High-Level Low-Power System Design Optimization

David Pursley+, Tung-Hua Yeh*
+Cadence Design Systems, Inc., USA, *Cadence Design Systems, Inc., Taiwan

Abstract –*High-level decisions have the most impact on power consumption, but the effect of those decisions cannot be known until the hardware is implemented. This paper walks the reader through an industrial high-level low-power design methodology that enables the designer to consider and quantitatively evaluate a broad range of hardware implementations to find the most power-efficient architecture. This paper concludes with two industry case studies using this high-level low power methodology.*

INTRODUCTION

It is well understood that decisions with the most impact in terms of quality of results (QoR) are made early in the design process. When optimizing for power, experts estimate that optimal architectural decisions can reduce power by 80% or more. [1] Stated as the inverse, architectural decisions that are poor for power can lead to 5X greater power consumption than more power-efficient architectures.

For this reason, in the ideal world multiple architectures would be identified and fully evaluated in terms of power, performance, and area. Then the best architecture would be implemented in hardware, knowing that the design constraints will be met. This ideal approach is shown in Figure 1.

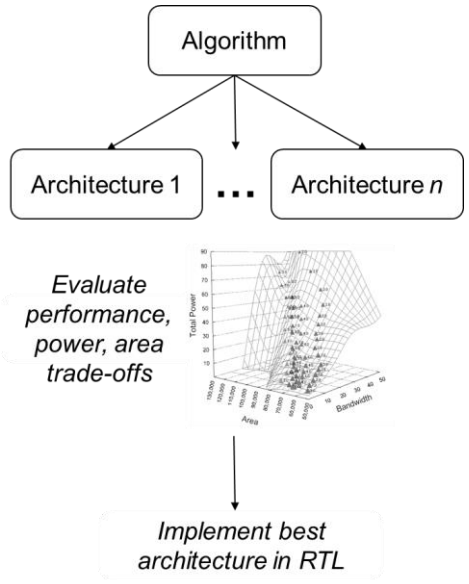

FIGURE 1. IDEAL FLOW FROM ALGORITHM TO IMPLEMENTATION

Unfortunately, in real world production applications this ideal flow is rarely, if ever, realized. Despite the best efforts of industry and academia, there is no analysis mechanism that allows implementation trade-offs to be made with certainty or even strong causality at the architectural level, especially when it comes to power optimization. One reason is that the power consumption is greatly affected by both the *architecture* and the *microarchitecture*.

"Architecture:" Collection of high-level decisions about the overall structure of the hardware. Three common examples are block partitioning, communication interfaces, and storage (memory) architecture. Informally, architectural decisions can be thought of as those done as a block diagram on a whiteboard in the early stages of design.

"Microarchitecture:" Collection of lower-level implementation decisions about the hardware implementation. Three common examples are pipeline depth, datapath structure, and register allocation. Informally, these decisions are usually part of the implementation stage, whether writing the RTL by hand or via high-level synthesis (HLS).

After the architectural and microarchitectural decisions are fixed, there is little opportunity for significant power trade-offs in the RTL flow. It is estimated that the total amount of savings available after these decisions is 20%. [1]

While architectural and microarchitectural decisions have the most impact, the extent of their impact is not known until well into the implementation of the hardware. But at that point, it is too late to make any substantive architectural changes.

As a result designers have no choice but to make decisions early in the design process based on "gut instinct." For experienced designers, that is often good enough, as any decisions that are sub-optimal are "close enough," and there is still a good chance the overall area and performance targets will be met based on previous experience. However, it does risk that significant optimization was missed.

That risk is especially prevalent when it comes to power optimization, as most designers don't have the same intuitive feel for power implications of their decisions as they do on performance and area.

PROPOSED METHODOLOGY

A more pragmatic methodology modifies the flow slightly into something that is achievable today. Instead of attempting to analyze and make implementation decisions directly from the algorithm or high-level architecture, the proposed methodology automatically creates many implementations from the original algorithm and then quantitatively evaluates each of them to determine the power, performance, and area trade-offs they each represent.

Specifically, the proposed methodology uses high-level synthesis (HLS) to automatically generate multiple RTL implementations architectural models. Commercial HLS tools are well-known to be able to do advanced power, performance, and area trade-offs, giving the designer high quality RTL implementations from which to select. [2]

Each RTL implementation is fully evaluated by state RTL tools, including power estimation, to provide feedback on the architecture and the resulting implementations. This methodology is shown in Figure 2.

Accepted for publication at 2017 International Symposium on VLSI Design, Automation and Test (2017 VLSI-DAT)

This methodology provides an easy, automated flow from algorithm to quantitative power, performance, and area metrics. It provides an exploration capability not present in the typical RTL design flow, where only one RTL implementation is created, evaluated, and optimized.

The obvious benefit of the proposed methodology is that the best implementation can be selected for implementation in silicon. This removes the guesswork on exactly how to implement the specified algorithm to meet the given constraints.

However, remember that it is the architectural decisions, not implementation decisions, that provide the most benefit. For that, we need to extend the methodology further.

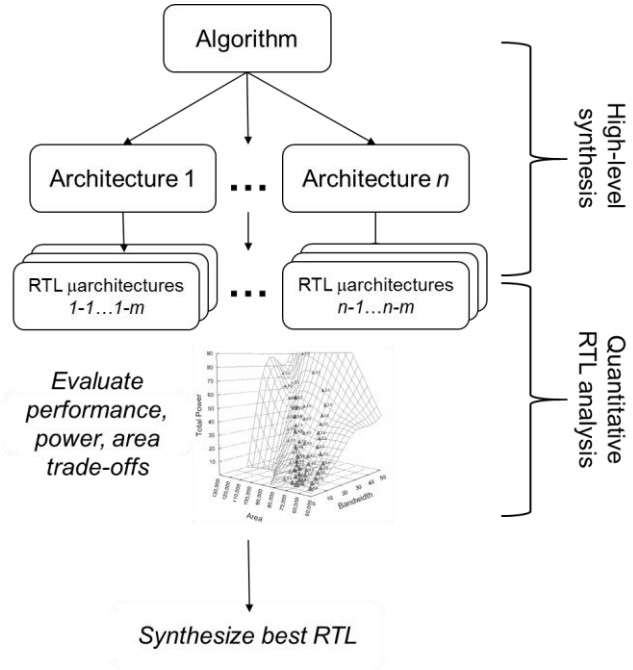

FIGURE 2. PROPOSED METHODOLOGY

Directed exploration

Directed exploration extends this methodology to aid, or direct, the designer to the most problematic spots for optimization. For example, when optimizing for area, it will direct the designer to the specific portions of the algorithm that are resulting in the most area. In the simplest case, you can imagine this would point to a segment of code that results in several high cost multipliers. (Often, it is not that simple as sharing, registers, and muxing can vastly change area implications.)

Directed exploration is especially helpful when optimizing for power. Unlike area, power can vary greatly for any given implementation, so quantitative analysis becomes a requirement.

Furthermore, different input stimuli can exercise the hardware in ways that vastly change the power. For example, video encoders and decoders have much worse power characteristics when the video stream contains a lot of motion.

Power dissipation also changes over time. Wireless modems are quite different when attempting to acquire a signal vs. tracking a locked signal. Similarly, a processor has different power dissipation when booting an OS vs. waiting at idle vs. running a specific application. Power dissipation changes even when time is measured in microseconds or nanoseconds, which becomes a key concern when optimizing for peak power.

Getting an accurate measure of RTL power consumption during design exploration has been a major challenge. This is complicated by the fact that different tools are used at different stages of the design, for example at RTL vs. place and route. Recently, commercial tools have emerged that deliver time-based RTL power analysis with system-level runtimes and capacity, as well as high-quality estimates of gates and wires based on production implementation technology. [3] The accuracy of these tools is within 15% of signoff estimates.

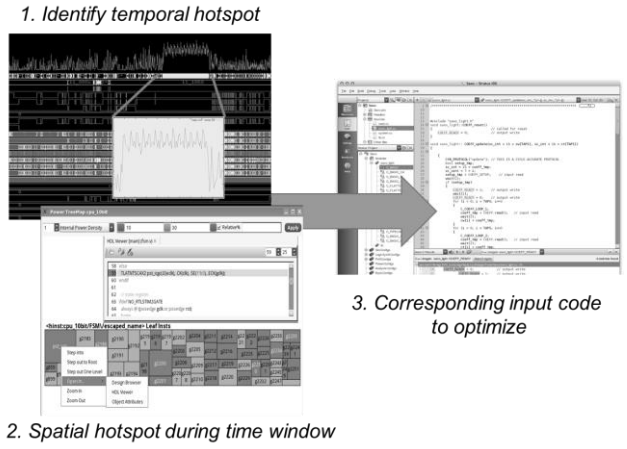

FIGURE 3. IDENTIFYING POWER HOTSPOTS IN TIME AND SPACE

As shown in Figure 3, accurately identifying high-power "hotspots" in the RTL, both in time (waveforms) and space (RTL code), allows the hotspot to be mapped directly to the corresponding code in the algorithm. This gives the designer very precise feedback on the exact behavioral code that should be optimized to reduce power, completing the feedback loop shown in Figure 4.

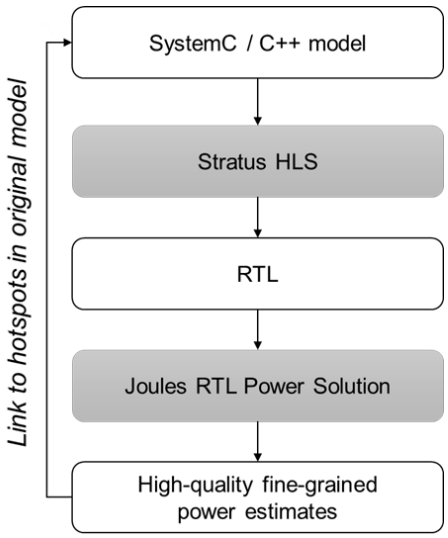

FIGURE 4. DIRECTED µARCHITECTURAL EXPLORATION

RESULTS

This section details applications of the above methodology. The first is an inverse discrete cosine transformation (IDCT), selected because it is easily understood, widely used, and non-proprietary. The second example details the results of this methodology for an industrial software defined radio (SDR) application committed to production silicon.

Simple example: Inverse discrete cosine transform

The DCT and IDCT is a commonly used transformation in signal processing. The specific implementation we used here is a two-dimensional IDCT of the sort that would typically be used in image decoding, such as in a JPEG or MPEG decoder. [4]

In hardware, this is often implemented as two one-dimensional IDCT's in succession, one for row-processing and one for column-processing. To improve overall throughput., the two IDCT's are done in parallel with intermediate value storage between them as shown in Figure 5.

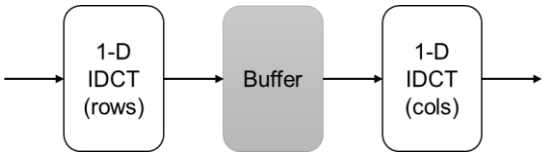

FIGURE 5. SIMPLE 2-D IDCT PROCESSING UNIT

We implemented the two 1-D IDCT blocks as a SystemC algorithm, modified from C source code from a publicly available source. [5]

We applied the quantitative trade-off analysis methodology to evaluate multiple design decisions. Specifically, we varied the following architectural and microarchitectural choices to explore the design space.

Buffer architecture: We varied its implementation as either single-write/single-read or high-performance dual-read/dual-write.

Latency: We constrained the HLS tool to implement the IDCT loop in 8, 16, 32 clock cycles.

Loop pipelining: For the implementations with longer latencies, we allowed the HLS tool to pipeline the IDCT computation loop so it can begin execution every 8 or 16 cycles.

Clock frequency: We varied clock frequency supplied to the HLS tool from 100MHz to 400MHz in a 65nm low power technology.

Note that certain combinations of decisions were not possible, such as a lower performance memory architecture with a high-performance loop pipelining microarchitecture. Also, the highest frequencies were not realizable (could not close timing) with certain combinations of microarchitectural decisions. The non-implementable combinations were excluded from analysis.

In total, 61 different RTL microarchitectures were considered, as shown in Figure 6. Each was evaluated for area (as reported by logic synthesis), power (as reported by gate-level vector-based power analysis), and overall bandwidth (determined via RTL simulation).

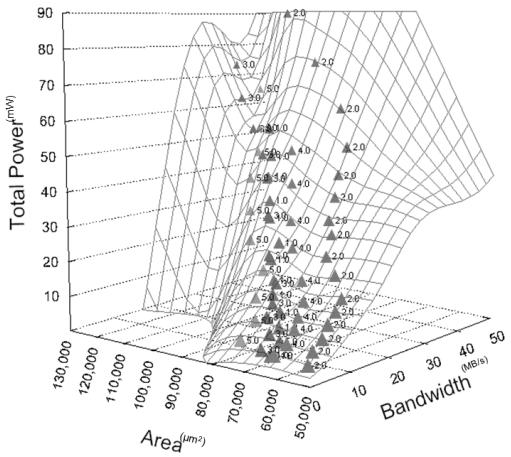

FIGURE 6. IDCT TRADE-OFFS FROM 61 IMPLEMENTATIONS

The microarchitectures covered a large design space. After excluding any points that were pareto dominated by another point, throughput varied by 4.8x, area varied by 2.2x, power varied by 7.5x, and energy per IDCT varied by 2.6x.

It was interesting that the most energy-efficient microarchitecture changed with throughput, as shown in in Figure 7.

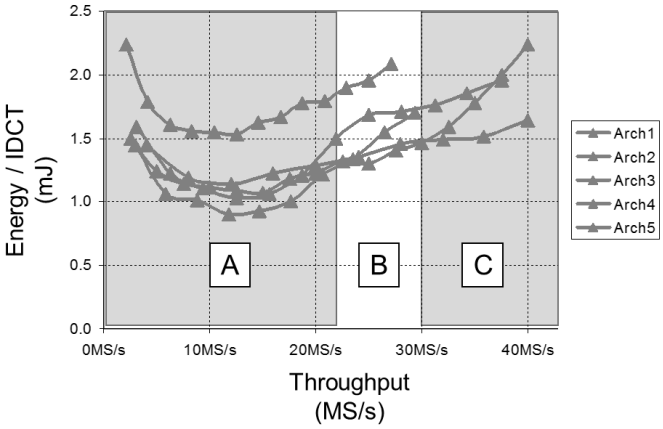

FIGURE 7. ENERGY EFFICIENCY OF IDCT MICROARCHITECTURES

In Region B, where throughput is 22 to 30 million samples per second, an unpipelined microarchitecture taking 16-clock cycles per loop iteration was the most energy-efficient.

In Region C, where throughput is greater than 30 million samples per second, a highly pipelined microarchitecture was most energy efficient. In fact, at the highest throughputs this pipelined microarchitecture was 35% more energy efficient than the microarchitecture from Region B.

In Region A, where throughput is less than 22 million samples per second, a different pipelined microarchitecture was more energy-efficient (but larger) than the microarchitecture from Region B.

This underlines the fact the "best" architectural and microarchitectural decisions are not always an obvious choice, especially when optimizing for power.

Industrial example: software-defined radio

This methodology, including directed microarchitectural exploration, was used to implement approximately 8 million gates in an industrial software-defined radio receiver application. The application included three major blocks, tightly coupled including feedback. Details of the application are proprietary.

The performance requirements were determined *a priori* by the overall system. All blocks had throughput requirements (1 sample every *n* cycles), and two of the blocks also had latency requirements (no less than *m* cycles from input to output). Clock frequency was fixed at 500MH in a 28nm technology library.

Area was the primary optimization goal, with dynamic power reduction as a secondary goal. Area was reported after logic synthesis, and power was measured via vector-based RTL power estimation.

Block A: This block was the most complex, including not only computation but also control of the other blocks. Initial optimization efforts focused on three of the four functions included in the block. However, after some analysis it was the fourth function causing the performance bottleneck. After additional optimization, two RTL implementations stood out as especially relevant from an area perspective, being within 0.5% of each other.

Power analysis then determined the slightly larger version had 34% less power consumption. Some of the area optimization had severely decreased the effectiveness of fine-grained clock gating.

Block B: This block was less complex than Block A, but replicated several times so it had significant impact on area and power. After a number of synthesis runs with some initial exploration of performance constraints, the results were analyzed to determine where potential area optimizations may be found. One computational kernel was found to be dominating both area and power, and optimization was focused on this portion of the design.

A majority of improvement came from applying two types of optimizations. One optimization creates custom datapath components for sequences of operations, which generally reduces the area for the computation, but prevents some sharing. The other optimization partitions the datapath, which parallelizes computations but allows sharing within them. These were applied separately and in tandem.

In the end, the version where both optimizations were applied was the best overall implementation in terms of both area and power. It was 23% smaller than the largest point. Unfortunately, the same testbench (simulation vectors) was not used for all points, preventing a power comparison of all points. But this smallest implementation also consumed the least power of the ones that were measured.

Block C: This block was very small, accounting for 3% of the total area and 2% of the total power. The same high-level techniques were applied by adjusting constraints on the HLS tool

Several microarchitectures were generated by varying the constraints given to the HLS tool. No directed exploration was done given its small contribution to overall area and power. This resulted in one pareto optimal point that was 9% smaller and 5% less power than the worst data point.

Overall results: By using the methodology of quantitative trade-off analysis, including directed exploration, this industrial application was significantly smaller and more power-efficient than originally budgeted. This methodology is credited with reducing the area of the design by >25% with a 4x improvement in power, compared to the estimates based on the previous-generation hand-written RTL.

CONCLUSION

The proposed methodology leverages the ability of HLS to generate multiple RTL implementations so that they may be quantitatively compared in terms of area, performance, and power. Beyond simply allowing more data points to be generated, this methodology also helps focus the designers' optimization efforts on wherever it will have the most impact.

We found that quantitative analysis is especially important when it comes to power optimization. One reason is that most real-world designers don't have the same intuitive feel for power implications as they do on performance and area. Moreover, as seen in Block A of the industrial software-defined radio example, sometimes very small changes can have huge implications for power.

REFERENCES

[1] G. Delp, Y. Trivedi, "Design and Verification of Low Power SoCs," *ISQED09 Embedded Tutorial*, 2009.

[2] Cadence® Stratus™ High-Level Synthesis, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/synthesis/stratus-high-level-synthesis.html

[3] Cadence Joules™ RTL Power Solution, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/power-analysis/joules-rtl-power-solution.html

[4] Wikipedia, "JPEG," http://en.wikipedia.org/wiki/JPEG.

[5] Independent JPEG Group, http://www.ijg.org/files/, 2016.