IPSJ Transactions on System LSI Design Methodology Vol.8 12–25 (Feb. 2015)

[DOI: 10.2197/ipsjtsldm.8.12]

Invited Paper

High-level Synthesis for Low-power Design

Zhiru Zhang[1,a)] Deming Chen[2,b)] Steve Dai[1,c)] Keith Campbell[2,d)]

Received: November 13, 2014, Released: February 12, 2015

Abstract: Power and energy efficiency have emerged as first-order design constraints across the computing spectrum from handheld devices to warehouse-sized datacenters. As the number of transistors continues to scale, effectively managing design complexity under stringent power constraints has become an imminent challenge of the IC industry. The manual process of power optimization in RTL design has been increasingly difficult, if not already unsustainable. Complexity scaling dictates that this process must be automated with robust analysis and synthesis algorithms at a higher level of abstraction. Along this line, high-level synthesis (HLS) is a promising technology to improve design productivity and enable new opportunities for power optimization for higher design quality. By allowing early access to the system architecture, high-level decisions during HLS can have a significant impact on the power and energy efficiency of the synthesized design. In this paper, we will discuss the recent research development of using HLS to effectively explore a multi-dimensional design space and derive low-power implementations. We provide an in-depth coverage of HLS low-power optimization techniques and synthesis algorithms proposed in the last decade. We will also describe the key power optimization challenges facing HLS today and outline potential opportunities in tackling these challenges.

Keywords: high-level synthesis, low-power design, algorithm, compiler optimization, hardware acceleration

1. Introduction

As modern system-on-a-chip (SoC) integrates billions of transistors and an increasing number of heterogeneous cores to meet the rapid scaling in application requirements, it has become important to manage the design complexity by raising the level of abstraction beyond the register-transfer level (RTL). Notably, electronic system-level (ESL) design methodology offers a very attractive option for enabling the next leap in productivity for integrated circuit (IC) design [35], [66], [100]. An ESL design flow allows users to effectively model the software/hardware components of the SoC system using high-level specifications and quickly converge on optimized low-level implementations via automated compilation and synthesis tools. For example, an analysis on C-based industrial system designs demonstrates the potential to achieve seven times decrease in code size and up to three orders of magnitude reduction in simulation and verification cycles [94].

Concurrently, the IC industry is undergoing a significant transition from performance-oriented design to power-constrained design, as power and energy efficiency are now first-order design constraints across the computing spectrum. In order to meet the stringent power requirements, designers often have to optimize the initial RTL by applying a variety of low-power techniques such as clock gating, power gating, and multiple voltage islands, where functional, structural, temporal, and spatial information must be jointly considered. The manual process in such a complex "four-dimensional" space has become increasingly difficult, if not impossible, to effectively optimize the RTL design within a short turnaround time. Additionally, power scaling requires designers to assess and optimize the system architecture as early as possible in the design flow [75]. As shown in **Fig. 1**, attempting to reduce power at the RT level typically has much less impact than at the behavioral and system levels with high-level decisions such as hardware/software partitioning, bus width sizing, scheduling, resource sharing, pipelining, etc. Clearly, raising the level of abstraction beyond RTL is crucial for achieving fast power closure.

In light of the escalating design complexity and power-limited

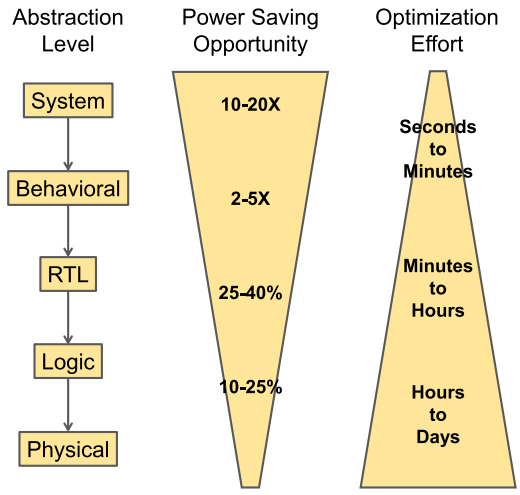


Fig. 1 Power saving opportunity and effort at different levels of abstraction derived from Ref. [81].

[1] School of Electrical and Computer Engineering, Cornell University, Ithaca, New York
[2] Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, Urbana, Illinois
a) zhiruz@cornell.edu
b) dchen@illinois.edu
c) stevedai@csl.cornell.edu
d) kacampb2@illinois.edu

scaling, high-level synthesis (HLS) [20], [27], [65] has emerged as a cornerstone of ESL design automation. HLS automatically transforms untimed behavioral description of an algorithm into optimized cycle-accurate hardware implementation in RTL. During the HLS flow, scheduling assigns each operation to a time step corresponding to a clock cycle. Resource allocation selects the numbers and types of hardware modules to be used, and resource binding assigns operations to these allocated modules. HLS extracts parallelism in the input behavioral description through control data flow analysis to synthesize the datapath and control logic of the hardware implementation. Because it enables automatic generation of optimized hardware from high-level programming languages and facilitates effective design space exploration of software and hardware architectures, HLS is a promising direction to significantly improve design productivity and at the same time address the increasing difficulty in meeting power constraints [100].

There has been a rich body of research on power optimization in HLS in the past two decades [10], [14], [22], [43], [45], [50], [54], [60], [80]. Some excellent surveys on the early efforts are given in Refs. [79], [81], [86]. In this paper, we focus on surveying the more recent developments of low-power HLS techniques that are proposed in the last decade. Inspired by the taxonomy proposed by Ranganathan for power-efficient computing in Ref. [83], we introduce a high-level categorization of the current and emerging low-power techniques and provide in-depth discussions on each category of techniques.

The rest of the paper is structured into three major sections. Section 2 provides an overview of general HLS methodologies for low-power design, followed by a detailed categorization of the important power optimizations, such as high-level techniques of scaling down power, applying low-power technologies, matching work to energy-efficient options, and performing cross-layer analysis. Section 3 demonstrates case studies of several representative low-power techniques in HLS, focusing on how to identify optimization opportunities and leverage them effectively at the high level. In particular, we describe an advanced compilation flow that targets CUDA on FPGA to effectively explore multi-level-granularity parallelism for performance and energy improvement. Finally, Section 4 addresses the outstanding challenges in low-power HLS and proposes some general directions for discovering additional power optimization options.

2. Optimizations for Low Power

Conventional optimizations for low-power design focus on cutting power at the circuit level. For example, designers often use technologies with different V_{th} threshold voltages to trade-off the performance of less timing-critical logic for reduced static leakage power [48]. Similarly, it would be useful to scale down the V_{DD} supply voltage to reduce the dynamic switching power on less performance-driven logic [37]. Other techniques such as power gating [44], [77], which turns off the voltage for the inactive part of the circuit, and clock gating [92], [96], which disables inactive registers, are widely implemented to achieve static and dynamic power savings, respectively. In general, the recurring theme in low-power design is to avoid waste by reducing unnecessary activities and slowing down over-provisioned resources that are running at a higher performance level than what is required by the workload [83].

The same concept of avoiding waste extends naturally into the domain of HLS, which presents a greater number and wider variety of optimization opportunities that are otherwise infeasible at the lower levels of design. With scheduling and binding, HLS generates cycle-accurate hardware circuit from untimed behavioral description. Scheduling, the process of assigning time step to each operation, can be performed intelligently, for example, to enable additional slacks for scaling down the power for more resources [47]. Binding, the process of assigning operations to hardware resources, can likewise be performed cleverly by assigning less timing-critical operations to low-power technologies such as modules with high V_{th} and low V_{DD} [12], [49]. The first-order task in low-power HLS is in leveraging the expanded design space in high-level scheduling and binding to identify and maximize the number of Pareto-optimal design points. We may consider a cross-layer approach that integrates upstream HLS with downstream physical implementation flow such as technology mapping and place-and-route to jointly realize additional power saving. Apart from lower level techniques, we may also transform algorithms and applications at the compiler level to achieve inherent high-level power efficiency such as improved memory locality and reduced runtime complexity [83]. We can also trade-off accuracy for low power and even spend additional power to save power [63], [68].

In this paper, we examine the low-power HLS optimization techniques in accordance with a high-level design methodology. Instead of a bottom-up approach that delves into specific circuit-level techniques or the types of power saving targeted by the optimizations, we establish a top-down view inspired by Ref. [83] that broadly categorizes the techniques based on their intuitive approaches to reducing inefficiency in the synthesized circuit. This allows us to emphasize high-level optimization strategies without losing sight of the low-level techniques. An overview of the categorization and a summary of relevant low-power optimization techniques under each category are shown in **Table 1**.

2.1 Using Low-power Technologies

Ideally, a low-power design should consume only the power that is absolutely necessary for meeting the design constraints and performance requirements. As a result, an intuitive approach is to use power-efficient instances of hardware modules. Binding operations to low-power resources such as high-V_{th} module, low-V_{DD} island, and power-efficient memory is a common technique for low-power HLS design. The process of statically binding to low-power technologies involves complex trade-offs between power and performance. Otherwise, the design would have been bound to the lowest-power resources in the first place. While HLS has the ability to freely explore the design space in terms of scheduling and binding, optimizations must be carefully crafted to balance the benefit of using low-power resources with any negative effect on performance and cost. As an example, by exploiting the timing slacks, defined as the amount of extra delay that an operation can tolerate without violating the given timing constraints,

Table 1 High-level categorization of current low-power optimization techniques.

Optimization Category	Relevant Techniques
Using low-power technologies	Multiple threshold voltage [15], [49], [91], multiple supply voltage [12], [39], [47], [59], non-volatile memory [57]
Scaling down the power for under-utilized resources	Clock gating [1], [45], power gating [14], [16], [18], [31], behavioral observability don't care [22], [23], soft constraints [24], dynamic voltage and frequency scaling [46], [64], [82]
Matching work to energy-efficient options	Heterogeneous substrates [2], [3], [78], [87], [99], C-like parallel programming targeting FPGA [29], [72], [97]
Cross-layer analysis	Interconnect optimization [11], [13], [43], floorplanning optimization [38], [88], memory optimization [7], [21], [61], [95], high-level/logic synthesis [28]
Trading-off other metrics for power	Approximate hardware synthesis from behavioral description [67], error-constrained bit-width optimization [26], [63], [70]
Spending power to save power	Resource over-provisioning for hotspot reduction [68], dynamic voltage and frequency scaling [46], [64], [82]

we can bind operations to low-power resources without incurring any significant performance degradation. Unfortunately, different operations often compete for the limited timing slack. It is therefore important to optimally budget the use of low-power resources based on the available slack, a problem generally known as the timing budgeting problem [9], [36], [58]. Jiang et al. [47] combines scheduling with timing budgeting to assign operations to the appropriate time step and bind them to the appropriate resource implementation to optimize slack distribution for low power.

Minimizing the leakage current is an important technique to reduce the static power consumption of the circuit. Because leakage current decreases exponentially with increase in V_{th}, a given technology can manipulate the substrate bias to raise the V_{th} as long as performance constraints are met. Advances in technology allows cells with different V_{th} values on the same chip where the lower V_{th} cells switch faster but suffer from high leakage while the higher V_{th} cells switch slower but have lower leakage. Selecting the module with the appropriate V_{th} given the design constraints is an interesting optimization problem in HLS. Khouri and Jha [49] develop a heuristic algorithm that greedily targets the frequently idle modules as candidates for high-V_{th} implementation, for which the selection considers the potential of the module for leakage reduction. Tang et al. [91] propose to first synthesize the design with low-V_{th} modules and then replace module instances for nodes with positive slacks with corresponding high-V_{th} implementations. Chen et al. [15] follow the same approach of slack-based module replacement to reduce leakage power, but also consider the reliability of the synthesized circuit in terms of negative bias temperature instability.

It is also possible to run different blocks of the chip with different V_{DD}'s. While modules with higher V_{DD} incur shorter delay, those with lower V_{DD} experience longer delay. Running the non-critical part of the logic in a low voltage island can greatly reduce both the dynamic and static power of the circuit because switching power and leakage power are proportional to V_{DD}^2 and V_{DD}, respectively. Chen et al. [12] present a network flow-based low-power resource binding algorithm that jointly considers both multiple-V_{DD} and switching activity. The objective is to bind all operations to functional units so that the number of operations with low V_{DD} is maximized and the total switching activities of functional units are minimized. Gu et al. [39] assume that operations are initially assigned customized voltages and propose the problem of partitioning these voltages into an appropriate number of voltage domains for a power-optimal design, referred to as the voltage partitioning problem. Liu et al. [59] address the NP-hard voltage partitioning problem with an approximation algorithm where the accuracy depends on the ratio of maximum to minimum initial voltages. The time complexity of the algorithm is on the order of the product among the numbers of voltage domains, initial voltages, and functional units.

Apart from the well-established low-power techniques, there is a recent trend in adopting emerging technologies such as non-volatile memory (NVM) for low-power design, examples of which include STT-RAM [52], CBRAM [53], and RRAM [19]. NVM has the potential to replace traditional volatile memory at different levels of the memory hierarchy and offers ultra-low standby leakage power, small area, and instant on/off capability. However, NVM typically incurs large write power and latency. It is therefore necessary to carefully synthesize the proper memory architecture under design constraints when utilizing NVM. HLS enables the optimization and synthesis of hybrid memory architectures that consider the trade-off between emerging NVM and traditional SRAM. Li et al. [57] propose an optimization that minimizes the power consumption under given memory bandwidth, chip area, and performance requirements for hybrid NVM and SRAM architectures by intelligently deciding memory allocations, memory hierarchy, and memory type.

HLS provides the high-level flexibility in synthesizing the proper hardware architecture that incorporates critical design decisions that are more difficult to explore at the lower levels. It is important to note that using low-power technologies in HLS should not be a spearheaded effort in indiscriminately applying low-power modules anywhere in the design. Instead, it must be a global optimization effort that examines the trade-off between performance and power consumption of available technologies and determines how, where, and when to apply low-power technologies. Leveraging these technologies may even necessitate algorithmic improvements. In the case of NVM, it is necessary to optimize the memory writes in the program to alleviate the large write power.

2.2 Scaling Down the Power for Under-utilized Resources

Apart from statically binding operations to low-power resources, HLS can also be leveraged to dynamically scale down the power for under-utilized resources. In particular, clock gating is a typical technique to turn off the inactive logic in the circuit. At a given point in time, it is often the case that a large amount

of logic is inactive, but the corresponding registers continue to be triggered by the clock even though the registered values are not changing. We can achieve significant saving in switching power of the register and the clock nets by inserting additional logic to disable the clock signals into these inactive registers. Ahuja et al. [1] discuss methods of applying clock gating at various levels of granularities at the behavioral level and propose priority for clock gating decisions. Huang et al. [45] address the area overhead of the clock gating logic and devise an optimization to minimize this overhead.

Power gating, on the other hand, turns off the inactive logic by shutting off the power to the corresponding block of the circuit, which further minimizes the standby static power consumption. Dal and Mansouri [31] develop a synthesis technique that binds operations with maximally overlapping lifetimes to the same power island, which consists of a cluster of logic whose power can be shut off independently from the rest of the circuit. Choi et al. [18] and Chen et al. [14], [16] tackle the inherent overhead in state-retention storage required to preserve the logic states during power gating cycles and develop scheduling and binding algorithms to minimize the total size of required retention storage.

To enable the benefits of operation gating in HLS, the concept of observability don't care (ODC) has been generalized at the behavioral level to identify unnecessary computations and guide the scheduler in maximizing the efficiency of clock gating [22], [23]. Soft constraints [24] can be applied within the scheduler to promote gating opportunity. We will provide a more detailed discussion on behavioral ODC in Section 3.1.

In addition to turning off inactive logic, it is also possible to slow down some of the operations for power reduction by applying dynamic voltage and frequency scaling (DVFS), which varies the operating voltage and frequency of the circuit over time based on the performance demand of the design [93]. Many scheduling techniques [46], [64], [82] have been developed to take advantage of DVFS to achieve power reduction. Overall, we note that static analysis at the high level reveals runtime power optimization opportunities that are much more difficult to identify at the lower levels. Unlike RTL at which the clock boundaries are fixed, HLS has the freedom to define the hardware structure (datapath) as well as the schedule (control) to maximize power scaling for under-utilized resources.

2.3 Matching Work to Energy-efficient Options

While it is desirable to leverage HLS to consider resource optimizations at the device level, we must step back and examine system-level possibilities. As power scaling saturates performance for single-core and multi-core systems, we are encountering a shift toward heterogeneous multiprocessor systems that take advantage of the unique characteristics of different types of computing substrates and combine them synergistically to boost performance per unit of power. Examples of heterogeneous platforms include CPU-FPGA [2], [3], [99] and CPU-GPU-FPGA [87] systems for different levels of computation needs. More recently, Microsoft has integrated FPGAs in its datacenters for Bing search engine's page rank processing and reported 95% increase in throughput with only 10% power overhead [78]. It is evident that FPGA has the potential of becoming an integral part of these future heterogeneous systems and is in fact an attractive option for energy-efficient computing. FPGA is fully customizable for specialized applications and excels at dataflow-intensive designs that exploit both fine-grained and coarse-grained parallelism [42]. Unfortunately, the availability and power advantages of heterogeneous systems are usually overshadowed by the difficulty in programming them. The lack of a common programming model among CPU, GPU, FPGA, and other emerging processors complicates the heterogeneous development process and hinders widespread adoption. A high-level design methodology has the potential to unify the development effort on heterogeneous processors and enable new compilation techniques and synthesis optimizations to simultaneously extract the benefits of each substrate.

In particular, there are multiple lines of effort in targeting C-like parallel programming languages to CPU-FPGA [29], [72], [97]. This effort aims to enable cross-compilation of different parts of a program written in a single unified programming language onto the most suitable substrates based on the characteristics of the corresponding workloads. Papakonstantinou et al. [72] adapt the CUDA programming model [69] into a new FPGA design flow to enable efficient compilation of CUDA kernels onto FPGA. Known as FCUDA, this CUDA-to-FPGA flow efficiently maps coarse and fine-grained parallelism expressed in CUDA kernels onto FPGA's reconfigurable fabric. In Section 3.4, we will provide a detailed case study of FCUDA with insights on the suitability of CUDA for FPGA and the power advantages of such high-level framework. Similarly, there is parallel effort in industry in targeting OpenCL [51] at FPGA. There are multiple developments on a new heterogeneous parallel programming and compilation environment for OpenCL on a CPU/FPGA system in which the host code is compiled onto the CPU and kernels are accelerated on the FPGA [29], [97]. With optimizations such as memory coalescing and kernel pipelining, such OpenCL compilers have the potential to greatly improve performance per unit power over homogeneous substrates.

2.4 Cross-layer Analysis

Rather than looking for optimization opportunities locally at a specific stage of the synthesis flow, examining the optimization problem holistically provides new dimensions to the solution space. For example, Zheng et al. [101] propose a place-and-route directed HLS flow that iterates among scheduling, binding, and physical implementation, while Tan et al. [89] develop a technology mapping-aware scheduling algorithm. Similarly in the context of power optimization, inferring low-level physical implementation details at the high level can effectively reduce power consumption even if the high-level estimates are not perfectly accurate. Such cross-layer analysis may consider logic synthesis, technology mapping, place-and-route, and memory synthesis jointly with HLS so the upstream HLS synthesized design is friendly for downstream physical implementation and optimized for both the front-end and back-end flows.

Much of the work in cross-layer analysis focuses on optimizing interconnect power. As an example, studies show that FPGA's in-

terconnects contribute to more than 70% of total power consumption [56]. As a result, low-power HLS techniques tackle resource allocation and binding because they are the key steps in determining the interconnects. High-level power estimation is often used to guide the selection of binding/allocation solutions. Chen et al. [11], [13] extend the method in Ref. [8] to efficiently calculate switching activities using high-level CDFG simulation and use switching activities to estimate power consumption. Based on a glitch-aware technology mapper [17], Cromar et al. [28] further consider glitches, spurious switching activities, in their power estimation for additional power reduction during binding. We will present a detailed case study of this work in Section 3.2. Furthermore, Hsieh et al. [43] propose to activate unused flip-flops to block the propagation of glitches to reduce glitch power on interconnects associated with the output of functional units in a design. Stammermann et al. [88] and Gu et al. [38] devise optimization algorithms to simultaneously perform floorplanning and binding.

Memory is another important source of power consumption, and memory bandwidth is often the bottleneck for optimal designing of performance and power. Memory partitioning is a popular technique for power optimization. Lyuh and Kim [61] and Wang and Hu [95] present algorithms to reduce power consumed by memory accesses by leveraging memory banking and exploiting multiple memory operating modes. The algorithm consists of simultaneous memory access scheduling/binding and operating mode assignment to assign discrete memory operations to different banks for performance and power optimization. Benini et al. [7] propose an algorithm to optimally synthesize a multi-bank on-chip SRAM architecture for power reduction based on the dynamic execution profile of memory access frequencies. They argue that mapping the most frequently accessed addresses onto on-chip memory guarantees power efficiency. Cong et al. [21] further propose a technique that statically calculates memory access frequencies in polynomial time. This technique requires little to no profiling and can support more partitioning schemes.

It is important to consider low-level physical information during high-level design [85] to enable more holistic power optimizations. By adopting techniques that perform cross-layer analysis that examines or integrates different steps of the design flow, we can significantly reduce the turnaround time for achieving power closure while maintaining a high-level design methodology.

2.5 Trading-off Other Metrics for Power

No matter how much effort we put in to avoid waste by exploring a large variety of low-power options, there is always a power floor, the bare minimum amount of power, that must be consumed for the given application specifications and technology constraints. Moving below the power floor requires intelligently compromising the functionality of the application, often in the form of accuracy.

Mallik et al. [63] propose an error-constrained optimization during synthesis that trade-off quantization error with power. The optimization intelligently allocates bit-width using profiling and high-level models to estimate quantization errors and power consumption, respectively. Constantinides et al. [26] devise a procedure to automatically analyze the sensitivity of outputs to rounding of internal variables for assigning bit-widths. Osborne et al. [70] further applies clock gating to vary bit-width dynamically at runtime based on branch statistics. Nepal et al. [67] extend the approximation beyond data type and bitwidth and develop a tool called ABACUS to synthesize approximate hardware circuits from behavioral descriptions. The algorithm produces variants on the abstract syntax tree of the behavioral description by performing a range of approximate transformations, including data type simplification, operation approximation, and arithmetic substitution. Similar to other low-power design techniques described earlier, working at the high level of abstraction provides a wider range of approximation solutions to efficiently identify the optimal inexact circuits that represent the Pareto trade-off between power and accuracy, thus performing global transformations on the program whose power optimization effects are not limited by lower-level implementations.

2.6 Spending Power to Save Power

It is common for synthesis to incur additional cost in one area of the design to realize significant improvement in another area of the design. In performance-centric HLS, for example, we pay some additional cost in area and power for synthesizing additional hardware logic to achieve significant improvement in overall design throughput [30], [90]. Similarly in low-power optimization, we pay the additional cost of dynamic power management hardware to conserve the overall power of the system through dynamic voltage and frequency scaling.

On the other hand, power consumption is not necessarily a direct function of the number of resources that are used, but also the way that power is distributed among these resources, which determines the dynamic operating conditions of the synthesized circuit. In particular, the temperature profile of the chip has a strong influence on the leakage power, a phenomenon known as thermal-induced leakage [32]. The intuitive notion that the minimum number of resources consumes the least amount of power may not be always applicable. In fact, Ni and Memik [68] find that over-provisioning resources allows for more optimal distribution of power density, reducing hotspots and subsequently minimizing leakage power. They propose a resource allocation and binding technique that allows certain degree of redundancy to achieve a low-power design considering thermal-induced leakage.

3. Case Studies

Low-power design requires that the system architecture be optimized as early as possible in the design flow [75]. In this section, we use several case studies to elaborate on how we can capitalize on the high-level observations and approaches described in Section 2 to reduce inefficiency in the synthesized design. In particular, we illustrate examples of analysis that identify optimization opportunities in the design and act on the results of the analysis. Approaches include algorithmic improvement in the synthesis engine and programmer-directed optimizations in the source design. Techniques involve mathematical modeling of the design, which provide a systematic method to globally optimize

design preferences, an important objective of which is power. We will show that although gate-level details are considered in some of these techniques, high-level power optimization trumps gate-level power analysis in efficiency and effectiveness.

3.1 Behavioral Observability Don't Care

Clock gating is an important technique in RTL synthesis for scaling down the power for under-utilized resources, and a simple way to identify clock gating opportunity is based on stability conditions [33]. The value stored in a register may not change from cycle to cycle. Even if it changes, the value may not need to be propagated down the pipeline if the downstream circuit does not subsequently use the result. This condition of observability don't care (ODC) dictates that significant power saving can be achieved by clock-gating the unobservable branches of the circuit without affecting the desired functionality [4].

To exploit this benefit in HLS, the concept of ODC has been generalized at the behavioral level to identify unnecessary computations and guide the scheduler in maximizing the efficiency of clock gating [22]. While behavioral-level ODC helps alleviate the otherwise intensive designer input needed to identify clock gating candidates, it also reveals clock gating opportunities not visible at the RTL level. For example, the algorithm in **Fig. 2** (a) can be correctly implemented with the architecture in Fig. 2 (b). However, none of the registers is able to take advantage of ODC-based clock gating because the comparison is scheduled after the multiplication, and the observability condition is not determined before the registers. In contrast, if the same algorithm is implemented with the architecture in **Fig. 3** by scheduling the comparison before the multiplication, the two registers and the multiplier on the B path can be safely ignored when the comparison selects the A path at the multiplexer. In this case, the B path with the multiplier is unobservable, and the two registers on the B path can be clock-gated, from which the multiplier is also data-gated because its inputs will remain stable. It is important to note that although the architecture in Fig. 3 requires two extra 1-bit registers to pipeline the result of the comparison, the power overhead from these two small 1-bit registers is negligible compared to the power reduction achieved from clock-gating the two large 32-bit registers.

```
int DUT(int A, int B)
{
    int B2 = B*B;
    bool c = A>0;
    int r = c ? A:B2;
    return r;
}
```

(a)

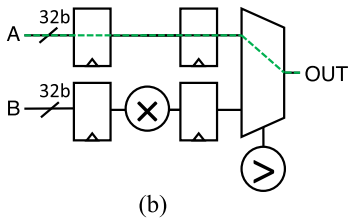


Fig. 2 (a) C code of a simple example design. (b) A hardware implementation of the design which cannot be clock-gated because the observability condition is scheduled after the multiplication.

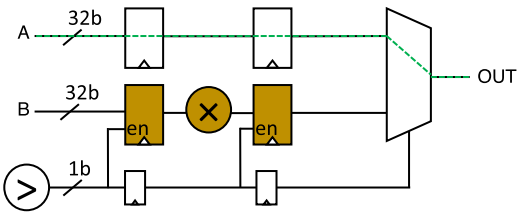


Fig. 3 Clock gating of the registers on the B path is possible when the observability condition is scheduled before the multiplication.

The above example shows that clock gating may not be applicable on some architectures among many alternatives of the same functionality. At the RTL level, the hardware architecture has been fixed, which limits the amount of clock gating that can be performed. Because HLS performs scheduling to synthesize the architecture of choice for the design, HLS provides the freedom to choose the right schedule and a good architecture that maximize clock gating opportunity. Comparing the architecture in Fig. 3 with that in Fig. 2 (b), scheduling the comparison before the multiplication allows more registers to be clock-gated. At the same time, it is evident that a basic strategy for operation gating is to evaluate the gating conditions earlier in the design. In other words, it requires a scheduling optimization that creates a favorable schedule in which the gating condition c is executed prior to another operation v whose observability depends on the gating condition. In this context, it is natural to extend the mathematical-programming-based SDC scheduling formulation [25] for operation gating.

With the SDC scheduling, an integer-valued scheduling variable s_v is used to represent the time step for which operation v is executed. A system of integer-difference constraints in the form of $s_u - s_v \leq d$, where d is a constant integer, encapsulates a range of scheduling constraints such as dependency, latency, and clock period, and optimizes the schedule for all the operations. In the context of operation gating, additional soft constraints [24] are introduced in the form of $s_c - s_v - w \leq d$, where w is a penalty variable used to encourage gating condition c to be executed prior to the operation v. Unlike hard constraints that must be obeyed for the correctness of the design, low power is a soft constraint treated as design preference and will be honored whenever possible given the hard constraints. Reference [22] relates the penalty in the soft constraint to the estimated power dissipation of the operation in question. Minimizing the penalty thus achieves maximum power saving.

3.2 Glitch Reduction

The switching activity of a circuit can be divided into two categories: functional transitions and glitches. Functional transitions are required for the circuit to function correctly while glitches represent useless logic computations resulting from imbalances in timing before the outputs become stabilized. Thus glitches clearly represent wasted power, but accurately characterizing them requires low-level (gate netlist level) analysis. At the same time, the opportunities for binding optimizations that can reduce glitch power are greatest in high-level synthesis. Thus a cross layer approach is required.

HLPower [28] is such a cross-layer approach, targeting glitch power reduction through glitch-aware binding that minimizes switching activity. The HLPower binding engine does register binding first and functional unit binding second. The register binding is done with a conventional algorithm. The functional unit binding algorithm is customized, as shown in **Fig. 4**, to reduce glitch power. First, for each operation type, a control step

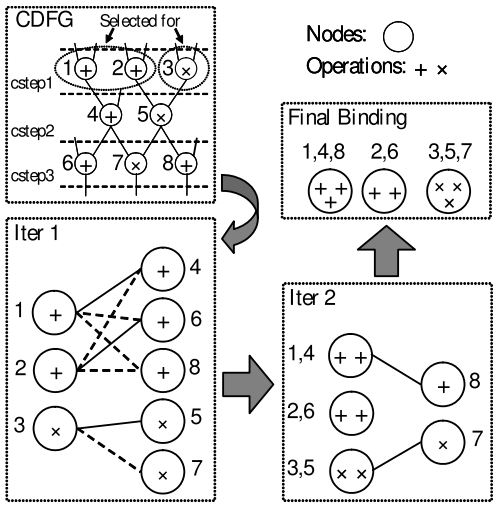


Fig. 4 The HLPower binding algorithm.

that uses the most of that resource is identified. Next, the corresponding resource contraint defining operations are selected as the base set for bipartite matching. A bipartite graph is then constructed from the base operation set to the remaining operations. Edges are created for each operation pair that can be shared.

For each potential pairing, a partial technology-mapped netlist representing that binding assignment is constructed, including any multiplexer that needs to be inserted for sharing. Switching activity is computed for the partial netlist using node-level transition density and switching activity analysis. The switching activity is then used, in addition to a mux size metric, to weigh each edge to direct the bipartite matching. The optimal pairs are grouped together as shown in Fig. 4 and then iteratively matched again with the remaining operation. This process is repeated until all operations have been assigned a sharing group or no feasible pairings remain. The binding is now complete as each group represents a physical functional unit that the member operations are bound to. Experimental results show a decrease in dynamic power of 22% over a previous approach that does not consider glitches.

3.3 Polyhedral Optimization

Listing 1: Nested loop amenable to polyhedral modeling.

```
for (a = 0; a < 15; a++)
    for (b = 0; b <= a; b++)
        X[a][b] = f(X[a][b+1], X[a+1][b]);
```

Profiling an application often reveals a few "hot" instructions that are taking the majority of the energy, typically in the body of a nested loop. Thus optimizing such nested loops can lead to substantial overall energy savings. When the nested loops involve very regular array access patterns (e.g., matrix multiply), an analytical model known as the polyhedral model can be used to model the iteration and memory access space of the nested loop. The model can then be used to perform transformations which reorder the iterations for improved locality and even expose parallelism to perform effective performance/power design trade-offs.

The polyhedral model is based on linear algebra, enabling analytical solutions to such optimization problems. A basic example of a nested loop amenable to polyhedral modeling is shown in Listing 1. Each inner loop iteration is represented with a vector $\vec{i}$ (equivalent to (a, b) in the example) normalized such that $\vec{i} \geq \vec{0}$. We can model the loop bounds constraints as system of linear inequalities:

$$\mathbf{A}\vec{i} \leq \vec{b} \tag{1}$$

where $\mathbf{A}$ is a coefficient matrix and $\vec{b}$ is a constant vector. We can also map each array access in each iteration to an array index vector $\vec{s}$ through a linear function with coefficient matrix $\mathbf{M}$ and constant offset $\vec{o}$:

$$\vec{s} = \mathbf{M}\vec{i} + \vec{o} \tag{2}$$

Each coefficient matrix $\mathbf{M}$ provides information about inter-iteration dependencies that must be satisfied. Through the application of a transformation matrix $\mathbf{T}$, we can now project the iteration domain defined by Eq. (1) into a new space resulting in a reordering of the memory accesses defined by Eq. (2):

$$\vec{i'} = \mathbf{T}\vec{i} \tag{3}$$

$$\vec{s'} = \mathbf{M}\mathbf{T}^{-1}\vec{i'} + \vec{o} \tag{4}$$

The physical result of this transformation can take many forms, ranging from loop iteration reversal to loop header permutations to more advanced transformations such as diagonal traversal (skewing). Furthermore, these transformations can easily be combined through matrix multiplication. There are two primary benefits to performing such transforms:

- Memory accesses can be reordered to improve locality and thus increase data reuse.
- By making transformations such that inner loop iterations do not depend on each other, iterations of the inner loop(s) can be parallelized.

Finding the right transformations to get a particular memory access pattern or to increase parallelism has now been reduced to a linear algebra problem. When high-level synthesis is extended with polyhedral optimization, as one might imagine, new optimization opportunities present themselves that can result in substantial energy savings.

3.3.1 Retuning a Polyhedral Optimizer for HLS

In one study [102], a source-to-source polyhedral optimization engine called CLooG [5], tuned for targeting software compilers for CPUs, is retuned to target the Vivado HLS engine for FPGAs [98]. Three modifications and extensions to the engine are made:

- **Turn off polyhedra seperation**: Polyhedra seperation fragments the iteration space where appropriate to avoid computation masking "if" statements in inner loops, which can be problematic for branch predictors on CPUs. Such masking "if" statements are not a problem for the hardware generated by HLS, so turning off this feature actually improves the quality of results due to the reduced loop complexity.
- **Floating-point bounds computation strength reduction**: Polyhedral transformations often result in a new iteration

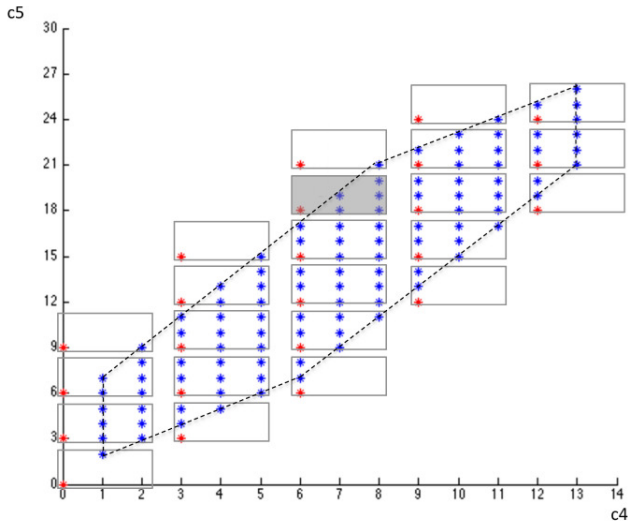


Fig. 5 Illustration of tiling for odd-shaped iteration spaces. Red points are tile origins and blue points are unmasked iteration points within each tile. An exemplary tile that straddles the iteration space border is highlighted in grey.

domain that involves complex loop limit computations with floor and ceiling functions, divisions, and min and max functions which are performed as floating point computations. By transforming these loop limit equations, computation strength can be dramatically reduced: Ceiling and floor functions with divisions inside can be reduced to integer divisions; integer divisions in equations can be replaced with multiplications with the use of a little algebra; and finally nested min and max function trees can be balanced for more parallelism. Further savings can be obtained through bitwidth reduction of the loop index, which can safely be performed if the compiler knows the limits of each loop index variable.

- **Tiling with masking**: Partitioning the iteration space into tiles that are small enough to fit into registers exposes enormous data reuse and parallelism opportunities to the Vivado HLS engine. Unfortunately, the optimal transformed iteration domain often has an odd shape (e.g., a parallelogram due to a diagonal memory access pattern) which is difficult to tile. The solution, illustrated in **Fig. 5**, is to use regular tiles to cover the entire shape and for those tiles on the boundary, suitable empty iterations will be introduced to mask out unnecessary iterations outside of the boundary. This would greatly simplify boundary conditions and improve parallelism for hardware generation.

The net result of applying all three modifications is a 60% reduction in energy consumption, a 55% reduction in area, and a 18% reduction in latency, a triple win [102]!

3.3.2 Enabling Inter-loop Pipelining

In another study [103], the PoCC polyhedral compiler [76] is extended to enable the Vivado HLS engine [98] to pipeline one nested loop's output into another nested loop's input where Vivado was previously unable due to incompatibilities between the two memory access patterns. With the polyhedral model, transforming the two loops such that their memory access patterns are compatible for pipelining is simply a matter of finding an iteration space to memory access space mapping matrix **M** from Eq. (2) that results in feasible transformed iteration domains in both nested loops. In this study, row access, column access, and diagonal access pattern matrices are tried and the best is chosen. Loop unrolling and pipelining directives are then added to the output code to direct Vivado to take advantage of the newly exposed parallelism.

The result is a 29.6× speedup on average over the unoptimized source code and a 6× speedup over code with the same Vivado optimization directives added, but without the polyhedral transforms needed to make some of the Vivado optimizations work. If Ref. [102] is any indication, significant energy savings are also likely with these optimizations.

3.3.3 Lessons

This case study provides two insights into how to think about power reduction:

(1) **Power reduction can be a win-win**: With so many untapped optimization opportunities at the behavioral level, one should be wary of considering area, latency, and power optimization to be a zero-sum game. All three can win simultaneously when complexity is reduced by using the right model.

(2) **Strength reduction is a power saver**: Think about the hardware that must be *synthesized* to execute an instruction. Where possible, reduce floating point to fixed point to integer computation, reduce division to multiplication to shifting, etc. Be aware of how programming languages can, by their semantics, limit the strength reductions the compiler can perform.

3.4 FCUDA

As we approach the point of diminishing returns for single-threaded performance and power efficiency optimization, increasingly the programmer needs to be in the loop to identify parallelism opportunities that compilers can exploit to increase performance per watt. Parallel extensions to the C programming language such as OpenMP and MPI have emerged to meet this challenge through a single level of parallelism. Taking things to the next level, the CUDA and OpenCL programming languages give the programmer the power to explicitly express hierarchical and multidimensional spatial parallelism, grouping threads into multiple instances of "blocks" or "workgroups" with multidimensional indices to express massive parallelism on the order of thousands of threads.

At the same time, in data centers where getting the most performance out of a given amount of rack space is paramount, computational *density* per watt has started to become the key limiting factor due to skyrocketing power demands and thermal dissipation limits. This is the domain where specialized hardware such as ASICs and FPGAs win over GPUs and CPUs, in many cases by multiple orders of magnitude.

Thus an interesting study would try to find the best of both worlds: take massively parallel applications implemented in massively parallel programming languages and figure out how to map them to ASICs or FPGAs. FCUDA [40], [41], [71], [72], [73], [74] is one such study.

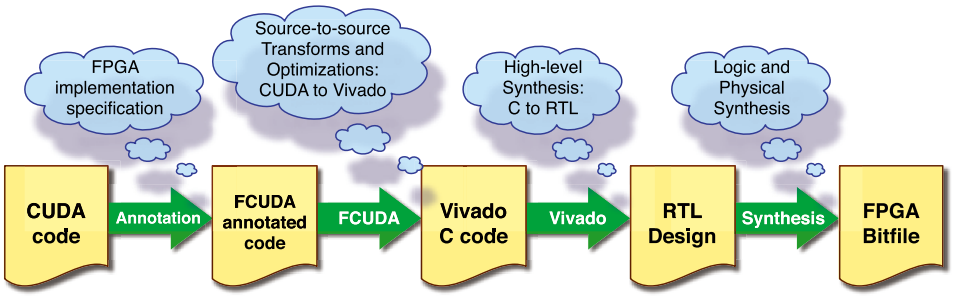


Fig. 6 The FCUDA compilation process.

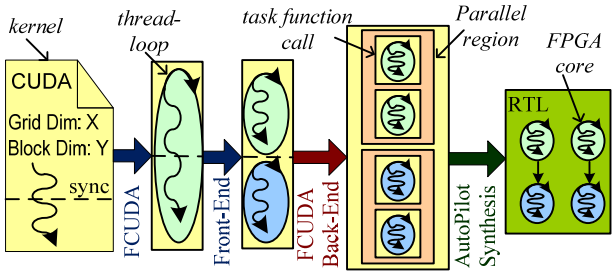


Fig. 7 Translating CUDA parallelism to FPGA hardware parallelism.

The FCUDA compilation process is illustrated in **Fig. 6**. The input to the compiler is kernels written in CUDA with programmer specified annotations to guide and parameterize the FPGA implementation. The annotated CUDA code is then run through a source-to-source transformation engine which creates explicit thread and block loops to transform the CUDA code into equivalent high-level synthesis friendly C code. Several transformations and optimizations are performed at this stage, including separating memory access from computation to effectively optimize each part, scratch pad memory partitioning and main memory access coalescing, common subexpression elimination, loop fission about CUDA barriers to maintain correctness, and the insertion of parallel annotations and loop unrolling directives.

These annotations then direct the Vivado HLS engine [98] to generate a hierarchically parallel hardware design which resembles the hierarchical parallelism of the CUDA model: inner loop unrolling results in multiple threads running in each core with shared logic and scratchpad memory; the cores are grouped into core-clusters with each cluster sharing a common main memory data communication interface; finally, the core clusters are then physically tiled and connected to a memory interface to complete the design.

Figure 7 provides some detail about how CUDA parallelism is translated to hardware parallelism for each thread block. First a loop over all threads is created. Next, loop fission is performed across FCUDA barriers to maintain correctness. Loop unrolling is then performed to enable groups of threads to execute in parallel. The result after synthesis is then multiple synchronized cores that are specialized to execute a thread block.

The resulting hardware design uses 5–16% of the energy as the equivalent GPU implementation with competitive latency [74]. Further latency and energy savings can be achieved by reducing the computation word bitwidth from 32 to 16 or even 8 bits. Such savings are much more difficult to achieve on a comparable GPGPU platform that is typically optimized for a single bitwidth.

3.4.1 Extensions

As one might imagine, power reduction opportunities abound when an embarrassingly parallel GPGPU programming model meets the highly flexible FPGA platform. Several FCUDA extensions have been developed to explore some of this optimization potential:

- **Design space exploration**: The hierarchically parallel nature of the CUDA input specification presents the FCUDA compiler with many design choices. The Multilevel Granularity Parallelism Synthesis (ML-GPS) framework [74] is an FCUDA extension that searches four dimensions of this design space: thread coarsening granularity, scratchpad memory partitions, cores per core-cluster, and core clusters per accelerator to find a performance optimized solution. The framework uses Vivado [98] as an accurate back-end quality-of-results estimating tool, but minimizes the number of Vivado invocations through regression analysis from a few key datapoints and through the use of a multidimensional binary-search algorithm. The binary search algorithm leverages the convex nature of the array partition and unroll factor search space to find the minimum latency point.
- **Throughput optimization**: In the original FCUDA compiler, the programmer shows the compiler how to separate computation from memory accesses. For some kernels with interleaved memory and computation, this seperation is a job better suited to the compiler. The Throughput Oriented Performance Porting (TOPP) framework [71] is an FCUDA extension that performs deeper code analysis and transformations to optimize throughput and act as an automated way to insert FCUDA annotations. The framework works by lowering the input CUDA code to a Hierarchical Region Graph (HRG) of AST nodes that capture control flow, data dependencies, and thread invariance with leaf nodes identified as computations, memory accesses, or synchronizations. With this intermediate representation, the TOPP engine can perform code motions to remove synchronization points and move memory accesses closer to where they are used. As a bonus, this shortening of variable lifetimes enables a graph coloring algorithm to reduce scratchpad memory usage through sharing.
- **Pipelining multiple kernels**: Taking things to the next level, Gurumani et al. [40] take a stereo matching algorithm implemented with 13 CUDA kernels along with host code and map it to a single FPGA using FCUDA to produce kernel building blocks. The kernel pipeline is shown in **Fig. 8**. With all the

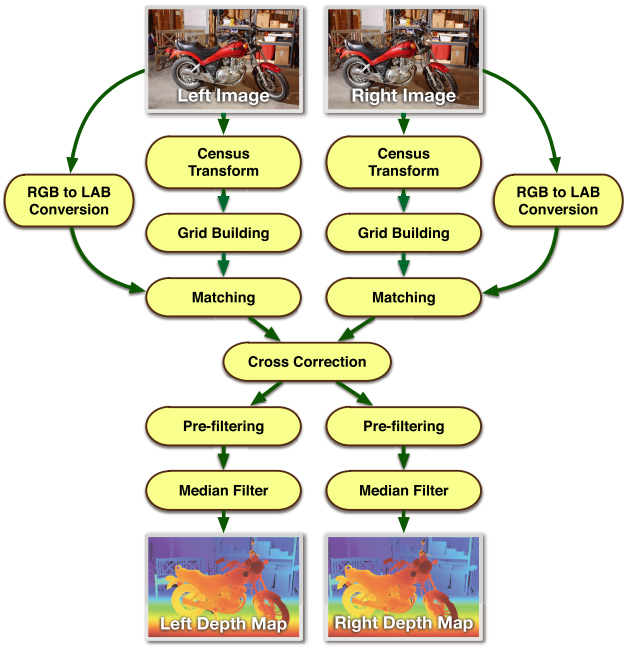


Fig. 8 Stereo matching CUDA kernel pipeline from Ref. [40]. Sample images courtesy of Ref. [84].

kernels being mapped to the same FPGA, each kernel, each with its own design space, is now competing with the other kernels for the same resources, greatly complicating the design space search problem. Furthermore, buffers need to be inserted between dependent kernels to efficiently stream data, but choosing each buffer size and the communication quanta is a difficult latency/area trade-off that can affect both upstream and downstream kernels. To achieve an efficient implementation, application specific knowledge as well as an analytical model is used to find an optimal solution.

- **Sharing data with a network-on-chip**: Gurumani et al. [41] propose the use of a Network-on-chip architecture to enable cores to communicate with each other to distribute scratchpad memory pressure and reduce main memory accesses. Their architecture is a 2D mesh of FCUDA cores. Each core has its own memory and its own router to connect it to the network. Each router contains a directory to keep track of where each memory location is stored in the network (if any). Thus if one core has data another needs, the data can be forwarded to avoid a high-latency, power-intensive main memory access. Due to the lack of memory coherence among thread blocks in the CUDA programming model, the directory can safely ignore writes, greatly reducing the design complexity and thus area and power overheads.

3.4.2 Lessons

This case study teaches some important lessons about effective power reduction:

(1) **Keep the programmer in the loop**: Where it is easy for the programmer to provide help, compiler designers should consider taking advantage of such inputs. With programmer annotations for parallelism and data movements, the original FCUDA compiler is able to reduce the waste resulting from worst case assumptions. The use of application specific knowledge in the kernel pipelining study also suggests the programmer can greatly help the compiler automate multi-kernel FPGA mapping.

(2) **Decouple computation from memory access**: Entangled memory accesses and computation make power optimization difficult. Considering each part separately is a good technique for decomposing the power optimization problem.

(3) **Optimize memory accesses first**: Memory accesses can consume more power than computation. Intelligently partitioning scratchpad memories into banks, coalescing main memory accesses into bursts, and sharing on-chip memory help to reduce wasteful energy "spending" on memory accesses.

(4) **Avoid redundant computation**: When targeting a GPU, having all threads redundantly compute the same result can make sense if the synchronization overhead would be greater. But with the high flexibility of the FPGA platform, this waste can be identified and avoided with common subexpression elimination and by identifying thread invariant instructions.

4. Future Directions

Considering the existing state-of-the-art in low-power design and high-level synthesis, we have some thoughts about future directions that have great potential:

(1) **Effective power modeling and analysis**: Effective high-level power optimization requires accurate performance, power, and area (PPA) models, but these models can be difficult to create at the high level due to the many abstraction layers in the design flow and the intricate interactions between these layers. We believe that it is important to build models of the common HLS building blocks (e.g., functional units, memories, interconnects, etc.) that are parameterized by configuration parameters and timing constraints and produce area and energy estimates at each level of abstraction. Such PPA estimates need to be propagated efficiently up the design flow and applied correctly in the high-level framework to achieve power benefits. The need for such robust parameterized power modeling is already evident in the GPU domain so architects can quickly co-optimize performance and power [55]. It is natural for HLS, which enables high-level exploration of a large design space, to soon adopt this trend. It is also crucial to continue advancing the research on algorithmically integrating scheduling and binding in HLS with key logic synthesis and physical planning techniques such as technology mapping and place-and-route to enable effective high-level power analysis tools that predict power consumption, identify power hot spots, and provide early prescriptive feedback to the designers.

(2) **Advanced power management**: Increasing complexity also calls for greater flexibility in the synthesized hardware. In the context of low-power design, new HLS techniques are needed in synthesizing dynamic yet lightweight power management units alongside the main design to monitor real-time power and workload and adapt to the characteristic of the execution at runtime. Similar to the power management micro-controller used in advanced multicore pro-

cessors (e.g., Refs. [34], [62]), the synthesized power management hardware may incorporate predictive modeling to anticipate performance and power requirements and apply intelligent control algorithms to minimize power consumption based on the analysis. How to carefully balance the area/performance overhead introduced by power management and the potential power saving is a very interesting research problem.

(3) **Better HLS programming languages**: Programming languages define a contract between the programmer and the compiler and determine the division of labor between the two. Most existing programming languages for hardware design through HLS are based on C, which requires the programmer to do a lot of micromanagement to effectively control the computation. This micromanagement as well as the Von Neumann based C programming model itself may limit the HLS engine's ability to do optimizations due to worst case assumptions and memory model assumptions. Better programming languages are needed that allow the programmer to do what humans do best: specify concepts, algorithms, and high-level organization and free the HLS engine to do what machines do best: the grunt work of platform mapping, scheduling, optimization, and design space exploration.

One way to gain insight into better language design is to analyze a sampling of code at all levels of granularity from the token to the module level and distinguish the components that do useful work from those that are only there to meet the requirements of the language. When some of the not-so-useful code gets synthesized as hardware, it represents wasted energy that could be saved through better language design. Language designers need to consider how easily a programmer can understand the semantics as well as whether the language provides sufficient information for the compiler to efficiently target heterogenous platforms.

(4) **Advanced strength reduction**: Strength reduction is a great way to simplify generated hardware and reduce power consumption on FPGA and ASIC platforms. Advanced techniques are needed to perform algebraic transformations using commutativity, associativity, and distributivity axioms; to reduce floating-point computations to fixed-point; and to reduce bit widths. Profiling-based variable range analysis is one possible approach. Transformations that reduce precision may be acceptable, but the programming language needs to make expressing such precision requirements natural (see third point).

Given the enormous design space and flexibility of custom hardware, HLS is uniquely suited to exploit strength reduction all the way to the gate level. In the HLS paradigm, we are not limited by some unspoken rule about how the functional units should behave, what their physical structure should be, or even how many different types there are, as long as collectively they implement the behavior specified by the designer and are coarse-grained enough to make synthesis scalable. One research direction could explore dividing conventional functional units into their building blocks (e.g., the carry lookahead logic in an adder), dividing instructions into corresponding micro-ops, and then seeing if strength reduction could optimize out some of the micro-ops (e.g., if only some bits of the result are needed).

(5) **More advanced computation models**: Polyhedral models are great for nested loops, but do not work as well for other control flow structures. GPGPU hierarchically parallel models are great for highly-regular, embarrassingly parallel algorithms, but don't work as well for less regular or parallel computations. More models are needed to cover the rest of the application space and enable more advanced program transformations. As more applications are covered, more code becomes efficiently high-level synthesizable. At the same time, existing models can be stretched to extend them to more applications. For example, polyhedral models have been shown to be more applicable than previously thought [6].

Computation models also need to be extended for power estimation. These models could also be extended to enable mapping to low-level physical design models. For example, spatial information present in the polyhedral and GPGPU models could be mapped to physical placement models to reduce wire lengths, cutting power consumption.

(6) **Advanced memory management and architecture generators**: Reducing the distance bits have to travel is a great way to reduce energy usage. Advanced memory architecture generators are needed to keep the data closer to where it is used. Avoiding cache coherence issues is a great way to reduce complexity in generated interconnects, but the programming language model needs to make the lack of coherence obvious (see third point). Network-on-chip architectures are one approach, particularly when combined with advanced computation models (see fifth point). A more radical approach is to fragment the global memory space into many separate memory spaces. The partitioning could be done explicitly through programming models like MPI or implicitly through automatic partitioning algorithms. Memory partitioning could then guide hardware partitioning and help create efficient placement solutions.

Finally, nonvolatile processor-in-memory architectures could enable a new power saving paradigm: instead of performing energy intensive writes of data to off-chip storage to insure that it persists after power-off, simply save the data closest to where it is likely to be used.

(7) **Application-specific compilers**: Reusable, efficient high-level hardware building blocks would be a great way to accelerate hardware design that involves many commonly used algorithms. The large parameterization space of these building blocks in terms of latency, power, interface requirements, electrical specifications, target technology, and floorplan makes effective reuse a challenge. A possible approach to this problem is the creation of application-specific HLS engines. Compiler designers could look at the best hardware designs for a specific application and attempt to generalize, a much simpler proposition than trying to generalize *all* hardware designs.

5. Conclusions

In this paper, we survey recent low-power optimization and design techniques in high-level synthesis and provide a systematic (sometimes detailed) study of the most promising solutions so far in this very important domain of research. We also describe the key power optimization challenges facing high-level synthesis today, some lessons we learned, and some potential opportunities to further improve quality of results of high-level synthesis in the future. As we have seen, there are numerous directions to be explored in low-power design for high-level synthesis. Furthermore, low-power design is inextricably linked to other quality-of-results metrics such as latency, throughput, and area. Thus for a truly holistic, cross-layer approach that has the most potential, all of these optimization goals should be considered.

Acknowledgments We thank the editorial committee for their helpful comments.

References

[1] Ahuja, S., Lakshminarayana, A. and Shukla, S.K.: Power reduction using high-level clock-gating, *Low Power Design with High-Level Power Estimation and Power-Aware Synthesis*, pp.119–129, Springer (2012).

[2] Altera: Cyclone V SoCs: Lowest system cost and power, Altera (online), available from ⟨http://www.altera.com/devices/processor/soc-fpga/cyclone-v-soc/cyclone-v-soc.html⟩ (accessed 2014-10-31).

[3] Anthony, S.: Intel unveils new Xeon chip with integrated FPGA, touts 20x performance boost, ExtremeTech (online), available from ⟨http://www.extremetech.com/extreme/184828-intel-unveils-new-xeon-chip-with-integrated-fpga-touts-20x-performance-boost⟩ (accessed 2014-10-31).

[4] Babighian, P., Benini, L. and Macii, E.: A scalable ODC-based algorithm for RTL insertion of gated clocks, *Design, Automation, and Test in Europe* (*DATE*), pp.500–505 (2004).

[5] Bastoul, C.: Code generation in the polyhedral model is easier than you think, *Int'l Conf. Parallel Architecture and Compilation Techniques* (*PACT*), pp.7–16 (2004).

[6] Benabderrahmane, M.-W., Pouchet, L.-N., Cohen, A. and Bastoul, C.: The polyhedral model is more widely applicable than you think, *Compiler Construction*, pp.283–303, Springer (2010).

[7] Benini, L., Macii, A. and Poncino, M.: A recursive algorithm for low-power memory partitioning, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.78–83 (2000).

[8] Bogliolo, A., Benini, L., Riccó, B. and De Micheli, G.: Efficient switching activity computation during high-level synthesis of control-dominated designs, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.127–132 (1999).

[9] Bozorgzadeh, E., Ghiasi, S., Takahashi, A. and Sarrafzadeh, M.: Optimal integer delay budgeting on directed acyclic graphs, *Design Automation Conf.* (*DAC*), pp.920–925 (2003).

[10] Chandrakasan, A.P., Potkonjak, M., Mehra, R., Rabaey, J. and Brodersen, R.W.: Optimizing power using transformations, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.14, No.1, pp.12–31 (1995).

[11] Chen, D., Cong, J., Fan, Y. and Wan, L.: LOPASS: A low-power architectural synthesis system for FPGAs with interconnect estimation and optimization, *IEEE Trans. Very Large-Scale Integration Systems* (*TVLSI*), Vol.18, No.4, pp.564–577 (2010).

[12] Chen, D., Cong, J., Fan, Y. and Xu, J.: Optimality study of resource binding with multi-Vdds, *Design Automation Conf.* (*DAC*), pp.580–585 (2006).

[13] Chen, D., Cong, J., Fan, Y. and Zhang, Z.: High-level power estimation and low-power design space exploration for FPGAs, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.529–534 (2007).

[14] Chen, Y.-G., Geng, H., Lai, K.-Y., Shi, Y. and Chang, S.-C.: Multi-bit retention registers for power gated designs: Concept, design, and deployment, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.33, No.4, pp.507–518 (2014).

[15] Chen, Y., Xie, Y., Wang, Y. and Takach, A.: Minimizing leakage power in aging-bounded high-level synthesis with design time multi-Vth assignment, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.689–694 (2010).

[16] Chen, Y.-G., Shi, Y., Lai, K.-Y., Hui, G. and Chang, S.-C.: Efficient multiple-bit retention register assignment for power gated design: Concept and algorithms, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.309–316 (2012).

[17] Cheng, L., Chen, D. and Wong, M.D.: GlitchMap: An FPGA technology mapper for low power considering glitches, *Design Automation Conf.* (*DAC*), pp.318–323 (2007).

[18] Choi, E., Shin, C., Kim, T. and Shin, Y.: Power-gating-aware high-level synthesis, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.39–44 (2008).

[19] Chueh, Y.-L. and Huang, C.-H.: Resistive random access memory, *US Patent App. 14/197,697* (2014).

[20] Cong, J., Liu, B., Neuendorffer, S., Noguera, J., Vissers, K. and Zhang, Z.: High-level synthesis for FPGAs: From prototyping to deployment, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.30, No.4, pp.473–491 (2011).

[21] Cong, J., Jiang, W., Liu, B. and Zou, Y.: Automatic memory partitioning and scheduling for throughput and power optimization, *ACM Trans. Design Automation of Electronic Systems* (*TODAES*), Vol.16, No.2, p.15 (2011).

[22] Cong, J., Liu, B., Majumdar, R. and Zhang, Z.: Behavior-level observability analysis for operation gating in low-power behavioral synthesis, *ACM Trans. Design Automation of Electronic Systems* (*TODAES*), Vol.16, No.1, p.4 (2010).

[23] Cong, J., Liu, B. and Zhang, Z.: Behavior-level observability don't-cares and application to low-power behavioral synthesis, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.139–144 (2009).

[24] Cong, J., Liu, B. and Zhang, Z.: Scheduling with soft constraints, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.47–54 (2009).

[25] Cong, J. and Zhang, Z.: An efficient and versatile scheduling algorithm based on SDC formulation, *Design Automation Conf.* (*DAC*), pp.433–438 (2006).

[26] Constantinides, G.A.: Word-length optimization for differentiable nonlinear systems, *ACM Trans. Design Automation of Electronic Systems* (*TODAES*), Vol.11, No.1, pp.26–43 (2006).

[27] Coussy, P. and Morawiec, A. (Eds.): *High-level synthesis: From algorithm to digital circuit*, Springer (2008).

[28] Cromar, S., Lee, J. and Chen, D.: FPGA-targeted high-level binding algorithm for power and area reduction with glitch-estimation, *Design Automation Conf.* (*DAC*) (2009).

[29] Czajkowski, T.S., Neto, D., Kinsner, M., Aydonat, U., Wong, J., Denisenko, D., Yiannacouras, P., Freeman, J., Singh, D.P. and Brown, S.D.: OpenCL for FPGAs: Prototyping a compiler, Technical report, hgpu.org (2013).

[30] Dai, S., Tan, M., Hao, K. and Zhang, Z.: Flushing-enabled loop pipelining for high-level synthesis, *Design Automation Conf.* (*DAC*), pp.1–6 (2014).

[31] Dal, D. and Mansouri, N.: Power optimization with power islands synthesis, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.28, No.7, pp.1025–1037 (2009).

[32] Fallah, F. and Pedram, M.: Standby and active leakage current control and minimization in CMOS VLSI circuits, *IEICE Trans. Electronics*, Vol.88, No.4, pp.509–519 (2005).

[33] Fraer, R., Kamhi, G. and Mhameed, M.K.: A new paradigm for synthesis and propagation of clock gating conditions, *Design Automation Conf.* (*DAC*), pp.658–663 (2008).

[34] Friedrich, J., Le, H., Starke, W., Stuechli, J., Sinharoy, B., Fluhr, E.J., Dreps, D., Zyuban, V., Still, G., Gonzalez, C., et al.: The POWER8 processor: Designed for big data, analytics, and cloud environments, *Int'l Conf. IC Design & Technology* (*ICICDT*), pp.1–4 (2014).

[35] Gerstlauer, A., Haubelt, C., Pimentel, A.D., Stefanov, T.P., Gajski, D.D. and Teich, J.: Electronic system-level synthesis methodologies, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.28, No.10, pp.1517–1530 (2009).

[36] Ghiasi, S., Bozorgzadeh, E., Huang, P.-K., Jafari, R. and Sarrafzadeh, M.: A unified theory of timing budget management, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.25, No.11, pp.2364–2375 (2006).

[37] Gonzalez, R., Gordon, B.M. and Horowitz, M.A.: Supply and threshold voltage scaling for low power CMOS, *IEEE Journal of Solid-State Circuits*, Vol.32, No.8, pp.1210–1216 (1997).

[38] Gu, Z., Wang, J., Dick, R.P. and Zhou, H.: Unified incremental physical-level and high-level synthesis, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.26, No.9, pp.1576–1588 (2007).

[39] Gu, Z.P., Yang, Y., Wang, J., Dick, R.P. and Shang, L.: TAPHS: Thermal-aware unified physical-level and high-level synthesis, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.879–885 (2006).

[40] Gurumani, S., Rupnow, K., Liang, Y., Cholakkail, H. and Chen, D.:

High level synthesis of multiple dependent CUDA kernels for FPGAs, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.305–312 (2013).

[41] Gurumani, S., Tolar, J., Chen, Y., Liang, Y., Rupnow, K. and Chen, D.: Integrated CUDA-to-FPGA synthesis with network-on-chip, *IEEE Symp. Field Programmable Custom Computing Machines* (*FCCM*), pp.21–24 (2014).

[42] Hauck, S. and DeHon, A.: *Reconfigurable computing: The theory and practice of FPGA-based computation*, Morgan Kaufmann (2010).

[43] Hsieh, C.-T., Cong, J., Zhang, Z. and Chang, S.-C.: Behavioral synthesis with activating unused flip-flops for reducing glitch power in FPGA, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.10–15 (2008).

[44] Hu, Z., Buyuktosunoglu, A., Srinivasan, V., Zyuban, V., Jacobson, H. and Bose, P.: Microarchitectural techniques for power gating of execution units, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.32–37 (2004).

[45] Huang, S.-H., Tu, W.-P. and Li, B.-H.: High-level synthesis for minimum-area low-power clock gating, *Journal of Information Science and Engineering*, Vol.28, No.5, pp.971–988 (2012).

[46] Jha, N.K.: Low power system scheduling and synthesis, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.259–263 (2001).

[47] Jiang, W., Zhang, Z., Potkonjak, M. and Cong, J.: Scheduling with integer time budgeting for low-power optimization, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.22–27 (2008).

[48] Kao, J.T. and Chandrakasan, A.P.: Dual-threshold voltage techniques for low-power digital circuits, *IEEE Journal of Solid-State Circuits*, Vol.35, No.7, pp.1009–1018 (2000).

[49] Khouri, K.S. and Jha, N.K.: Leakage power analysis and reduction during behavioral synthesis, *IEEE Trans. Very Large-Scale Integration Systems* (*TVLSI*), Vol.10, No.6, pp.876–885 (2002).

[50] Khouri, K.S., Lakshminarayana, G. and Jha, N.K.: High-level synthesis of low-power control-flow intensive circuits, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.18, No.12, pp.1715–1729 (1999).

[51] Khronos: The OpenCL specification (online), available from ⟨https://www.khronos.org/registry/cl/specs/opencl-2.0.pdf⟩ (2014).

[52] Kultursay, E., Kandemir, M., Sivasubramaniam, A. and Mutlu, O.: Evaluating STT-RAM as an energy-efficient main memory alternative, *Int'l Symp. Performance Analysis of Systems and Software* (*ISPASS*), pp.256–267 (2013).

[53] Kund, M., Beitel, G., Pinnow, C.-U., Rohr, T., Schumann, J., Symanczyk, R., Ufert, K.-D. and Muller, G.: Conductive bridging RAM (CBRAM): An emerging non-volatile memory technology scalable to sub 20nm, *Int'l Electron Devices Meeting* (*IEDM*), pp.754–757 (2005).

[54] Landman, P.E. and Rabaey, J.M.: Power estimation for high level synthesis, *European Conf. Design Automation*, pp.361–366 (1993).

[55] Leng, J., Hetherington, T., ElTantawy, A., Gilani, S., Kim, N.S., Aamodt, T.M. and Reddi, V.J.: GPUWattch: Enabling energy optimizations in GPGPUs, *Int'l Symp. Computer Architecture* (*ISCA*), pp.487–498 (2013).

[56] Li, F., Chen, D., He, L. and Cong, J.: Architecture evaluation for power-efficient FPGAs, *Int'l Symp. Field-Programmable Gate Arrays* (*FPGA*), pp.175–184 (2003).

[57] Li, S., Li, A., Liu, Y., Xie, Y. and Yang, H.: Nonvolatile memory allocation and hierarchy optimization for high-level synthesis, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*) (2015).

[58] Lin, C., Xie, A. and Zhou, H.: Design closure driven delay relaxation based on convex cost network flow, *Design, Automation, and Test in Europe* (*DATE*), pp.63–68 (2007).

[59] Liu, H.-Y., Lee, W.-P. and Chang, Y.-W.: A provably good approximation algorithm for power optimization using multiple supply voltages, *Design Automation Conf.* (*DAC*), pp.887–890 (2007).

[60] Lyuh, C.-G. and Kim, T.: High-level synthesis for low power based on network flow method, *IEEE Trans. Very Large-Scale Integration Systems* (*TVLSI*), Vol.11, No.3, pp.364–375 (2003).

[61] Lyuh, C.-G. and Kim, T.: Memory access scheduling and binding considering energy minimization in multi-bank memory systems, *Design Automation Conf.* (*DAC*), pp.81–86 (2004).

[62] Ma, K., Li, X., Chen, M. and Wang, X.: Scalable power control for many-core architectures running multi-threaded applications, *ACM SIGARCH Computer Architecture News*, Vol.39, No.3, pp.449–460 (2011).

[63] Mallik, A., Sinha, D., Banerjee, P. and Zhou, H.: Low-power optimization by smart bit-width allocation in a SystemC-based ASIC design environment, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.26, No.3, pp.447–455 (2007).

[64] Manzak, A. and Chakrabarti, C.: Variable voltage task scheduling algorithms for minimizing energy, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.279–282 (2001).

[65] Martin, G. and Smith, G.: High-level synthesis: Past, present, and future, *IEEE Design & Test of Computers*, Vol.26, No.4, pp.18–25 (2009).

[66] Martin, G., Bailey, B. and Piziali, A.: *ESL design and verification: A prescription for electronic system level methodology*, Morgan Kaufmann (2010).

[67] Nepal, K., Li, Y., Bahar, R. and Reda, S.: ABACUS: A technique for automated behavioral synthesis of approximate computing circuits, *Design, Automation, and Test in Europe* (*DATE*), pp.1–6 (2014).

[68] Ni, M. and Memik, S.O.: Thermal-induced leakage power optimization by redundant resource allocation, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.297–302 (2006).

[69] Nvidia: CUDA Compute unified device architecture programming guide, Technical report, Nvidia (2007).

[70] Osborne, W.G., Coutinho, J.G.F., Luk, W. and Mencer, O.: Power-aware and branch-aware word-length optimization, *IEEE Symp. Field Programmable Custom Computing Machines* (*FCCM*), pp.129–138 (2008).

[71] Papakonstantinou, A., Chen, D., Hwu, W., Cong, J. and Liang, Y.: Throughput-oriented kernel porting onto FPGAs, *Design Automation Conf.* (*DAC*), pp.1–10 (2013).

[72] Papakonstantinou, A., Gururaj, K., Stratton, J., Chen, D., Cong, J. and Hwu, W.: FCUDA: Enabling efficient compilation of CUDA kernels onto FPGAs, *Symp. Application Specific Processors* (*SASP*) (2009).

[73] Papakonstantinou, A., Gururaj, K., Stratton, J., Chen, D., Cong, J. and Hwu, W.: Efficient compilation of CUDA kernels for high-performance computing on FPGAs, *ACM Trans. Embedded Computing Systems* (*TECS*), Vol.13, No.2, pp.25:1–25:26 (2013).

[74] Papakonstantinou, A., Liang, Y., Stratton, J., Gururaj, K., Chen, D., Hwu, W. and Cong, J.: Multilevel granularity parallelism synthesis on FPGAs, *IEEE Symp. Field Programmable Custom Computing Machines* (*FCCM*) (2011).

[75] Pedram, M.: Low power design methodologies and techniques: An overview, *Microprocessor Report*, Vol.486 (1999).

[76] Pouchet, L.-N., Bondhugula, U., Bastoul, C., Cohen, A., Ramanujam, J. and Sadayappan, P.: Hybrid iterative and model-driven optimization in the polyhedral model, Technical Report 6962, INRIA Research Report (2009).

[77] Powell, M., Yang, S.-H., Falsafi, B., Roy, K. and Vijaykumar, T.: Gated-Vdd: A circuit technique to reduce leakage in deep-submicron cache memories, *Int'l Symp. Low Power Electronics and Design* (*ISLPED*), pp.90–95 (2000).

[78] Putnam, A., Caulfield, A.M., Chung, E.S., Chiou, D., Constantinides, K., Demme, J., Esmaeilzadeh, H., Fowers, J., Gopal, G.P., Gray, J., et al.: A reconfigurable fabric for accelerating large-scale datacenter services, *Int'l Symp. Computer Architecture* (*ISCA*), pp.13–24 (2014).

[79] Raghunathan, A. and Jha, N.K.: Behavioral synthesis for low power, *Int'l Conf. Computer Design: VLSI in Computers and Processors*, pp.318–322 (1994).

[80] Raghunathan, A. and Jha, N.K.: SCALP: An iterative-improvement-based low-power data path synthesis system, *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems* (*TCAD*), Vol.16, No.11, pp.1260–1277 (1997).

[81] Raghunathan, A., Jha, N.K. and Dey, S.: *High-level power analysis and optimization*, Springer (1998).

[82] Raje, S. and Sarrafzadeh, M.: Variable voltage scheduling, *Int'l Symp. Low Power Design*, pp.9–14 (1995).

[83] Ranganathan, P.: Recipe for efficiency: Principles of power-aware computing, *Comm. ACM*, Vol.53, No.4, pp.60–67 (2010).

[84] Scharstein, D., Hirschmüller, H., Kitajima, Y., Krathwohl, G., Nesic, N., Wang, X. and Westling, P.: High-resolution stereo datasets with subpixel-accurate ground truth, *German Conference on Pattern Recognition*, pp.31–42 (2014).

[85] Scheffer, L.: A roadmap of CAD tool changes for sub-micron interconnect problems, *Int'l Symp. Physical Design* (*ISPD*), pp.104–109 (1997).

[86] Shang, L., Dick, R.P. and Jha, N.K.: High-level synthesis algorithms for power and temperature minimization, *High-Level Synthesis*, pp.285–297, Springer (2008).

[87] Showerman, M., Enos, J., Pant, A., Kindratenko, V., Steffen, C., Pennington, R. and Hwu, W.-M.: QP: A heterogeneous multi-accelerator cluster, *LCI Int'l Conf. High-Performance Clustered Computing* (2009).

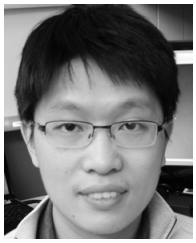
[88] Stammermann, A., Helms, D., Schulte, M., Schulz, A. and Nebel, W.: Binding, allocation and floorplanning in low power high-level synthesis, *Int'l Conf. Computer-Aided Design* (*ICCAD*), p.544 (2003).

[89] Tan, M., Dai, S., Gupta, U. and Zhang, Z.: Mapping-Aware

Constrained Scheduling for LUT-Based FPGAs, *Int'l Symp. Field-Programmable Gate Arrays* (*FPGA*), pp.1–10 (2015).
[90] Tan, M., Liu, B., Dai, S. and Zhang, Z.: Multithreaded Pipeline Synthesis for Data-Parallel Kernels, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.718–725 (2014).
[91] Tang, X., Zhou, H. and Banerjee, P.: Leakage power optimization with dual-Vth library in high-level synthesis, *Design Automation Conf.* (*DAC*), pp.202–207 (2005).
[92] Téllez, G.E., Farrahi, A. and Sarrafzadeh, M.: Activity-driven clock design for low power circuits, *Int'l Conf. Computer-Aided Design* (*ICCAD*), pp.62–65 (1995).
[93] Vittoz, E.A.: Low-power design: Ways to approach the limits, *Int'l Solid-State Circuits Conf.* (*ISSCC*), pp.14–18 (1994).
[94] Wakabayashi, K.: C-based behavioral synthesis and verification analysis on industrial design examples, *Asia and South Pacific Design Automation Conf.* (*ASP-DAC*), pp.344–348 (2004).
[95] Wang, Z. and Hu, X.S.: Power aware variable partitioning and instruction scheduling for multiple memory banks, *Design, Automation, and Test in Europe* (*DATE*), p.10312 (2004).
[96] Wu, Q., Pedram, M. and Wu, X.: Clock-gating and its application to low power design of sequential circuits, *IEEE Trans. Circuits and Systems I: Fundamental Theory and Applications*, Vol.47, No.3, pp.415–420 (2000).
[97] Xilinx: All programmable abstractions, Xilinx (online), available from ⟨http://www.xilinx.com/products/design-tools/all-programmable-abstractions.html#software-based⟩ (accessed 2014-11-06).
[98] Xilinx: Vivado design suite, Xilinx (online), available from ⟨http://www.xilinx.com/products/design-tools/vivado.html⟩ (accessed 2014-11-7).
[99] Xilinx: Zynq-7000 all programmable SoC, Xilinx (online), available from ⟨http://www.xilinx.com/products/silicon-devices/soc/zynq-7000/⟩ (accessed 2014-10-31).
[100] Zhang, Z. and Chen, D.: Challenges and opportunities of ESL design automation, *Int'l Conf. Solid-State and Integrated Circuit Technology* (2012).
[101] Zheng, H., Gurumani, S.T., Rupnow, K. and Chen, D.: Fast and effective placement and routing directed high-level synthesis for FPGAs, *Int'l Symp. Field-Programmable Gate Arrays* (*FPGA*), pp.1–10 (2014).
[102] Zuo, W., Li, P., Chen, D., Pouchet, L.-N., Zhong, S. and Cong, J.: Improving polyhedral code generation for high-level synthesis, *Int'l Conf. Hardware/Software Codesign and System Synthesis* (*CODES+ISSS*) (2013).
[103] Zuo, W., Liang, Y., Rupnow, K., Li, P., Chen, D. and Cong, J.: Improving High Level Synthesis Optimization Opportunity Through Polyhedral Transformations, *Int'l Symp. Field-Programmable Gate Arrays* (*FPGA*) (2013).

Zhiru Zhang is an assistant professor in the School of ECE at Cornell University and a member of the Computer Systems Laboratory. His current research focuses on high-level design automation for heterogeneous computing. His work has been recognized with the Best Paper Award from TODAES and a best paper nomination at ICCAD. In 2006, he co-founded AutoESL Design Technologies, Inc. to commercialize his Ph.D. dissertation research on high-level synthesis. AutoESL was acquired by Xilinx in 2011 and the AutoESL tool was rebranded as Vivado HLS after the acquisition.

Deming Chen is an associate professor in the ECE department of University of Illinois at Urbana-Champaign. His research interests include high-level synthesis, nano-centric CAD techniques, GPU optimization, reconfigurable computing, hardware/software co-design, and computational biology. He is an associate editor for TCAD, TODAES, TVLSI, TCAS-I, JCSC, and JOLPE. He obtained various awards, including five Best Paper Awards, ACM SIGDA Outstanding New Faculty Award, and IBM Faculty Award.



Steve Dai is a Ph.D. student in Computer Systems Laboratory at Cornell University working with Professor Zhiru Zhang. He received his B.S. in Electrical Engineering from University of California, Los Angeles (UCLA) in 2011 and his M.S. in Electrical Engineering from Stanford University in 2013. He is broadly interested in electronic design automation, with emphasis on high-level synthesis and parallel programming for reconfigurable computing.

Keith Campbell is a Ph.D. student working in the Coordinated Science Laboratory at the University of Illinois at Urbana-Champaign with Professor Deming Chen. He received his B.S. in Electrical Engineering from the Illinois Institute of Technology in 2008. His current research focuses on the intersection of high-level synthesis and circuit validation and reliability. Other research interests include compiler design, high-level programming language design, and alternative computer architectures.

(Invited by Editor-in-Chief: *Hiroyuki Tomiyama*)