Advanced power management techniques: going beyond intelligent shutdown

Luca Benini
DEIS Universitá di Bologna
lbenini@deis.unibo.it

Abstract— Well into the System-on-Chip era, power consumption has emerged as one of the most critical challenges to design complexity scaling. Moving from a critical assessment of current technologies and architectures, we survey the distinguishing features of a design methodology that aims at energy consumption reduction, under guaranteed *quality of service* **(QoS), as a main objective in system design.**

I. Introduction

Systems on chips (SoCs) containing tens of millions of transistors are currently designed and produced. Billion-transistor chips are not a distant dream in the future. Several challenges arise from the complexity of designing multimillion- and billion-transistor chips. Design methods (and tools) must follow the rapid pace of evolution dictated by technology and application pull. As of today, a few fundamental trends of evolution have emerged.

- Technology is increasingly hard to master, both from a cost and a reliabili perspective. Hence, application-specific hardware platforms are becoming not viable from an economic viewpoint. Large-scale SoCs must be flexible and programmable after fabrication (to sell in large volumes), and they must contain a large amount of functional redundancy (to tolerate technology-induced faults and bring yield to acceptable levels).

- Systems on chips are designed using pre-existing components. Reuse becomes key for designing chips fast enough, because no design team can successfully complete a multi-million transistor design "from scratch" in the six to nine month window dictated by market pressure. Design methodologies must support component re-use in a plug-and-play fashion.

- While the traditional computing equipment market has reached a saturation region where margins are extremely low, great promise is shown by the emergence of "ambient intelligence" devices. These devices will materialize the vision of ubiquitous interactive access to world-wide information and communication resources from tiny wireless terminals, coupled extensive data collection and integration from a myriad of geographically dispersed sensors.

These trends have jointly contributed to the creation of a unprecedented *power crisis*: on one side technology and complexity control push for highly-reusable, standardized and programmable architectures, which are notoriously much less power efficient than dedicated hardware [11]. On the other hand, the largest market potential is for chips that can be easily embedded in tiny, battery powered (even battery-less) devices, while at the same time performing computationally intensive tasks.

Low power design has thus become a critical need for current and future SoCs. In this paper we will first review the trends of evolution of current technologies and architectures, with the purpose of identifying the most critical bottlenecks in system power consumption. Then, we will focus on a few leading-edge low power design techniques that can provide immediate relief to the power crisis.

II. Technology trends

It is a well-known fact that rapidly evolving silicon technology creates smaller and faster CMOS transistors, with decreasing switching power. At a fist sight, then, technology works for reducing power consumption. However, die area is not shrinking with technology (in other words, chips pack a quadratically increasing number of transistors). This would not be a problem if the power dissipation of basic CMOS gates decreased fast enough to compensate the increase in integration density. Unfortunately, this is not the case, as detailed next.

First, because of bounded propagation speed of electromagnetic waves, as opposed to the ever faster switching speed of transistors, silicon technology is becoming increasingly interconnect-dominated. Delays on wires are already dominating: global wires spanning a significant fraction of the chip have propagation delays exceeding the clock period. The same holds for switching power: the gate capacitance of a minimum-size transistor is decreasing with technology, but interconnect capacitance per unit length is not decreasing at the same speed (because of sidewall contributions) and interconnect length is not decreasing for global wires. Hence, the C factor in the well known switching power equation $P_{sw} = kCV_{DD}^2 f$ (where k represents switching activity) does not scale down as fast as minimum feature size [4, 13].

Second, in order do reap performance benefits (in other words, to satisfy quality of service requirements), chip clock

frequency is scaled faster than technology [5]. This result is achieved by clever architectural optimizations that reduce the number of logic stages to be traversed within the clock cycle time. From a power viewpoint this is clearly a problem, because power is directly proportional to switching frequency f. Performance constraints (or objectives) are also the main reason why supply voltage scaling is not drastic as one would desire for power minimization purposes. In fact, transistor switching speed decreases as $(V_{DD} - V_T)^{\alpha}$, with $1 < \alpha < 2$. This cannot be tolerated if performance is tightly constrained.

Third, deep submicron transistors are increasingly leaky in the OFF state. Source-to-drain current due to subthreshold conduction is the dominant cause of leakage, but drain-to-gate currents due to electron tunnelling across the gate oxide is also becoming significant. Furthermore, random variations of the number of dopant atoms in the channel region cause poor threshold control, and. many transistors have a threshold significantly lower than nominal. Unfortunately, subthreshold conduction is exponentially dependent on threshold voltage and transistors with lower threshold leak exponentially more than nominal transistors. As a result, chip standby power is becoming a significant concern. Clearly, subthreshold leakage also a heavy limiter to threshold voltage V_T reduction, with obvious negative impact on supply voltage scaling [5].

From this brief overview, it is apparent that power is critical in many ways. Power density grows significantly with technology, thereby creating huge problems in power supply distribution. Total power consumption in active state increases as well, with obvious negative consequences in battery lifetime as well as thermal design. Finally, even stand-by power is growing fast with technology, impacting battery lifetime in portable applications. For these reasons, technologists are pointing at power as the one of the most likely show-stoppers to technology scaling if adequate countermeasures are not taken.

III. Architectural trends

As seen in the previous section, technology evolution is not going to solve the power consumption problem. On the contrary, many technologists refer to design innovation at the circuit, logic, architectural level as a way out from the crisis. Unfortunately, trends in this area are not favorable to energy efficiency at all. Even though a significant research effort is being devoted to power minimization, mainstream architectural design is moving toward energy-hungry architectures.

Most systems-on-chip are nowadays designed with a high degree of programmability. The majority contains one or more core processors, and many instantiate several programmable coprocessors (e.g., VLIW units for numerical computations, programmable IO processors, etc.); a few recent designs even embed a significant amount of bit-programmable logic (FPGA fabrics) [15]. Programmability is a common requirement when designing large-scale SoCs for two main reasons. First, it ensures functional flexibility, which widens the spectrum of applicability and the potential production volume. Second, it leaves margins for post-fabrication bug fixing and tuning, thereby enhancing yield. High yield and volume of sales are required to amortize ballooning mask development and fabrication costs. Second, processor-based architectures emphasize reuse of hardware modules (e.g., the cores themselves, the memories, etc.) as well as software components (e.g. libraries, operating systems, compilers). Reuse-centric methodologies are instrumental to reduce development risk and cost, and minimize time-to-market.

Yet, flexibility and reuse come at a price. The power-performance ratio (i.e., the energy) required by a processor to carry out a given task (e.g., MPEG decoding) is several orders of magnitude (3 or more) higher than what could be achieved with an application-specific architecture [14, 11]. The causes of the energy inefficiency of processor-based architectures are well understood. Processors have to fetch and decode instructions to control execution. This is a huge overhead with respect to an application-specific architecture with a hardwired controller.

Instruction execution relies on heavily shared memory structures (register files), while dedicated architectures can exploit distributed registers with faster read/write time and enormously lower power consumption per access. The sequential instruction execution flow inevitably limits the parallelism that can be extracted from an application, and therefore emphasizes storage requirements. In general processor-based architectures are memory-hungry and require complex memory hierarchies to achieve satisfactory performance thanks to the exploitation of the spatial and temporal locality principles. As a result, a dominant fraction of the power consumed by a processor is spent in accessing the memory hierarchy (i.e., in communicating with memory structures and in extracting/storing information in them). Storage-related costs are minimized in application-specific architectures thanks to the exploitation of distributed and custom-sized memories and the minimization of spurious memory accesses created as an artefact of sequential instruction execution.

Advanced processors, often required to attain performance goals, are even more power-hungry than simple processors, because they rely on various forms of speculative execution to increase the average number of instructions executed in a clock cycle. Well-known performance-enhancement techniques, such as speculating past branches, value prediction, prefetching, imply the execution of redundant instructions which can be committed only after they completed. In case of a wrong prediction, a double overhead is often incurred. Not only computation has been executed needlessly, but also its effects must be rolled back.

Fine-grained programmable fabrics can be one or more orders of magnitude more energy-efficient than processors [14] for some classes of computations, but they still incur a very significant overhead with respect to dedicated logic. Recent data shows that computation performed by an embedded FPGA fabric in a hybrid FPGA-ASIC chip is more than two orders of magnitude more power consuming and more than ten times slower than the same computation in dedicated logic [15]. The overhead in this case is mainly due to communication between

fine-grain programmable logic elements, which is performed on massively redundant programmable wiring resembling a multi-stage network, as opposed to dedicated, instance specific wires. Additionally, programmable logic blocks are often much more complex than the basic gates they mimic when programmed.

In summary, it quite clear that the power crisis is caused by the conjunction of two factors: technology does not help us as much as we need, and architectural evolution trends are working against energy efficiency. To escape this losing scenario, a multi-pronged approach is needed, which improves energy efficiency without compromising flexibility, while minimizing performance overhead. This hard to do, but it can be done, as we shall see in the following sections.

IV. Designing low-power hardware

In a nutshell, we believe that the key for addressing the power consumption challenge lies in two simple ideas: (i) exploit flexibility not only at the functional level, but also in the power dimension (i.e., design *power-manageable architectures*), (ii) enable application-dependent specialization without compromising reuse and design flow streamlining. The first idea implies adding hardware resources that do not have a computational task, but they dedicated to controlling the power level of functional units, detecting and exploding idleness, and locally trading off performance with power. The second idea requires tuning a hardware platform to a specific computational task, without significantly increasing design time and effort (i.e., with a push-button process that does not require re-design of hardware blocks).

The practical embodiments of the two above principles often require both hardware and software support. In this section we analyze hardware implications. In particular, we focus on power manageable hardware, because design practices in this area are well established and widely spread. On the contrary, application-dependent specializations for power efficiency are performed much more in a ad-hoc fashion and they have more limited applicability, therefore they are not covered here. The interested reader can refer to a few recently published books and survey papers for additional information [2, 10, 11].

A. Power-manageable architectures

Power manageable architectures ultimately aim at eliminating idle power consumption (i.e., the power consumed by a hardware component when it is not in use) and run-time slack in active state, through the run-time control of clock activity and frequency, supply voltage and device threshold. The benefit of idle power elimination is obvious, while run-time slack elimination translates to cubic power savings (and quadratic energy savings) if the circuit is slowed down while at the same time reducing the power supply. Even though slow-down coupled with dynamic voltage scaling is more effective than idle power reduction [3], the two techniques are not mutually exclusive. Clearly, slowing down a circuit reduces its idleness, but in many cases it cannot eliminate it: consider for example a MP3 player. When the device is active it operates under tight performance constraints (namely, real-time playback), and obviously it is not possible to stretch execution time beyond the duration of a music track, even if after it has been played out, the player remains idle for hours.

Leakage is a major concern in idle-power reduction, because it impacts battery lifetime even if the circuit is completely idle. Quiescent power specifications tend to be very tight. In fact, CMOS technology has traditionally been extremely power-efficient when transistors are not switching, and system designers expect low leakage from CMOS chips. To meet leakage power constraints, *multiple-threshold* and *variable threshold* circuits have been proposed [9]. In multiple-threshold CMOS, the process provides two different thresholds. Low-threshold transistors are fast and leaky, and they are employed on speed-critical sub-circuits. High-threshold transistors are slower but exhibit low sub-threshold leakage, and they are employed in non-critical units/paths of the chip.

Unfortunately, multiple-threshold techniques tend to loose effectiveness as more transistors become timing-critical. Variable-threshold circuits overcome this shortcoming by dynamically controlling the threshold voltage of transistors through substrate biasing. When a variable-threshold circuit becomes quiescent, the substrate of NMOS transistors is negatively biased, and their threshold increases because of the well known *body-bias effect*. A similar approach can be taken for PMOS transistors (which require positive body bias). Variable-threshold circuits can in principle solve the quiescent leakage problem, but they require standby control circuits that modulate substrate voltage. Needless to say, accurate and fast body-bias control is quite challenging, and requires carefully designed closed-loop control [9].

Idle power is not only caused by leakage, but it is also due to unneeded switching activity. Clock switching within idle functional units is the best known example of this problem. Clock gating is used to eliminate unneeded clock activity. Most low-power processors implement both hardware and software controlled clock gating through dedicated power-down instructions. A radical way to eliminate idle power (both leakage and dynamic) is to disconnect a unit from its power supply. Unfortunately, in this case state information is lost.

In active state, supply voltage can be controlled to reduce power, albeit in the limited ranges allowed in aggressively scaled technologies. *Multiple-voltage* and *variable voltage* techniques have been developed to this purpose [11]. In multiple-voltage circuits two or more power supply voltages are distributed on chip. Similarly to the multiple-threshold scheme, timing-critical transistors can be powered at a high voltage, while most transistors are connected to the low voltage supply. Multiple voltages are also frequently used to provide standard voltage levels (e.g., 3.3 V) to input-output circuits, while powering on-chip internal logic at a much lower voltage to save power. The main challenges in the multiple-voltage approach are in the design of multiple power distribution grids and of power-efficient level-shifters to interface low-voltages

with high-voltage sections.

Variable-voltage techniques offer the possibility of modulating the power supply dynamically during system operation. In principle, this is a very powerful technique, because it gives the possibility to trade off power for speed at run time, and to finely tune performance and power to non-stationary workloads. In practice, the implementation of this technique requires considerable design ingenuity. First, voltage changes require non-negligible time, because of the large time constants of power supply circuits. Second, the clock speed must be varied consistently with the varying speed of the core logic, when supply voltage is changed.

Power manageable hardware can be designed at different levels of granularity. While early embodiments allowed only coarse-grained control (e.g., peripheral vs. core logic), the trend is toward fine-grained support to power management. An example of a state-of-the-art power management support is the "Voltage islands" technology introduced by IBM [6], where it is possible to instantiate multiple regions on a chip each powered by its own variable voltage supply, clocked by a dedicated variable frequency clock generation unit, and equipped with independent threshold voltage control circuitry.

It is important to remember that power manageable hardware is only one side of the equation. Transitions between power states (e.g. from idle to active, from fast to slow) have significant cost in performance and energy. Hence, virtually all power management schemes require significant software support both at design (and compilation) time and at execution time. The main challenges are to avoid power state transitions that cannot be amortized and to select the most suitable power state for a given operating condition.

V. Power management software

We generalize the notion of operating system (OS) to the middleware that provides support for the operation of SoCs. Note that system support middleware in current SoCs is usually quite simple, designed for a specific integrated core processor, under the assumption that a processor provides global, centralized control for the system. In the context of future SoCs, the prevailing paradigm will be peer to peer interaction among several, possibly heterogeneous processing elements. Thus, we think that system software will be designed as a modular distributed system. Each programmable component will be provided with system software to support its own operation and to manage its communication to other modules.

The system software creates an abstraction of the underlying hardware platform. In a nutshell, we can view the system as a network of components. Each component models a computational or storage unit. We observe that:

- Each component can operate at various service levels, providing corresponding performance and energy consumption levels. This abstracts the physical implementation of components with adjustable voltage and/or frequency levels, as well as with the ability to disable their functions in full or in part.
- Even if units can be power managed independently, power minimization for a complex system requires information exchange between its components. Locally optimal choices do not generally lead to a globally optimal solution.

Dynamic power management entails selecting the appropriate component state to service a workload with the minimum energy consumption. DPM *policies* are the control algorithms for state transitions [1]. Transitions among states have a finite delay and energy penalty. Thus, the definition of policies that maximize performance under energy constraints, is a non-trivial problem [1]. Whereas policies can be implemented in hardware (as a part of the control-unit of a component), much more flexibility and ease of integrations is achieved by software implementations. Thus a policy can be seen as a program that is executed at run-time by the system software. The fundamental premise for the applicability of DPM is that systems (and their components) experience non-uniform workloads during operation time. Such an assumption is valid for most systems, both when considered in isolation and when inter-networked. A second assumption of DPM is that it is possible to predict, with a certain degree of confidence, the fluctuations of workload. Workload observation and prediction should consume little energy.

Power management policies can be implemented in system kernel and be tightly coupled to process management. Indeed, process management has knowledge of currently-executing tasks and tasks coming up for execution. Process managers know also which components (devices) are needed by each task. Thus, policy implementation at this level of system software enjoys both a global view and an outlook of the system operation in the near future. Predictive wake-up of components is possible with the knowledge of upcoming tasks and required components. The system software can be designed to improve effectiveness of power management. Indeed, power management exploits idle times of components. The scheduler can sequence tasks for execution with the additional goal of clustering component operation, thus achieving fewer but longer idle periods. The inter-relation between power manager and scheduler is even stronger for variable-voltage units. In this area numerous variable-voltage scheduling algorithms have been proposed [11], which aim at running processing elements as slow as possible (to allow supply voltage minimization), without violating performance constraints. Experiments with implementing DPM policies at different levels of system software [8] have shown increasing energy saving as the policies have deeper interaction with the system software functions.

A. Application-level power management

A reasonable questions is why not letting the application programs control the service levels and energy cost of the underlying hardware components. There are typically two objections to such an approach. First, application software should be independent of the hardware platform for portability rea-

sons. Second, system software supports generally multiple tasks. When a task controls the hardware, unfair resource utilization and deadlocks may become serious problems.

For these reasons, it has been suggested [7] that application programs contain system calls that request the system software to control a hardware component, e.g., by turning it on or shutting it down, or by requesting a specific frequency and/or voltage setting. The request can be accepted or denied by the operating system, that has access to the task schedule information and to the operating levels of the components. The advantage of this approach is that OS-based power management is enhanced by receiving detailed service request information from applications, and thus is in a position to take better decisions.

Another approach is to let the compiler extract directly the power management requests from the application programs at compile time. This is performed by an analysis of the code, followed by a code modification phase that inserts power management requests in critical points of the computation. For instance, in a variable-voltage unit, voltage and speed change requests can be issued immediately after data-dependent control-flow branching, when the behavior of the program is such that additional slack is created with respect to the worst case execution time [12].

VI. Conclusion

Moving from the analysis of technology and architectural trends in gigascale silicon integration, we observed that power consumption has become one of the most menacing showstoppers. Thus, leading edge low power design techniques are more critical than ever. Power management has emerged as the most promising approach to tackle the power crisis, because it makes it possible to finely control the power-performance tradeoff at run time. The paper surveyed hardware and software technologies for power management support, focusing more on fundamental ideas than on specific embodiments. Even though many solutions exist, much work still needs to be done especially in light of the evolution of SoC architectures toward communication-dominated networks-on-chip.

References

[1] L. Benini, A. Bogliolo, G. De Micheli, “A Survey of Design Techniques for System-Level Dynamic Power Management,” *IEEE Transactions on Very Large-Scale Integration Systems*, vol. 8, no. 3, pp. 299-316, June 2000.

[2] L. Benini, G. De Micheli, “System-Level Power Optimization: Techniques and Tools,” *ACM Transactions on Design Automation of Electronic Systems*, vol. 5, no. 2, pp. 115-192, April 2000.

[3] T. Burd, et al., ”A dynamic voltage scaled microprocessor system,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1571-1580, Nov. 2000.

[4] R. Ho, K. Mai, M. Horowitz, “The Future of wires,” *Proceedings of the IEEE*, January 2001.

[5] T. Karnik, S. Borkar, V. De, ”Sub-90nm technologies – Challenges and opportunities for CAD,” *IEEE/ACM International Conference on Computer-Aided Design* pp. 203-206, Nov. 2002.

[6] D. Lackey, et al., ”Managing power and performance for Systems-on-chip designs using voltage islands,” *IEEE/ACM International Conference on Computer-Aided Design* pp. 195-202, Nov. 2002.

[7] Y.-H. Lu, T. Simunic, G. De Micheli, ”Software controlled power management,” *IEEE/ACM International Workshop on Hardware-Software Codesign*, pp. 157-161, Sept. 1999.

[8] Y.-H. Lu, G. De Micheli, ”Comparing system level power management policies,” *IEEE Design & Test of Computers*, vol. 18, no. 2, pp. 10-19, March-April 2001.

[9] S. Martin, K. Flautner, T. Mudge, D. Blaauw, ”Combined dynamic Voltage scaling and Adaptive body Biasing for lower power microprocessors under dynamic workloads,” *IEEE/ACM International Conference on Computer-Aided Design* pp. 721-727, Nov. 2002.

[10] P. R. Panda, F. Catthor, N. D. Dutt, K. Danckaert, E. Brockmeyer, C. Kulkarni,A. Vandercappele, P. G. Kjeldsberg, “Data and Memory Optimization Techniques for Embedded Systems”, *ACM Transactions on Design Automation of Electronic Systems*, vol. 6, no. 2, pp. 149-206, April 2001.

[11] M. Pedram, J. Rabaey, *Power-aware design methodologies* Kluwer, 2002.

[12] D. Shin, J. Kim, S. Lee, ”Intra-task voltage scheduling for low-energy hard real-time applications,” *IEEE Design & Test of Computers*, vol. 18, no. 2 pp. 20-30, March-April 2001.

[13] T. Theis, “The future of Interconnection Technology,” *IBM Journal of Research and Development*, Vol. 44, No. 3, May 2000, pp. 379-390.

[14] H. Zhang, V. Prabhu, V. George, M. Wan, M. Benes, A. Abnous, J. Rabaey, “A 1-V Heterogeneous Reconfigurable DSP IC for Wireless Baseband Digital Signal Processing,” *IEEE Journal of Solid-State Circuits*, vol. 35, no. 11, pp. 1697–1704, Nov. 2000.

[15] P. Zuchowski et al., ”A hybrid ASIC and FPGA architecture,” *IEEE/ACM International Conference on Computer-Aided Design* pp. 187-194, Nov. 2002.