2013 39th Euromicro Conference Series on Software Engineering and Advanced Applications

Fast Evaluation of Power Consumption of Embedded Systems using DIPLODOCUS

Feriel Ben Abdallah and Ludovic Apvrille
Institut Mines-Telecom, Telecom ParisTech, LTCI CNRS
Telecom ParisTech - LabSoC
EURECOM, CS 50193 - 06904 Sophia Antipolis cedex
Email: feriel.benabdallah, ludovic.apvrille@telecom-paristech.fr

***Abstract*— DIPLODOCUS is a UML profile targeting the design space exploration of Systems-on-Chip at a high level of abstraction. It is supported by an open source toolkit named TTool, which enables the designer to develop, simulate and formally verify SoC models. Up to now, performance criteria such as CPU and bus loads were the main metrics used for the exploration. The main objective of our research works is to provide methods and tools to quickly estimate the power consumption at the first steps of a system design. This work enhances DIPLODOCUS with power consumption modelling and analysis. Both software and hardware power management techniques can now be modelled in an abstract way. So enhanced DIPLODOCUS offers the possibility to evaluate different power management policies, and to estimate the system power consumption.**

I. Introduction and motivation

Power management is a critical design constraint particularly in mobile devices as a high power consumption reduces the lifetime of battery-operated systems and increases heat dissipation which requires expensive cooling technologies. To address this issue, many power reduction techniques have been proposed in the literature. For example, the dynamic voltage/frequency scaling (DVFS) technique [8] adjusts on the fly the operating frequency as well as the voltage of electronic components with respect to changing workloads or to a given power management strategy. The clock gating [5] is a special case of DVFS with two frequencies (nominal frequency and zero frequency). It consists of freezing the clocks of inactive elements (e.g., processors) in order to save power. This technique is widely used because it is conceptually simple and it is ideally suited for implementing self-managed components. Dynamic power reduction techniques are usually complemented with a manager. The latter takes decision on the choice of operation modes according to the execution of applications in the system, e.g., according to the computational complexity of tasks. With the advent of very complex applications on mobile devices, managers have also become very complex, commonly working closely with Operating Systems, thus taking decisions from information on the load of the systems, the types of tasks under execution, etc.

One critical problem is that reliable results on the power savings techniques can only be obtained at late design stages, which might be too late for design consideration [6]. It is noteworthy that wrong estimates of power consumption may lead to a poor system performance or even to system failures, e.g, in a system where critical tasks are running, a wrong estimate of the remaining charge of the battery can cause a system failure, also a bad estimate of required energy consumption may cause inadequate choice of the architecture which can be very expensive. Thus, it is imperative to address power consumption early in the design flow. To efficiently guide early choices, high-level power estimation and optimization would be extremely beneficial. Moreover, to explore systems on chip in a exhaustive and efficient way, tools and unified interfaces for timing, performance, and power analysis exploration are needed. In this context, Model Driven Development (MDD) [3] was used for power estimation at high level of abstraction ([16], [9]). This methodology allows developers to use models to specify what system functionality is required and what architecture is to be used instead of the use of a programming language defining how the system is implemented. MDD is based essentially on meta models, models and model transformation concepts and it aims at constructing an abstraction layer to avoid dealing with details and allow model reutilization. DIPLODOCUS [1] is a UML-based environment introduced a few years ago to address the design space exploration (DSE) of Systems-on-Chip. A UML profile customizes UML for a given domain, using its extension capabilities [18].

This paper presents how DIPLODOCUS has been extended to efficiently support power consumption evaluation. Basically, our solution relies on the definition of new UML diagrams to capture in a fast and abstract way power management techniques and DVFS operational modes. Also, the overall power consumption of a system mapped onto a given hardware architecture can be evaluated.

This paper is organized as follows: Section 2 presents research work related to high level power evaluation techniques. Section 3 introduces main features of DIPLODOCUS. Section 4 describes the extensions introduced to DIPLODOCUS to include power consumption / management aspects in DSE. Section 5 presents the experimental results, and finally section 6 concludes the paper.

978-0-7695-5091-6/13 $26.00 © 2013 IEEE
DOI 10.1109/SEAA.2013.8

II. Related work

Several research works focus on the evaluation of power consumption at a high level of abstraction. For example, Tiwari et al. [21] proposed a first approach for evaluating the power consumption in embedded systems at a high-level through the introduction of the concept of Instruction Level Power Analysis (ILPA). ILPA defines a base cost for each processor instruction in order to compute the average of the overall power consumption of a given realistic benchmark program. One important drawback of this technique is the evaluation of inter-instruction overheads that depend on the neighboring instructions, which is not necessarily reflected in the cost computed solely from base cost. Moreover, since ILPA is built around an instruction set simulator (ISS), it is considered as a slow technique for large DSE (Design Space Exploration) [19].

In order to have a better accuracy, Laurent et al. [14] proposed the FLPA (Functional Level Power Analysis) which cut processors into functional block like processing unit, internal memory, instruction management unit, and others, in contrast to ILPA that considers the processor as a black box model. Laurent et al. establish a functional analysis for each block to specify and discard the non consuming blocks. Then, in order to estimate the overall consumption, different measurements are then used to figure out the parameters that affect the power consumption of each consuming block. Power models of these blocks are finally determined using linear regression with a set of power measurements. Even though the FLPA technique seems more accurate than the ILPA, it is still considered as an expensive technique for early power system-level DSE. Indeed, the designer needs an optimized definition of the CPU blocks and a time consuming evaluation of their power parameters.

We can overcome the limitations of measurement-based power estimation methodologies (ILPA, FLPA) by raising the level of abstraction at which the system description, analysis and simulations are performed. Raising the level of abstraction means describing the applications and the power management mechanisms in a higher (more abastract) level. We call this methodology Model Driven Power estimation (MDP). Our methodology allows for rapid estimation of power consumption in the earliest stages of development, i.e., it allows for design space exploration at the model level, considering not only functional aspects but also those related to energy consumption such as temperature, battery, etc. In this context, Arpinen et al. [2] proposed extensions to the OMG UML profile MARTE (Modeling and Analysis of Real-Time and Embedded systems) [17]. Marte defines some features to characterize power consumption in embedded systems (power package, power supply package, battery package, and cooling supply package). Arpinen et al. added features allowing the designer to model dynamic power management policies basing on finite state machines. This approach is still at the conceptual level, and no analysis tools have been developed yet to evaluate energy dissipation.

On the other hand, DIPLODOCUS / TTool is an advanced DSE environment for functional aspects of embedded systems but it still lack the concepts of power consumption definition and analysis. The purpose of this work is to extend it to integrate the power consumption aspects of an embedded system. We provide the possibility to establish a fast simulation and analysis of results when executing an application scenario with different power management policies. Our approach relies on the use of abstract power estimation models, using an UML-based estimation methodology. More precisely we propose extensions to the DIPLODOCUS UML profile in order to add the missing concepts for the definition and analysis of power consumption aspects. The following section briefly introduces DIPLODOCUS on which power consumption extensions are then applied.

III. DIPLODOCUS

DIPLODOCUS is a UML profile targeting the design space exploration of embedded systems at a high level of abstraction. It is implemented by an open source toolkit named TTool [1] and is based on the Y methodology [11]. Indeed, DIPLODOCUS clearly separates application and architecture modeling (Fig. 1). Its abstractions allows for fast simulation and formal analysis techniques. In particular, DIPLODOCUS communication channels carry only unvalued samples. The main objective of our UML profile is to help designers to spot a suitable hardware/software partitioning regarding functionality, performance, silicon area, power consumption metrics. The DIPLODOCUS modeling methodology is based on three main stages:

Application modeling: The application platform is modeled as a network of tasks communicating with channels, events, and requests. Each task behavior is described with a UML activity diagram extended with communication operators and abstract computational complexity operators (e.g., EXECI x, i.e. execute x int operations). Three communication and synchronization operators have been defined [13]:

1) **Channels** are characterized by a point-to-point unidirectional communication between two tasks. Channel types are:
   a) Blocking Read/Blocking Write (BR-BW)
   b) Blocking Read/Non Blocking Write (BR-NBW)
   c) Non Blocking Read/Non Blocking Write (NBR-NBW)
2) **Events** are characterized by a point-to-point unidirectional asynchronous communication between two tasks. Events are stored in an intermediate FIFO queue which may be finite or infinite.
3) **Requests** are characterized by a multi-point to one point unidirectional asynchronous communication between tasks. A unique infinite FIFO between senders and the receiver is used to store all incoming requests. Consequently, a request cannot be lost.

Architecture modeling: The architecture platform is modeled as a set of interconnected parametrizable execution nodes

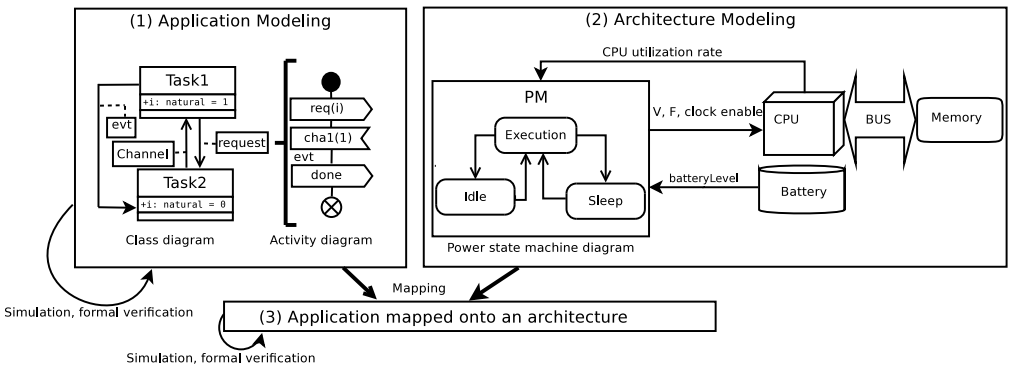


Fig. 1. DIPLODOCUS modeling methodology integrating the power manager

(CPUs, HW accelerators, DSPs), storage nodes (memories), and communication nodes (buses and bridges). It is described with a UML deployment diagram where DIPLODOCUS links and nodes have their own UML stereotype.

Mapping: The mapping phase is described using a set of interconnected hardware nodes on which tasks, channels, events and requests are mapped.

DIPLODOCUS is amenable for modeling and analyzing embedded systems using fast simulation techniques. The simulation speed benefits from the high abstraction level of both application and architecture models, as compared to simulations usually performed at lower abstraction levels (e.g. TLM level, RTL level, etc.). Additionally, formal analysis [12] may be applied using automatic code generation to the formal languages such as LOTOS [7] or UPPAAL [4].

IV. Power Management for DIPLODOCUS UML profile

The main objective of this work is to define a UML profile allowing designers to evaluate the impact on their systems on various power management policies, and therefore to capture candidate power management policies - handmade one, predefined policies - and then to evaluate the energy consumption resulting from these different policies. As a first approach, we focus our work on CPUs power consumption only. The power consumption of a processor is caused by two factors: the first one is the dynamic dissipation due to the charging and discharging of load capacitances, the second one is the static dissipation due to the leakage current. The dynamic power consumption defined in [22] is given by:

$$P_{dynamic} = C * f * V_{dd}^2 * \alpha \quad (1)$$

where V_{dd} is the supply voltage, f is the operating frequency and α is the switching activity. α represents the percentage of CMOS circuit switches during one clock tick and C is the total circuit capacitance. The static power consumption defined in [22] is given by:

$$P_{static} = I_{leakage} * V_{dd} \quad (2)$$

Two main sources of leakage currents are the sub-threshold drain-source and gate leakage effects. The static power is not dependent of operating frequency, and thus the static power does not contribute to the computational performance of components in a similar manner as the dynamic power [20]. Eq (1) informs us about the CPU's power-related characteristics that should be defined. Thus, we extend DIPLODOCUS CPU metaclass to a new stereotype named $EnergyAwareCPU$ (Fig. 3). Note the DIPLODOCUS CPU model abstracts the hardware processing unit and its operating system. Its behaviour can be customized by the following parameters (amongst others): cache miss ratio, number of cores and scheduling algorithm [12]. As for V_{dd} and f in Eq(1), we set them in a new designed profile named $DynamicPowerManager$. This choice is explained by the fact that recent processors commonly support several voltage and frequency levels i.e., they have different power Operating Points (OP), and they can be switched from one OP to another one. We associate with each $EnergyAwareCPU$ a $DynamicPowerManager$. The idea is to build up a FPSM ($FinitePowerStateMachine$) for each CPU, and then link it with a defined power management policy that is in charge of selecting the current OP, and which therefore triggers the transitions between OPs (f, v). Fig. 2 represents the power manager metamodel that covers both software and hardware parts, and which are mainly composed of a paramatrizable FPSM that we call $CPUFinitePowerStateMachine$ and a $PowerPolicyManager$.

A. CPU Finite Power State Machine

In order to define the FPSM, we extend the UML metaclass $StateMachine$ to a new stereotype named $CPUFinitePowerStateMachine$ which contains two subclasses representing states $PowerState$, and transitions between states $PowerStateTransition$. $PowerState$ is mainly characterized with frequency, voltage values, and the static power consumption defined in Eq (2). In the scope of an energy-aware design space exploration, the designer can experiment with any number of CPU power states. The stereotype $PowerStateTransition$ extends the UML metaclass $Transition$ by introducing one extra attribute defining the duration of each transition between two states. Each transition has power and delay costs and is triggered according to power

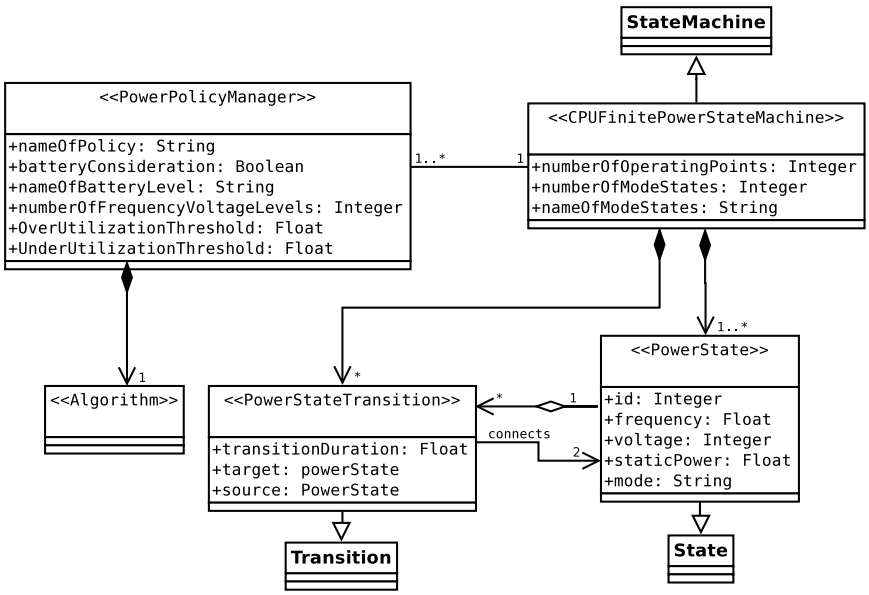


Fig. 2. Dynamic Power Manager Metamodel

management policies specified by the designer.
Let us consider a basic but representative example. We consider the following CPU modes: *Execution*, *Sleep*, and *Idle* (Fig.4). Switching from one OP to another may produce a significant energy gain at the cost of performance: this is a way to avoid energy waste when the system does not need high computation resources. The choice of the adequate OP within a certain time interval T, is a critical task for the power manager because of the tradeoff between the cost of staying in or changing the OPs. For instance, if entering a lower power state requires power supply shutdown, returning from this state to the active one requires restoring the context.

B. Power Policy Manager

Power management algorithms called policies can be implemented in either hardware or software. The PPM (Power Policy Manager) is defined through the stereotype $PowerPolicyManager$ (see Fig. 2). The PPM is an entity meant to decide - using a given policy - which operating point is the most suitable one to minimize the system power consumption. $PowerPolicyManager$ contains a list of decision policies and a boolean attribute that specifies whether the battery is taken into consideration or not. When designers are not meant to describe their own power management policies, the DIPLODOCUS DPM profile includes predefined policies that rely on the CPU utilization rate, on timers, and on battery level metrics. These three parameters can be used in one single policy, but for the sake of clarity we present their use separately. More complex policies can be easily built, so as to offer more design space exploration possibilities.

1) Policy 1: CPU rate of utilization: Many studies have shown that the energy consumption increases with the CPU rate of utilization [15]. One possible way to control a CPU

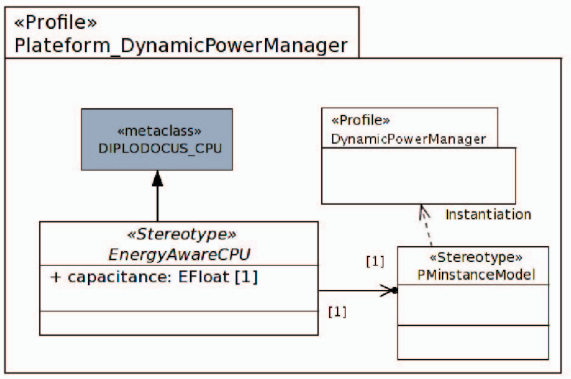
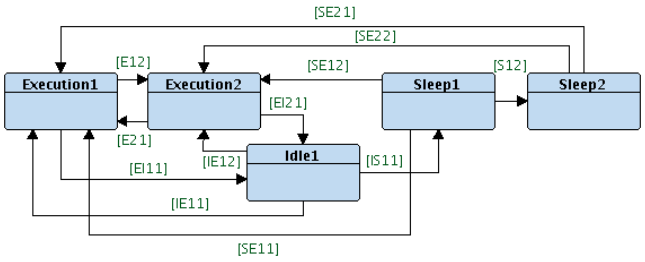


Fig. 3. DIPLODOCUS Extension for DPM

energy consumption is to define *thresholds*- defined based on aspects related to energy consumption- stating whether the CPU has an over utilization or an under utilization. As CPUs have several operating points, we propose to set for each operating point $O_k(f_k, v_k)$ two corresponding thresholds: U_{o_k} and U_{u_k} for over and under utilization, respectively. For a given time interval t, we can measure the utilization rate, say U, of the CPU under its actual operating point O_k, and with U_{o_m} and U_{u_m} as thresholds. We can then define the power manager policy as follows (see algorithms 1 and 2). The first algorithm, manages the under utilization case where we seek the first operating point O_k whose f_k and v_k are lower than those of the actual operating point. O_k should satisfy $U > U_{u_k}$ condition. Algorithm 2 manages the over utilization case where we seek the first operating point O_k whose f_k and v_k are bigger than the actual operating point. O_k should satisfy $U < U_{o_k}$ condition. In case of



Transition	Condition (policy 1)
E12	$U < U_{u_{Execution1}} and\ U < U_{o_{Execution2}}$
E21	$U > U_{o_{Execution2}}$
SE21	$U > U_{o_{Sleep2}}\ and\ U > U_{o_{Execution2}}$
SE22	$U > U_{o_{Sleep2}}\ and\ U > U_{o_{Execution2}}$

Fig. 4. Example of CPU Finite Power State machine

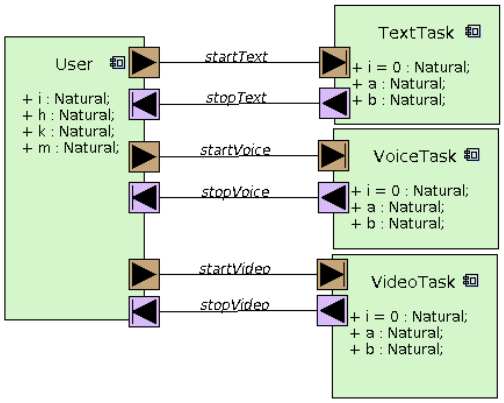


Fig. 5. Component based diagram of the Application

conflict between operating points, we always select the under utilization operating point.

2) Policy 2: battery level: One important point when defining the power manager policy related to the set of available operating points is to consider the battery levels (see Table I). This policy could depend on the importance of executed tasks, just because some of them could be critical, and must ensure for example their dealine. Thus, we give the designer the possibility to change the behavior of the power manager according to battery level and priority of running tasks.
Let us now consider the simple model we presented in Fig. 4. We define in Table I operating points accordingly to battery levels, with $f_5 < f_4 < f_3 < f_2 < f_1$ and $v_5 < v_4 < v_3 < v_2 < v_1$. Note that a couple ($f_5$, v_5) consumes much less energy than the couple (f_1, v_1). So the battery related policy of this example consists of decreasing the number of available operating points of execution, with the use of low frequency/voltage couples, according to the battery levels, in order to save energy. For example, as long as battery power is good tasks are allowed to run at the highest frequency (f_1 in our example). Whereas, when the battery becomes critical the system scales back to lower frequency and lower voltage running mode (in our example (f_3,v_3)).

3) Policy 3: Timeout: The timeout policy is an industry standard for DPM [10]. It is defined as follows: when an idle time starts, a timer (with duration T_f) is triggered. If after T_f the system remains idle, then the power manager switches the system to off state, until it receives an interruption that marks the end of the idle time. Timeout policy is very common technique of power reduction in embedded systems thanks to its simplicity. The drawback of Timeout policy is that it wastes energy while waiting for the timeout to expire.

V. Experimental results

In this section, we report on experiments using a very simple case study. We compare two power-management policies based on their respective impact on the energy consumption for a given embedded system, i.e. with the same tasks, the same hardware architecture, and the same mapping. The first policy simply assumes the CPU frequency is fixed to a given value during all the system execution - which is the usual way DIPLODOCUS operates, whereas the second one consists in selecting different CPU frequencies according to the computational requirements of the application tasks.

A. Application Model

Fig. 5 presents the DIPLODOCUS task diagram of our proposed application. The diagram has three periodic tasks named *Text*, *Voice* and *Video* - their period is the same -, as well as a main control task called *User*. The *User* task abstracts three possible scenarios: (i) concurrent execution of *Text*, *Voice* and *Video* tasks. (ii) The concurrent execution of *Text* and *Voice*, and (iii) execution of the *Text* task only. The behaviour of the three periodic tasks is abstracted with a computational cost operator *EXECI x* whose value x is different for each task. For example, the *Video* task requires a higher computational power than the *Voice* task. The latter requires higher computational power than *Text* task.
The fact that we can have several scenarios with different computational requirements makes the energetic analysis interesting with this abstract application. Indeed, the power manager will switch from an operating point to another one differently in each scenario because the computational requirements of each scenario strongly differ.

B. Architecture Model and Mapping

For the easiness of results presentation, we use one single CPU onto which the four tasks are mapped.

C. Power Manager Model

1) Power states and Power transitions: As explained in the previous sections, the power states and transitions of

Algorithm 1 Under utilization Case

$U < U_{u_m}$
$k = m + 1$
boolean $chosenstate = 0$
while $chosenstate == 0$ and $k < M + 1$ **do**
  pick operating point k;
  calculate U;
  if $U < U_{u_k}$ **then**
    $k = k + 1$;
  else
    if $U > U_{o_k}$ **then**
      choose operating point O_{k-1};
    else
      choose operating point O_k;
    end if
    $chosenstate = 1$;
  end if
end while
if $chosenstate == 0$ **then**
  choose the default designer policy in this case
end if

Algorithm 2 Over utilization Case

$U > U_{o_m}$
$k = m - 1$
boolean $chosenstate = 0$;
while $chosenstate == 0$ and $k > 0$ **do**
  pick operating point k;
  calculate U;
  if $U > U_{o_k}$ **then**
    $k = k - 1$;
  else
    choose operating point O_k;
    $chosenstate = 1$;
  end if
end while
if $chosenstate == 0$ **then**
  choose the default designer policy in this case
end if

Policy	$Execution_1$	$Execution_2$	$Idle_1$	$sleep_1$	$sleep_2$
Battery critical	f_3,v_3	f_3,v_3	f_3,v_3	f_4,v_4	f_5,v_5
Battery low	f_1,v_1	f_3,v_3	f_3,v_3	f_4,v_4	f_5,v_5
Battery good	f_1,v_1	f_2,v_2	f_3,v_3	f_4,v_4	f_5,v_5

TABLE I
POWER MANAGEMENT POLICY WITH CONSIDERATION OF BATTERY

the CPU are modelled using the DPM profile. For the first case, we model the FPSM by a single state with frequency $f_1 = 400\ MHz$ and voltage $v_1 = 1.72\ V$, which constitute the unique CPU operating point. For the second case, the FPSM is composed of two power state, linked by two power transitions. RUNNING1 characterized by the operating point ($f_{21} = 400\ MHz$, $v_{21} = 1.72\ V$) and RUNNIN2 characterized by the operating point ($f_{22} = 133\ MHz$, $v_{22} = 1.55\ V$). We consider that the transitions between different modes takes zero time units. Note that the transition duration is parametrizable through the DPM profile.

2) Power Policy: An important issue in real time applications is to respect time constraints. To do so, we suggest not to overpass a certain CPU load value. Also, this metric can be very useful for power consumption optimization since it gives a possible way to control the CPU frequency and voltage by the definition of two thresholds stating whether the CPU has an over utilization or an under utilization. Over and under utilizations can also be defined according to the right respect of time constraints, or not. These thresholds are defined based on aspects related to energy consumption such as heat dissipation and battery charge. Finally, we use the following formula to compute the CPU load every time a task starts or ends execution:

$$CPU_{Load} = \sum_{i=1}^{n} (C_i * TimePerCycle)/T_i \qquad (3)$$

where C_i is the computational complexity in cycles of the task i, T_i is its associated period, and n the number of running tasks. $C_i = C_{iE} + C_{iC}$ where C_{iE} is the computational time

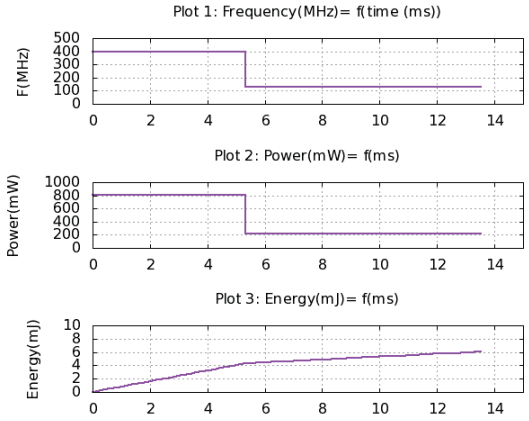


Fig. 6. Simulation results of the second policy

of the execution operations, and C_{iC} is the computational time of communication operations. $C_{iE} = \sum_{i=1}^{n} n_{EXECI}$.
$C_{iC} = a_R * t_c/t_b + a_W * t_c/t_b$, with a_R the number of read samples, a_W the number of writen samples, t_b bus size, and t_c channel size. The first policy relying on only one Operating Point, under and over utilization thresholds are defined only for the second policy: 30% and 80% for $OP = (f_{21}$,$v_{21})$, and 0% and 90% for $OP = (f_{22}$,$v_{22})$, respectively.

D. Results and discussion

We model the following scenario, the user sends requests to use Video, Voice and Text, then he/she uses Voice and Text without Video, and finally he/she uses Text only. When we

use the first policy, we start with a CPU load of 68%. Once the Video is over, the CPU load becomes 28%, and at the end of the scenario the CPU load is 8%. Notice that we cannot run all this scenario with the lowest frequency as the time constraint would not be respected (the CPU load would be greater than 1), and so we can use only $OP = (f_1,\ v_1)$ OP. When we use the second policy with our modeled power manager, the scenario starts with a CPU load similar to the one of the first policy (i.e., 68%), then it changes to 28% when the video ends, which is lower than the threshold 30%. Our algorithm checks the FPSM associated with the CPU to know the possible frequencies that the system may use, in order to lower the consumption, but without having an over utilization in the CPU load. The power manager decides to switch to the operating point $(f_{22},\ v_{22})$, in which time constraints are still respected. Thus, the power consumption decreases without altering the functional constraints. The CPU load becomes 84% when running the Text and Voice, and then 24% when running the Text only. For the first policy we have a total energy consumption of $11.278\ mJ$, whereas for the second policy we have a total energy consumption of $6.08\ mJ$. Fig. 6 shows the simulation results of the second policy. Notice that the slope of the energy consumption curve is reduced when the frequency decreases which explains the energetic gain. Naturally an adequate choice would be to choose the second policy from an energy saving perspective. We clearly show that our modelling approach enabled us to perform a costless energetic analysis, in a high abstract level. It is noteworthy to mention that we have used a very short simulation time for the easiness of presentation.

VI. Conclusion and future directions

The Design Space Exploration as proposed by the DIPLODOCUS UML profile was missing an important metric: power consumption evaluation. Thus, this paper explains how DIPLODOCUS can be extended with the possibility to model both hardware and software-related power management techniques: definition of operation modes, and description of policies to select operation modes. An intuitive case study demonstrates the relevance of our approach.

Yet, two important points are still to be studied. First, we need to validate the relevance of the power consumption abstractions that have been defined by comparing the results obtained for large-scale applications, that is, we need to compare results obtained at DIPLODOCUS levels with the results that would be obtained with the implementation of these applications. Second, we intend to formally verify safety properties (e.g., deadlock absence) of power managers modeled in TTool.

References

[1] Ludovic Apvrille. Ttool for diplodocus: An environment for design space exploration. In *Proceedings of the 8th international conference on New technologies in distributed systems*, NOTERE '08, pages 28:1–28:4, New York, NY, USA, 2008. ACM.

[2] Tero Arpinen, Erno Salminen, Timo D. Hmlinen, and Marko Hnnikinen. Marte profile extension for modeling dynamic power management of embedded systems. *Journal of Systems Architecture - Embedded Systems Design*, pages 209–219, 2012.

[3] Colin Atkinson and Thomas Kühne. Model-driven development: A metamodeling foundation. *IEEE Softw.*, 20(5):36–41, 2003.

[4] Gerd Behrmann, Re David, Kim G. Larsen, M. Oliver Mller, Paul Pettersson, Wang Yi, Johan Bengtsson, Fredrik Larsson, Alexandre David, Tobias Amnell, Arne Skou, Carsten Weise, Thomas Hune, Ansgar Fehnker, Klaus Havelund, Judi Romijn, Franck Cassez, Franois Laroussinie, Patricia Bouyer, and Justin" Pearson. Uppaal: Model checking timed automata.

[5] Luca Benini and Giovanni de Micheli. *Dynamic Power Management: Design Techniques and CAD Tools*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[6] Luca Benini and Giovanni De Micheli. System-level power optimization: Techniques and tools. *ACM TRANSACTIONS ON DESIGN AUTOMATION OF ELECTRONIC SYSTEMS*, 5(2):115–192, 2000.

[7] Tommaso Bolognesi and Ed Brinksma. Introduction to the iso specification language lotos. *Comput. Netw. ISDN Syst.*, 14(1):25–59, March 1987.

[8] Kihwan Choi, Wonbok Lee, Ramakrishna Soma, and Massoud Pedram. Dynamic voltage and frequency scaling under a precise energy model considering variable and fixed components of the system power dissipation. In *IEEE/ACM International Conference on Computer Aided Design*, November 2004.

[9] Saadia Dhouib, Eric Senn, Jean-Philippe Diguet, Johann Laurent, and Dominique Blouin. Model driven high-level power estimation of embedded operating systems communication services. *Embedded Software and Systems, Second International Conference on*, 0:475–481, 2009.

[10] Q. Jiang, H.-S. Xi, and B.-Q. Yin. Adaptive optimisation of timeout policy for dynamic power management based on semi-markov control processes. *Control Theory Applications, IET*, 4(10):1945 –1958, october 2010.

[11] Bart Kienhuis, Ed F. Deprettere, Pieter van der Wolf, and Kees A. Vissers. A methodology to design programmable embedded systems - the y-chart approach. In *Embedded Processor Design Challenges: Systems, Architectures, Modeling, and Simulation - SAMOS*, pages 18–37, London, UK, UK, 2002. Springer-Verlag.

[12] D. Knorreck, L. Apvrille, and R. Pacalet. Fast simulation techniques for design space exploration. In *47th International Conference Objects, Models, Components, Patterns*, volume 33, pages 308–327, Zurich, Switzerland, June 2009.

[13] Daniel Knorreck, Ludovic Apvrille, and Renaud Pacalet. Formal system-level design space exploration. *Concurrency and Computation: Practice and Experience*, 25(2):250–264, 2013.

[14] Johann Laurent, Eric Senn, Nathalie Julien, and Eric Martin. High Level Energy Estimation for DSP Systems. In *in Proc. Int. Workshop on Power And Timing Modeling, Optimization and Simulation PATMOS*, pages 311–316, 2001.

[15] Chia-Hung Lien, Ying-Wen Bai, and Ming-Bo Lin. Estimation by software for the power consumption of streaming-media servers. *Instrumentation and Measurement, IEEE Transactions on*, 56(5):1859 –1870, oct. 2007.

[16] Ons Mbarek, Amani Khecharem, Alain Pegatoquet, and Michel Auguin. Using model driven engineering to reliably accelerate early low power intent exploration for a system-on-chip design. In *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, SAC '12, pages 1580–1587, New York, NY, USA, 2012. ACM.

[17] O M G Document Number and Associated Files. Uml profile for marte : Modeling and analysis of real-time embedded systems. *Engineering*, 15(November):738, 2009.

[18] OMG. Unified Modeling Language (OMG UML). (November), 2007.

[19] Roberta Piscitelli and Andy Pimentel. A high-level power model for mpsoc on fpga. *IEEE Computer Architecture Letters*, 99, 2011.

[20] Jan Rabaey. *Low Power Design Essentials*. Springer Publishing Company, Incorporated, 1st edition, 2009.

[21] Vivek Tiwari, Sharad Malik, and Andrew Wolfe. Power analysis of embedded software: A first step towards software power minimization. *IEEE Transactions on VLSI Systems*, 2:437–445, 1994.

[22] Wei Zhang, James Williamson, and Li" Shang. chapter Power Dissipation, pages 41–80. Springer US, 2011.