Power Elastic Systems: Discrete Event Control, Concurrency Reduction and Hardware Implementation

Andrey Mokhov, Delong Shang, Danil Sokolov, Fei Xia, Alex Yakovlev, Yu Zhou

Technical Report Series
NCL-EECE-MSD-TR-2009-151

September 2009

Contact: fei.xia@ncl.ac.uk

NCL-EECE-MSD-TR-2009-151

School of Electrical, Electronic & Computer Engineering
Merz Court, Newcastle University
Newcastle upon Tyne, NE1 7RU
UK
http://async.org.uk

Power Elastic Systems: Discrete Event Control, Concurrency Reduction and Hardware Implementation

Andrey Mokhov, Delong Shang, Danil Sokolov, Fei Xia, Alex Yakovlev, Yu Zhou

***Abstract*—The design of microelectronic systems becomes increasingly power-driven, moving towards systems where power and energy appear as dynamic resources. This paper focuses on the aspect of power-adaptive and power-resilient systems, with the property that is collectively called here Power Elasticity, centered on the concept of treating power as a quantifiable system resource. A new approach to power elasticity is proposed based on discrete event control, conveniently represented in a Petri net modeling framework. A simple mechanism aimed at reducing power stressing by using concurrency reduction and an efficient technique for its implementation, the entirely novel idea of 'soft arbitration', are presented. Our approach paves the way for designing systems with fine granularity of power and timing control, thus significantly more robust and better optimized to the operational conditions in a wide variety of applications.**

INTRODUCTION

Microelectronic system design is becoming more energy conscious, because of limited energy supply (scavenged energy or low battery) and excessive heat with associated thermal stress and device wear-out. At the same time, the high density of devices per die and the ability to operate with a high degree of parallelism, coupled with environmental variations, create almost permanent instability in voltage supply (cf. Vdd droop), making systems highly power variant. In the not so long past *low power design* was targeted merely at the reduction of capacitance, Vdd and switching activity, whilst maintaining the required system performance. In many current applications, the design objectives are changing to maximizing the performance within the dynamic power constraints from energy supply and consumption regimes. Such systems can no longer be simply regarded as low power systems, but rather as power-adaptive or power-resilient systems. It is also possible now to imagine designs where systems are optimized to work under both dynamic power and performance constraints.

When systems are subjected to varying environmental conditions, with voltage and thermal fluctuations, timing tends to be the first issue affected. Most systems are still designed with global clocking and the design is often made overly pessimistic to avoid failures due to timing variations. To reduce these margins designers now consciously allow parts of the system to fail, albeit rarely, to maintain the overall balance between the increased performance gains due to margin cuts and reasonably low error rate [1, 2], often combined with dynamic frequency and voltage scaling. Elsewhere designers are moving towards timing elasticity and a wider use of asynchronous design methods. It has been shown that the latter, materialized into the so called elastic voltage scaling, can lead to 30-40% average power savings under the same level of performance [3]. The former technique may be well suited for CPU pipelines but the latter seems to be more universal and appropriate for more heterogeneous systems such as SOCs and even 3D die-stacks. This trend is set to continue in a widening scope of embedded applications and systems with multiple cores and heterogeneity. These methods have so far been developed based on the assumption of a relative rigidity of energy supply levels. Computations tend to be scheduled based on a prior knowledge of the energy requirements. Vdd droops are usually accommodated through reliable operation. However, the notion of elasticity can be taken further than simply stretching delays to accommodate varying conditions.

We would like to investigate elasticity in terms of energy supply and consumption. The ultimate goal is to design systems in such a way that, while maintaining functionality requirements and preserving behavioral equivalence in computation, the computational execution can be altered so as to meet the energy mode requirements. This concept of systems being limited by applicable power and designing systems according to such limitations (called power-elastic design in this paper) is different from conventional low power design [4].

We believe that this problem cannot be solved in its entirety without actually introducing a measure of energy (or power) into the deep levels of the system design abstraction, for example in the form of quantized resources. We also believe that this can be done very elegantly within the computational and behavioral models based on token games, such as Petri nets. Given that there exist powerful methods for the analysis and synthesis of Petri nets, as well as their mapping into logic circuits, the overall prospects of achieving an algorithmic and potentially automated way of obtaining efficient power controls and their hardware implementation are realistic. The overall discipline of designing systems with dynamic power allocation is called here *power elasticity*. In this paper we develop an approach to power elasticity suitable for deriving simple and low-cost hardware for fast response control of energy use. This complements the existing concept of timing elasticity based on dynamic adjustment of computational delays, also at the fine grain level, using asynchronous logic techniques. Together they pave the way for designing systems with fine granularity of power and timing control, and thereby being significantly more robust and better optimized to the operational conditions in a wide variety of (mostly embedded) applications.

Contributions and organization of this paper

The main contributions of this paper are the proposal of the power elastic view of system design and specific power elastic design and implementation techniques, including especially concurrency reduction modeling, analysis and design as well as

soft arbitration. The rest of this paper is organized as follows: In Section 2 we will review existing techniques available for use at the front-line of power management and control, namely the "actuator and sensor" mechanisms a power controller needs for monitoring and manipulating system power behavior. In Section 3 we will describe our automatic power control regime based on power profiling and feedback control concepts. In Section 4 we will explore Petri net techniques and describe initial investigations of power elastic design based on concurrency reduction and soft arbitration techniques. Then discussions and future work vision conclude the paper.

EXISTING POWER CONTROL MECHANISMS

A handful of front-line power saving mechanisms has been used by the semiconductor industry to reduce circuit power consumption [5]. These techniques can be divided by the type of power consumption they address (dynamic power or static power) and by the stage of circuit life during which they are employed (design time or run time), as shown in FIGURE 1.

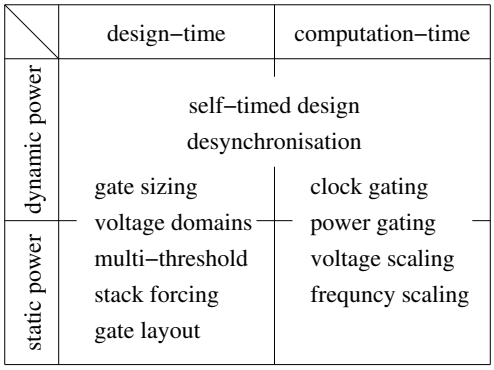

FIGURE 1 POWER CONTROL MECHANISMS

In *clock gating*, registers whose stored values do not change for several computation cycles are isolated from their clock, thus reducing switching activity of the registers and the clock tree which cause up to 30% of dynamic power consumption. There is a trade-off between the granularity of the clock gating and the area overheads introduced by the gating logic.

For reducing both switching and leakage power, a concept of *voltage domains* is often employed. For this a circuit is partitioned into islands with independent power supplies. The islands may consist of several computation blocks or correspond to IP cores. The speeds of the blocks outside the critical path can be individually traded for power savings.

Voltage scaling utilizes the over-conservative margins used to contain process variation and variable operating conditions. The clock period is calculated for the worst-case conditions, however, most of the time the circuit operates in normal conditions and can run faster. Therefore, supply voltage can be safely reduced while the circuit still runs within the given clock period. Voltage scaling is often combined with circuit partitioning into multiple voltage domains to provide more flexibility and granularity in power control.

Leakage current is eliminated completely by the *power gating* approach, where the power supply is disconnected from a computation block if it stays inactive for extended period of time. Certain precautions need to be taken to retain the correct values on the interface of un-powered block for recovery.

Voltage scaling can be naturally integrated with self-timed or asynchronous circuits. These circuits are free of a rigid clock and function at the best possible speed for given operating conditions. There are several approaches to synthesis of self-timed systems [6], however they require significant changes to the conventional design flow. Recently a less intrusive desynchronization technique found its way to commercial products [3]. It introduces elements of self-timed designs into synchronous circuits at the late stage of conventional design flow, thus re-using the time-proved synchronous EDA tools.

Many low-level optimization techniques are applied at the circuit synthesis stage and cannot be controlled later on [5]. For example, gate sizing allows decreasing the number of hazards, which cause up to 20% of dynamic power consumption, by carefully adjusting the arrival time of gate inputs causing the glitch. Leakage current can be effectively reduced with the stack forcing technique where extra transistors are inserted in series to the short transistor stacks. When doing technology mapping, often a library with two implementations of gates are used, standard and low-power. At the lithography stage, the mask data of individual gates can be tweaked to reduce leakage current. These low-level power optimization techniques reduce the speed of the gate, and are therefore usually applied to gates outside the critical path.

All of these techniques can be regarded as providing multiple discrete operation modes for parts of a system with various degrees of power consumption and performance.

POWER ELASTIC BASICS

Here we explore the basic concept of a feedback control strategy with the optimal use of applicable power as its goal. The ultimate aim is to derive and implement at low cost an appropriate power elastic control law for any given system.

Power profiling

Applicable power is the quantity of power that can be applied, determined by two factors. One is the availability of power from energy source(s), especially important in the case of variable and non-deterministic sources such as batteries and scavengers. The second factor is other limitations on power application. For example, under a stable power supply, the operating temperature may restrict the quantity of power applicable. Applicable power can be characterized as the upper bound of power as a function depending on time and space:

$B_p = B_p(x, y, z, t) = B_p(S, t)$

is the upper bound of applicable power at a particular location on chip at a particular time, where x, y, and z are the 3-dimensional indexes of location which can be unified into a general space index S and t is time. The space factor may represent the fact that different parts of a chip may have different temperature characteristics thus different applicable power bounds. In cases of on-chip VLSI, the Newtonian view of the space factor being continuous is not realistic, as existing and future implementable power control mechanisms do not support the infinitesimal fine grain manipulation of power in space. In general, chips are divided into a finite number of areas or blocks based on a finite set of discrete functions and a finite set of discrete engineering implementation techniques. There is always a lower bound for block size for power control beyond which further block division is technically unrealistic or provides no benefit. This lower bound of block size and finite chip size imply an upper bound for the number of blocks.

The space factor can then be simplified to an integer index:

$B_p = B_p(i, t)$

is the upper bound of applicable power for the ith block, where i is the integer index of discrete blocks.

Existing power control mechanisms in general implement coarse grain power manipulation through the use of a limited set of operating modes. Switching among these modes is not usually applied very frequently in time. Switching between different power modes too frequently implies very short stable stays in any mode and risks negating any benefit by the overhead of mode switching. In other words, both the values of t and B_p are also in the discrete domain and thus the operation of power control via switching among multiple power modes is a *discrete event system*. FIGURE 2 illustrates $B_p(i, t)$ as continuous and discrete concepts.

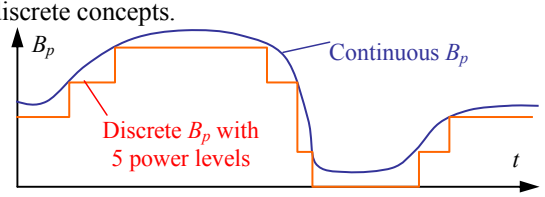

FIGURE 2 CONTINUOUS AND DISCRETE POWER BOUNDS

A description of applicable power in time and space in the form of FIGURE 2 is known here as a *power profile*. Many techniques can be used to obtain power profiles for the purpose of power control system design. These can be static methods including energy source and computation intensity predictions, or dynamic based on sensor data in real time.

Architecture for power elastic circuits

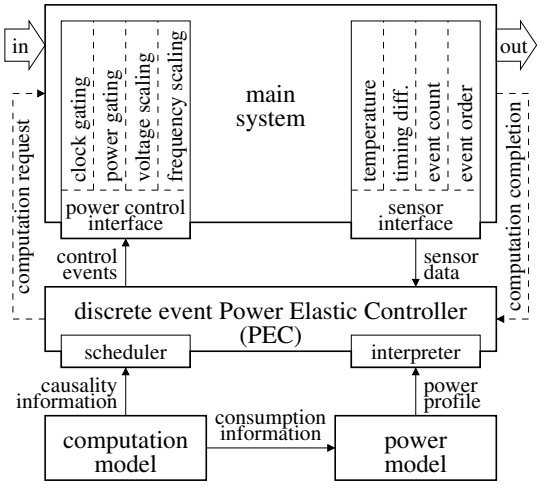

FIGURE 3 POWER ELASTIC ARCHITECTURE

The generic architecture for power-elastic circuits is depicted in FIGURE 3. Its idea is to extend a circuit with a discrete even Power Elastic Controller (PEC) which ensures the power profile is kept within a given boundary by, for example, reducing the concurrency of the circuit whist preserving behavioral equivalence. The PEC decides which computational blocks of the circuit should operate to maintain the required power consumption and regulates the clock gating, power gating and voltage scaling interfaces. This decision is made based on a set of power consumption rules, data from the sensors, such as temperature, delay difference, switching activity, etc. Optionally, causality information can be derived from the computation model and used to schedule the activation of the circuit components in the optimal order.

Power elastic transformation

The PEC is at the centre of the power elastic approach. It is therefore of paramount importance a method of synthesizing such a controller for any given system be developed. The synthesis process of the PEC should take characterization input from the system power and computation models. Here the power model describes system power profiles, and the computation model is a reduced representation of the functional computation behavior of the controlled system/circuit, concentrating on the control path. With these as inputs, the synthesis process carries out a *power elastic transformation* which finds a concrete implementation computation control model. This is then applied through the PEC, resulting in a sequence of execution which satisfies the power profiles without losing equivalence to the functional computation model. This concept is illustrated in FIGURE 4.

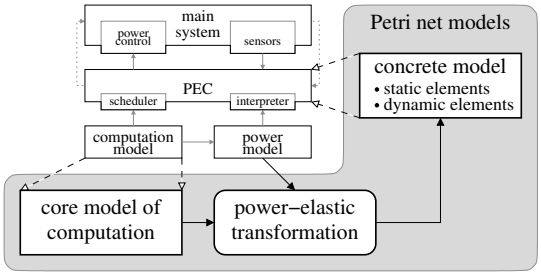

FIGURE 4 PEC SYNTHESIS

A unified method of modeling needs to be developed to make this process of power elastic transformation systematic. Petri nets have been used to represent discrete event systems for the purpose of their analyses and synthesis for a long time [7, 8]. Petri net models can be used to directly represent such issues as causality, concurrency and synchronization. The flow relations in Petri nets can be used to represent the relations between such computation elements as tasks and threads, including their relative concurrency and cross-dependencies determined by the computation and communication relations and information events. The execution semantics [8, 9] readily derivable from a Petri net core computation model can be used in the process of a power elastic transformation which preserves equivalences. In power models, power profiles can for example be represented by quantizing $B_p(i, t)$ into the number of tokens in a *power place*. This concept is demonstrated through stochastic modeling and analyses in [10].

The PEC for a block does not have to be a centralized processor with relatively high power and communication cost. Petri net modeling of discrete event control and asynchronous circuits allow the PEC to be implemented from a collection of small circuits distributed spatially within a block to reduce operational cost and communication bandwidth needs. And generic methods of direct mapping of Petri nets to circuits [6] facilitate this spatial distribution of the PEC.

The process of power elastic transformation can be either static (design time), where a non-variable PEC is synthesized once for a system, or dynamic, where the PEC is tuned during run time, or hybrid, where the PEC synthesis will have both dynamic and static elements. Discussions on these choices are outside the scope of this paper. Here we present relevant and

useful techniques for all these choices.

POWER ELASTIC TECHNIQUES

We have developed power elastic techniques mainly based on concurrency reduction. In this first attempt at investigating power elasticity, we have concentrated on the simplest form of power profile, i.e. Boolean power modes. A block is either on or off; a thread or task is either started or paused; and only such Boolean power control mechanisms as power and clock gating are envisaged. Generalizing these techniques to more complex operation modes is a subject for future exploration.

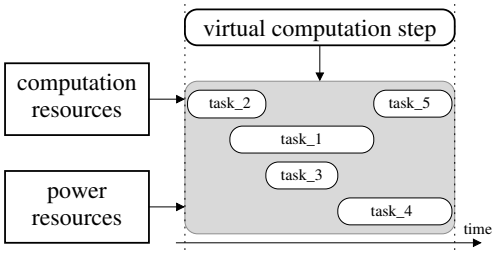

FIGURE 5 CONCURRENCY REDUCTION

The concept of concurrency reduction is illustrated in FIGURE 5. The member tasks within a logically atomic computation step could be executed fully concurrently if resources permit this. However, when a resource like power is in short supply or being regulated to avoid Vdd droop, as in the example in FIGURE 5 where no more than two tasks can be executed simultaneously, the system may choose to execute some of the tasks sequentially, thus trading latency for power.

Concurrency relations and concurrency reduction

As an intuitive concurrency reduction example, FIGURE 6(a) depicts a Petri net core computation model consisting of three concurrent threads (i.e., *a*, *b*, and *c*). Each thread involves the sequential execution of two tasks, e.g., *a.1* and *a.2* for thread *a*. Cross-dependency relations exist between (*a.1, b.2)* and (*b.1, c.2*). Suppose, from the power model, applicable power is quantized into two power units, and the execution of a task requires one power unit. The concrete control model in FIGURE 6(b) illustrates dynamic scheduling based on arbitration, whereby at most two tasks can be scheduled simultaneously, and the scheduling result is only determined during run time.

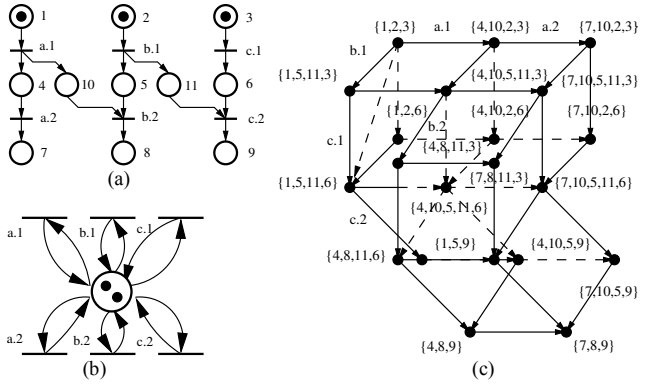

FIGURE 6 CORE PN MODEL (a), ITS RG (c), AND A CONTROL MODEL (b)

The behavior or execution semantics of a Petri net system can be described by its *Reachability Graph* (RG) where $RG=(S, T, F, M_0)$. *S* is the set of all possible markings of the net; *T* is the set of the transitions when considering both *interleaving* and *step* firing semantics; *F* are the transition functions (or next-state functions) of $s'=f(s,t)$ where $s', s \in S$ and $t \in T$; and M_0 is the initial marking of the net. FIGURE 6(c) shows the RG of the example net. In the example, the marking {1,5,11,6} is reachable from the initial marking {1,2,3}, following the interleaving transition sequences of (*b.1,c.1*) or (*c.1,b.1*), or a step transition of {*b.1,c.1*}. Other "step" arcs are not explicitly marked in the diagram to reduce clutter.

For two transitions t_1 and t_2, $t_1 < t_2$ if t_1 precedes t_2 in every transition sequence of the RG. An *N-ary concurrency relation* upon *T* is defined as the set of *N*-tuples, where for each tuple, $\neg(t_1 < t_2)$ holds for every pair of tuple elements t_1 and t_2 ($t_1 \neq t_2$). This example has a *highest concurrency relation arity* of ternary. This highest arity may represent the maximum number of simultaneously active blocks or tasks in a system.

An *N*-ary concurrency relation recursively implies *M*-ary concurrency relations for all $M \leq N$, on all *M*-tuples that are subtuples of an *N*-tuple belonging to the *N*-ary relation. For example, the ternary relation tuple (*a.1,b.1,c.1*) implies three binary concurrency tuples, i.e., (*a.1,b.1*), (*a.1,c.1*), and (*b.1,c.1*). Concurrency relations of the example include 4 ternary tuples and 9 binary tuples, as listed in TABLE 1. The generalization of this property is only true in both directions for a subclass of systems with *distributive concurrency*, as pointed out in [11].

TABLE 1 CONCURRENCY RELATIONS IN THE SYSTEM OF FIGURE 6

Ternary concurrency	{(a.1,b.1,c.1),(a.2,b.1,c.1),(a.2,b.2,c.1), (a.2,b.2,c.2)}
Binary concurrency	{(a.1,b.1),(a.1,c.1),(b.1,c.1),(a.2,b.1),(a.2,c.1), (a.2,b.2), (b.2,c.1),(a.2,c.2),(b.2,c.2)}

Concurrency reduction means the removal of a subset of the *N*-tuples from an *N*-ary concurrency relation. The removal of a tuple will remove all its supertuples. For example, the removal of (*b.1,c.1*) eliminates its parent tuples of (*a.1,b.1,c.1*) and (*a.2,b.1,c.1*) in the concurrency relation list.

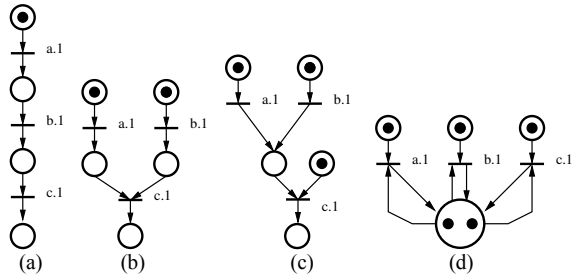

FIGURE 7 CONCRETE CONTROL MODELS TO ELIMINATE *(a.1,b.1,c.1)*: SUPER-LINEAR (a), AND-CAUSALITY (b), OR-CAUSALITY (c) AND ARBITRATING (d)

Both static and dynamic control mechanisms are discussed in this section for concurrency reduction. The difference is that static control applies a single partial order to the tuple elements (i.e., execution of tasks) whereas dynamic control applies multiple orders (which one of the orders takes place is only determined during run time). Static control is further divided into *super-linear* and *and-causal* cases, whereas dynamic control is divided into *or-causal* and *arbitrating* ones. FIGURE 7 lists these control structures by Petri net models in reducing the highest concurrency relation arity of (*a.1,b.1,c.1*) to binary.

Super-linear control imposes a complete order on the tuple elements. With this control, the state cube formed by *a.1*, *b.1*,

and *c.1* in FIGURE 6(c) is replaced by the local *RG* in FIGURE 8(a). As a result, all the local binary concurrency tuples incurred by (*a.1,b.1,c.1*) are eliminated.

And-causal control expresses an AND enabling condition for a task's execution. With FIGURE 7(b), *c.1* is enabled when both *a.1* and *b.1* have fired. The partial order in this example is {(*a.1,c.1*),(*b.1,c.1*)}, and the corresponding local *RG* is shown in FIGURE 8(b). With and-causality, only one local binary concurrency tuple is maintained, i.e., (*a.1,b.1*).

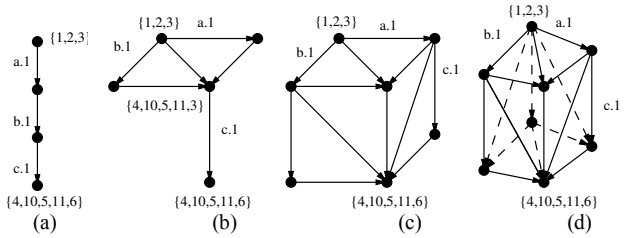

FIGURE 8 LOCAL RGS AFTER CONCURRENCY REDUCTION: SUPER-LINEAR CONTROL (a), AND-CAUSALITY (b), OR-CAUSALITY (c), AND ARBITRATION (d)

Or-causal control [12] expresses an OR enabling condition for a task's execution. With FIGURE 7(c), *c.1* is enabled when either *a.1* or *b.1* has fired. The speciality of or-causality is that it imposes two (mutually exclusive) partial orders: {(*a.1,c.1*)} and {(*b.1,c.1*)}. It is only known at run time which order takes place. The local *RG* with or-causal control is shown in FIGURE 8(c), where all local binary concurrency tuples are maintained.

Finally, with the *arbitrating* control of FIGURE 7(d) (a 2-of-3 arbitration), all three tasks are enabled, but at most two of them can be executed simultaneously. FIGURE 8(d) shows the local *RG*, by which it is evident that all the three binary tuples are maintained. In addition, all three "step" arcs corresponding to the tuples are enabled at the initial state, whereas only one "step" arc is allowed in the or-causal control.

Dynamic control based on arbitration preserves system concurrency the best, followed by or-causal control, whereas static control has a high reduction of concurrency.

Concurrency reduction causes *performance degradation*. This is because of two factors: the stretched execution time as a direct consequence of concurrency reduction, and the extra delays incurred by concurrency control (i.e. PEC). The former factor can be determined from the Petri net control structures, whereas the latter is related to the controller implementation.

Suppose the execution delays of tasks *a.1*, *b.1*, and *c.1* are t_{a1}, t_{b1} and t_{c1}, respectively (assuming positive and bounded delays). Further suppose that the delays required by implementing the super-linear, and-causal, or-causal, and arbitrating controllers are d_{sl}, d_{ac}, d_{oc}, and d_{ab}, respectively.

Before concurrency reduction, the triple tasks in the example can be executed within a period of $max(t_{a1},t_{b1},t_{c1})$. With super-linear control, the execution period is $t_{a1}+t_{b1}+t_{c1}+d_{sl}$. With and-causal control, the execution period is $max(t_{a1},t_{b1})+t_{c1}+d_{ac}$. With or-causal control, the execution time is $max(min(t_{a1},t_{b1})+t_{c1},max(t_{a1},t_{b1}))+d_{oc}$, which can be further refined to $max(t_{a1}+t_{c1},t_{b1})+d_{oc}$ should the run-time order is {(*a.1,c.1*)}, or otherwise to $max(t_{b1}+t_{c1},t_{a1})+d_{oc}$.

Similar to or-causal control, arbitration-based control has an execution period dependent on run time token-game results. If the imposed partial order turns out to be {(*a.1,b.1*)} (or {(*b.1,a.1*)}) during run time, the execution time is $max(t_{a1}+t_{b1},t_{c1})+d_{ab}$. Other cases can be similarly derived.

Not considering controller delays, static controls degrade performance more than dynamic controls. Arbitration-based control has a superset of execution paths of the or-causal control and can render even lower performance degradation.

The control mechanisms are compared in Table 2, in an *N*-ary to *M*-ary reduction. Here the control structures, the orders they impose on the tuple elements, and the effects on concurrency relations and performance degradation (without controller delays) are described.

In the context of FIGURE 3 and FIGURE 4, the highest concurrency arity *M* would be derived from the power model and direct power place modeling exists in the arbitration case. Petri net techniques are used to derive concrete models for the PEC in the form of FIGURE 7 from core computation models in the form of FIGURE 6(a). More details, including discussions on the distribution of concurrency reducing PEC algorithms among small circuits across a block, can be found in [13].

Soft arbiters

Arbitration-based concurrency reduction has been shown to be simple to design and potentially efficient in operation in the example above. However, with current and future on-chip and 3D systems likely to have high degrees of concurrency and complex relations among concurrent blocks and threads, large *M* and *N* numbers are likely prevalent in real systems. This could make PEC's using other (such as static) concurrency reduction techniques more attractive because of comparatively lower implementation and operational costs, unless efficient multi-client, multi-resource arbiters can be found. We demonstrated a distributed arbiter architecture for large (10×10) implementations with good scalability in [14], but issues like performance and cost persist.

Fortunately, unlike hard enumerable resources such as software threads and hardware blocks, power resource is different in that it allows a degree of *softness* in arbitration. Suppose *N* clients need a system resource and no more than *M* of them can access it simultaneously, because e.g.

1. the system does not have enough *hard resources* to serve more than *M* clients, e.g. it has only *M* processing units;
2. we want to reduce concurrency for power management issues (exceeding *M* may lead to inefficient energy consumption or overheating).

TABLE 2 COMPARISON OF DIFFERENT CONTROL MECHANISMS FOR CONCURRENCY REDUCTION

Control Scheme	Structure	Partial Order Imposed	Concurrency Reduction Effects	Performance degradation
super-linear	*M*! arrangements	single total order	all relations removed	largest
and-causal	And enabling conditions between $\left\lceil \frac{N}{M} \right\rceil$ groups of *M*-ary subtuples	single partial order between Groups	up to *M*-ary relations maintained but restricted to within a group	second largest
or-causal	Or conditions on a *M*-ary subtuple to enable the next new tuple element	multiple	all *M*-ary relations maintained	dynamic, smaller than static controls
arbitrating	*M*-of-*N* arbitration	multiple	all *M*-ary relations maintained	dynamic, more flexible than or-causal control

In case 1 the bound has to be strict. Simultaneously granting M+1 clients inevitably leads to at least one clash of two clients at the same hard resource, leading to logical errors and system failure. However, case 2 is more flexible by nature. Random, occasional events have little effect on the inertial, statistical characteristics of power consumption. Therefore, in this case the bound may be relaxed and granting access to more or less than M clients may be allowed as long as the rate of such imprecise granting is acceptable. We call arbiters with relaxed bounds on the number of issued grants *soft arbiters*.

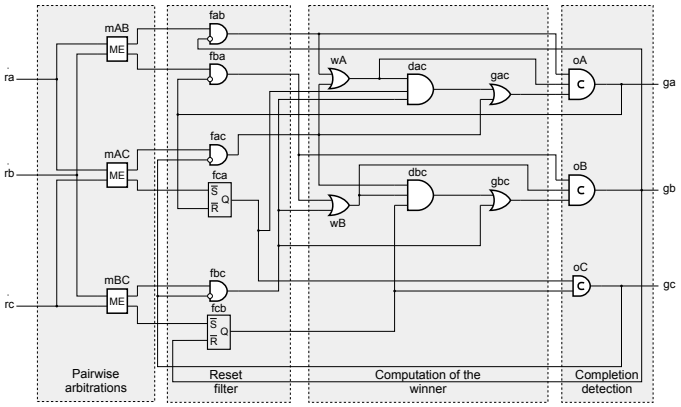

FIGURE 9 STRICT 1-OF-3 ARBITER

Implementations of soft arbiters can in general be smaller and faster than strict ones. For example, the implementation of a 1-of-3 strict arbiter presented in [15] is given in FIGURE 9. It has a construction with four layers: pair-wise arbitration, reset filters, computation of the winner, and finally the layer of completion detection. But if it is allowed to issue two grants occasionally (instead of at most one) then its implementation can be simplified dramatically as shown in FIGURE 10. Details of this simplification process are discussed in [16].

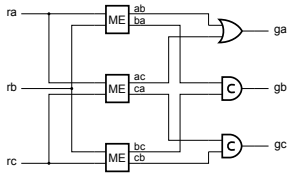

FIGURE 10 SOFT 1-OF-3 ARBITER

Let us study the behavior of the simplified arbiter. Table 3 shows the grants issued for given orders of requests. The arbiter always gives the grant to the first request, plus to request *ra* if it came second. Thus, for a single burst of incoming requests, the probability for the arbiter to issue two grants is 1/3. On the other hand, if there is a constant flow of incoming requests abcabcabc... then the arbiter will be giving one and two grants in an alternating fashion, effectively behaving as an 1.5-of-3 soft arbiter (the series of grants will be *ga*-*gb*-{*ga*, *gc*}-*gb*-{*ga*, *gc*}-... etc).

Soft arbitration provides an opportunity to build small and fast arbiters for power-related concurrency management. Modeling issues and the implementation of soft arbiters with real-time control over the degree of softness are discussed in detail in [16]. Also shown in [16], it is possible to implement real-time adaptive thresholding for variable M values in soft arbiters to support dynamically variable power profiles.

TABLE 3 ANALYSIS OF SOFT ARBITER WITH RESPECT TO REQUEST ORDERS

Request order	*ab/ba*	*ac/ca*	*bc/cb*	Issued grant(s)
abc	*ab*	*ac*	*bc*	*ga*
acb	*ab*	*ac*	*cb*	*ga*
bac	*ba*	*ac*	*bc*	*ga, gb*
bca	*ba*	*ca*	*bc*	*gb*
cab	*ab*	*ca*	*cb*	*ga, gc*
cba	*ba*	*ca*	*cb*	*gc*

DISCUSSIONS AND FUTURE WORK

In this paper we presented the concept and general method of power elastic design and initial explorations on concurrency reduction, a method of power elastic control, and soft arbiters, a promising technique for concurrency reduction.

This article is aimed to serve as a position paper to lead the way for a series of systematic developments for Power Elastic Control, including modeling (deterministic and stochastic), adaptive discrete event control/policy algorithms, interfaces with the sensors and actuators, architectures, and hardware-software implementations. These provide rich future research opportunities especially towards finding systematic methods, for which our chosen formalism, Petri nets, has great potential. Initial steps in this research have already been taken [10, 13, 14, 15, 16]. A large portion of this research is in the domain of developing power elastic electronics for energy harvesting environments.

The concepts of concurrency regulation and soft arbitration have significance outside the domain of power control. For instance, network bandwidth is a resource with similar inertial, statistical properties to power, and thus suitable to be managed through concurrency regulation with soft arbitration [17].

REFERENCES

[1] S. Das, et al., "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance", IEEE J. Solid-State Circuits, pp.32-48, Jan. 2009.

[2] K. Bowman, et al., "Circuit Techniques for Dynamic Variation Tolerance", DAC'09, July 2009.

[3] E. Tuncer, J. Cortadella, L. Lavagno, "Enabling adaptability through elastic clocks", DAC'09, pp. 8-10, July 2009.

[4] J. Liu et. al., "Power-aware scheduling under timing constraints for mission-critical embedded systems," DAC 2001.

[5] S. Henzler, *Power Management of Digital Circuits in Deep Sub-Micron CMOS Technologies*, Springer-Verlag, 2006.

[6] CDT paper on synthesis (title withheld for blind review)

[7] A. Benveniste et al., "Diagnosis of Asynchronous Discrete Event Systems, A Net Unfolding Approach", IEEE Trans. Auto. Control, vol. 48, pp. 714-727, 2001.

[8] Ph. Darondeau, et al., "Synthesis of Nets with Step Firing Policies", Fundamenta Informaticae, vol. 94, 2009.

[9] E. Best and M. Koutny, "Petri Net Semantics of Priority Systems", TCS-96(1), pp. 175-216, 1992.

[10] Stochastic analyses technical report (title withheld for blind review).

[11] Concurrency semantics (title withheld for blind review).

[12] Or causality (title withheld for blind review).

[13] Concurrency reduction report (title withheld for blind review).

[14] NxM arbiters (title withheld for blind review).

[15] Multi-way arbitration (title withheld for blind review).

[16] Soft arbitration technical report (title withheld for blind review).

[17] S. Yang, S. Furber, Y. Shi, L. A. Plana, "An admission control system for QoS provision on a best-effort GALS interconnect", ACSD'08, 2008.