Power-Aware Architectural Synthesis

Robert P. Dick†, Li Shang‡, and Niraj K. Jha⋆

† Dept. of Electrical Engg. & Computer Science
Northwestern University
Evanston, IL, U.S.A.

dickrp@ece.northwestern.edu

‡ Dept. of Electrical & Computer Engg.
Queen's University
Kingston, Ontario, Canada

lshang@queensu.ca

⋆ Dept. of Electrical Engg.
Princeton University
Princeton, New Jersey, U.S.A.

jha@ee.princeton.edu

I. Introduction

Power consumption is of great important parameters of modern electronic systems. It impacts the performance, cooling costs, packaging costs, and reliability of integrated circuits (ICs), as well as the lifespans of battery-powered electronics. Therefore, it is important to consider power consumption during design.

The complexity and scope of automatic design continues to increase. It is now possible to automatically design, i.e., synthesize, complex ICs and systems from high-level specifications without designer intervention. Architectural synthesis has been an active research area for more than a decade. Since addressing power consumption at higher levels of the design process increases the potential for improvement, researchers have developed a wide range of power optimization and management techniques to address IC power consumption issues during architectural synthesis. In this article, we present techniques for the synthesis of low-power ICs and systems. In particular, we focus on power-aware behavioral synthesis and system synthesis.

The rest of this article is organized as follows. In Section I-A, we introduce and define behavioral synthesis and system synthesis. In Section II, we describe the contributors to IC and system power consumption and discuss a number of techniques to improve power and thermal characteristics. Many of these techniques will prove useful in both behavioral synthesis and system synthesis. In Sections III and IV, we provide details on behavioral synthesis and system synthesis and indicate areas of active research. Section V points out a few commercial architectural synthesis products. We conclude in Section VI.

A. Architectural synthesis overview

In 1958 and 1959, Jack Kilby and Robert Noyce built the first ICs. Although the simple applications of early ICs enabled fully manual design, within ten years, engineers were designing large-scale integration (LSI) ICs containing tens of thousands of transistors. In the late 1960s and early 1970s, fully manual design became impractical and engineers began automating the design process. Table I gives a chronology of areas of active research and development in electronic design automation. Note that the first research in an area may have appeared before the area was of wide interest, e.g., some researchers had already made great progress in behavioral synthesis before the 1990s. As indicated in Table I, tasks that consist of simple actions repeatedly applied were the most straightforward and the first to be automated. However, as design complexity continued to increase, engineers found it necessary to automate increasingly complicated and creative tasks that had once required the efforts of skilled designers. In recent years, two trends are apparent: higher levels of the design process have been automated and power consumption has become a first-order design characteristic. In the past five years, these trends have converged; research on power-aware architectural synthesis has proceeded at a rapid pace.

Fig. 1 illustrates the conventional levels or stages of digital system design. Physical design, i.e., deciding on the physical locations and shapes of transistors, func-

TABLE I

CHRONOLOGY OF ACTIVE RESEARCH AND DEVELOPMENT TOPICS IN ELECTRONIC DESIGN AUTOMATION

1958–1965	Manual design
1965–1975	Schematic capture, automated mask production
	Circuit simulation
	Automatic routing
1975–1985	Automated placement
	Design rule checking
	Layout vs. schematic checking
1985–1990	Hardware description languages
	Logic synthesis
	Static timing analysis
1990–1995	Behavioral synthesis
	Formal verification
1995–2000	Hardware-software co-synthesis
	SoC synthesis
	Low-power design becomes critical
2000–2005	NoC synthesis, platform-based design
	Synthesis for new processes, e.g., microfluidics and MEMS
	Thermal and reliability issues become critical

tional units, and processors, as well as communication, clock distribution, and power distribution networks, was largely automated in the 1960s and 1970s. However, this area remains open, with continued improvement over past work and new algorithms to deal with changes brought about by process scaling. Combinational logic synthesis, the efficient design of combinational networks that implement Boolean expressions, advanced rapidly in the 1970s and 1980s. Register-transfer level (RTL) optimizations, such as retiming, underwent substantial advances in the 1990s.

This trend of automating increasingly high levels of the design process continues to this day. Sophisticated algorithms are now used to automatically design, or synthesize, very large scale integration (VLSI) circuits and hardware-software systems, starting from high-level descriptions of application behavior. These synthesis algorithms automatically make design decisions at many levels, ranging from architectural level to physical level, in order to optimize performance, energy consumption, thermal characteristics, price, and reliability. Power consumption is now a critical cost for synthesized architectures. It influences packaging and cooling costs, performance, reliability, and battery lifespan. Moreover, optimizing power and thermal characteristics greatly increases the complexity of synthesis. This article gives a taxonomy of synthesis problems, describes how state-of-the-art synthesis algorithms solve these problems, and indicates trends that will influence future work in the field.

Architectural synthesis may be broken into two main areas: behavioral synthesis and system synthesis. This article describes these areas and explains methods of reducing power consumption during synthesis. However, each area is broad; they cannot be exhaustively covered here. System synthesis has its roots in hardware-software co-synthesis, with much current activity in system-on-chip (SoC) synthesis and network-on-chip synthesis. This article describes hardware-software co-synthesis and SoC synthesis but defers to Marculescu's article in this chapter for a detailed treatment of network-on-chip synthesis [1].

Behavioral synthesis and system synthesis share a few common challenges. In both cases, starting from an abstract description of the application to be implemented, constraints on the costs of the system (e.g., price, performance, and power consumption), and a database of resources that may be used to implement the application, it is necessary to determine which processing and communication resources will be used in the final

design (*allocation*[1]), determine the resource that will be used for each particular operation and communication event (*assignment*), and provide a means of controlling the times at which all events occur (*scheduling*). These tasks are challenging; both the allocation/assignment and scheduling problems are NP-complete [2][2]. In summary, behavioral and system synthesis share a number of hard problems.

Behavioral synthesis differs from system synthesis as indicated in Fig. 1 and Table II. Behavioral synthesis and system synthesis can, in some cases, start from the same sorts of specifications. However, in behavioral synthesis, most operations are fine-grained, i.e., they can be represented by short instruction sequences for a general-purpose processor and may be implemented in hardware as a combinational network or a shallow pipeline, e.g., multiplication. In system-level synthesis, tasks are generally coarse-grained. They may be complex procedures requiring numerous general-purpose instructions or highly sequential hardware implementations, e.g., fast Fourier transform. In behavioral synthesis, it is generally assumed that the entire specification is implemented in synthesized hardware. In system synthesis, hardware and software are both used in the implementation. Differences in task granularity and implementation style (hardware-only or hardware-software) lead, in turn, to other differences between behavioral synthesis and system synthesis.

The simplicity of components, e.g., functional units and wires, in behavioral synthesis simplifies clocking, scheduling, and interface problems; it is usually possible to assume a globally synchronous system, a discrete-time schedule in which all operations take small integer numbers of clock cycles, and straightforward interfacing between components. However, the simplicity of individual components is offset by their quantity. Quickly determining the impact of architectural decisions on the floorplans[3] and thermal profiles of designs containing hundreds or thousands of frequently parallel operations is extremely challenging.

In system synthesis, the number of components is limited. However, they are generally more complicated than arithmetic functional units, e.g., instruction processors or protocol translators. The system synthesis algorithm may not have control over the implementation of each complex component. Therefore, providing for global synchronization and communication is more challenging. Interface synthesis, i.e., synthesizing the interfaces between hardware components as well as software and hardware, is an active area of research in system synthesis. Unlike behavioral synthesis, tasks in system synthesis need not take a small integer number of clock cycles: time values are modeled as reals, not integers. Moreover, some operations may have dramatically higher execution times than others. Therefore, a number of discrete time domain scheduling algorithms that are promising in behavioral synthesis are not directly applicable in system synthesis.

[1]Some behavioral synthesis researchers define *allocation* to be the assignment of tasks and communication events to resources as well as the selection of resources.

[2]Garey and Johnson provide an introduction to the theory of NP-completeness [2]. For the purpose of this article, the implications can be summarized as follows: nobody has ever developed and reported an algorithm that can quickly produce optimal solutions to large instances of these problems and there is strong evidence (but no proof) that such an algorithm cannot be implemented using conventional, deterministic, computers.

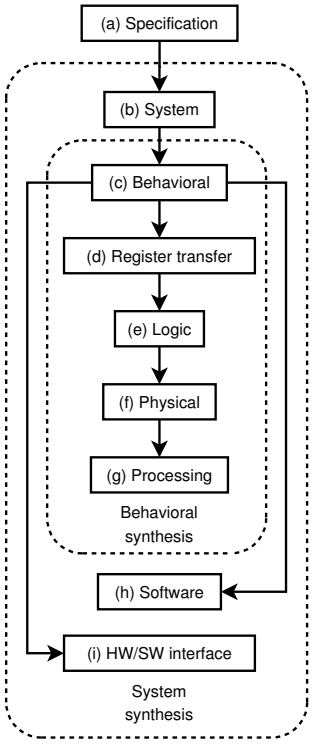

Fig. 1. Digital system design levels

II. Challenges of Low-Power Synchronous System Synthesis and Design

This section introduces the fundamentals necessary to understand the sources of power consumption in synchronous digital systems (Section II-A). It then describes a number of techniques that may be used during

[3]A floorplan indicates the positions of all architectural components in an IC.

TABLE II
DIFFERENCES BETWEEN BEHAVIORAL SYNTHESIS AND SYSTEM SYNTHESIS.

	Behavioral synthesis	System synthesis
Implementation	Hardware, IC	Hardware-software system
Timing model	Discrete	Continuous
Processing element model	Combinational networks, shallow pipelines, registers, multiplexers,	Processors, protocol translators, memories
Communication resource model	Wires	Protocols over busses, network-on-chip, or wires

behavioral synthesis and system synthesis to improve power and thermal characteristics (Sections II-B– II-E).

A. Power overview

With increasing system integration, as well as aggressive technology scaling, power consumption has become a major challenge in digital system design. In high-performance computer systems, power and thermal issues are key design concerns. Power management and optimization techniques are essential for minimizing system power consumption and temperature to permit reliable operation. For portable devices, prolonging battery lifetimes and minimizing packaging costs are primary design challenges. Power also interacts with other design metrics, such as performance, cost, and reliability, thereby further increasing design complexity. For example, the failure rate of electronic devices is a strong function of system temperature, which is in turn controlled by system power dissipation. Therefore, increasing power consumption results in the need for more complicated cooling and packaging solutions to sustain system reliability, which in turn increases costs. As projected by International Technology Roadmap for Semiconductors (ITRS) [3], power will continue to be a limiting factor in future technologies. There is an increasing need to address power issues in a systematic way at all levels of the design process.

In digital CMOS circuits, power dissipation is the sum of dynamic power, $P_{dynamic}$, and static power P_{static}. Dynamic power, $P_{dynamic}$, results from charging and discharging of the capacitance of CMOS gates and interconnect during circuit switching, P_{switch}, and the power during transient short-circuits when inputs are in transition, $P_{short\ circuit}$. For synchronous CMOS designs, switching power is one of the dominant sources of power consumption. It is a function of physical capacitance, C, switching activity, s [4], clock frequency, f, and supply voltage, V_{dd}:

$$P_{switch} = \frac{1}{2} s C {V_{dd}}^2 f \quad (1)$$

In CMOS, the other major source of power consumption, static power, P_{static}, results from leakage current. Leakage current has five basic components: reverse-biased PN junction current, subthreshold leakage, gate leakage, punch-through current, and gate tunneling current. Of these five components, subthreshold and gate leakage will remain dominant during the next few years. The subthreshold leakage power is given by

$$P_{subthreshold} = I_{sub} \frac{W}{L} V_{dd} e^{\frac{-V_{th}}{nV_T}} \quad (2)$$

where I_{sub} and n are technology parameters, W and L are device geometries, V_{th} is the threshold voltage, and V_T is the thermal voltage constant [4]. Gate leakage is the current between the gate terminal and any of the other three terminals (drain, source, body). As a result of technology scaling, gate leakage increases exponentially due to decreasing gate oxide thickness.

From Equations (1) and (2), it can be seen that total power consumption may be reduced by attacking operating voltage, capacitance, switching activity, threshold voltage, transistor size, and temperature. In real designs, the variables upon which total power consumption depends are often closely related: reducing one may increase another. In addition, reducing power consumption may have a negative impact on other design metrics. A synthesis algorithm must simultaneously consider, and trade off, these design metrics.

B. Operating voltage oriented techniques

Reducing operating voltage, V_{dd}, is one of the most promising techniques for reducing dynamic power consumption. As indicated by Equation (1), $P_{dynamic}$ is quadratically related to V_{dd}. All other things being equal, halving V_{dd} reduces $P_{dynamic}$ to 1/4 of its initial value. However, this reduction has a negative impact on circuit

[4]The product of physical capacitance, C, and switching activity, s, is also called switched capacitance.

performance [5]:

$$f = \frac{k(\mathrm{V_{dd}} - \mathrm{V_{th}})^{\alpha}}{\mathrm{V_{dd}}} \quad (3)$$

where k is a design-specific constant and α is a process-specific constant ranging from one to two. As a result, for low values of $\mathrm{V_{th}}$ and $\alpha \simeq 2$, and all other things being equal, halving $\mathrm{V_{dd}}$ implies halving clock frequency, f. However, some of the following paragraphs describe techniques for reducing operating voltage without degrading performance.

Multiple simultaneous operating voltages: ICs contain timing critical and non-critical combinational logic paths between memory elements (latches and flip-flops). It is possible to selectively decrease the operating voltage(s) of gates on the non-critical paths, thereby reducing $P_{dynamic}$ without reducing performance. Multiple voltage techniques may be used within with architectural synthesis. Although it is not essential, processing elements[5] sharing the same voltage are often placed in contiguous regions called voltage islands to simplify power distribution. In addition, communication between different voltage regions relies on level converters. This physical requirement for contiguous regions dramatically changes the IC floorplan, thereby changing communication power consumption, wire delays, and thermal properties. These changes, in turn, impact the original design properties, e.g., combinational path criticality, optimal clock frequency, and operation cycle times. It is necessary to consider the consequences of using multiple voltages at multiple design levels, i.e., architectural and physical. Recent multiple voltage behavioral [6] and system [7] synthesis techniques allow solution of the voltage level assignment problem concurrently with one or more of the other following problems: processing element selection, assignment of tasks to processors, scheduling, and floorplanning.

Dynamic voltage (and frequency) scaling: In addition to varying the operating voltages of sub-circuits by position, it is possible to vary operating voltages in time. Dynamic voltage scaling is generally carried out in conjunction with frequency scaling to prevent timing violations. It allows an IC to adaptively adjust operating voltage to minimize power consumption without violating timing constraints. Dynamic voltage and frequency scaling (DVFS) interacts closely with scheduling: some schedules allow timing slack to be used for power minimization without the violation of deadlines while others leave little opportunity for power minimization.

[5]When used in a general context, the term processing element will be used to refer to both functional units, e.g., multipliers and adders, as well as system-level processing elements, e.g., microprocessor cores.

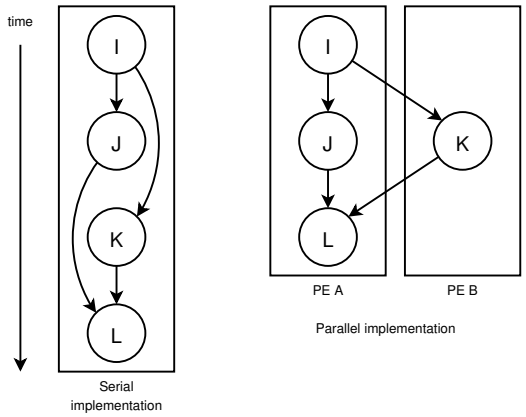

Fig. 2. Series and parallel implementations of a dataflow graph.

Synthesis algorithms have been developed for both off-line DVFS and on-line DVFS [8], for which predictions of future system behavior are used to amortize the cost of voltage and frequency changes over longer low-power periods.

Scheduling and timing: Scheduling is the process of selecting the orders and execution start times of operations and communication events. In some cases, a system's original schedule may not permit the reduction of operating voltage(s) without performance degradation. For example, some operations may be immediately followed by other operations, leaving little spare time for voltage reduction. Changing operation start times and orders can open up opportunities for greater reductions in power consumption via operating voltage reduction. It is also possible to change the number of clock cycles and frequency for an operation, thereby allowing a decrease in power consumption without degrading computational throughput.

Power for performance and area techniques: Even if it seems that attempts to reduce the operating voltage for tasks on critical timing paths will result in performance degradation, it is sometimes possible to buy back the lost performance at a cost in area. Consider the example in Fig. 2. In the serial implementation shown on the left, the processing element must operate at a high voltage at all times to meet the performance requirements. By adding another processing element and parallelizing the operations, as shown by the parallel implementation to the right, it is possible to finish execution early, thereby providing enough timing slack to permit operating voltage to be halved, reducing dynamic power consumption to 1/4 its initial value. This general technique of buying voltage reduction at the cost of performance and gaining back the lost performance through increased area and/or

design complexity forms the basis of a number of techniques in low-power architectural synthesis [9]–[11].

C. Switched capacitance-oriented techniques

It is possible to reduce both active device and interconnect capacitance via a number of synthesis techniques. Reducing a CMOS gate in size reduces the capacitance driven by the previous gate. However, this also increases resistance to the power and ground rails, increasing the delay of the subsequent gate. In many cases, several devices are not on the critical timing path of the system. Their sizes may be reduced to reduce driven capacitance. This technique shares properties with operating voltage reduction. However, the potential for improvements in dynamic power consumption is generally smaller because the relationship between power and capacitance-dependent delay is sub-quadratic, i.e., reducing operating voltage is generally a better choice than reducing capacitance. However, reducing capacitance does come with one additional advantage: the resulting decrease in gate size also reduces area. During architectural synthesis, it is common for libraries to contain functionally equivalent processing elements with different power and performance properties. Many of these differences have their sources in differing internal gate and wire capacitance values. However, in architectural synthesis, this problem is often encompassed by processing element selection and assignment of operations to processing elements.

Interconnect self-capacitance may be decreased by three techniques, one local and two architectural. Decreasing interconnect width reduces capacitance at the cost of increased delay. However, it is also possible to simultaneously reduce interconnect delay and capacitance by decreasing wire length. Finally, one can change the assignment of tasks to processing elements to reduce or eliminate the inter-processing element switched capacitance necessary for data communication. The lengths of wires are decided by the impact of architectural decisions, such as the allocation of processing elements and the assignment of operations to processing elements, upon the floorplan ultimately produced. The impact of interconnect coupling on effective capacitance is becoming increasingly important. It can be addressed during architectural synthesis via bus planning as well as coding techniques.

Switched capacitance, the product of physical capacitance and switching activity, reflects the actual run-time load of the circuit. Recent studies [12] have demonstrated that switched capacitance minimization is a much more efficient power optimization technique than physical capacitance reduction. Switched capacitance reduction techniques have been developed at all levels of the design hierarchy. Architecture-level techniques [13], such as power management, data encoding, glitch suppression, architectural transformation, are widely used in low-power behavioral and system synthesis.

D. Leakage power techniques

Most work in low-power synthesis explicitly targets dynamic power consumption. This is not surprising. Even at the 90 nm process node, dynamic power accounts for over 90% of total power in modern processors. However, research indicates that a half or more of the total power consumption will result from leakage at the 25 nm process node [3]. Subthreshold leakage is an exponential function of chip temperature. As a result, increasing temperature from 25°C to 100°C can result in subthreshold leakage being the dominant source of power consumption.

As indicated in Equation (3), it is necessary to reduce V_{th} in unison with V_{dd} in order to maintain good performance. However, reduction in threshold voltage increases subthreshold leakage. This problem may be addressed by using multiple threshold voltages [14], such as multi-V_{th}, adaptive body biasing, etc. During synthesis, high threshold voltages can be assigned to functional units along non-critical timing paths to reduce subthreshold leakage while functional units on critical paths operate at lower threshold voltages to maintain performance.

Power gating reduces subthreshold leakage power consumption by inserting sleep transistors in series with pull-up or pull-down paths of functional units to control their leakage power dependent on the sleep transistor inputs [14]. NMOS transistors with high threshold voltages are typically used as sleep transistors. This circuit topology is known as MTCMOS. Other techniques, e.g., exploiting the transistor stack effect, transistor sizing, and supply voltage scaling, may also be used to minimize subthreshold leakage power consumption.

E. Temperature-oriented techniques

All other things being equal, increasing IC power consumption increases temperature. Using temperature-aware techniques in architectural synthesis is a complex task. IC temperature is affected by many factors, including IC dynamic and leakage power profile, interconnect power profile, as well as the packaging and cooling solution. Many of these power profiles are only available after physical design, i.e., floorplanning. Although power optimization techniques can reduce average chip temperature, local thermal hotspots due to unbalanced chip power profiles may result in thermal emergencies,

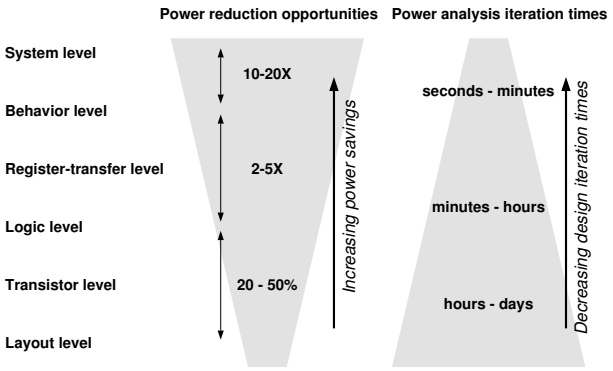

Fig. 3. Benefits of high-level power analysis and optimization [12].

e.g., reliability problems due to electromigration. In addition, subthreshold leakage power consumption has an exponential relationship with chip temperature. Without temperature optimization, leakage power can dominate power consumption. To address IC thermal problems, it is critical to integrate architectural synthesis with physical synthesis and thermal analysis to form a complete thermal optimization flow [6]. Thermal modeling and analysis also need to be incorporated into the inner optimization loop to guide IC synthesis. However, detailed thermal characterization requires 3D full chip-package thermal analysis, which may have high computational complexity. Thermal analysis may easily become the performance bottleneck for thermal-aware synthesis.

F. Potential of power optimization at different design levels

Although power minimization techniques were first developed at the device level, postponing power optimization until this stage of the design process neglects opportunities at higher levels. As indicated by Fig. 3, considering power minimization at earlier stages of the synthesis or design process has a number of advantages. It yields greater potential for improvement. Moreover, it indirectly improves solution quality because many candidate designs may be considered at higher levels of synthesis due to the use of more abstract (hierarchical) system modeling.

III. Low-power behavioral synthesis

Behavioral synthesis, or high-level synthesis, is the automatic design of an IC starting from an implementation-independent description of the design's behavior, a description of the functional units and communication resources available, and constraints on performance and power.

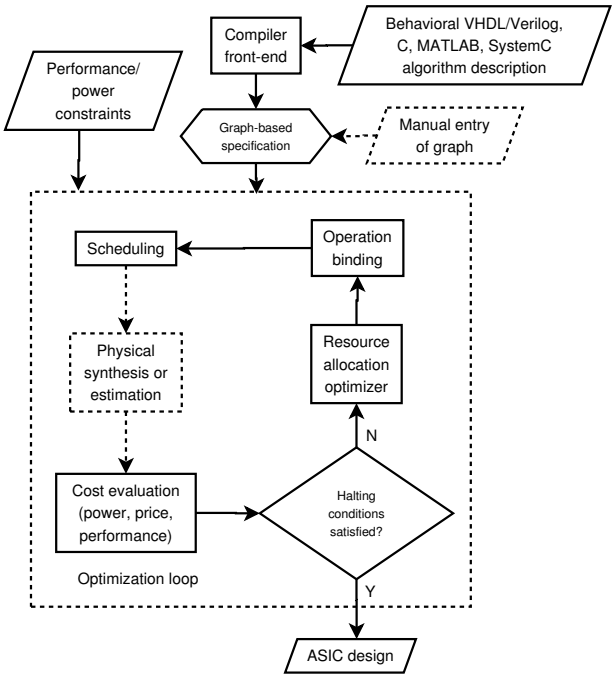

Fig. 4. Behavioral synthesis algorithm

```
void FIR_filter(int n, int order,
  int * a, int * x, int * y)
{
  int i, j;
  for (i = order - 1; i < n; ++i) {
    y[i] = 0;
    for (j = 0; j < order; ++j) {
      y[i] += a[j] * x[i - j];
    }
  }
}
```

Fig. 5. Finite impulse response filter code

Fig. 4 gives an overview of a behavioral synthesis optimization flow. Note that, although this flow is representative, other high-level meta-algorithms exist. For example, it would be possible to use a mixed integer linear program (MILP) solver on a unified behavioral synthesis problem formulation, in which case there would be no allocation, binding, and scheduling optimization loop.

Although the input to a behavioral synthesis system can take many forms, the most common are software language, hardware description language, or graph based specifications. An example C input file for a finite impulse response filtering algorithm is shown in Fig. 5. As shown in Fig. 4, regardless of the starting point, behavioral synthesis systems use compilers [15], [16]

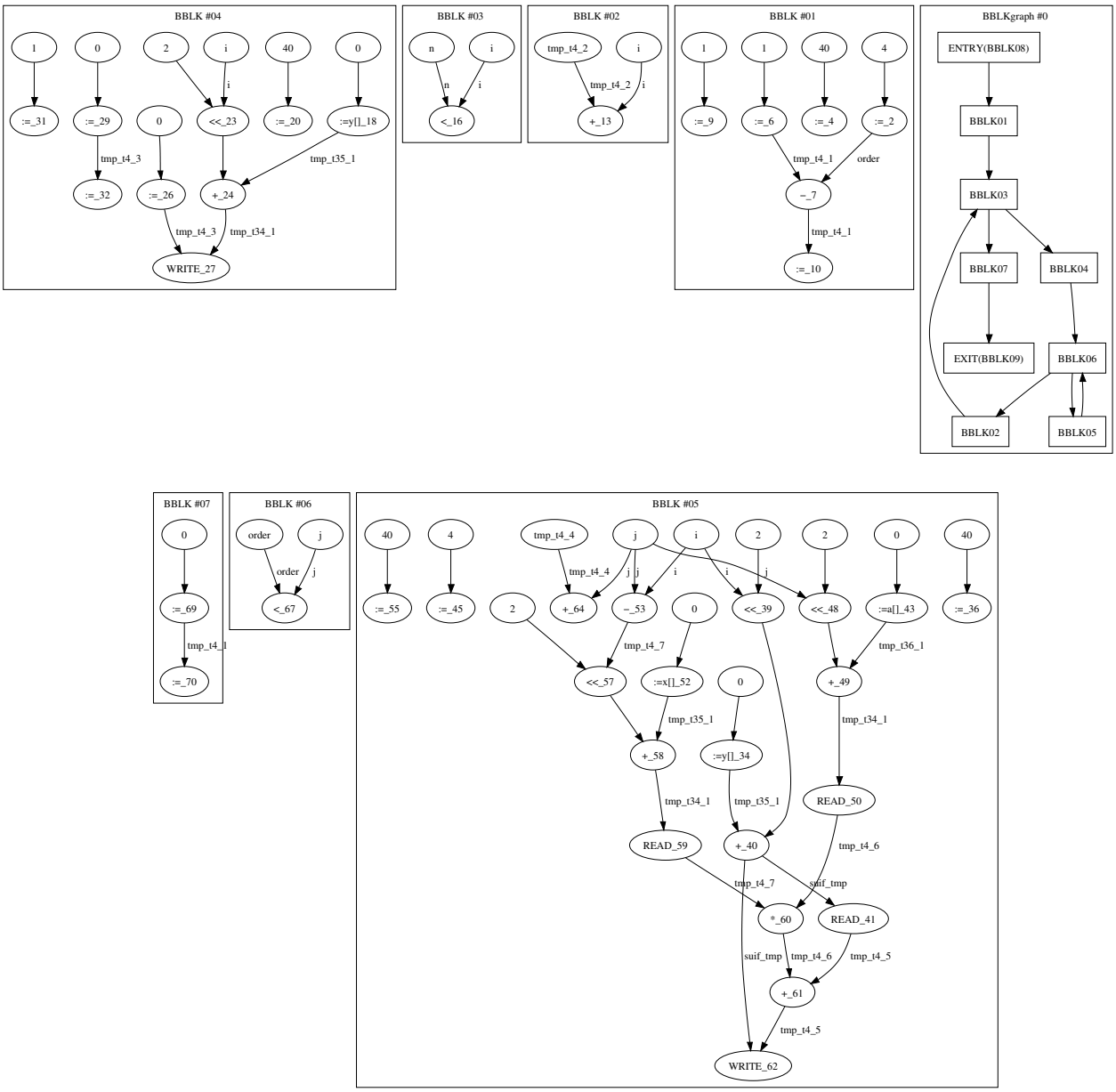

Fig. 6. Finite impulse response filter CDFG

to convert specifications into (possibly synchronous) data flow graphs (DFGs) or control-data flow graphs (CDFGs) for further optimization. Translation and performance optimization of the code in Fig. 5 results in the control-data flow graph shown in Fig. 6[6]. In this figure, the graph to the upper-right shows the flow of control among the basic blocks, i.e., straight-line sequences of code that may be represented with data flow graphs. Within each basic block, the nodes without incoming edges represent variables or constants and the other nodes represent operations on the data arriving on the incoming arcs. In addition to a description of the algorithm to be implemented, behavioral synthesis tools require models for the hardware resources that may be used in the implementation. For example, the user may provide a library of performance and power models for the available functional units, e.g., adders, multipliers,

[6]Figure courtesy of Rajarshi Mukherjee and Dr. David Zaretsky at Northwestern University.

and registers. These models may be provided as part of the resource library or automatically generated by commercial timing and power analysis tools.

A behavioral synthesis algorithm does functional unit allocation, operation binding, and scheduling to optimize performance, IC area, and possibly power. Power optimization, e.g., minimizing switched wire capacitance, may require physical information and, therefore, floorplanning block placement within behavioral synthesis. The product of behavioral synthesis is a complete RTL description of the synthesized system. This output is generally used as an input to a logic synthesis tool as indicated by Steps (c) and (d) in Fig. 1.

A. Dynamic power optimization

Extensive research has been conducted in low-power behavioral synthesis. In the past, IC power consumption was dominated by dynamic power. Therefore, most low-power synthesis research has focused on dynamic power optimization. Dynamic power is a quadratic function of supply voltage. Therefore, voltage reduction is commonly used to reduce power consumption in behavioral synthesis. However, reducing operating voltage requires global design changes, i.e., changes to functional unit allocation, assignment of operations to functional units, and schedules.

Optimal scheduling using multiple supply voltages is an NP-hard problem. Johnson and Roy developed a behavioral scheduling algorithm, called minimum energy schedule with voltage selection (MESVS) that uses integer linear programming (ILP) to optimize the energy consumption of a DSP datapath by using multiple supply voltages [17]. Voltage scaling may have a negative impact on circuit performance. In this work, timing requirements are enforced via ILP constraints. MESVS is limited to discrete voltage level selection. Later, Johnson and Roy proposed MOVER [18], which allows continuous voltage assignment. MOVER also uses an ILP-based method to conduct voltage selection and operation partition, and then derive a feasible schedule with minimum area overhead. Optimal ILP-based solutions generally have high computation complexity. Chang and Pedram developed a dynamic programming based method to address the multiple voltage scheduling problem in datapath circuits [19]. Under timing constraints, this approach reduces supply voltages along non-critical paths to maximize power reduction with low area overhead. Raje and Sarrafzadeh developed a heuristic-based voltage assignment algorithm, with computational complexity $\mathcal{O}\left(N^2\right)$, to minimize power consumption [20]. Although it is demonstrated that voltage reduction can greatly reduce power, incremental gains decrease with the number of voltage levels. In addition, incorporating multiple on-chip supply voltages complicates IC design.

In addition to voltage scaling, researchers have developed behavioral synthesis algorithms that minimize switching activity and driven capacitance. Chatterjee and Roy designed a behavioral synthesis system for low-power DSPs [21]. In this work, application data flow graphs were transformed to reduce switching activity, thereby reducing power consumption. Chandrakasan et al. designed HYPER-LP [22], a behavioral synthesis system. HYPER-LP uses algorithmic transformations enable voltage scaling and effective capacitance reduction. Kumar et al. [23] developed a profile-driven behavioral synthesis algorithm, using profiling to characterize the run-time activities of data flow graph based system models. Low-power behavioral synthesis is then conducted to minimize estimated system switching activity. Chang and Pedram proposed an allocation and binding technique to minimize the switching activity in registers [24]. In this work, statistical methods are used to characterize the switching activities of registers. A max-cost flow algorithm was then proposed to conduct power-optimal register assignment. Chang and Pedram also proposed a low-power binding technique to minimize the power consumption of datapath functional units [25], in which power optimization is formulated as a max-cost multi-commodity flow problem. Dasgupta and Karri proposed a simultaneous binding and scheduling techniques to reduce switching activity, hence the power consumption, of buses [26]. Mehra et al. proposed behavioral synthesis techniques for low-power real-time applications. By preserving locality and regularity in input behavior during resource assignment, this technique reduces the need for global buses, thereby reducing power consumption. Ercegovac et al. proposed a behavioral synthesis system [27] that uses multiple precision arithmetic units to support low-power ASIC synthesis. In this work, system resource allocation is conducted through multi-gradient search and task assignment is based on a modified Karmarkar-Karp's number partitioning heuristic.

A few researchers have developed high-level synthesis algorithms that combine numerous power optimization techniques. Musoll and Cortadella proposed several high-level power optimization techniques, including loop interchange, operand reordering, operand sharing, idle units, and operand correlation, for reducing the activities of functional units [28]. Raghunathan and Jha designed SCALP [29], an iterative-improvement-based behavioral synthesis system, for low-power data intensive applications. SCALP provides a rich set of behavioral optimization techniques, including architectural transfor-

mation, scheduling, clock selection, module selection, and hardware allocation and assignment. Khouri et al. showed how to perform low power behavioral synthesis for control-flow intensive algorithms [30]. This work uses an iterative improvement framework to perform design space exploration. Behavioral power optimization techniques, including loop unrolling, module selection, resource sharing and multiplexer network restructuring, are done concurrently.

B. Physical-aware power optimization

In conventional behavioral synthesis, physical implementation details were generally ignored when making architectural decisions. Continued process scaling has required fundamental changes to IC synthesis. At present, physical design details must be considered during all stages of IC synthesis. Many of the techniques use physical information, e.g., floorplan block placements, to better optimize switched capacitance [31]–[34], as explained in Section II-C. Although they do not use a floorplan, Lyuh et al. optimize assignment of communication events to interconnect buses, and the order of (capacitively coupled) wires within buses, to reduce effective switched capacitance [35]. Prabhakaran and Banerjee proposed a simultaneous scheduling, binding and floorplanning algorithm to address the power consumption of interconnect during behavioral synthesis. Zhong and Jha presented an interconnect-aware low-power behavioral synthesis algorithm, called ISCALP, that minimizes power consumption in in interconnects through interconnect-aware binding [36]. Recently, Gu et al. designed a fast, high-quality incremental floorplanning and behavioral synthesis system that concurrently optimizes performance, power, and area [37].

C. Leakage power optimization

As a result of technology scaling, leakage power consumption is becoming increasingly significant in digital CMOS circuits. Khouri and Jha [38] were the first to propose a method of reducing leakage power consumption during behavioral synthesis. They proposed an iterative algorithm to minimize leakage power consumption during behavioral synthesis using dual-V_{th} technology. Through each iteration, a greedy prioritization approach is used to identify the functional unit with maximum leakage power reduction potential, and then replace it with a higher-V_{th} functional unit. Gopalakrishnan and Katkoori proposed KnapBind [39], a leakage-aware resource allocation and binding algorithm to minimize datapath leakage power consumption. This work maximizes the idle time of datapath modules. MTCMOS functional modules with large idle time slots are placed into sleep mode when they are idle. Tang et al. proposed a heuristic to minimize leakage power consumption during behavioral synthesis [40]. The synthesis problem is formulated as the maximum weight independent set problem. Datapath components with maximum or near-maximum leakage saving potentials are identified and replaced with low-leakage library modules. Leakage power is a strong function of chip temperature. Mukherjee et al. proposed a temperature-aware resource binding technique to minimize leakage power consumption during behavioral synthesis [41]. The proposed iterative resource binding technique minimizes chip peak temperature by balancing the chip power profile, thereby reducing leakage power.

D. Thermal optimization

Increasing performance requirements and system integration are dramatically increasing IC power density, hence chip temperature. Thermal effects are becoming increasingly important during IC design. Mukherjee et al. addressed thermal issues during behavioral synthesis [42]. They proposed temperature-aware resource allocation and binding algorithms to minimize chip peak temperature. Gu et al. designed TAPHS, a thermal-aware unified physical and behavioral synthesis system [6]. TAPHS incorporates a complete set of integrated behavioral and physical thermal optimization techniques, including voltage assignment, voltage island generation, and thermal-aware floorplanning, to jointly optimize chip temperature, power, performance and area. Thermal-aware behavioral synthesis algorithms must determine the temperature profiles of a tremendous number of candidate designs. Recently, researchers have developed and publicly released fast and accurate thermal analysis tools specifically for this purpose [43].

IV. Low-power system synthesis

System synthesis has its roots in hardware-software co-synthesis. Early hardware-software co-synthesis algorithms took, as input, a high-level description of the application's required functionality, descriptions of available hardware, e.g., instruction processors and application-specific integrated circuits (ASICs), as well as performance and power requirements. The hardware-software co-synthesis algorithm automatically produced a design for the desired application, often consisting of application-specific and general-purpose processors mounted on a printed circuit board. The main focus of most hardware-software co-synthesis algorithms is partitioning applications between instruction processors and application-specific cores/ICs.

SoC synthesis algorithms target hardware-software systems implemented on single ICs. Although their func-

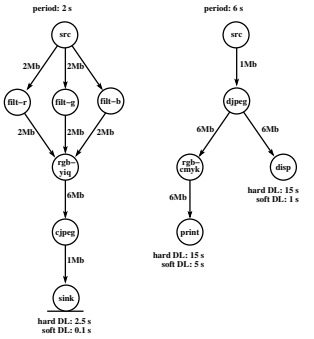

Fig. 8. Example image processing application specification

tionality overlaps with hardware-software co-synthesis algorithms, SoC synthesis algorithms also place great weight on synthesizing (heterogeneous) communication busses or networks. In addition, some consider the interaction between architectural and physical design in order to better solve the entire SoC synthesis problem.

Fig. 7 illustrates a system synthesis optimization flow. Although this flow is representative, some flows, e.g., those using constructive algorithms, may differ. Initially, a description of the algorithm to be implemented is provided in a high-level language such as MATLAB, C, or SystemC. This description is then translated into a graph representation by a compiler front-end. Note that these first stages may be omitted if a graph-based specification is available. One such graph format, shown in Fig. 8[7], is a task set composed of multiple directed acyclic graphs in which nodes represent tasks and edges represent data dependencies. Timing constraints may be expressed as deadlines (DL) on nodes. Different tasks may be invoked periodically with different periods.

In addition to the required functionality, a database containing price, power consumption, execution time, and other characteristics of processing elements and communication resources is also provided. A portion of one such database is shown in Table III [44].

Potential architectures consisting of processing element allocations, assignments of tasks to processing elements, and a schedule of all tasks and communication events are then optimized. Costs such as price, power, and execution time are then evaluated. The process repeats until acceptable solutions are produced. The resulting architectures are then completed by using behavioral synthesis to generate application-specific cores or FPGA configurations for the hardware-implemented tasks and using a compiler to generate executable code for the software-implemented tasks. Note that many existing system synthesis algorithms only solve subsets of the entire system synthesis problem.

[7]Figure from the E3S benchmark suite [45].

TABLE III
PORTION OF PROCESSING ELEMENT PERFORMANCE AND POWER CONSUMPTION DATABASE [44]

AMD K6-2E 400 MHz/ACR		
Price ($)	Idle power (mW)	
33	160	
Type	Time (μs)	Power (W)
Angle to Time Conversion	1.5	10
Basic Floating Point	2.9	10
CAN Remote Data Request	0.35	10
Fast Fourier Transform	1600	10
RGB to YIQ Conversion	16000	10
Image Rotation	2100	10
Text Processing	2800	10
⋮	⋮	⋮

NEC VR5432 167 MHz		
Price ($)	Idle power (mW)	
33	250	
Type	Time (μs)	Power (W)
Infinite Impulse Response Filter	83	2.5
Inverse Discrete Cosine Transform	840	2.5
Inverse Fast Fourier Transform	16000	2.5
Matrix Arithmetic	36000	2.5
⋮	⋮	⋮

A. Low-power hardware-software co-synthesis algorithms

Low-power co-synthesis algorithms form the basis for later work on low-power SoC synthesis. They build upon power-aware allocation, assignment, and scheduling optimization engines and further improve power consumption with point techniques such as multiple voltage levels and domain-specific scheduling algorithms. Dick and Jha developed a synthesis algorithm for low-power distributed systems [46] that simultaneously optimizes power consumption and price while honoring hard real-time deadlines. Dave et al. developed a constructive algorithm to solve the low-power multi-rate distributed system co-synthesis problem [47]. Shang and Jha presented a method of synthesizing low-power systems containing dynamically reconfigurable FPGAs [48].

Much of the early work in low-power hardware-software co-synthesis was based on the assumption that processing elements are off-the-shelf parts with strict

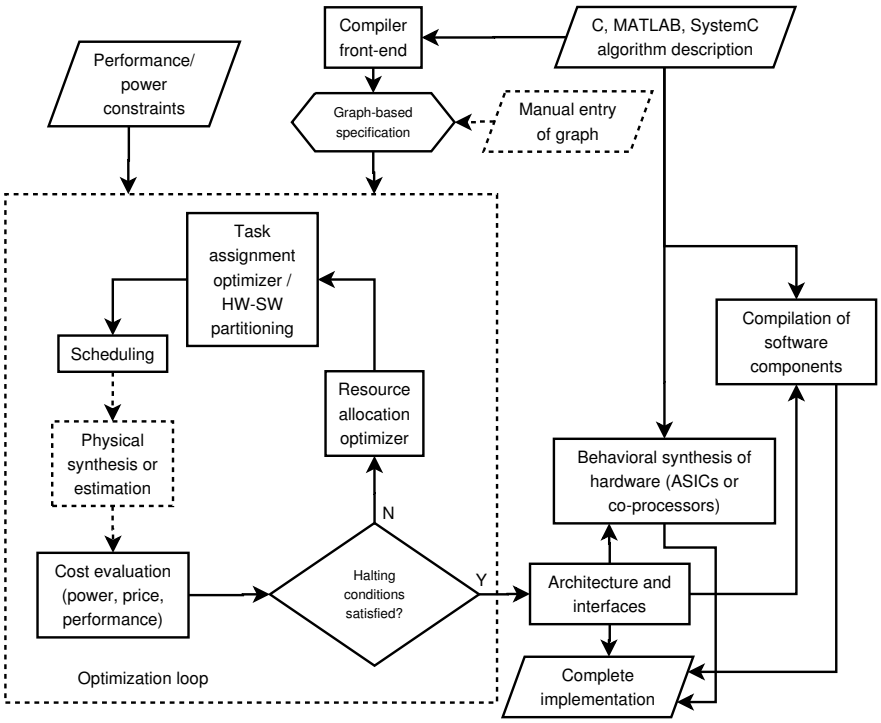

Fig. 7. System synthesis algorithm

constraints on operating voltages. Later work relaxed this assumption, considering multiple operating voltages and DVFS (described in Section II). Gruian and Kuchcinski developed a dual-voltage task scheduling algorithm for reducing power consumption [49]. Kirovski and Potkonjak developed an integrated DVFS and system synthesis algorithm for independent tasks mapped to a bus-based multiprocessor [50]. Schmitz and Al-Hashimi developed a genetic algorithm to incorporate DVFS into an energy minimization technique for distributed embedded systems [51]. It takes the power variations of tasks into account while performing DVFS. An off-line voltage scaling heuristic is proposed that is fast enough for use in system synthesis, starting from real-time periodic task graphs. Yan et al. proposed a scheduling algorithm that uses DVFS and adaptive body biasing to jointly optimize both dynamic and leakage power consumption [52]. Analytical solutions are derived to determine the optimal supply voltage and bias voltage. Then, the optimal energy consumption is determined under real-time constraints.

DVS can also be applied to communication links. Naturally, performing simultaneous DVS in the processors and communication links in a distributed system can yield greater power savings than performing DVS in the processor alone. Luo et al. presented such a method [53]. In addition to honoring real-time constraints, their scheduling algorithm also efficiently distributes timing slack among tasks and multi-hop communication events.

Quality of service (QoS) is an important consideration in designing systems for real-time multimedia and wireless communication applications. Qu and Potkonjak proposed a technique for partitioning a set of applications among multiple processors and determining a DVFS schedule to minimize energy consumption under constraints on QoS [54]. The applications are assumed to be independent, have the same arrival times and no deadline constraints.

B. Low-power system-on-chip synthesis algorithms

The low-power system-on-chip problem combines elements of hardware-software co-synthesis problem and behavioral synthesis problem. Like hardware-software co-synthesis, tasks may be implemented with general-purpose instruction processors or application-specific hardware accelerators. However, the synthesis algorithm potentially has greater control over the details of hardware implementation, opening new options for power optimization.

Methods of estimating SoC power consumption are essential to enable design exploration and synthesis.

Bergamaschi et al. developed an SoC analysis tool that estimates power and may be used within a system synthesis flow [55]. Lajolo et al. described a number of ASIC and instruction processor power estimation techniques that may be used in system synthesis [56]. Based on these power estimation algorithms, synthesis algorithms may select and optimize SoC designs.

Power estimation techniques can be used to guide the search for high-quality solutions during the synthesis of low-power or low-temperature SoCs. Givargis et al. developed a method of pruning the set of SoC candidate architectures in order to efficiently arrive at low-power designs [57]. They determine which elements of the solution are independent from each other, thereby decomposing the problem into small, independent problems. Fei and Jha describe a functional partitioning method for synthesizing low-power real-time distributed embedded systems whose constituent nodes are SoCs [58]. The input specification, given as a set of task graphs, is partitioned and each portion is implemented as an SoC. Hung et al. give a method of using voltage islands and thermal analysis within SoC synthesis to minimize peak temperature [7]. Hong et al. presented an algorithm to select a processor core and instruction/data cache configuration to best enable DVFS [59].

Communication networks have a large impact on the power consumption, performance, and feasibility of SoC designs. As a result, a number of researchers have worked on low-power, communication-centric SoC synthesis. Dick and Jha developed a low-power SoC synthesis algorithm that optimizes power consumption, performance, and area [60]. It uses floorplanning block placement to estimate communication delay, power consumption, and wire congestion. Lyonnard et al. developed a low-power SoC synthesis algorithm that gives great attention to communication network synthesis [61]. Instead of estimating physical characteristics via floorplanning, this work focuses on logical bus structure and communication protocol modeling. Hu et al. optimize SoC bus bit-width under a fixed processing element allocation, task assignment, and schedule [62]. Results for a seven core H.263 encoder are presented. Thepayasuwan et al. used simulated annealing to design bus topologies and demonstrated results for a JPEG SoC design [63]. They do parasitic extraction for performance estimation and reduce power consumption by minimizing bus length. They propose using the algorithm as a synthesis post-processing step. Hu et al. presented a method of using voltage islands in SoC designs that minimizes power consumption, area overhead, and number of voltage islands [64]. Pasricha et al. developed an algorithm for floorplan-aware synthesis of bus topologies that meet combinational delay constraints imposed by bus cycle times [65]. This work assumes a fixed IP core allocation and task assignment. It minimizes bus count and bus width under explicit communication throughput constraints.

Conventional SoC designs typically contain a limited number of modules, connected by on-chip buses or point-to-point links. However, as the number of on-chip modules grows in the coming years, bus or point-to-point link communication will face serious problems due to increasing global wire delay. To address these issues, in SoC designs, buses are gradually being replaced by more sophisticated on-chip communication networks [66]. On-chip networks may consume a significant portion of SoC power budgets [67]. Therefore, power and power-related design problems, such as thermal issues [68], are of great concern in network-on-chip designs. The design and synthesis of on-chip networks supporting multi-hop routing has grown into an active and broad research area. Readers may refer to Marculescu's article in this chapter for a detailed treatment of this area [1].

V. Commercial products

Although complete and general low-power system synthesis tools are not yet available, a number of supporting tools have been released. As described in Section III, the performance and power consumption of functional units, can be automatically determined via logic synthesis and analysis tools such as PrimeTime and PrimePower from Synopsys, Encounter from Cadence, Blast Power and Blast Fusion QT from Magma Design Automation, as well as Synplify from Synplicity.

Behavioral synthesis has reached a level of maturity at which a number of commercial products are available. Cynthesizer from Forte Design Systems synthesizes a SystemC algorithm to an RTL description. The Get2Chip synthesis tool, now owned by Cadence, translates Superlog to RTL. A number of synthesis tools target FPGAs. The DSP Synthesis tool from AccelChip starts from MATLAB, CoDeveloper from Impulse Accelerated Technologies starts from C, Mitrion's virtual processor starts from a C-like language, and BINACHIP's FREEDOM compiler starts from (digital signal processing) instruction processor executables.

VI. Conclusions

As indicated in Section V, behavioral synthesis is a commercially supported alternative to RTL design. A number of companies offer solutions to portions of the system synthesis problem. Both areas remain open with

active research on new application domains, new synthesis algorithms, and new implementation technologies. Power and thermal optimization techniques in behavioral synthesis and system synthesis are necessary to improve performance, battery life, reliability, product size, and cooling costs. During the next five years, we can expect behavioral and system synthesis to continue to displace and supplement manual architectural design for high-complexity products that are produced in limited volumes, e.g., application-specific embedded systems. In addition, we can expect continued research on power-aware and thermal-aware synthesis and the industrial application of mature techniques.

REFERENCES

[1] Companion article in VLSI Handbook.

[2] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Company, NY, 1979.

[3] "International Technology Roadmap for Semiconductors," 2006, http://public.itrs.net.

[4] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw, "Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2002, pp. 721–725.

[5] K. A. Bowman, B. L. Austin, J. C. Eble, X. Tang, and J. D. Meindl, "A physical alpha-power law MOSFET model," *IEEE J. Solid-State Circuits*, vol. 34, pp. 1410–1414, Oct. 1999.

[6] Z. P. Gu, Y. Yang, J. Wang, R. P. Dick, and L. Shang, "TAPHS: Thermal-Aware Unified Physical-Level and High-Level Synthesis," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2006, pp. 879–885.

[7] W.-L. Hung, G. Link, Y. Xie, N. Vijaykrishnan, N. Dhanwada, and J. Conner, "Temperature-aware voltage islands architecting in system-on-chip design," in *Proc. Int. Conf. Computer Design*, Oct. 2005.

[8] N. K. Jha, "Low power system scheduling and synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2001, pp. 259–263.

[9] L. Goodby, A. Orailoglu, and P. M. Chau, "Microarchitecture synthesis of performance-constrained, low-power VLSI designs," in *Proc. Int. Conf. Computer Design*, Oct. 1994.

[10] A. Raghunathan and N. K. Jha, "An iterative improvement algorithm for low power data path synthesis," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 597–602.

[11] R. S. Martin and J. P. Knight, "Power profiler: Optimizing ASICs power consumption at the behavioral level," in *Proc. Design Automation Conf.*, June 1995.

[12] A. Raghunathan, N. K. Jha, and S. Dey, *High-level Power Analysis and Optimization*. Kluwer Academic Publishers, MA, 1997.

[13] J. Rabaey and M. Pedram, Eds., *Low Power Design Methodologies*. Kluwer Academic Publishers, MA, 1996.

[14] A. Agarwal, C. H. Kim, S. Mukhopadhyay, and K. Roy, "Leakage in nano-scale technologies: mechanisms, impact and design considerations," in *Proc. Design Automation Conf.*, June 2004, pp. 6–11.

[15] M. W. Hall, J. M. Anderson, S. P. Amarasinghe, B. R. Murphy, S.-W. Liao, E. Bugnion, and M. S. Lam, "Maximizing multiprocessor performance with the SUIF compiler," *IEEE Trans. Computers*, Dec. 1996.

[16] P. P. Chang, S. A. Mahlke, W. Y. Chen, N. J. Water, and W. mei W. Hwu, "IMPACT: An architectural framework for multiple-instruction-issue processors," in *Proc. Int. Symp. Computer Architecture*, May 1991.

[17] M. Johnson and R. K. Roy, "Optimal selection of supply voltages and level conversion during datapath scheduling under resource constraints," in *Proc. Int. Conf. Computer Design*, Oct. 1996, pp. 72–77.

[18] M. C. Johnson and K. Roy, "Datapath scheduling with multiple supply voltages and level converters," *ACM Trans. Design Automation Electronic Systems*, vol. 2, no. 3, pp. 227–248, 1997.

[19] J. Chang and M. Pedram, "Energy minimization using multiple supply voltages," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 1996, pp. 157–162.

[20] S. Raje and M. Sarrafzadeh, "Variable voltage scheduling," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 1995, pp. 9–14.

[21] A. Chatterjee and R. K. Roy, "Synthesis of low power linear DSP circuits using activity metrics," in *Proc. Int. Conf. VLSI Design*, Jan. 1994, pp. 261–264.

[22] A. P. Chandrakasan, M. Potkonjak, R. Mehra, J. Rabaey, and R. Brodersen, "Optimizing power using transformations," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 1, pp. 12–31, Jan. 1995.

[23] N. Kumar, S. Katkoori, L. Rader, and R. Vemuri, "Profile-driven behavioral synthesis for low power VLSI systems," *IEEE Design & Test of Computers*, vol. 13, no. 9, pp. 70–84, Sept. 1995.

[24] J. M. Chang and M. Pedram, "Register allocation and binding for low power," in *Proc. Design Automation Conf.*, June 1995.

[25] J. Chang and M. Pedram, "Module assignment for low power," in *Proc. European Design Automation Conf.*, Sept. 1996, pp. 376–381.

[26] A. Dasgupta and R. Karri, "Simultaneous scheduling and binding for power minimization during microarchitecture synthesis," in *Proc. Int. Symp. Low-Power Design*, Apr. 1994.

[27] M. Ercegovac, D. Kirovski, and M. Potkonjak, "Low-power behavioral synthesis optimization using multiple precision arithmetic," in *Proc. Design Automation Conf.*, June 1999, pp. 568–573.

[28] E. Musoll and J. Cortadella, "High-level synthesis techniques for reducing the activity of functional units," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 1995, pp. 99–104.

[29] A. Raghunathan and N. K. Jha, "SCALP: An iterative-improvement-based low-power data path synthesis system," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 11, pp. 1260–1277, Nov. 1997.

[30] K. S. Khouri, G. Lakshminarayana, and N. K. Jha, "High-level synthesis of low power control-flow intensive circuits," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1715–1729, Dec. 1999.

[31] D. W. Knapp, "Fasolt: A program for feedback-driven data-path optimization," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 11, no. 6, pp. 677–695, June 1992.

[32] J. P. Weng and A. C. Parker, "3D scheduling: High-level synthesis with floorplanning," in *Proc. Design Automation Conf.*, June 1992.

[33] Y. M. Fang and D. F. Wong, "Simultaneous functional-unit binding and floorplanning," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1994.

[34] W. E. Dougherty and D. E. Thomas, "Unifying behavioral synthesis and physical design," in *Proc. Design Automation Conf.*, June 2000.

[35] C.-G. Lyuh, T. Kim, and K.-W. Kim, "Coupling-aware high-level interconnect synthesis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 23, no. 1, pp. 157–164, Jan. 2004.

[36] L. Zhong and N. K. Jha, "Interconnect-aware low power high-level synthesis," *IEEE Trans. Computer-Aided Design of Inte-*

grated Circuits and Systems, vol. 24, no. 3, pp. 336–351, Mar. 2005.

[37] Z. P. Gu, J. Wang, R. P. Dick, and H. Zhou, "Incremental Exploration of the Combined Physical and Behavioral Design Space," in *Proc. Design Automation Conf.*, June 2005, pp. 208–213.

[38] K. S. Khouri and N. K. Jha, "Leakage power analysis and reduction during behavioral synthesis," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 10, no. 6, pp. 876–885, Dec. 2002.

[39] C. Gopalakrishnan and S. Katkoori, "KnapBind: An area-efficient binding algorithm for low-leakage datapaths," in *Proc. Int. Conf. Computer Design*, Oct. 2003, pp. 430–435.

[40] X. Tang, H. Zhou, and P. Banerjee, "Leakage power optimization with dual-Vth library in high-level synthesis," in *Proc. Design Automation Conf.*, June 2005, pp. 202–207.

[41] R. Mukherjee, S. O. Memik, and G. Memik, "Peak temperature control and leakage reduction during binding in high level synthesis," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2005, pp. 251–256.

[42] R. Mukherjee, S. O. Memik, and G. Memik, "Temperature-aware resource allocation and binding in high-level synthesis," in *Proc. Design Automation Conf.*, June 2005.

[43] Y. Yang, Z. P. Gu, C. Zhu, L. Shang, and R. P. Dick, "Adaptive Chip-Package Thermal Analysis for Synthesis and Design," in *Proc. Design, Automation, and Test in Europe*, Mar. 2006, pp. 844–849.

[44] "Embedded microprocessor benchmark consortium," http://www.eembc.org.

[45] R. P. Dick, "E3S: The embedded system synthesis benchmarks suite," E3S link at http://robertdick.org/tools.html.

[46] R. P. Dick and N. K. Jha, "MOGAC: A Multiobjective Genetic Algorithm for Hardware-Software Co-Synthesis of Distributed Embedded Systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 10, pp. 920–935, Oct. 1998.

[47] B. P. Dave, G. Lakshminarayana, and N. K. Jha, "COSYN: Hardware-software co-synthesis of heterogeneous distributed embedded systems," *IEEE Trans. VLSI Systems*, vol. 7, no. 1, pp. 92–104, Mar. 1999.

[48] L. Shang and N. K. Jha, "Hardware-software co-synthesis of low power real-time distributed embedded systems with dynamically reconfigurable FPGAs," in *Proc. Int. Conf. VLSI Design*, Jan. 2002, pp. 345–352.

[49] F. Gruian and K. Kuchcinski, "LEneS: Task scheduling for low-energy systems using variable supply voltage processors," in *Proc. Asia & South Pacific Design Automation Conf.*, Jan. 2001, pp. 449–455.

[50] D. Kirovski and M. Potkonjak, "System-level synthesis of low-power hard real-time systems," in *Proc. Design Automation Conf.*, June 1997, pp. 697–702.

[51] M. Schmitz and B. M. Al-Hashimi, "Considering power variations of DVS processing elements for energy minimization in distributed systems," in *Proc. Int. Symp. System Synthesis*, Nov. 2001.

[52] L. Yan, J. Luo, and N. K. Jha, "Combined dynamic voltage scaling and adaptive body biasing for heterogeneous distributed real-time embedded systems," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2003, pp. 30–37.

[53] J. Luo, L.-S. Peh, and N. K. Jha, "Simultaneous dynamic voltage scaling of processors and communication links in real-time distributed embedded systems," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 2003.

[54] G. Qu and M. Potkonjak, "Energy minimization with quality of service," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2000, pp. 43–49.

[55] R. A. Begamaschi, Y. Shin, N. Dhanwada, S. Bhattacharya, W. E. Dougherty, I. Nair, J. Darringer, and S. Paliwal, "SEAS: A system for early analysis of SoCs," in *Proc. Int. Conf. Hardware/Software Codesign and System Synthesis*, Oct. 2003, pp. 150–155.

[56] M. Lajolo, A. Raghunathan, S. Dey, L. Lavagno, and A. Sangiovanni-Vincentelli, "Efficient power estimation techniques for HW/SW systems," in *Proc. Alessandro Volta Memorial Wkshp. Low-Power Design*, Mar. 1999.

[57] T. Givargis, F. Vahid, and J. Henkel, "System-level exploration for Pareto-optimal configurations in parameterized systems-on-a-chip," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 2001, pp. 25–30.

[58] Y. Fei and N. K. Jha, "Functional partitioning for low power distributed systems of systems-on-a-chip," in *Proc. Int. Conf. VLSI Design*, Jan. 2002.

[59] I. Hong, D. Kirovski, G. Qu, M. Potkonjak, and M. B. Srivastava, "Power optimization of variable voltage core-based systems," *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 18, no. 12, pp. 1702–1714, Dec. 1999.

[60] R. P. Dick and N. K. Jha, "MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis," in *Proc. Design, Automation & Test in Europe Conf.*, Mar. 1999, pp. 263–270.

[61] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for hetereogeneous multiprocessor system-on-chip," in *Proc. Design Automation Conf.*, June 2001, pp. 518–523.

[62] J. Hu, Y. Deng, and R. Marculescu, "System-level point-to-point communication synthesis using floorplanning information," in *Proc. Int. Conf. VLSI Design*, Jan. 2002.

[63] N. Thepayasuwan, V. Damle, and A. Doboli, "Bus architecture synthesis for hardware-software co-design of deep submicron systems on chip," in *Proc. Int. Conf. Computer Design*, Jan. 2003.

[64] J. Hu, Y. Shin, N. Dhanwada, and R. Marculescu, "Architecting voltage islands in core-based system-on-a-chip designs," in *Proc. Int. Symp. Low Power Electronics & Design*, Aug. 2004, pp. 180–185.

[65] S. Pasricha, N. Dutt, and E. Bozorgzadeh, "Floorplan-aware automated synthesis of bus-based communication architectures," in *Proc. Design Automation Conf.*, June 2005.

[66] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. Design Automation Conf.*, June 2001, pp. 684–689.

[67] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: a power-performance simulator for interconnection networks," in *Proc. Int. Symp. Microarchitecture*, Nov. 2002, pp. 294–305.

[68] L. Shang, L.-S. Peh, A. Kumar, and N. K. Jha, "Thermal modeling, characterization and management of on-chip networks," in *Proc. Int. Symp. Microarchitecture*, Dec. 2004, pp. 67–80.