in the literature also multiplied the frequencies by the load. Nonetheless, our experimental results showed that the load—which defines the frequency in the first place—had no further effect on the final accuracy of the model. Hence, for optimization purposes (limiting the input size of the model), we omitted its use.

Most power models in the literature are constructed using data fitting techniques like linear regression and tend to be of a polynomial form [23,24,26,27]. These models assume a linear relationship between the inputs and the outputs, which is not always the case [23], thus increasing the error when the change in power consumption is not linear [17]. Although we also use data-based techniques for identification, the model we construct is a nonlinear autoregressive model with exogenous input (NARX) [79], since they would overcome the shortcomings of regression-based ones [36,37], giving it an advantage over linear regression models.

The neural network is composed of two layers. The hidden layer contains neurons with sigmoids as activation functions, and the output layer contains a single linear neuron. Figure 7 shows the general structure of the NARX model with the output feedback and the time delay line (TDL), which in our model is equal to 1.

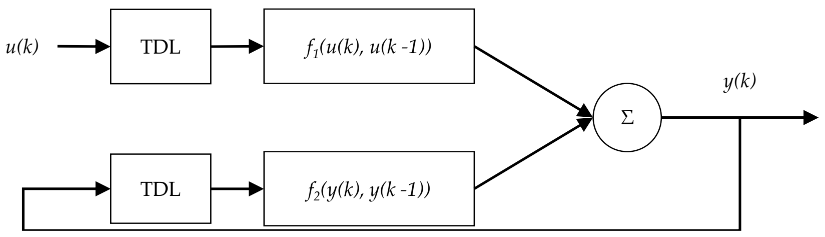


Figure 7. The general structure of the NARX neural network model. TDL: Time Delay Line.

Since this work is focused around the SoC, in order to minimize the influence of other system components, all communication and secondary peripherals Wi-Fi, modem, cameras, screen, etc.) were disabled and assumed to consume a static constant amount of power in that state [28,80]. The measured consumed power becomes:

$$P_{Consumed} = P_{SoC} + P_{Static} \tag{2}$$

During our experimentation, we found the static power P_{Static} to be marginal, thanks to the power management capabilities of modern devices, but we still included it in the model for better accuracy.

5.2.2. Thermal Model

Temperature dynamics are different from those of the other studied variables. For a starter, it is continuous, and going to be sampled (discretized). In addition, it does not depend only on the inputs, but also on its own previous values. Since our aim is to monitor the operating state of the SoC, we will not focus on the mechanics of heat generation and transfer. Thus, we focused our attention on finding variables affecting the temperature.

Recorded data show that temperature is directly correlated with the frequency of the CPU and the GPU. The data also confirm the correlation in the measurements between the recorded temperature and power consumption, as concluded in the literature [45,46]. Therefore, the considered inputs of the model of the temperatures are only the frequencies of the CPU and the GPU, along with the MOR and the power consumption.

To better represent these dynamics, we tried several model and settled on an Autoregressive–Moving-Average with Exogenous Inputs (ARMAX) model. ARMAX models use the regression of inputs and previous outputs, along with the moving average to simulate or predict the current output as follows [81]:

$$A(q^{-1})y(k) = B(q^{-1})u(k-\tau) + C(q^{-1}e(k)) \tag{3}$$

$y(k)$ is the output to estimate (or predict), $u(k)$ is the exogenous (X) variable or the input, and $e(t)$ is the moving average (MA) variable. q^{-1} is the delay operator. $A(q^{-1})$ holds of the autoregressive parameters, $B(q^{-1})$ holds the input parameters, and $C(q^{-1})$ holds the MA parameters [81].

In the case of this work, the ARMAX model is ideal to model the temperature, as it takes into account previous inputs and values of the output. In the identification process, temperature T_{SoC} is the output $y(k)$ to be estimated, and the inputs are:

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

6. Monitoring: Residuals Generation and Evaluation

The design of a monitoring system goes through two major steps. The first is the residuals generation, whereas the second consists of their evaluation. In theory, generated residuals have a value of zero in normal operating conditions. However, in practice, due to uncertainties, it is uncommon that residuals are equal to zero. Thus, two types of approaches can be found in the specialized literature for residual evaluation:

- **Model-based approaches** use the analytical expression of the residuals to generate thresholds that take into account parameter and model uncertainties along with measurement noises [82–86].
- **Signal-based approaches** consider residuals as a signal and do not take into account its physical origin. The idea consists of the extraction of statistical and probabilistic properties of residual signal in normal operation and using them as a reference to make a decision (generate an alarm, for instance) [87–90].

6.1. Residual Generation

In this work, raw residuals are generated by comparing the current outputs of the system with those estimated by the reference model (cf. Figure 3). They are expressed as follows:

$$\begin{cases} r_T = (T_{estm}(t) - T_{meas}(t))/T_{estm}(t) \\ r_P = (P_{estm}(t) - P_{meas}(t))/P_{estm}(t) \\ r_{i_V} = (V_{i_{estm}}(t) - V_{i_{meas}}(t))/V_{i_{estm}}(t) \\ r_{i_f} = (f_{i_{estm}}(t) - f_{i_{meas}}(t))/f_{i_{estm}}(t) \end{cases} \tag{4}$$

where $i = \{1, ..., n, GPU\}$ denotes either the GPU or the core number of the CPU, the subscript $_{estm}$ denotes model estimations and $_{meas}$ is for measured values (or readings).

This approach for residuals generation is well suited for the detection of progressive deviations in the characteristics of the system and thus allows for early detection of degradation. Interpretations of these drifts in characteristics, on the other hand, are performed using relationships of causality arising from the expertise of manufacturers and system operators. Hence, this method employs a large part of the available information (physical knowledge, expertise, measures, etc.).

6.2. Residual Evaluation

This step allows us to determine the presence or absence of a fault. The raw residuals generated in the previous paragraph are evaluated each with methods suited for their type. The purpose of this evaluation is to generate a normalized residual R which can only take two values; 1 or 0. This residual serves as an indicator for users of the presence (or not) of faults in the system.

To generate the normalized residual, we need to define thresholds values. In this work, taking into account how residuals are generated, their evaluation is carried out using signal-based approaches. Additionally, since each subsystem is modeled by the most appropriate method for its dynamics, thresholds calculation is also adapted to the said dynamics.

6.2.1. Frequency and Voltage Residuals Evaluation

Analysis of the frequency raw residuals r_f and those of the voltage r_V (given in Section 7) show that these signals are equal to zero, but include momentary differences during the moments when the frequency or the voltage changes. These spikes are caused by a delay in the computation of estimated values relative to measured ones. To take this delay into account in the decision-making process, the maximum values of the delay, noted τ_{ref}, is identified and then used to calculate normalized residuals R_f and R_V as follows:

$$R_x = \begin{cases} 1 & \text{if } r_x \neq 0 \text{ and } \tau_{x_d} > \tau_{f_{ref}}, \\ 0 & \text{elsewhere} \end{cases} \tag{5}$$

$x = \{f, v\}$ denotes either the frequency or the voltage, and τ_{x_d} is the value of the computed delay for each of these two variables.

6.2.2. Power and Temperature Residuals Evaluation

Analysis of the raw residuals r_P and r_T show that these signals have a high-frequency noise with an average close to zero. They are also normally distributed around that average (Figures 9b and 10b), which indicates that 99.7% of the residuals will be present between values of the mean μ plus or minus three times the standard deviation σ (in our case, it is 99.2% for the temperature residuals and 99.4% for the power residuals). However, errors will still arise from estimation and scheduling errors [91]. These errors are especially prevalent in periods of high loads where readings and estimations might be out of synchronization (as seen with the frequency and voltage).

Knowing that drifts of characteristics are rather contained in the average value of the signal, and in order to minimize the effect of the noise and avoid false alarms coming from estimation and synchronization errors, we use the moving average method to compute averaged residuals for power and temperature, r_{m_P} and r_{m_T}, respectively, from raw residuals r_P and r_T, respectively (Moving average is the unweighted sum of data over a window of n samples). Then, we use μ and σ of the averaged signals as thresholds to form an envelope around each of the residuals (Equation (6)). Thus, normal operating thresholds are generated as follows for power (temperature averaged residuals and thresholds are obtained using the same formula):

$$\begin{cases} r_{m_P} & = & \frac{1}{n}\sum_{i=1}^{n} r_P \quad , \\ th_{r_P}^{+} & = & \mu_{r_{m_P}} + 3 \times \sigma_{r_{m_P}} , \\ th_{r_P}^{-} & = & \mu_{r_{m_P}} - 3 \times \sigma_{r_{m_P}} \end{cases} \tag{6}$$

To further simplify the process of decision making, we generate normalized residuals R_P and R_T from the averaged residual r_{m_P} and r_{m_T} as follows:

$$\begin{cases} R_x = 1 & \text{if} \quad \|r_{x_m}\| > \|th_{r_x}\|, \\ R_x = 0 & \text{elsewhere} \end{cases} \tag{7}$$

$x = \{P, T\}$ denotes either the power or the temperature.

6.3. Fault Isolation

After fault detection comes isolation of the faulty subsystems, which remains an area of ongoing and developing research. In this paper, fault isolation is achieved through the incremental and interconnected structure of the model (Figure 1). Indeed, since in our model, each subsystem represents a functional component of the system and describes a relationship between its inputs and outputs during normal operation. In the case of presence of faults, these latter will first be detected in the faulty subsystems readings and alarms. Once a fault is detected, the next step is either to analyze its propagation and effect on the rest of the subsystem or just isolate it. In this work, we aim for isolation. Since the subsystems are interconnected, a fault would normally propagate. Thus, outputs from the model of the faulty component are replaced with readings which will allow for the rest of the subsystems in the model to continue generating the same outputs as measured.

7. Validation of the Model and Monitoring Algorithm

The results presented in this section were recorded in a default usage and benchmarking scenario. This scenario is composed of three main steps. The first is a series of standard benchmarks (for both CPU and GPU). These benchmarks are PCMark [92], 3D mark [93], AnTuTu Benchmark [94], and Geekbench 4 [95]. The second step is a less complex video playback task, and the third one is the device left idle for a while.

Since we used two experimental setups, in this section, we will alternate showing results from both setups to highlight the flexibility and portability of the models and algorithm we propose here.

7.1. Model Validation

7.1.1. Frequency and Voltage Models

Evolution of the measured frequencies from the CPU cores of the Android phone and the frequencies estimated by the model of the governor is given in Figure 8a. It shows near-identical results with a small delay at the instants of frequency change. This delay explained by the time needed by the model to assess the change. The maximum recorded delay is $\tau_{f_{ref}} = 0.1$ s, during a period of high loads, due to scheduling delays [91]. This value will be used in the residual evaluation step. Analyzing Figure 8b, we arrive at the same conclusion for the voltage model with the difference of the maximum recorded delay being $\tau_{Vref} = 0.12$ s.

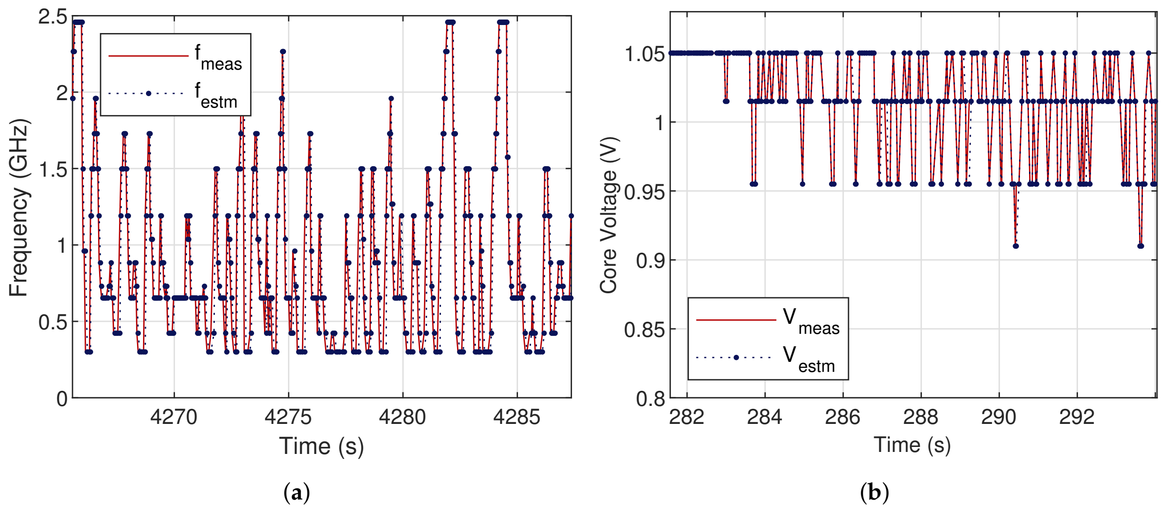


Figure 8. Frequency and voltage models performances. (**a**) Frequency model estimations vs. system readings. (**b**) Voltage model estimations vs. system readings.

7.1.2. NARX Power Model

To train and evaluate the NARX model, the data recovered during the default scenario, is split—as per standard training and validation procedure—into three sets; a training set (60%), a validation set (20%), and a test set (20%). The first two sets are used during the training process, while the test set is used to judge the performance of the model. Once the model was validated and tested offline, it was then further tested online to evaluate its estimation speed and its capacity to withstand scheduling delays during periods of high-loads and input lag.

Table 1 shows the training, validation, and test results for the NARX model in addition to the online test results which are required to validate the model in our use case. In Figure 9a, the measured power draw from the phone is displayed against values estimated by the NARX model, in the online test. The estimations follow the measured values accurately with minimal errors. The model has an accuracy of 97.12%, and the recorded Mean Absolute Error (MAE) 0.0168 W, whereas the Mean Squared Error (MSE) is 7.04×10^{-4}. Figure 9b shows the estimation error distribution of the NARX power model for the online test. The errors have a mean value of 1.178×10^{-4} W and a standard deviation of 0.0265, with 99.4% within $\mu \pm 3 \times \sigma$. The Kolmogorov–Smirnov (KS) test confirms that this distribution fits the profile of a standard normal distribution.

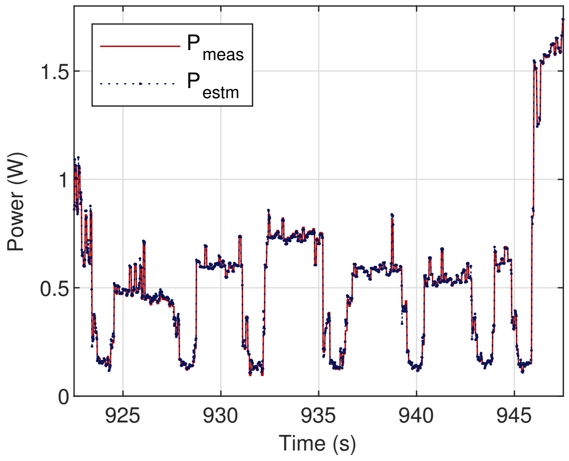
Table 1. Results from the training and validation of the Nonlinear Autoregressive model with eXogenous input (NARX) model for the Android phone. MAPE: Mean Absolute Percentage Error. MSE: Mean Squared Error.

Set	Training	Validation	Test	Online Test
MSE	2.32×10^{-4}	2.68×10^{-4}	3.17×10^{-4}	$\mathbf{7.04 \times 10^{-4}}$
MAPE (%)	2.13	2.19	2.40	**2.88**

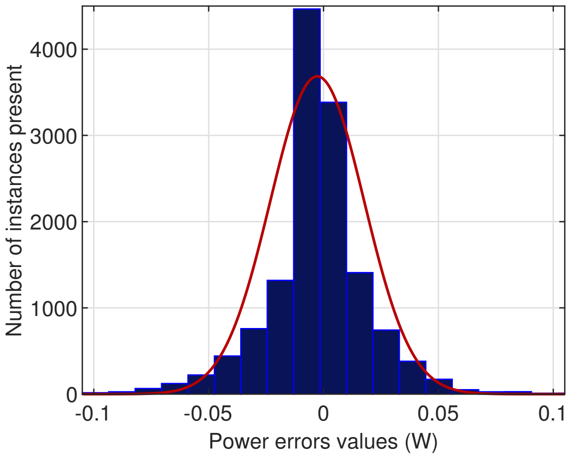
We also compared our results with the accuracy of established and recent power models; *Trepn* which measured at 94.5% [22], *Snapdragon Profiler* (measured at 95.2%), *PowerBooter* which reported 96% accuracy [17,23], *PETrA* (96% [29]), the models proposed by Walker et al. (96.2% [33]), Yoon et al. (94.9% [7]), Kim et al. (96.2% [26]), and *GreenOracle* (∼10% [31]).

Finally, we compared the power draw overhead caused by our modeling and monitoring program to the profilers we could test; *PowerBooter*, *Trepn*, and *Snapdragon Profiler*. The first caused an increase of 9% in power consumption during a 15 min test. While the second caused an increase of 8%, in our standardized benchmarking test. The *Snapdragon Profiler* caused an increase of 5%. Compared to the three aforementioned profilers, our modeling and monitoring scheme caused an increase of only 3% in power consumption, during the same test.

While we are pleased that our NARX model delivered results that are on par or even better than established works, it is worth noting that some of these profilers have higher levels of granularity than our model. Furthermore, some works, like *PowerBooter*, have not been updated for several years, and manufacturer-specific profilers were trained using its brands' SoCs, and thus would probably perform with them.



(**a**)

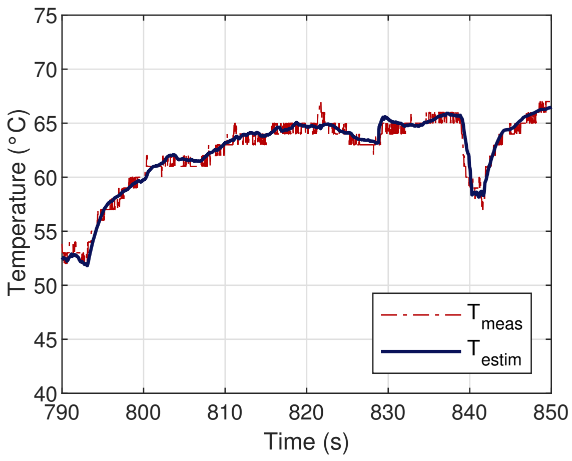


(**b**)

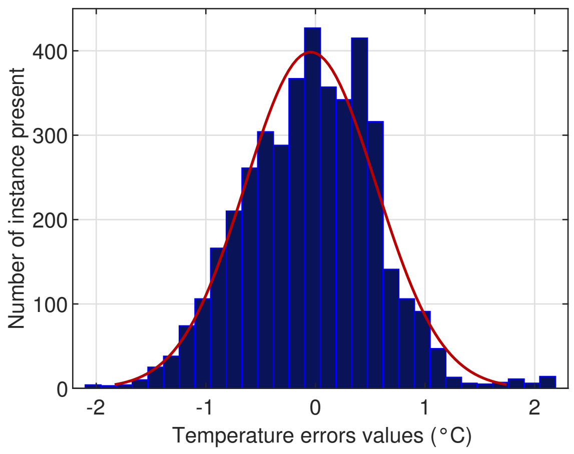
Figure 9. NARX power model perfomance. (**a**) NARX model power estimations compared to the system measurements. (**b**) Distribution of the NARX power model estimation errors.

7.1.3. ARMAX Temperature Model

In system identification, only two sets of data are needed. The first set is used to train the model, and the second is used to validate and test it. For this model, 70% of the data are used as a training set and the rest as a validation set. Finally, as in the case of the NARX power model, the model was tested online. Table 2 shows the MAPE and MAE (%) from both sets and the online test. The ARMAX model has an MAE of 0.4741 °C (1.48%). The estimation errors from the ARMAX model are normally distributed (KS test), as they were in the case of the power model, with 99.2% of the values within $\mu \pm 3 \times \sigma$. Figure 10b displays the distribution of these errors.



(**a**)



(**b**)

Figure 10. The Autoregressive–Moving-Average with Exogenous Inputs (ARMAX) temperature model perfomance. (**a**) ARMAX model temperature estimations compared to system readings. (**b**) Distribution of the ARMAX Temperature estimation errors.

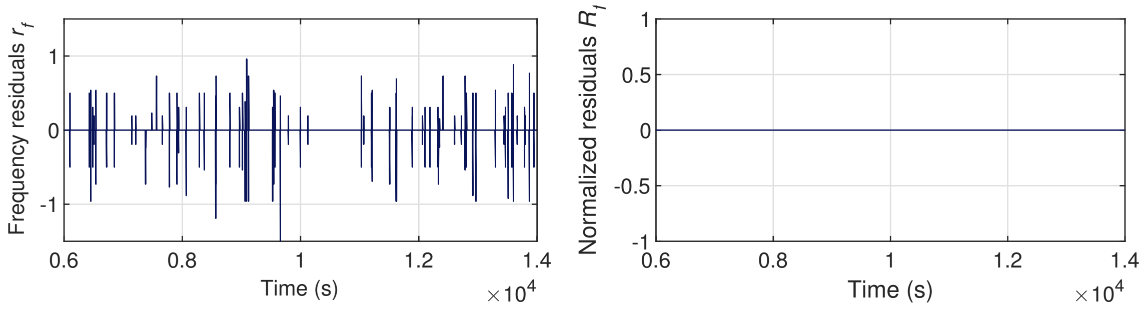
Table 2. Results from the identification and validation of ARMAX model for the development board.

Set	Identification	Validation (Test)	Online Test
MSE	0.35	0.58	**0.83**
MAPE (%)	0.73	0.75	**1.48**

Figure 10a displays the evolution of the estimated temperature on the development board, compared with the measured temperature of the CPU. The estimations delivered by our model are accurate in both low-frequency changes (heating and cooling), as well as high-frequency ones (weak temperature changes). The figure also shows that the model takes into account the initial conditions. Hence, temperature modeling is validated.

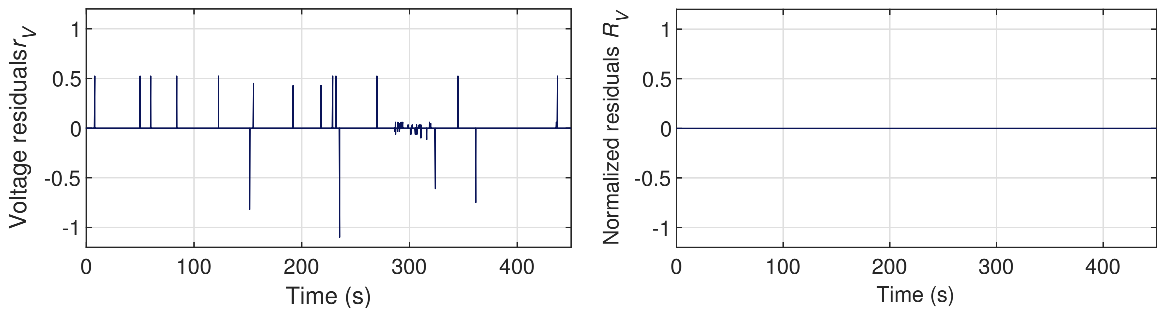
7.2. Validation of the Fault Detection and Isolation (FDI) Algorithm

Profiles of the raw residuals r_f and normalized residual R_f during our tests on the smartphone are given in Figure 11. This latter shows that raw residuals are sensitive to delays of estimation by the model at the moments of frequency changes, resulting in the appearance of a number of spikes. However, the normalized residual R_f is equal to zero over the duration of the test. Hence, no false alarm is recorded. Figure 12, allows us to draw the same conclusion for the voltage raw residuals r_v and the normalized residuals R_v, which also present no false alarms.



(**a**) (**b**)

Figure 11. Profile of the residuals r_f and R_f during normal operation: (**a**) raw residuals r_f, (**b**) normalized residuals R_f.



(**a**) (**b**)

Figure 12. Profile of the residuals r_V and R_V during normal operation scenario: (**a**) raw residuals r_V, (**b**) normalized residuals R_V.

Raw residuals r_P, averaged residuals r_{m_P}, and normalized residuals R_P from the tests on the smartphone, are given in Figure 13a–c, along with the computed thresholds (in red). In the raw residuals r_P, 99.4% of data is within the normal operating envelope, giving rise to some false alarms. The number of false alarm is reduced to naught by using the averaged residuals r_{m_P}. Thus, averaged residuals greatly improve the accuracy of the power normalized residuals. Figure 13 also shows all the temperature residuals r_T, r_{m_T}, and R_T along with the thresholds (red colored lines) from the tests on the development board, and allows as to draw the same conclusion for the temperature residuals.

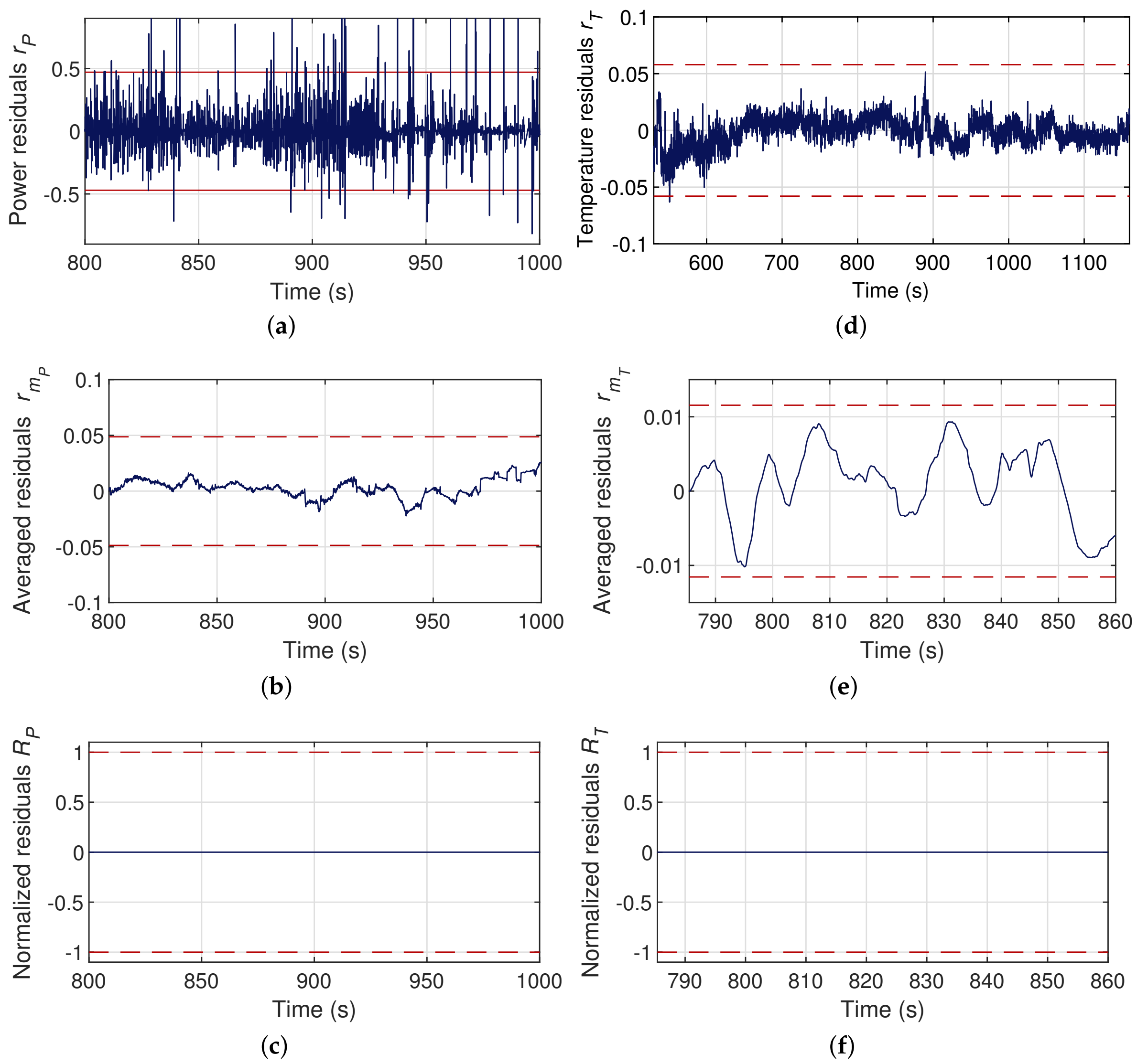


Figure 13. Power and temperature residuals assessment in normal operation. (**a**) Power raw residuals r_P. (**b**) Power averaged residuals r_{m_P}. (**c**) Power normalized residuals R_P. (**d**) Temperature raw residuals r_T. (**e**) Temperature averaged residuals r_{m_T}. (**f**) Temperature normalized residuals R_T.

8. Testing the FDI Algorithm

To test the residuals sensibility to faults and degradation, we propose three different faulty scenarios with faults originating either from the environment, or the software, or the hardware. During these experimentation process, the device will be put into the same benchmarking and usage as the previous section but in the middle of that usage, a fault will be presented as described below. A demonstration video of this process have been published online (Supplementary Material) [96].

8.1. Control Faults

In the first faulty scenario, we reproduce a possible governor bug or a system clock malfunction where frequency no longer corresponds to the load. In this scenario, a constant frequency is forced onto the CPU. Figure 14 shows the evolution of both measured and estimated frequencies on the development board. At $t = 46$ s, the measured frequency is locked to a constant frequency that does not match the input load. This fault is instantly detected by the raw residual r_f as shown in Figure 15a. Consequently, the normalized residual R_f rises from 0 to 1, generating an alarm (Figure 15b).

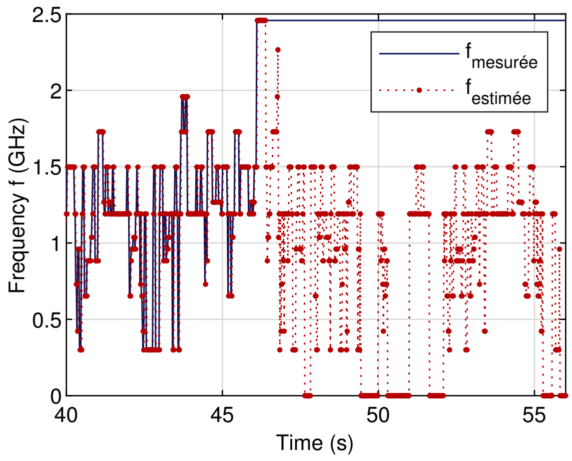
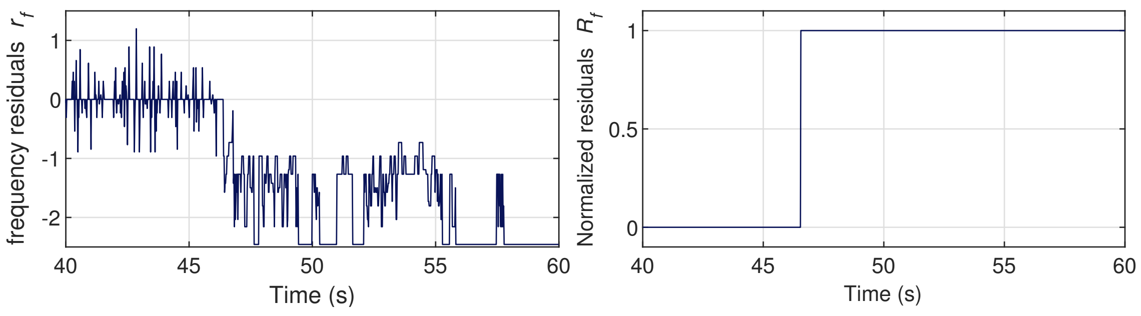


Figure 14. Estimated frequency from loads against measured frequency that is blocked at the maximum value around $t = 46$ s.



(**a**) (**b**)

Figure 15. Profile of the frequency residuals during faulty operation. Frequency is blocked at the maximum value around $t = 46$ s at which time an alarm is generated. (**a**) Raw residuals r_f. (**b**) Averaged residuals R_f.

8.2. Hardware or Component Faults

This scenario is intended to simulate faulty component, the presence of foreign bodies on the chip (like the accumulation of dust), or the change in the chips characteristics due to wear either by time or by overuse. Such faults and drifts are generally noticed through a decrease in power consumption.

As already described in Section 5.2.1, the neural net model is trained to monitor the power consumed by the SoC (and the static power consumed by the rest of the peripherals by extension). Thus, to simulate the change in power profile caused by one of the aforementioned reasons, In this experiment, we plugged an external peripheral—a USB lamp.

In Figure 16, the values of the measured and estimated power consumption values from the development board are drawn against each other. It shows a noticeable difference between the two values compared to Figure 9a. When computing and evaluating the raw residuals (Figure 18a), we find that most values fall within the thresholds envelope. Nevertheless, further analysis using normalized residuals (Figure 18b) shows that the average of the residuals fall outside of the threshold envelope and so an alarm is generated (Figure 18c).

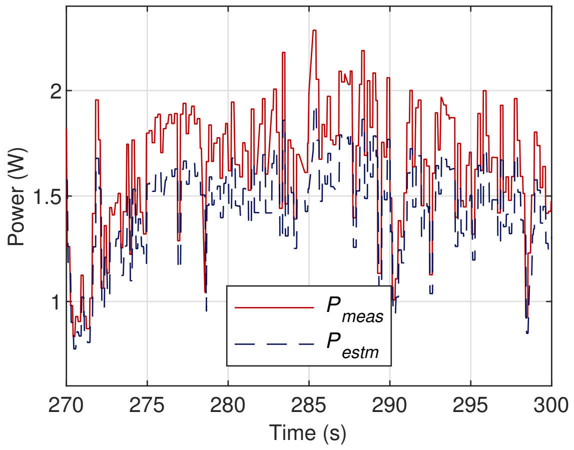


Figure 16. Measured power values drawn against estimated ones during the faulty power scenario experiment.

8.3. Environmental Faults

While the faults discussed in the previous paragraph come from either the software or the chip itself, the fault we are introducing in this faulty scenario is generally due to the environment around the chip. In this part of the test, the chip will be heated, allowing us to test the ability of the fault detection algorithm to detect abnormal heating of the SoC, which can be caused by a failure in the cooling system, or electrostatic charges, or even radiations.

In order to simulate such faults, we had to heat the devices up to a temperature higher than their running temperatures, without damaging their components or compromising their structural integrity. The phone was sealed in a waterproof bag and put into a hot water bath at a temperature of 80 °C, whereas the development board was exposed, at proximity to a 1000 W light projector.

Figure 17 shows the evolution of measured and estimated SoC temperatures, on the smartphone. The difference between the two curves becomes clear at $t = 180$ s, about 20 s after the submersion of the phone into the water. Figure 18 shows the reaction of residuals r_T (Figure 18d), r_{m_T} (Figure 18e), and R_T (Figure 18f). The latter is detected around $t = 190$ s where the residual r_{m_T} goes beyond normal operating envelope. Then, an alarm is generated (R_T rises from 0 to 1).

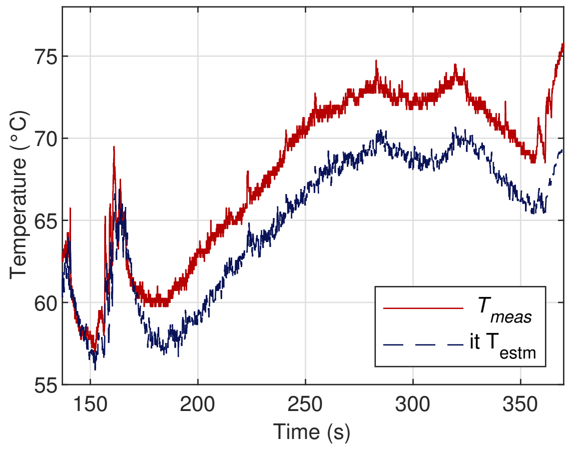
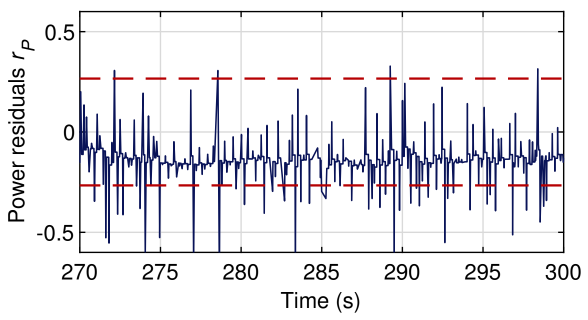
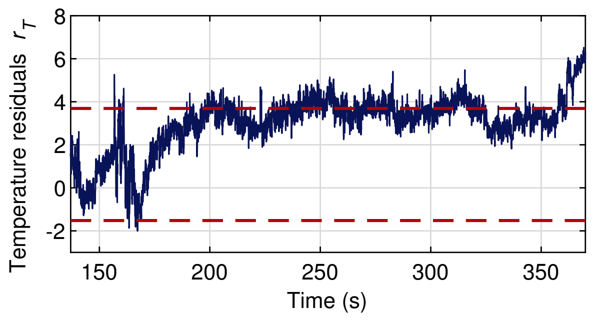


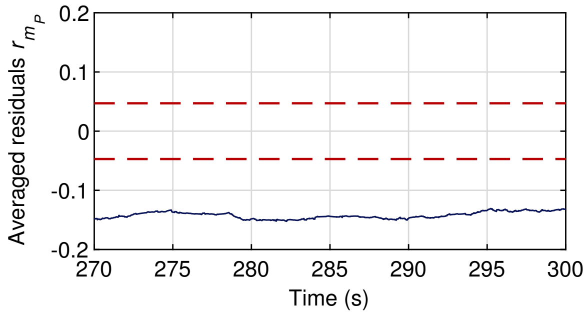
Figure 17. Measured temperatures and simulated temperatures of the smartphone while put in a heated environment. Divergence starts around $t = 190$ s.



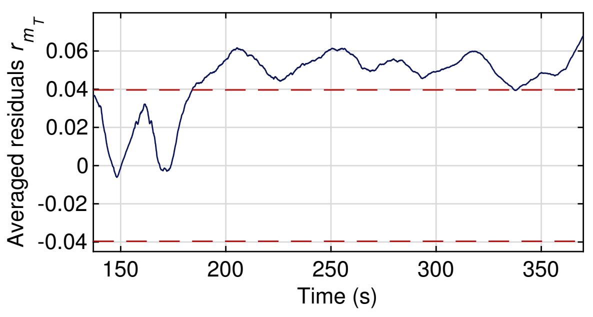
(**a**)



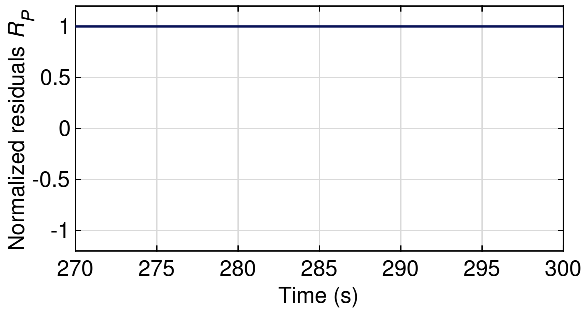
(**d**)



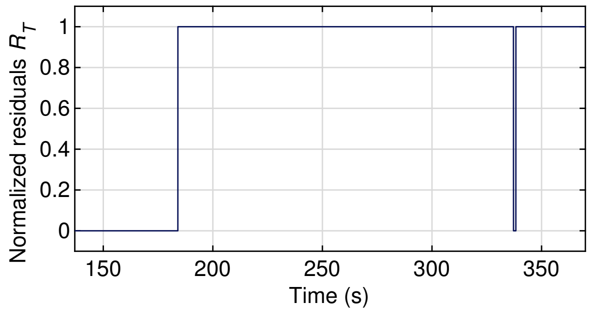
(**b**)



(**e**)



(**c**)



(**f**)

Figure 18. Power residuals assessment during faulty scenario of the introduction of an external component, and temperature residuals during the heating experiment. (**a**) Power raw residuals r_P. (**b**) Power averaged residuals r_{m_P}. (**c**) Power normalized residuals R_P. (**d**) Temperature raw residuals r_T. (**e**) Temperature averaged residuals r_{m_T}. (**f**) Temperature normalized residuals R_T.

9. Conclusions

This work has tackled the problem of monitoring a CPU-GPU SoC in an embedded system intended for safety-critical use, hence making its monitoring an obligation. We propose a monitoring scheme that acquires data from the device. Then, using this data, we built and trained a novel incremental interconnected model for the estimation of characteristic variables of the system.

The unique structure of the model streamlines the otherwise difficult process of modeling an SoC, making it relatively easier and more focused on the actual use of the model rather than building it. The latter is composed of subsystems to each built to estimate one variable rendering the process of fault isolation easier. Moreover, this interconnected structure gives the model builder the freedom and flexibility to adapt the model to the use case by including new subsystems and hence enlarging the scope of the model, or omitting some of them to focus on certain variables. The process of building each of the subsystems was also detailed, from the reverse-engineering of the algorithms to the use of data-based techniques (neural net and regression).

Experimental results validated all of the subsystems and the incremental model delivers estimation on the fly. Furthermore, our NARX power model has an accuracy of 97.12%; one of the highest in the literature. The choice of a nonlinear neural net to model power consumption proved its merit since the NARX power model outperforms the regression-based models because it accounts for the nonlinearity in the changes of power consumption. Finally, the ARMAX model's temperature results are accurate and on par with established models.

In the monitoring part, we presented a fault diagnosis module, aimed at the early detection of drifts from estimated characteristics. The fault diagnosis module uses residuals to generate alarms in case of faults. These residuals are generated by comparing estimations made by the developed model with readings from the system. To test the fault diagnosis module, it was first trained and validated during standard use. Then, three failure scenarios were tested, each to simulate a different fault. The fault diagnosis module quickly detected, localized, and isolated these faults. Thus, it is experimentally validated. In addition to fault detection, the information collected on the trend of the characteristic drift can be used for analysis and modeling of degradation phenomena in CPU-GPU chips, which will be discussed in future works.

Supplementary Materials: The following are available online at www.mdpi.com/xxx/s1, Video S1: Prototypage d'un système de surveillance des systèmes embarqués pour le projet MMCD.

Author Contributions: Conceptualization, O.D. and M.D.; Formal analysis, O.D. and M.D.; Methodology, O.D. and M.D.; Formal analysis, O.D.; Investigation, O.D.; Software, O.D.; Supervision, M.D.; Writing—original draft, O.D.; Writing—review & editing, O.D. and M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This paper is part of the "FUI 19" project. It is a publicly funded project through a grant by the *Banque Publique d'Investissement* (*Bpifrance*). *Bpifrance* is a public institution funding projects aiming to promote research, development, and industry in France.

Conflicts of Interest: The authors declare no conflict of interest. The funder had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

References

1. AMD. What Is Heterogeneous Computing? Available online: https://developer.amd.com/heterogeneous-computing-2/what-is-heterogeneous-computing/ (accessed on 23 January 2018).
2. Akdur, D.; Garousi, V.; Demirörs, O. A survey on modeling and model-driven engineering practices in the embedded software industry. *J. Syst. Archit.* **2018**, *91*, 62–82, doi:10.1016/j.sysarc.2018.09.007.
3. Djedidi, O.; Djeziri, M.A.; M'Sirdi, N.K. Data-Driven Approach for Feature Drift Detection in Embedded Electronic Devices. *IFAC PapersOnLine* **2018**, *51*, 1024–1029, doi:10.1016/j.ifacol.2018.09.714.
4. Djedidi, O.; Djeziri, M.A.; M'Sirdi, N.K.; Naamane, A. 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, Mardrid, Spain, 26–28 July 2017; SciTePress: Mardrid, Spain, 2017; Volume 1, pp. 338–345, doi:10.5220/0006470803380345.
5. Kim, D.; Chon, Y.; Jung, W.; Kim, Y.; Cha, H. Accurate Prediction of Available Battery Time for Mobile Applications. *ACM Trans. Embed. Comput. Syst.* **2016**, *15*, 1–17, doi:10.1145/2875423.
6. Guo, Y.; Wang, C.; Chen, X. Understanding application-battery interactions on smartphones: A large-scale empirical study. *IEEE Access* **2017**, *5*, 13387–13400, doi:10.1109/ACCESS.2017.2728620.
7. Yoon, C.; Lee, S.; Choi, Y.; Ha, R.; Cha, H. Accurate power modeling of modern mobile application processors. *J. Syst. Archit.* **2017**, *81*, 17–31, doi:10.1016/j.sysarc.2017.10.001.
8. Dzhagaryan, A.; Milenković, A.; Milosevic, M.; Jovanov, E. An Environment for Automated Measuring of Energy Consumed by Android Mobile Devices. In Proceedings of the 6th International Joint Conference on Pervasive and Embedded Computing and Communication Systems, Lisbon, Portugal, 25–27 July 2016; SCITEPRESS—Science and Technology Publications: Lisbon, Portugal, 2016; pp. 28–39, doi:10.5220/0005950800280039.
9. Romansky, S.; Borle, N.C.; Chowdhury, S.; Hindle, A.; Greiner, R. Deep Green: Modelling Time-Series of Software Energy Consumption. In Proceedings of the 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 17–22 September 2017; pp. 273–283, doi:10.1109/ICSME.2017.79.

10. Almusalli, F.A.; Zaman, N.; Rasool, R. Energy efficient middleware: Design and development for mobile applications. In Proceedings of the 2017 19th International Conference on Advanced Communication Technology (ICACT), Bongpyeong, Korea, 19–22 February 2017; pp. 541–549, doi:10.23919/ICACT.2017.7890149.
11. Niu, L.; Zhu, D. Reliability-aware scheduling for reducing system-wide energy consumption for weakly hard real-time systems. *J. Syst. Archit.* **2017**, *78*, 30–54, doi:10.1016/J.SYSARC.2017.06.004.
12. Li, T.; Yu, G.; Song, J. Minimizing energy by thermal-aware task assignment and speed scaling in heterogeneous MPSoC systems. *J. Syst. Archit.* **2018**, *89*, 118–130, doi:10.1016/j.sysarc.2018.08.003.
13. Caviglione, L.; Gaggero, M.; Lalande, J.F.; Mazurczyk, W.; Urbański, M. Seeing the unseen: Revealing mobile malware hidden communications via energy consumption and artificial intelligence. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 799–810, doi:10.1109/TIFS.2015.2510825.
14. Suarez-Tangil, G.; Tapiador, J.E.; Peris-Lopez, P.; Pastrana, S. Power-aware anomaly detection in smartphones: An analysis of on-platform versus externalized operation. *Pervasive Mob. Comput.* **2015**, *18*, 137–151, doi:10.1016/j.pmcj.2014.10.007.
15. Merlo, A.; Migliardi, M.; Fontanelli, P. Measuring and estimating power consumption in Android to support energy-based intrusion detection. *J. Comput. Secur.* **2015**, *23*, 611–637, doi:10.3233/JCS-150530.
16. Peltonen, E.; Lagerspetz, E.; Nurmi, P.; Tarkoma, S. Constella: Crowdsourced system setting recommendations for mobile devices. *Pervasive Mob. Comput.* **2016**, *26*, 71–90, doi:10.1016/j.pmcj.2015.10.011.
17. Hoque, M.A.; Siekkinen, M.; Khan, K.N.; Xiao, Y.; Tarkoma, S. Modeling, Profiling, and Debugging the Energy Consumption of Mobile Devices. *ACM Comput. Surv.* **2015**, *48*, 1–40, doi:10.1145/2840723.
18. Ahmad, R.W.; Gani, A.; Hamid, S.H.A.; Xia, F.; Shiraz, M. A Review on mobile application energy profiling: Taxonomy, state-of-the-art, and open research issues. *J. Netw. Comput. Appl.* **2015**, *58*, 42–59, doi:10.1016/j.jnca.2015.09.002.
19. Benkhelifa, E.; Welsh, T.; Tawalbeh, L.; Jararweh, Y.; Basalamah, A. Energy Optimisation for Mobile Device Power Consumption: A Survey and a Unified View of Modelling for a Comprehensive Network Simulation. *Mob. Networks Appl.* **2016**, *21*, 575–588, doi:10.1007/s11036-016-0756-y.
20. Google Inc. *Android 5.0 APIs*; Google Inc.: Mountain View, CA, USA, 2017.
21. Google Inc. Measuring Power Values | Android Open Source Project? Available online: https://source.android.com/devices/tech/power/values (accessed on 4 June 2020).
22. Qualcomm Innovation Center. *Trepn Profiler*; Qualcomm Innovation Center: San Diego, CA, USA, 2019.
23. Zhang, L.; Tiwana, B.; Qian, Z.; Wang, Z.; Dick, R.P.; Mao, Z.M.; Yang, L. 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, Scottsdale, AZ, USA, 24–28 October 2010; ACM Press: New York, NY, USA, 2010; p. 105, doi:10.1145/1878961.1878982.
24. Dong, M.; Zhong, L. Self-constructive high-rate system energy modeling for battery-powered mobile systems. In *MobiSys*; ACM Press: New York, NY, USA, 2011; p. 335, doi:10.1145/1999995.2000027.
25. Jung, W.; Kang, C.; Yoon, C.; Kim, D.D.; Cha, H. DevScope: A Nonintrusive and Online Power Analysis Tool for Smartphone Hardware Components. In Proceedings of the eighth IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis—CODES+ISSS '12, Tampere, Finland, 7–12 October 2010; ACM Press: New York, NY, USA, 2010; p. 353, doi:10.1145/2380445.2380502.
26. Kim, M.; Kong, J.; Chung, S.W. Enhancing online power estimation accuracy for smartphones. *IEEE Trans. Consum. Electron.* **2012**, *58*, 333–339, doi:10.1109/TCE.2012.6227431.
27. Kim, Y.G.; Kim, M.; Kim, J.M.; Sung, M.; Chung, S.W. A novel GPU power model for accurate smartphone power breakdown. *ETRI J.* **2015**, *37*, 157–164, doi:10.4218/etrij.15.0113.1411.
28. Djedidi, O.; Djeziri, M.A. Power Profiling and Monitoring in Embedded Systems: A Comparative Study and a Novel Methodology Based on NARX Neural Networks. *J. Syst. Archit.* **2020**, 101805, doi:10.1016/j.sysarc.2020.101805.
29. Di Nucci, D.; Palomba, F.; Prota, A.; Panichella, A.; Zaidman, A.; De Lucia, A. PETrA: A Software-Based Tool for Estimating the Energy Profile of Android Applications. In Proceedings of the 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), Buenos Aires, Argentina, 20–28 May 2017; pp. 3–6, doi:10.1109/ICSE-C.2017.18.

30. Shukla, N.K.; Pila, R.; Rawat, S. Utilization-based power consumption profiling in smartphones. In Proceedings of the 2016 2nd International Conference on Contemporary Computing and Informatics, IC3I 2016, Noida, India, 14–17 December 2016; pp. 881–886, doi:10.1109/IC3I.2016.7919046.
31. Chowdhury, S.A.; Hindle, A. GreenOracle: Estimating Software Energy Consumption with Energy Measurement Corpora. In Proceedings of the 13th International Workshop on Mining Software Repositories—MSR '16, Austin, TX, USA, 14–22 May 2016; ACM Press: New York, NY, USA, 2016; pp. 49–60, doi:10.1145/2901739.2901763.
32. Kim, K.; Shin, D.; Xie, Q.; Wang, Y.; Pedram, M.; Chang, N. FEPMA: Fine-Grained Event-Driven Power Meter for Android Smartphones Based on Device Driver Layer Event Monitoring. In Proceedings of the Design, Automation I& Test in Europe Conference and Exhibition (DATE), Dresden, Germany, 24–28 March 2014; pp. 1–6, doi:10.7873/DATE.2014.380.
33. Walker, M.J.; Diestelhorst, S.; Hansson, A.; Das, A.K.; Yang, S.; Al-Hashimi, B.M.; Merrett, G.V. Accurate and Stable Run-Time Power Modeling for Mobile and Embedded CPUs. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *36*, 106–119, doi:10.1109/TCAD.2016.2562920.
34. Pathak, A.; Hu, Y.C.; Zhang, M. 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, Bern, Switzerland, 10–13 April 2012; ACM: New York, NY, USA, 2012; pp. 29–42, doi:10.1145/2168836.2168841.
35. Xu, F.; Liu, Y.; Li, Q.; Zhang, Y. V-edge: Fast self-constructive power modeling of smartphones based on battery voltage dynamics. In Proceedings of the 10th USENIX Conference on Networked Systems Design and Implementation, Lombard, IL, USA, 2–5 April 2013; USENIX: Lombard, IL, USA, 2013; pp. 43–55.
36. Carvalho, S.A.; Cunha, D.C.; Silva-Filho, A.G. On the use of nonlinear methods for low-power CPU frequency prediction based on Android context variables. In Proceedings of the 2016 IEEE 15th International Symposium on Network Computing and Applications, Cambridge, MA, USA, 31 October–2 November 2016; pp. 250–253, doi:10.1109/NCA.2016.7778627.
37. Alawnah, S.; Sagahyroon, A. Modeling of smartphones' power using neural networks. *Eurasip J. Embed. Syst.* **2017**, *2017*, 22, doi:10.1186/s13639-017-0070-1.
38. Djedidi, O.; Djeziri, M.A.; M'Sirdi, N.K.; Naamane, A. 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), Montréal, QC, Canada, 28 October–2 November 2018; ACM Press: New York, NY, USA, 2018; pp. 79–82, doi:10.1145/3242102.3242139.
39. Zhang, Y.; Parikh, D. *Hotleakage: A Temperature-Aware Model of Subthreshold and Gate Leakage for Architects*; University of Virginia: Charlottesville, VA, USA, 2003.
40. Kim, J.M.; Kim, Y.G.; Chung, S.W. Stabilizing CPU frequency and voltage for temperature-aware DVFS in mobile devices. *IEEE Trans. Comput.* **2015**, *64*, 286–292, doi:10.1109/TC.2013.188.
41. Ferroni, M.; Nacci, A.A.; Turri, M.; Santambrogio, M.D.; Sciuto, D. Experimental Evaluation and Modeling of Thermal Phenomena on Mobile Devices. In Proceedings of the IEEE Euromicro Conference on Digital System Design (DSD), Funchal, Portugal, 26–28 August 2015; pp. 306–313, doi:10.1109/DSD.2015.20.
42. Chen, C.C.; Milor, L. Microprocessor Aging Analysis and Reliability Modeling Due to Back-End Wearout Mechanisms. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2015**, *23*, 2065–2076, doi:10.1109/TVLSI.2014.2357756.
43. Altieri, M.; Lesecq, S.; Puschini, D.; Heron, O.; Beigne, E.; Rodas, J. Evaluation and mitigation of aging effects on a digital on-chip voltage and temperature sensor. In Proceedings of the 2015 IEEE 25th International Workshop on Power and Timing Modeling, Optimization and Simulation, PATMOS 2015, Salvador, Brazil, 1–4 September 2015; pp. 111–117, doi:10.1109/PATMOS.2015.7347595.
44. Rahmani, A.M.; Haghbayan, M.H.; Miele, A.; Liljeberg, P.; Jantsch, A.; Tenhunen, H. Reliability-Aware Runtime Power Management for Many-Core Systems in the Dark Silicon Era. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* **2017**, *25*, 427–440, doi:10.1109/TVLSI.2016.2591798.
45. Mercati, P.; Paterna, F.; Bartolini, A.; Benini, L.; Rosing, T.Š. WARM: Workload-Aware Reliability Management in Linux/Android. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* **2017**, *36*, 1557–1570, doi:10.1109/TCAD.2016.2611501.

46. Zhou, J.; Yan, J.; Cao, K.; Tan, Y.; Wei, T.; Chen, M.; Zhang, G.; Chen, X.; Hu, S. Thermal-aware correlated two-level scheduling of real-time tasks with reduced processor energy on heterogeneous MPSoCs. *J. Syst. Archit.* **2018**, *82*, 1–11, doi:10.1016/J.SYSARC.2017.09.007.
47. Xie, Q.; Dousti, M.J.; Pedram, M. Therminator: A Thermal Simulator for Smartphones Producing Accurate Chip and Skin Temperature Maps. In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED), La Jolla, CA, USA, 11–13 August 2014; pp. 117–122, doi:10.1145/2627369.2627641.
48. Dousti, M.J.; Ghasemi-Gol, M.; Nazemi, M.; Pedram, M. ThermTap: An online power analyzer and thermal simulator for Android devices. In Proceedings of the IEEE International Symposium on Low Power Electronics and Design, Rome, Italy, 22–24 July 2015; pp. 341–346, doi:10.1109/ISLPED.2015.7273537.
49. Gao, Z.; Cecati, C.; Ding, S.X. A Survey of Fault Diagnosis and Fault-Tolerant Techniques Part I: Fault Diagnosis. *IEEE Trans. Ind. Electron.* **2015**, *62*, 3768–3774, doi:10.1109/TIE.2015.2417501.
50. Gao, Z.; Cecati, C.; Ding, S.X. A survey of fault diagnosis and fault-tolerant techniques-part II: Fault diagnosis with knowledge-based and hybrid/active approaches. *IEEE Trans. Ind. Electron.* **2015**, *62*, 3768–3774, doi:10.1109/TIE.2015.2419013.
51. Steininger, A. Testing and built-in self-test—A survey. *J. Syst. Archit.* **2000**, *46*, 721–747. doi:10.1016/S1383-7621(99)00041-7.
52. Deb, S.; Pattipati, K.R.; Raghavan, V.; Shakeri, M.; Shrestha, R. Multi-Signal Flow Graphs: A Novel Approach for System Testability Analysis and Fault Diagnosis. *IEEE Aerosp. Electron. Syst. Mag.* **1995**, *10*, 14–25, doi:10.1109/62.373993.
53. Sheppard, J. Maintaining diagnostic truth with information flow models. In Proceedings of the IEEE Conference Record. AUTOTESTCON '96, Dayton, OH, USA, 16–19 September 1996; pp. 447–454, doi:10.1109/AUTEST.1996.547773.
54. Zhang, G. Optimum Sensor Localization/Selection in A Diagnostic/Prognostic Architecture. Ph.D. Thesis, University System of Georgia, Atlanta, GA, USA, 2005.
55. Wang, F.W.; Shi, J.Y.; Wang, L. Method of diagnostic tree design for system-level faults based on dependency matrix and fault tree. In Proceedings of the 2011 IEEE 18th International Conference on Industrial Engineering and Engineering Management, IE and EM 2011, Changchun, China, 3–5 September 2011; pp. 1113–1117, doi:10.1109/IEEM.2011.6035351.
56. Cui, Y.; Shi, J.; Wang, Z. An analytical model of electronic fault diagnosis on extension of the dependency theory. *Reliab. Eng. I Syst. Saf.* **2015**, *133*, 192–202, doi:10.1016/j.ress.2014.09.015.
57. Gizopoulos, D.; Psarakis, M.; Adve, S.V.; Ramachandran, P.; Hari, S.K.S.; Sorin, D.; Meixner, A.; Biswas, A.; Vera, X. Architectures for online error detection and recovery in multicore processors. In Proceedings of the 2011 Design, Automation I& Test in Europe, Grenoble, France, 14–18 March 2011; pp. 1–6, doi:10.1109/DATE.2011.5763096.
58. Aggarwal, N.; Ranganathan, P. Configurable isolation: Building high availability systems with commodity multi-core processors. In *ACM Sigarch*; ACM Press: New York, NY, USA, 2007; Volume 35, pp. 470–481, doi:10.1145/1250662.1250720.
59. LaFrieda, C.; Ipek, E.; Martínez, J.F.; Manohar, R. Utilizing dynamically coupled cores to form a resilient chip multiprocessor. In Proceedings of the IEEE International Conference on Dependable Systems and Networks, Edinburgh, UK, 25–28 June 2007; pp. 317–326, doi:10.1109/DSN.2007.100.
60. Shyam, S.; Constantinides, K.; Phadke, S.; Bertacco, V.; Austin, T. 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, San Jose, CA, USA, 21–25 October 2006; ACM Press: New York, NY, USA, 2006; Volume 41, p. 73, doi:10.1145/1168857.1168868.
61. Austin, T.M. DIVA: A reliable substrate for deep submicron microarchitecture design. In Proceedings of theMICRO-32. Proceedings of the 32nd Annual ACM/IEEE International Symposium on Microarchitecture, Haifa, Israel, 16–18 November 1999; pp. 196–207, doi:10.1109/MICRO.1999.809458.
62. Meixner, A.; Bauer, M.E.; Sorin, D. Argus: Low-Cost, Comprehensive Error Detection in Simple Cores. In Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO 2007), Chicago, IL, USA, 1–5 December 2007; pp. 210–222, doi:10.1109/MICRO.2007.18.
63. Wang, N.J.; Patel, S.J. ReStore: Symptom-based soft error detection in microprocessors. *IEEE Trans. Dependable Secur. Comput.* **2006**, *3*, 188–201, doi:10.1109/TDSC.2006.40.

64. Li, M.L.; Ramachandran, P.; Sahoo, S.K.; Adve, S.V.; Adve, V.S.; Zhou, Y. Understanding the Propagation of Hard Errors to Software and Implications for Resilient System Design. *SIGOPS Oper. Syst. Rev.* **2008**, *42*, 265–276, doi:10.1145/1353535.1346315.
65. Sinha, S.; Kumar Goyal, N.; Mall, R. Early prediction of reliability and availability of combined hardware-software systems based on functional failures. *J. Syst. Archit.* **2019**, *92*, 23–38, doi:10.1016/j.sysarc.2018.10.007.
66. Zadegan, F.G.; Nikolov, D.; Larsson, E. On-Chip Fault Monitoring Using Self-Reconfiguring IEEE 1687 Networks. *IEEE Trans. Comput.* **2018**, *67*, 237–251, doi:10.1109/TC.2017.2731338.
67. Löfwenmark, A.; Nadjm-Tehrani, S. Fault and timing analysis in critical multi-core systems: A survey with an avionics perspective. *J. Syst. Archit.* **2018**, *87*, 1–11, doi:10.1016/j.sysarc.2018.04.001.
68. Gregg, B. CPU Utilization Is Wrong. Available online: http://www.brendangregg.com/blog/2017-05-09/cpu-utilization-is-wrong.html (accessed on 4 June 2020).
69. Saravanan, D.R. Understanding Processor Utilization on POWER Systems—AIX. Available online: https://www.ibm.com/developerworks/community/wikis/home?lang=en%5C#!/wiki/PowerSystems/page/UnderstandingCPUutilizationonAIX (accessed on 14 March 2020).
70. Qualcomm Inc. Trepn Power Profiler-FAQs-Qualcomm Developer Network. Available online: https://developer.qualcomm.com/forum/qdn-forums/increase-app-performance/trepn-profiler/27700 (accessed on 4 June 2020).
71. Leng, J.; Hetherington, T.; ElTantawy, A.; Gilani, S.; Kim, N.S.; Aamodt, T.M.; Reddi, V.J. GPUWattch: Enabling Energy Optimizations in GPGPUs. *ACM SIGARCH Comput. Archit. News* 2013, 41, 487–498, doi:10.1145/2485922.2485964.
72. Hong, S.; Kim, H. An integrated GPU power and performance model. In Proceedings of the 37th Annual International Symposium on Computer Architecture—ISCA '10, Saint-Malo, France, 19–23 June 2010; ACM: New York, NY, USA, 2010; Volume 38, p. 280, doi:10.1145/1816038.1815998.
73. Samsung. Samsung Opensource Release Center. Available online: https://opensource.samsung.com/ (accessed on 4 June 2020).
74. Samsung Inc. *Samsung Galaxy S5*; Samsung Inc: Seoul, Korea, 2014.
75. Arm Holdings. *Technologies | big.LITTLE—Arm Developer*; Arm Holdings: Cambridge, UK, 2018.
76. ARM Holdings *ARM Cortex-A9*; ARM Holdings: Cambridge, UK, 2007.
77. Brodowski, D.; Golde N. *CPU Frequency and Voltage Scaling Code in the Linux(TM) Kernel*; Google Inc: Mountain View, CA, USA, 2015.
78. Altamimi, M.L.; Naik, K. 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, Cancun, Mexico, 2–6 November 2015; ACM Press: New York, NY, USA, 2015; pp. 301–305, doi:10.1145/2811587.2811627.
79. Korjani, M.M.; Bazzaz, O.; Menhaj, M.B. Real time identification and control of dynamic systems. *Artif. Intell. Rev.* **2009**, *30*, 1–17, doi:10.1007/s10462-009-9111-z.
80. Lin, Y.D.; Rattagan, E.; Lai, Y.C.; Chang, L.P.; Yo, Y.C.; Ho, C.Y.; Chang, S.L. Calibrating parameters and formulas for process-level energy consumption profiling in smartphones. *J. Netw. Comput. Appl.* **2014**, *44*, 106–119, doi:10.1016/j.jnca.2014.04.014.
81. Fung, E.H.; Wong, Y.; Ho, H.; Mignolet, M.P. Modelling and prediction of machining errors using ARMAX and NARMAX structures. *Appl. Math. Model.* **2003**, *27*, 611–627, doi:10.1016/S0307-904X(03)00071-4.
82. Benmoussa, S.; Djeziri, M.A. Remaining useful life estimation without needing for prior knowledge of the degradation features. *IET Sci. Meas. I Technol.* **2017**, *11*, 1071–1078, doi:10.1049/iet-smt.2017.0005.
83. Djeziri, M.A.; Ould Bouamama, B.; Merzouki, R. Modelling and robust FDI of steam generator using uncertain bond graph model. *J. Process. Control* **2009**, *19*, 149–162, doi:10.1016/j.jprocont.2007.12.009.
84. Adrot, O.; Maquin, D.; Ragot, J. Fault detection with model parameter structured uncertainties. In Proceedings of the 5th European Control Conference, ECC'99, Karlsruhe, Germany, 30 August–3 September 1999.
85. Armengol, J.; Travé-Massuyès, L.; Vehí, J.; Lluís de la Rosa, J. A Survey on interval model simulators and their properties related to fault detection. *Annu. Rev. Control* **2000**, *24*, 31–39, doi:10.1016/S1367-5788(00)00002-X.
86. Li, L.; Zhou, D. *Fast and Robust Fault Diagnosis for a Class of Nonlinear Systems: Detectability Analysis*; 2004; doi:10.1016/j.compchemeng.2004.07.023.

87. Djeziri, M.A.; Merzouki, R.; Ould Bouamama, B.; Dauphin-Tanguy, G. Fault detection of backlash phenomenon in mechatronic system with parameter uncertainties using bond graph approach. In Proceedings of the 2006 IEEE International Conference on Mechatronics and Automation, ICMA 2006, Luoyang, China, 25–28 June 2006; Volume 2006; pp. 600–605, doi:10.1109/ICMA.2006.257639.
88. Basseville, M. On-board Component Fault Detection and Isolation Using the Statistical Local Approach. *Automatica* **1998**, *34*, 1391–1415, doi:10.1016/S0005-1098(98)00086-7.
89. Ge, C.; Yang, H.; Ye, H.; Wang, G. A Fast Leak Locating Method Based on Wavelet Transform. *Tsinghua Sci. Technol.* **2009**, *14*, 551–555, doi:10.1016/S1007-0214(09)70116-6.
90. Benmoussa, S.; Djeziri, M.A.; Ould Bouamama, B.; Merzouki, R. Empirical mode decomposition applied to fluid leak detection and isolation in process engineering. In Proceedings of the 18th Mediterranean Conference on Control and Automation, MED'10, Marrakech, Morocco, 23–25 June 2010; pp. 1537–1542, doi:10.1109/MED.2010.5547829.
91. Chen, J.; Du, C.; Xie, F.; Lin, B. Scheduling non-preemptive tasks with strict periods in multi-core real-time systems. *J. Syst. Archit.* **2018**, *90*, 72–84, doi:10.1016/j.sysarc.2018.09.002.
92. Futuremark. *PCMark for Android Benchmark*; UL: Espoo, Finland, 2019.
93. Futuremark. *3DMark*; UL: Espoo, Finland, 2018.
94. Cheetah Mobile. *AnTuTu Benchmark*; Cheetah Mobile: Beijing, China, 2019.
95. Primate Labs Inc. *Geekbench 4*; Primate Labs Inc.: Toronto, ON, Canada, 2019.
96. Djedidi, O.; Djeziri, M.; M'Sirdi, N. Prototyping of a Data-Driven Monitoring of Systems on Chip for Multifunction Modular Cockpit Display (MMCD) Project. Available online: https://hal-amu.archives-ouvertes.fr/hal-02293986v1 (accessed on 4 June 2020).