Analysis of Multi-Domain Scenarios for Optimized Dynamic Power Management Strategies

Jochen Zimmermann†, Oliver Bringmann†
†FZI Karlsruhe, Germany
{zimmermann,bringmann}@fzi.de

Wolfgang Rosenstiel†‡
‡University of Tuebingen, Germany
rosenstiel@informatik.uni-tuebingen.de

***Abstract*—Synchronous dataflow (SDF) models are gaining increased attention in designing software-intensive embedded systems. Especially in the signal processing and multimedia domain, dataflow-oriented models of computation are commonly used by designers reflecting the regular structure of algorithms and providing an intuitive way to specify both sequential and concurrent system functionality. Furthermore, dataflow-oriented models are qualified for capturing dynamic behavior due to data-dependent execution. In this work, we extend those data-dependent dataflow models to include dynamic power management (DPM) aspects of a target platform while still meeting hard timing requirements. We capture different system states in a multi-domain scenario approach and develop a state space based on this SDF representation for system analysis and optimization. By traversing the state space of the power-aware scenario modeling we derive a power management configuration with minimized energy dissipation depending on dynamic system behavior.**

I. Introduction

Besides of guaranteeing the strict observance of deadlines in real-time applications, power consumption has emerged as one of the most important factors in embedded system design, especially if systems are part of ultra-portable devices or depend on long battery lifetimes in general.

On the hardware side of embedded electronic systems, and especially in System-on-Chip (SoC) designs, there exist several approved low-power technologies like clock/power gating and multiple voltage islands. However, today's embedded functionality is growingly implemented in software due to flexibility and cost reduction reasons. Additional to compiler-based power optimization, dynamic voltage and frequency scaling (DVFS) with several different operating (power) modes of the system is known to be one of the most efficient low-power techniques on software execution level.

Synchronous dataflow (SDF) models [1], as well as their modifications cyclo-static (CSDF) and scenario-aware dataflow (SADF) models [2], got a lot of attraction recently in modeling and specification of embedded software. They allow to model dataflow-oriented applications in a very intuitive way which is usually a strong requirement in industrial development processes. Furthermore, one can easily express parallelism which eases application development for multicore and manycore systems.

SDF models are of special interest for (hard) real-time applications. They are statically analyzable as opposed to more general models like Kahn Process Networks (KPN) [3] which are proven to be undecideable. Hence, they allow to provide system performance guarantees at design time. [4] shows an approach to perform worst-case performance analysis for synchronous dataflow scenarios, e.g. worst-case throughput, but without considering any power-related issues. By definition, an SDF graph is a timeless, connected graph where each vertex (in SDF syntax called actor) represents some sort of processing and each edge transports data (tokens) between those actors. When an actor is executed (fires), it consumes a certain amount of input tokens and produces a certain amount of output tokens (input and output rates) on the FIFO channels it is connected to. A prerequisite for actor firing is the availability of a sufficient amount of input tokens. For modeling embedded software behavior, a so called Timed-SDF is commonly used where each actor is associated with a (usually) worst case execution time (WCET).

Many applications in the embedded domain are designed as dynamic streaming applications, which means that their actual behavior changes depending on their current state or input data. A scenario-aware synchronous dataflow (SASDF) model can be used to capture such application states and make them analyzable.

This work was partially supported by ITEA project VERDE under BMBF grant 01IS09012A and BMBF project RESCAR 2.0 under grant 01M3195E.

978-3-9810801-8-6/DATE12/©2012 EDAA

II. Related Work

Synchronous dataflow (SDF) models were first introduced by Lee and Messerschmitt [1]. Static schedulability of SDF graphs (SDFG) have been shown by Buck [5]. Recently, a lot of research work has been done on performance analysis and predictability, as well as multi- and manycore scheduling and mapping [6]. First approaches to include dynamic behavior in SDF models were researched by Buck [7] introducing actors whose dataflow behavior is affected by boolean-valued control tokens and extending analysis and scheduling techniques of those SDF models. Cyclo-static SDF (CSDF) models were introduced to overcome the limitation of static token production and consumption rates in usual SDF models by defining different phases of an actor's input or output. In general SDF context, scenario-awareness has been researched by Stuijk, Gheorghita et al. [8] [9], senario-aware voltage scaling was introduced in [10].

Optimizing the system energy consumption is a wide research field. Benini [11] and Iranli [12] give an overview of recent low-power techniques and dynamic power management (DPM). Optimization approaches at design time are usually related to energy-aware scheduling based on more general task graphs rather than SDF [13]. Both Andrei et al. [14] use an approach based on integer linear programming (ILP) to minimize the energy consumption. However, ILP problems are not able to capture dynamic runtime data-dependable behavior and their complexity grows exponentially with the exploration search space. Rong et al. [15] present a run-time approach for optimizing processor time-out values based on Markovian Processes. Probability-based DPM is shown in [16]. Isci et al. [17] try to maximize performance while still keeping a given power budget. In [18], a DVFS approach based on the feedback of output queues is presented for streaming applications.

III. Definition of Multi-Domain Scenarios

Scenarios are used to model different modes of operation to capture dynamic behavior [19]. In past research activities, mainly different worst case execution times of dynamic applications have been defined by scenarios e.g. capturing different

frame types in multimedia algorithms. In the SDF model of computation dynamic behavior means changing the properties of an actor during runtime.

However, application-oriented scenarios are not enough when it comes to terms of non-functional properties (NFP) like average power dissipation because they are not solely influenced by the application itself, but also by shared resources and especially the underlying platform configuration. Therefore, we suggest to consider also platform scenarios, e.g. computing resources running in different power modes with different execution times for the application (see Figure 1).

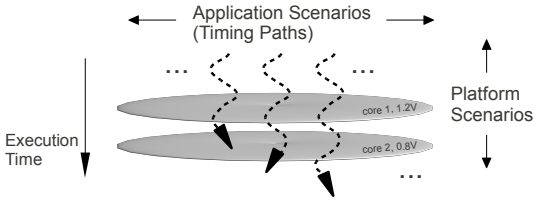

Fig. 1. Schematic Overview of Multi-Domain Scenarios

A. Application Scenario Models

Application scenarios [2] are used to capture dynamics in communication and computation of the application, e.g. data-dependent execution times. We especially use them to model different timing paths inside the application to encode controlflow properties. They are modeled either stochastically with markov chains or with finite state machines (FSM).

An *application scenario-aware dataflow graph* is a 4-tuple (V,E,Φ, Ψ) of an SDF-Graph (V,E) and contains both an SDF representation of the application including actor dependencies and an FSM describing the different application scenarios (see upper part of Figure 2). For each derived scenario a function Φ: $V \rightarrow \mathbb{R}^n$ determines an actors properties (execution time, power dissipation) in the actual state of the application scenario FSM.

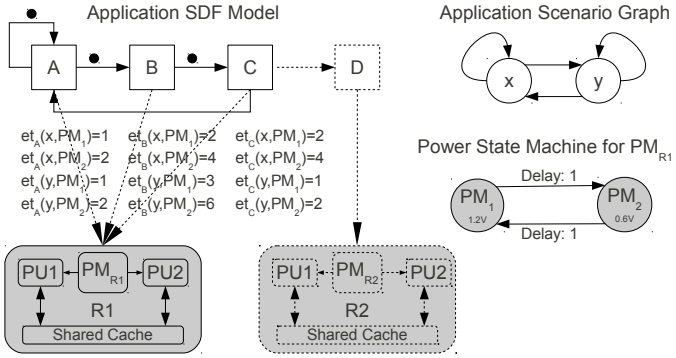

Fig. 2. Application Specification, Mapping and PSM

For further definitions of application scenarios, as well as proofs for boundedness and deadlock-freedom of markov-based scenario-aware dataflow graphs we refer to [20].

However, the execution time et_a of an actor a depends not only on the application scenario, but also on the power mode of the resource a is actually executed in. Let AS be the set of all application scenarios and PM^{R1} the set of power modes of resource $R1$. Then the execution time et_a after being mapped to resource $R1$ is defined as $et_a : AS \times PM^{R1} \rightarrow \mathbb{R}$. Furthermore, switching the power mode implies an additional switching delay. Without loss of generality and to ease the understanding, we choose the example graph in Figure 2 similar to [4] and in low power mode PM_2 twice the execution times as in PM_1. In general, the delays correspond to reduced operational frequencies due to reduced supply voltages and can be taken from target architecture specifications.

B. Platform Scenario Models

Platform scenarios describe different states of the underlying platform, e.g. cores, caches, and the memory subsystem. For platforms we use finite state machines to model different power states, the switching overhead as well as its implications on actors' properties (execution time, power consumption).

Definition 1 (PSM): A power-state machine (PSM) of system S is a 4-tuple $(V_{PSM}, E_{PSM}, \delta_{PSM}, \rho_{PSM})$ where V_{PSM} is a set of nodes defining the available power modes of S, $E_{PSM} \subseteq V_{PSM} \times V_{PSM}$ is a set of directed edges. Function $\delta_{PSM}(v_i, v_j) : V_{PSM} \times V_{PSM} \rightarrow \mathbb{R}$ defines the switching delay between nodes $v_i, v_j \in V_{PSM}$ and function $\rho_{PSM}(v_i, v_j) : V_{PSM} \times V_{PSM} \rightarrow \mathbb{R}$ defines the switching power between nodes $v_i, v_j \in V_{PSM}$.

Definition 2 (Platform Scenario): A platform scenario PS_i is a mode of operation of platform entities connected to one or more actors a_j and influencing the actors properties $Prop(a_j)$. A platform entity is directly used by actors to process data according to the appropriate platform state.

A *platform scenario graph* is a 4-tuple (Q, Σ, δ, q_0) and defines statically the switching points between different platform scenarios for each related platform entity. It consists of a finite set of system states Q, a finite set of transition events Σ in one or more assigned PSMs, a transition function $\delta : Q \times \Sigma \rightarrow Q$, and a starting state q_0. The transition function δ triggers the PSMs and assigns the switching overhead to the execution time of the actors mapped to the switching platform entity.

IV. Multi-Domain Scenario Model

We will derive a multi-domain scenario model by the development of a state space which contains states for combinations of application scenario execution and platform power modes such that a specified throughput is guaranteed.

A (max,+) algebra is used to capture SDF semantics.Let T be the set of input tokens of an actor a and for every token $\tau \in T$ t_τ is the arrival time of token τ. Furthermore, let et_a be the execution time of this actor. The (max,+) algebra reflects that the earliest time of tokens being available for the next actor after firing of actor a is defined as $max_{\tau \in T} t_\tau + et_a$. For (max +) algebra, mainly the standard semantics [21] and some extensions are used which are briefly defined in the following.

Let b and c be vectors, $b = [b_i]$ and $c = [c_i]$ with $b_i, c_i \in \mathbb{R}^{-\infty}$, d a scalar and M a matrix with column vectors m_j.

- $max(-\infty, d) = max(d, -\infty) = d$, the maximum operator with $-\infty$ being the neutral element
- $b+d$ denotes $[b_i+d]$, adding a scalar to each vector entity
- $b + c$ results in a vector $[b_i + c_i]$
- $\|b\| = max_i b_i$
- b_{norm} denotes $b - \|b\|$, the normalized vector b
- Md is defined as $max_j(m_j + d)$
- $b^{mod\ d} := [b_i \bmod d]$ denotes the modulo operation on each entity in vector b

An iteration in SDF semantics is the timespan where each channel contains exactly the same amount of tokens as in the initial state. The production times are contained in a so called *timestamp vector* υ which has as many entities as the number of initial tokens. This means that a timestamp vector υ_0 becomes a vector υ_1 after one iteration, but now containing the new production times. So, the timing of the following iteration can be determined by the timing of the actual iteration.

We define υ_{i,PM_j} as the timestamp vector of iteration i which was executed in power mode PM_j and PM^k as the power mode which is active in iteration k.

Furthermore, there exists a matrix $G_{AS,PM}$ for every application scenario AS and every power mode PM. $G_{AS,PM}$ contains information about the repetition vector of the corresponding SDFG, as well as the actor execution times and the scheduling which defines the execution order. Its entities quantify a minimum distance in time between the tokens indicated in column and row in consecutive iterations. An entity $-\infty$ means that there is no dependency between these tokens. The calculation of matrix $G_{AS,PM}$ is based on symbolic execution of actor firings according to the *minimum repetition vector*, actor execution time, and scheduling policy (for details please refer to [22]). As $G_{AS,PM}$ depends on the power mode, the possible switching overhead $\delta(PM^i, PM^{i+1})$ in time between different power modes PM in iteration i and $i+1$ has to be taken into account.

The timestamp vector υ_{i+1} can be calculated by

$$\upsilon_{i+1} = \begin{cases} G_{AS,PM} * \upsilon_i + \delta(PM^i, PM^{i+1}),\ PM^i \neq PM^{i+1} \\ G_{AS,PM} * \upsilon_i \ \text{,else} \end{cases} \tag{1}$$

Note that the use of iterations instead of single actor firings reflects that data-dependent scenarios are typically correlated to certain execution paths in a dataflow model which naturally limits the state space.

For the ASADF graph in Figure 2 and a given self-timed execution scheduling policy, calculation of matrix G_{y,PM_2} and υ_2 given that υ_1 is $[3,3,2]$ in power mode PM_1 with switching overhead $\delta(PM_1, PM_2) = 1$ results in

$$\upsilon_2 = G_{y,PM_2} * \upsilon_1 = \begin{bmatrix} 2 & -\infty & 4 \\ 2 & -\infty & 4 \\ -\infty & 6 & -\infty \end{bmatrix} \begin{bmatrix} 3 \\ 3 \\ 2 \end{bmatrix} + 1$$

$$= \begin{bmatrix} max\{2+3, -\infty+3, 4+2\} \\ max\{2+3, -\infty+3, 4+2\} \\ max\{-\infty+3, 6+3, -\infty+2\} \end{bmatrix} + 1 = \begin{bmatrix} 7 \\ 7 \\ 10 \end{bmatrix}$$

Considering the availability of the three tokens at $[3,3,2]^T$, their production times after executing application scenario y in power mode PM_2 are $[7,7,10]^T$. This is also illustrated in Figure 3 with a specified throughput constraint of $\frac{1}{4}$.

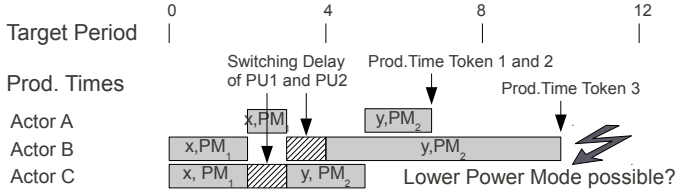

Fig. 3. Token production times calculated by timestamp vectors

A. Multi-Domain Scenario State Space

A DPM strategy can be deduced from the multi-domain scenario state space by chosing a certain platform state for a detected application scenario. The calculation of this state space implies some extensions on the definitions given above.

Definition 3 (Slack Vector): Let υ_i be the timestamp vector of iteration i and γ the throughput constraint. Given that σ_0 is defined as vector $[0,0,0]^T$, the slack vector σ_i of iteration i, $i > 0$, is defined as

$$\sigma_i := \sigma_{i-1} - (\frac{1}{\gamma} - (\upsilon_i - \upsilon_{i-1})) \tag{2}$$

Note that this definition of the slack vector is independent from the actual iteration and is calculated by the normalized timestamp vectors of the actual and the former iteration. This is the key for pruning the state space described in Section IV-B.

To be able to determine a dynamic power management strategy while still meeting (hard) timing requirements we assume in the following that there exists at least one application scenario where the specified throughput requirement is smaller than the actual throughput of the application.

Now we describe the algorithm to build the state space for feasible platform configuration solutions due to a given throughput constraint γ by execution. A state s in the state space S is a 2-tupel with a timestamp vector υ_i and a slack vector σ_i. The transition function Θ uses equation 1 to calculate the next timestamp vector υ_{i+1} and the slack vector σ_{i+1} in the next state using equation 2 for every combination of application scenario sequence and platform scenario sequence, e.g. power modes. Θ is annotated with a state as in the application scenario state machine and a state pm in the power state machine. Starting from an initial configuration s_0 the state space is built by executing transition function Θ on each valid state through breadth-first search order. If consecutive states are gained by one or multiple executions of transition function Θ we will refer to a *path* in the state space. In Section IV-B we present possibilities how to prune this state space.

If $\|\sigma\| > 0$ holds for any state s after applying transition function Θ, s is invalid due to violated performance constraints. In Figure 3, the slack vector in second iteration is

$$\begin{bmatrix} -1 \\ -1 \\ -2 \end{bmatrix} - \left(4 - \left(\begin{bmatrix} 7 \\ 7 \\ 10 \end{bmatrix} - \begin{bmatrix} 3 \\ 3 \\ 2 \end{bmatrix} \right) \right) = \begin{bmatrix} -1 \\ -1 \\ 2 \end{bmatrix}$$

This means that $\|\sigma\| = 2$, so the production time of token 3 violates the throughput constraint $\frac{1}{4}$ in the second iteration.

Obviously, there exists at least one valid path through the state space by always taking the power mode with the lowest execution times. Otherwise, there exists no mapping and scheduling of applications which fulfills the timing requirements at all.

B. Pruning the State Space

To get a closed form of the state space and to manage the complexity, we developed methods to prune the state space:

1) States are invalid due to violated performance constraints if $\|\sigma\| > 0$. Note that mode switching overheads are included in the calculation of the timestamp vectors.
2) Due to focus on DVFS strategy, a path in the fastest operation mode is skipped if there exists at least one sibling path in a lower power mode (assumed to have less power dissipation).
3) Since slack vectors σ_i depend on the relative distance between timestamp vectors υ_{i-1} and υ_i, the given throughput constraint γ, and the slack vector of the previous iteration σ_{i-1}, states s_k and s_l, $k \neq l$, can be joined if $\sigma_k = \sigma_l$. Obviously, $\upsilon_k^{mod\ \gamma} = \upsilon_l^{mod\ \gamma}$ holds which builds a cycle in the state space.

The proof that the state space is finite follows directly from the fact that the number of different slack vectors is limited by the throughput requirement. The rules to prune the state space are illustrated in Figure 4 assuming that a throughput of $\frac{1}{4}$ is required (maximum period is 4).

Figure 4 shows an excerpt of the state space's pruning process for the example graph in Figure 2. For each state in the state space a scalar e indicating the energy consumption of the execution of application scenario as in power mode pm can be calculated using standard equations for static and dynamic power dissipation (e.g. [14]) assuming that relevant properties are known, e.g. supply voltages, frequencies. Following the path with lowest power values minimizes the power dissipation for a detected application scenario sequence.

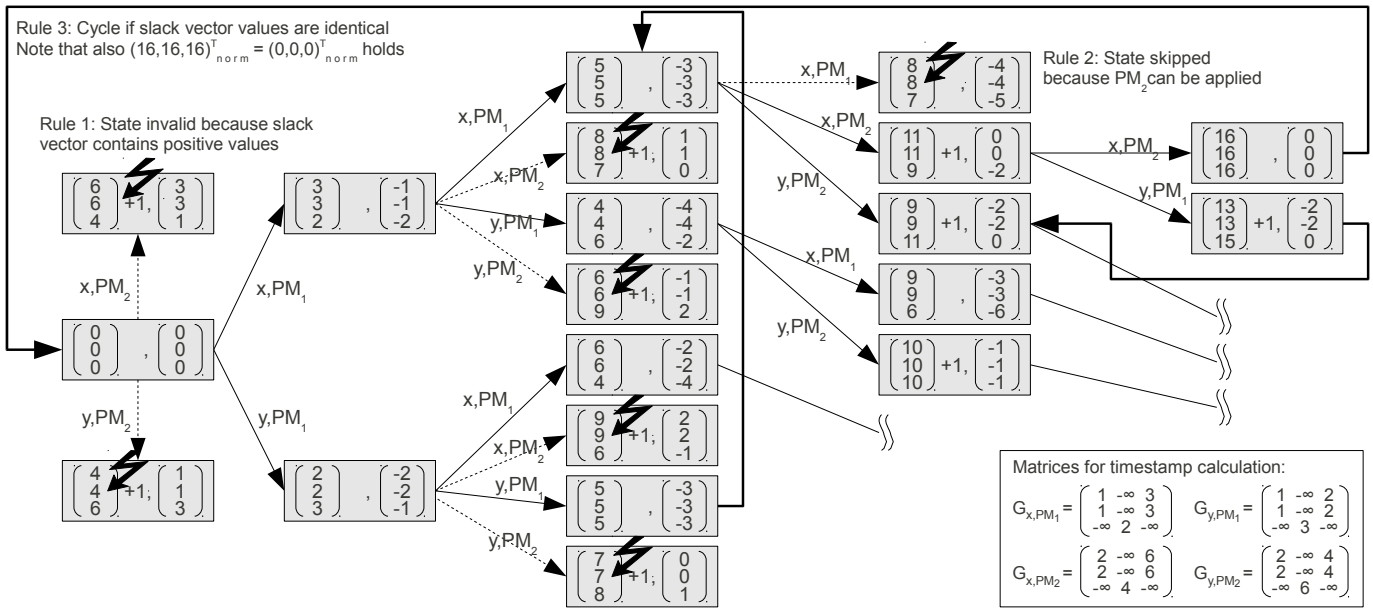

Fig. 4. Building and pruning the state space (required throughput is set to $\frac{1}{4}$)

V. Experiments

As a first experiment we calculated the whole state space of the example graph in Figure 2 which contains 77 states. The execution time on an Intel Core i5 system with 2.66GHz was approx. 2.8ms. We also used our approach to derive a DVFS strategy for a traffic sign detection algorithm (including filter and transformation algorithms) on an ARM-based SoC platform with supply voltages 0.975V-1.35V and appropriate frequencies 125MHz-720MHz. The different application scenarios or timing paths respectively, depend on the number of detected circles in the camera image. Calculated results show savings up to 66.1% in static and dynamic power dissipation.

Obviously, the number of states in the state space highly depends on the platform's DPM capabilities, the variations of the application due to different timing paths, and the performance constraints. Nevertheless, experiments with several algorithms (also synthetic benchmarks with randomly generated scenario graphs) and different performance requirements show that building and pruning the state space generally takes less than a few second. At a first glance, relaxing throughput requirements increases the state space due to more possibilities for switching into lower power modes. But this in turn results in rather similar values for the states' slack vectors which increases the possibility to prune the state space by building a cycle.

References

[1] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous data flow programs for digital signal processing," *IEEE Transactions on Computers*, vol. 36, no. 1, pp. 24–35, 1987.

[2] M. Geilen, "Synchronous dataflow scenarios," *ACM Trans. Embed. Comput. Syst.*, vol. 10, pp. 1–31, 2011.

[3] G. Kahn, "The semantics of a simple language for parallel programming," in *Information Processing '74: Proceedings of the IFIP Congress*. North Holland Publishing Co., 1974, pp. 471–475.

[4] M. Geilen and S. Stuijk, "Worst-case performance analysis of synchronous dataflow scenarios," in *CODES+ISSS '10: Proceedings of the 8th International Conference on Hardware/Software Codesign and System Synthesis*. ACM, 2010.

[5] J. T. Buck, "Scheduling dynamic dataflow graphs with bounded memory using the token flow model," Ph.D. dissertation, 1993.

[6] M. I. Gordon, W. Thies, and S. Amarasinghe, "Exploiting coarse-grained task, data, and pipeline parallelism in stream programs," in *ASPLOS '06: Proceedings of the 12th intern. conference on Architectural support for programming languages and operating systems*. ACM, 2006.

[7] J. T. Buck, "A dynamic dataflow model suitable for efficient mixed hardware and software implementations of dsp applications," in *CODES '94: Proceedings of the 3rd international workshop on Hardware/software co-design*.

[8] S. Stuijk, M. Geilen, and T. Basten, "Exploring trade-offs in buffer requirements and throughput constraints for synchronous dataflow graphs," in *DAC '06: Proceedings of the 43rd annual Design Automation Conference*. New York, NY, USA: ACM, 2006, pp. 899–904.

[9] S. V. Gheorghita, M. Palkovic, J. Hamers, A. Vandecappelle, S. Mamagkakis, T. Basten, L. Eeckhout, H. Corporaal, F. Catthoor, F. Vandeputte, and K. D. Bosschere, "System-scenario-based design of dynamic embedded systems," *ACM Transactions on Design Automation of Electronic Systems*, vol. 14, no. 1, pp. 1–45, 2009.

[10] S. V. Gheorghita, T. Basten, and H. Corporaal, "Intra-task scenario-aware voltage scheduling," in *CASES '05: Proceedings of the 2005 international conference on Compilers, architectures and synthesis for embedded systems*. New York, NY, USA: ACM, 2005, pp. 177–184.

[11] L. Benini, A. Bogliolo, and G. De Micheli, "A survey of design techniques for system-level dynamic power management," *IEEE Transactions on VLSI Systems*, vol. 8, no. 3, pp. 299–316, 2000.

[12] A. Iranli and M. Pedram, "System-level power management: An overview," 2007.

[13] R. Xu, R. Melhem, and D. Mosse, "Energy-aware scheduling for streaming applications on chip multiprocessors." IEEE, 2007.

[14] A. Andrei, P. Eles, Z. Peng, M. T. Schmitz, and B. M. Al Hashimi, "Energy optimization of multiprocessor systems on chip by voltage selection," vol. 15, no. 3, 2007.

[15] P. Rong and M. Pedram, "Determining the optimal timeout values for a power-managed system based on the theory of markovian processes: offline and online algorithms," in *DATE '06: Proceedings of the Conference on Design, Automation and Test in Europe*. ACM, 2006.

[16] S. Irani, R. Gupta, and S. Shukla, "Competitive analysis of dynamic power management strategies for systems with multiple power savings states," in *DATE '02: Proceedings of the Conference on Design, Automation and Test in Europe*. ACM, 2002.

[17] C. Isci, A. Buyuktosunoglu, C.-Y. Cher, P. Bose, and M. Martonosi, "An analysis of efficient multi-core global power management policies: Maximizing performance for a given power budget," in *MICRO '06: Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. ACM, 2006.

[18] A. Alimonda, S. Carta, A. Acquaviva, A. Pisane, and L. Benini, "A feedback-based approach to dvfs in data-flow applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 28, no. 11, 2009.

[19] P. Yang, P. Marchal, C. Wong, S. Himpe, F. Catthoor, P. David, J. Vounckx, and R. Lauwereins, "Managing dynamic concurrent tasks in embedded real-time multimedia systems," in *ISSS '02: Proceedings International Symposium System Synthesis*. ACM Press, 2002.

[20] B. Theelen, M. Geilen, T. Basten, J. Voeten, S. Gheorghita, and S. Stuijk, "A scenario-aware data flow model for combined long-run average and worst-case performance analysis," in *MEMOCODE '06: Proceedings of 4th ACM and IEEE International Conference on Formal Methods and Models for Co-Design*. ACM, 2006.

[21] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Synchronization and Linearity: An Algebra for Discrete Event Systems*. Wiley, 1992.

[22] M. Geilen, "Reduction techniques for synchronous dataflow graphs," in *DAC '09: Proceedings of the 46th Annual Design Automation Conference*. ACM, 2009.