Determining Application-specific Peak Power and Energy Requirements for Ultra-low Power Processors

Hari Cherupalli†, Henry Duwe‡, Weidong Ye‡, Rakesh Kumar‡, and John Sartori†

†University of Minnesota, ‡University of Illinois at Urbana-Champaign

cheru007@umn.edu {duweiii2,wye5,rakeshk}@illinois.edu jsartori@umn.edu

Abstract

Many emerging applications such as IoT, wearables, implantables, and sensor networks are power- and energy-constrained. These applications rely on ultra-low-power processors that have rapidly become the most abundant type of processor manufactured today. In the ultra-low-power embedded systems used by these applications, peak power and energy requirements are the primary factors that determine critical system characteristics, such as size, weight, cost, and lifetime. While the power and energy requirements of these systems tend to be application-specific, conventional techniques for rating peak power and energy cannot accurately bound the power and energy requirements of an application running on a processor, leading to over-provisioning that increases system size and weight. In this paper, we present an automated technique that performs hardware-software co-analysis of the application and ultra-low-power processor in an embedded system to determine application-specific peak power and energy requirements. Our technique provides more accurate, tighter bounds than conventional techniques for determining peak power and energy requirements, reporting 15% lower peak power and 17% lower peak energy, on average, than a conventional approach based on profiling and guardbanding. Compared to an aggressive stressmark-based approach, our technique reports power and energy bounds that are 26% and 26% lower, respectively, on average. Also, unlike conventional approaches, our technique reports guaranteed bounds on peak power and energy independent of an application's input set. Tighter bounds on peak power and energy can be exploited to reduce system size, weight, and cost.

1. Introduction

Ultra-low-power (ULP) processors have rapidly become the most abundant type of processor in production today. New and emerging power- and energy-constrained applications such as the internet-of-things (IoT), wearables, implantables, and sensor networks have already caused production of ULP processors to exceed that of personal computers and mobile processors [7]. The 2015 ITRS report projects that these applications will continue to rely on simple single-core ultra-low-power processors in the future, will be powered by batteries and energy harvesting, and will have even tighter peak power and energy constraints than the power- and energy-constrained ULP systems of today [2]. Unsurprisingly, low-power microcontrollers and microprocessors are projected to continue being the most widely-used type of processor in the future [3, 7, 17, 37].

ULP systems can be classified into three types based on the way they are powered [13]. As illustrated in Figure 1, some ULP systems are powered directly by energy harvesting (Type 1), while some are battery-powered (Type 3). Another variant is powered by a battery and uses energy harvesting to charge the battery (Type 2).

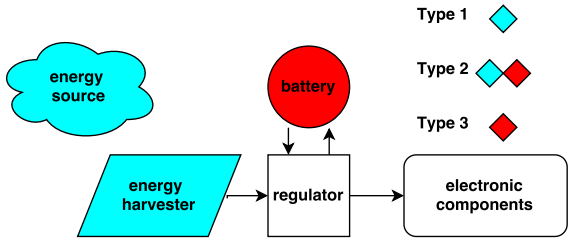


Figure 1: ULP systems are commonly powered by energy harvesting, battery, or a combination of the two, where harvesters are used to charge the battery.

For each of the above classes, the size of energy harvesting and/or storage components determine the form factor, size, and weight. Consider, for example, the wireless sensor node shown in Figure 2 [25]. The two largest system components that predominantly determine the overall system size and weight are the energy harvester (solar cell) and the battery.

Going one step further, since the energy harvesting and storage requirements of a ULP system are determined by its power and energy requirements, the peak power and energy requirements of a ULP system are the primary factors that determine critical system characteristics such as size, weight, cost, and lifetime [13]. In Type 1 systems, peak power is the primary constraint that determines system size, since the power delivered by harvesters is proportional to their size. In these systems, harvesters must be sized to provide enough power, even under peak load conditions. In Type 3 systems, peak power largely determines battery life, since it determines the *effective battery capacity* [10]. As the rate of discharge increases, effective battery capacity

ASPLOS '17, April 08-12, 2017, Xi'an, China

DOI: http://dx.doi.org/10.1145/3037697.3037711

Table 1: Specific energy and energy density for different battery types [5].

Battery Type	Specific Energy [J/g]	Energy Density [MJ/L]
Li-ion	460	1.152
Alkaline	400	0.331
Carbon-zinc	130	1.080
Ni-MH	340	0.504
Ni-cad	140	0.828
Lead-acid	146	0.360

Table 2: Power density for different types of energy harvesters. [35]

Harvester type	Power Density
Photovoltaic (sun)	100 mW/cm^2
Photovoltaic (indoor)	100 $\mu W/cm^2$
Thermoelectric	60 $\mu W/cm^2$
Ambient airflow	1 mW/cm^2

drops [10, 19]. This effect is particularly pronounced in ULP systems, where near-peak power is consumed for a short period of time, followed by a much longer period of low-power sleep, since pulsed loads with high peak current reduce effective capacity even more drastically than sustained current draw [19].

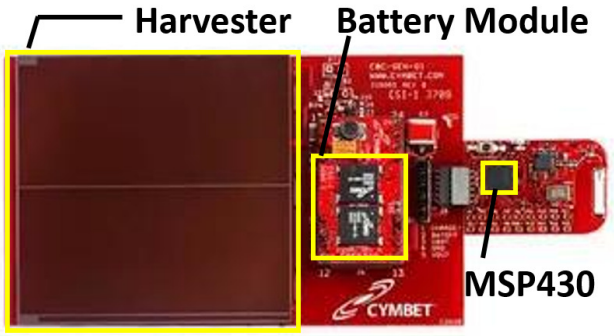


Figure 2: In most ULP systems, like this wireless sensor node, the size of the battery and/or energy harvester dominates the total system size.

In Type 2 and 3 systems, the peak energy requirement matters as well. For example, energy harvesters in Type 2 systems must be able to harvest more energy than the system consumes, on average. Similarly, battery life and effective capacity are dependent on energy consumption (i.e., average power) [19]. Figure 3 summarizes how peak power and energy requirements impact sizing parameters for the different classes of ULP systems.

Finally, Tables 1 and 2 list the energy and power densities for different types of batteries and energy harvesters, respectively. These data provide a rough sense of how size and weight of a ULP system scale based on peak energy and power requirements. A tighter bound on the peak power and energy requirements of a ULP system can result in a roughly proportional reduction in size and weight.

How are Peak Power and Energy Determined Today? There are several possible approaches to determine the peak

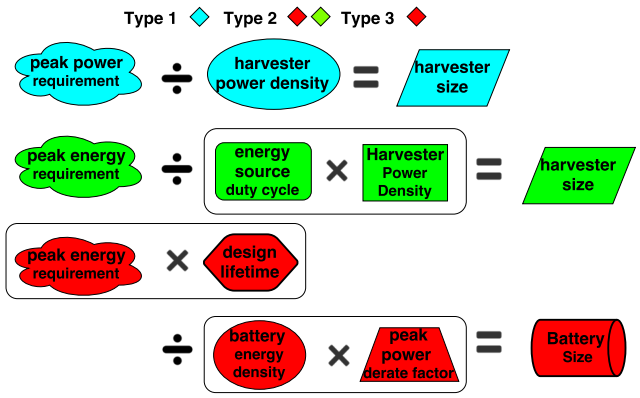


Figure 3: Harvester and battery size calculations for Type 1, 2, and 3 ULP systems depend on peak power and energy requirements.

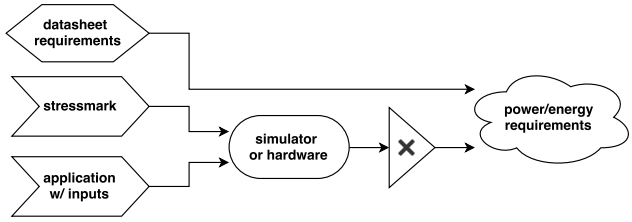
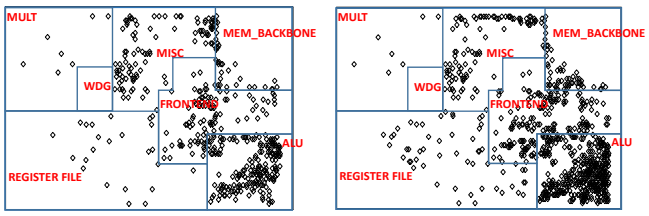


Figure 4: The conventional methodology for sizing energy harvesting and storage components involves determining peak power and energy requirements for a processor and selecting components that will provide enough power and energy to satisfy the requirements over the lifetime of the system.

power and energy requirements of a ULP processor (Figure 4).[1] The most conservative approach involves using the processor design specifications provided in data sheets. These specifications characterize the peak power that can be consumed by the hardware at a given operating point and can be directly translated into a bound on peak power. This bound is conservative because it is not application-specific; however, it is safe for any application that might be executed on the hardware. A more aggressive technique for determining peak power or energy requirements is to use a peak power or energy stressmark. A stressmark is an application that attempts to activate the hardware in a way that maximizes peak power or energy. A stressmark may be less conservative than a design specification, since it may not be possible for an application to exercise all parts of the hardware at once. The most aggressive conventional technique for determining peak power or energy of a ULP processor is to perform application profiling on the processor by measuring power consumption while running the target application on the hardware. However, since profiling is performed with specific input sets under specific operating conditions, peak power or energy bounds determined by profiling might be exceeded during operation if application inputs or system operating conditions are different than during profiling. To

[1] Peak power and energy are sometimes referred to as worst-case power and energy.



(a) Active gates at the peak cycle for tHold (452 gates). **(b)** Active gates at the peak cycle for PI (743 gates).

Figure 5: Different applications can have different activity profiles, resulting in peak power and energy requirements that are application-specific.

ensure that the processor operates within its peak power and energy bounds, a guardband is applied to profiling-based results.

Our Proposal: Determining Application-specific Peak Power and Energy Requirements

Most ULP embedded systems run the same application or computation over and over in a compute / sleep cycle for the entire lifetime of the system [1]. As such, the power and energy requirements of embedded ULP processors tend to be application-specific. This is not surprising, considering that different applications exercise different hardware components at different times, generating different application-specific loads and power profiles. For example, Figures 5a and 5b show the active (toggling) gates for two different applications (tHold and PI – see Table 3) during the cycles in which peak power is expended for each application. These figures were generated by running gate-level simulations of the applications on openMSP430 [20] and marking all gates that toggled in the cycle in which each benchmark expended its peak power. The figures show that PI exercises a larger fraction of the processor than tHold at its peak, leading to higher peak power. However, while the peak power and energy requirements of ULP processors tend to be application-specific, many conventional techniques for determining peak power and energy requirements for a processor are not application-specific (e.g., design-based and stressmark-based techniques). Even in the case of a profiling-based technique, guardbands must be used to inflate the peak power requirements observed during profiling, since it is not possible to generate bounds that are guaranteed for all possible input sets. These limitations prevent existing techniques from accurately bounding the power and energy requirements of an application running on a processor, leading to over-provisioning that increases system size and weight.

In this paper, we present a novel technique that determines application-specific peak power and energy requirements based on hardware-software co-analysis of the application and ultra-low-power processor in an embedded system. Our technique performs a symbolic simulation of an application on the processor netlist in which unknown logic values (Xs) are propagated for application inputs.[2] This allows us to identify gates that are guaranteed to not be exercised by the application for any input. This, in turn, allows us to bound the peak power and energy requirements for the application. The peak power and energy requirements generated by our technique are guaranteed to be safe for all possible inputs and operating conditions. Our technique is fully automated and provides more accurate, tighter bounds than conventional techniques for determining peak power and energy requirements. Our paper makes the following contributions.

- We present an automated technique based on symbolic simulation that takes an embedded system's application software and processor netlist as inputs and determines application-specific peak power and energy requirements for the processor that are guaranteed to be valid for all possible application inputs and operating conditions. This is the first approach to use symbolic simulation to determine peak power and energy requirements for an application running on a processor.
- We show that the application-specific peak power and energy requirements determined by our technique are more accurate, and therefore less conservative, than those determined by conventional techniques. On average, the peak power requirements generated by our technique are 27%, 26%, and 15% lower than those generated based on design specifications, a stressmark, and profiling, respectively, and the peak energy requirements generated by our technique are 47%, 26%, and 17% lower. Reduction in the peak power and energy requirements of a ULP processor can be leveraged to improve critical system metrics such as size and weight.
- Our technique can be used to guide optimizations that target and reduce the peak power of a processor. Optimizations suggested by our technique reduce peak power by up to 10% for a set of embedded applications.

2. A Case for Application-specific Input-independent Peak Power and Energy Requirements

We measured peak power consumption for a sample set of ULP benchmark applications (see Table 3) running on an MSP430F1610 processor.[3] Benchmark applications were run repeatedly with different inputs at an operating frequency of 8 MHz while sampling the voltage and current of the processor at a rate of 10 MHz using an InfiniiVision DSO-X 2024A oscilloscope, to ensure at least one sample per cycle. Power is calculated as the product of voltage and current. Figure 6 shows our test setup.

Figure 7a compares the peak power observed for different applications. The results show that peak power can be different for different applications. Thus, peak power bounds that are not application-specific will overestimate the peak power requirements of applications, leading to over-provisioning of energy harvesting and storage components that determine system size and weight. Figure 7a also shows that the peak power requirements of applications are significantly lower than the rated peak power of the chip (4.8 mW), so using design specifications to determine peak power requirements can lead to significant over-provisioning and inefficiency. The figure also confirms that peak power of an application depends on application inputs and can vary significantly for different inputs. This means that profiling cannot be relied

[2] Peak power and energy analyses can be offered as a cloud compilation service by the hardware system vendor in settings where the application developer does not have access to the processor description [6, 15, 24].

[3] MSP430 is one of the most popular processors used in ULP systems [8, 46].

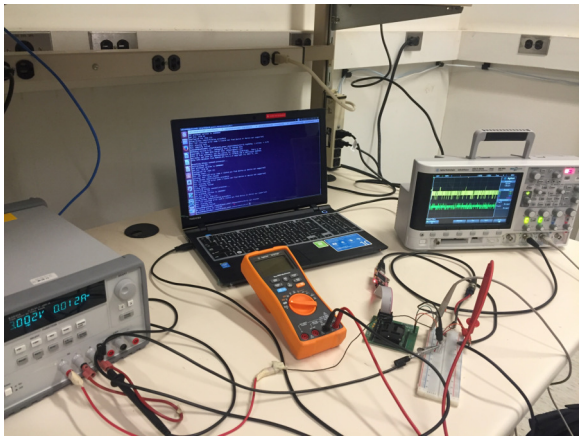
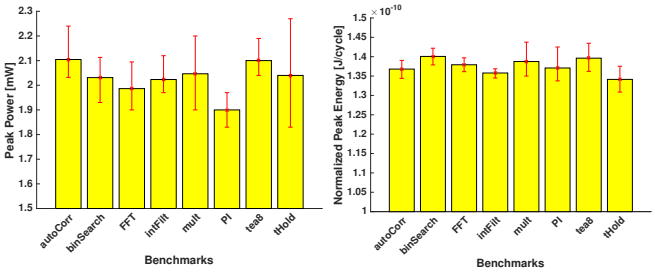


Figure 6: The test setup used to measure peak and average power on a ULP processor (MSP430).



(a) Peak Power **(b) NPE**

Figure 7: The peak power and normalized peak energy (normalized to an application's runtime in cycles) of a ULP processor are different for different applications and different inputs. The bars represent average across all inputs; error bars show the range of input-induced peak and average power variations. Measured variation between multiple runs of the same application and same input is less than 2%.

on to accurately determine the peak power requirement for a processor, since not all input combinations can be profiled, and the peak power for an unprofiled input could be significantly higher than the peak power observed during profiling. Since input-induced variations change peak power by over 25% for these applications (Figure 7a), a profiling-based approach for determining peak power requirements should apply a guardband of at least 25% to the peak power observed during profiling.

For energy-constrained ULP systems, like those powered by batteries (Type 2 and 3), peak energy as well as peak power determines the size of energy harvesting and storage components (Section 1). Thus, it is also important to determine an accurate bound on the peak energy requirements of a ULP processor. Figure 8 shows the instantaneous power profile for an application ($mult$), demonstrating that on average, instantaneous power can be significantly lower than peak power. Therefore, we can more accurately determine the optimal sizing of components in an energy-constrained system by generating an accurate bound on peak energy, rather than conservatively multiplying peak power by execution time.

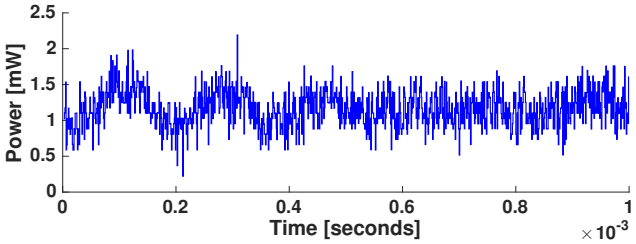


Figure 8: Measured instantaneous power of MSP430F1610 for the $mult$ benchmark is significantly lower, on average, than both the rated and observed peak power for the application.

Figure 7b characterizes the peak energy, normalized to application runtime in cycles, for different applications and input sets, showing that the maximum rate at which an application can consume energy is also application- and input-dependent. Therefore, conventional techniques for determining the peak energy requirements of a ULP processor have the same limitations as conventional techniques for determining peak power requirements. In both cases, the limitations of conventional techniques require over-provisioning that can substantially increase system size and weight.

In the next section, we describe a novel technique for determining the peak power and peak energy requirements of a ULP processor that is application-specific yet also input-independent.

3. Application-Specific Input-indep-endent Peak Power and Energy

Figure 9 provides an overview of our technique for determining application-specific peak power and energy requirements that are input-independent. The inputs to our technique are the application binary that runs on a ULP processor and the gate-level netlist of the ULP processor. The first phase of our technique, described in Section 3.1, is an activity analysis that uses symbolic simulation to efficiently characterize all possible gates that can be exercised for all possible execution paths of the application and all possible inputs. This analysis also reveals which gates can *never* be exercised by the application. Based on this analysis, we perform input-independent peak power (Section 3.2) and energy (Section 3.3) calculations to determine the peak power and energy requirements for a ULP processor.

3.1 Input-Independent Gate Activity Analysis

Since the peak power and energy requirements of an application can vary based on application inputs, a technique that determines application-specific peak power requirements must bound peak power for all possible inputs. Exhaustive profiling for all possible inputs is not possible for most applications, so we have created a novel approach for activity analysis that uses unknown logic values (Xs) for inputs to efficiently characterize activity for all possible inputs with minimum simulation effort.

Our technique, described in Algorithm 1, is based on symbolic simulation [9] of an application binary running on the gate-level netlist of a processor, in which Xs are propagated for all signal values that cannot be constrained based on the application. When the simulation begins, the states of all gates and memory locations that are not explicitly

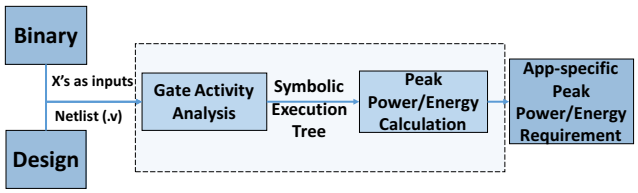


Figure 9: Our technique performs input-independent activity analysis that enables determination of accurate peak power and energy requirements for a ULP processor.

loaded with the binary are initialized to Xs. During simulation, all input values are replaced with Xs by our simulator. As simulation progresses, the simulator dynamically constructs an execution tree describing all possible execution paths through the application. If an X symbol propagates to the inputs of the program counter (PC) during simulation, indicating an input-dependent control sequence, a branch is created in the execution tree. Normally, the simulator pushes the state corresponding to one execution path onto a stack for later analysis and continues down the other path. However, a path is not pushed to the stack or re-simulated if it has already been simulated (i.e., if the simulator has seen the branch (PC) before and the processor state is the same as it was when the branch was previously encountered). This allows Algorithm 1 to analyze programs with input-dependent loops. When simulation down one path reaches the end of the application, an un-simulated state is loaded from the last input-dependent branch in depth-first order, and simulation continues. When all execution paths have been simulated to the end of the application (i.e., depth-first traversal of the control flow graph terminates), activity analysis is complete.[4]

During symbolic simulation, the simulator captures the activity of each gate at each point in the execution tree. A gate is considered active if its value changes or if it has an unknown value (X) and is driven by an active gate; otherwise, the gate is idle. The resulting annotated symbolic execution tree describes all possible instances in which a gate could possibly toggle for all possible executions of the application binary. As such, a gate that is not marked as toggled at a particular location in the execution tree can *never* toggle at that location in the application. As described in the next sections, we can use the information gathered during activity analysis to bound the peak power and energy requirements of an application.

3.2 Input-Independent Peak Power Requirements

The input to the second phase of our technique is the symbolic execution tree generated by input-independent gate activity analysis. Algorithm 2 describes how to use the activity-annotated execution tree to generate peak power requirements for a ULP processor, application pair.

The first step in determining peak power from an execution tree produced during gate activity analysis is to concatenate the execution paths in the execution tree into a single execution trace. We use a value change dump (VCD) file to record the gate-level activity in the execution trace. The execution trace contains Xs, and the goal of the peak power computation is to assign values to the Xs in the way that maximizes power for each cycle in the execution trace. The power of a gate in a particular cycle is maximized when the gate transitions (toggles). Since a transition involves two cycles, maximizing dynamic power in a particular cycle, c, of the execution trace involves assigning values to any Xs in the activity profiles of the current and previous cycles, c and $c-1$, to maximize the number of transitions in cycle c.

The number and power of transitions are maximized as follows. When the output value of a gate in only one of the cycles, c or $c-1$, is an X, the X is assigned the value that assumes that a transition happened in cycle c. When both

Algorithm 1 Input-independent Gate Activity Analysis

1. **Procedure** *Create Symbolic Execution Tree(app_binary, design_netlist)*
2. Initialize all memory cells and all gates in design_netlist to X
3. Load app_binary into program memory
4. Propagate reset signal
5. $s \leftarrow$ State at start of app_binary
6. Symbolic Execution Tree T.set_root(s)
7. Stack of un-processed execution paths, U.push(s)
8. **while** U != $\emptyset$ **do**
9. $e \leftarrow U$.pop()
10. **while** e.PC_next != X **and** !e.END **do**
11. e.set_inputs_X() // set all peripheral port inputs to Xs
12. $e' \leftarrow$ propagate_gate_values(e) // simulate this cycle
13. e.annotate_gate_activity(e,e') // annotate activity in tree
14. e.add_next_state(e') // add to execution tree
15. $e \leftarrow e'$ // process next cycle
16. **end while**
17. **if** e.PC_next == X **then**
18. **for all** $a \in$ possible_PC_next_vals(e) **do**
19. **if** $a \notin T$ **then**
20. $e' \leftarrow e$.update_PC_next(a)
21. U.push(e')
22. T.insert(a)
23. **end if**
24. **end for**
25. **end if**
26. **end while**

Algorithm 2 Input-independent Peak Power Computation

1. **Procedure** *Calculate Peak Power*
2. {E—O}_VCD $\leftarrow$ Open {Even—Odd} VCD File // maximizes peak power in even—odd cycles
3. T $\leftarrow$ flatten(Execution Tree) // create a flattened execution trace that represents the execution tree
4. **for all** {even—odd} cycles c $\in$ T **do**
5. **for all** toggled gates $g \in$ c **do**
6. **if** value(g,c) == X && value(g,c-1) == X **then**
7. value(g,c-1) $\leftarrow$ maxTransition(g,1) // returns the value of the gate in the first cycle of the gate's maximum power transition
8. value(g,c) $\leftarrow$ maxTransition(g,2) // returns the value of the gate in the second cycle of the gate's maximum power transition
9. **else if** value(g,c) == X **then**
10. value(g,c) $\leftarrow$!value(g,c-1)
11. **else if** value(g,c-1) == X **then**
12. value(g,c-1) $\leftarrow$!value(g,c)
13. **end if**
14. **end for**
15. {E—O}_VCD $\leftarrow$ value(*,c-1)
16. {E—O}_VCD $\leftarrow$ value(*,c)
17. **end for**
18. Perform power analysis using E_VCD and O_VCD to generate even and odd power traces, P_E and P_O
19. Interleave even cycle power from P_E with odd cycle power from P_O to form peak power trace, P_{peak}
20. peak power $\leftarrow$ max(P_{peak})

[4] Complex applications and processors might require heuristics for exploration of a large number of execution paths [11, 21]; however, our approach is adequate for ULP systems, which tend to have simple processors and applications. For example, complete analysis of our most complex benchmark takes 2 hours.

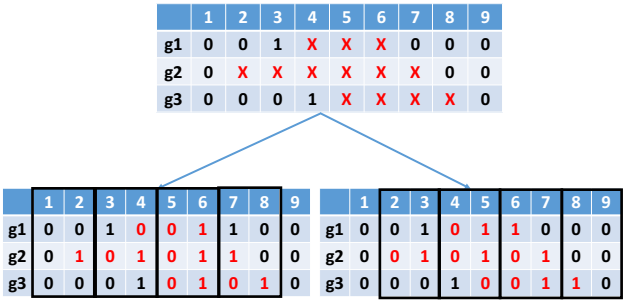


Figure 10: To determine a bound on peak power, we generate two different activity profiles – one that maximizes power in even cycles (left) and one that maximizes power in odd cycles (right).

values are Xs, the values are assigned to produce the transition that maximizes power in cycle c. The maximum power transition is found by a look-up into the standard cell library for the gate. Since constraining Xs in two consecutive cycles to maximize power in the second cycle may not maximize power in the first cycle, we produce two separate VCD files – one that maximizes power in all even cycles and one the maximizes power in all odd cycles. To find the peak power of the application, we first run activity-based power analysis on the design using the even and odd VCD files to generate even and odd power traces. We then form a peak power trace by interleaving the power values from the even cycles in the even power trace and the odd cycles in the odd power trace. This peak power trace bounds the peak power that is possible in every cycle of the execution trace. The peak power requirement of the application is the maximum per-cycle power value found in the peak power trace.[5]

Our VCD generation technique is illustrated in Figure 10. We use the example of three gates with overlapping Xs that need to be assigned to maximize power in every cycle. We show two assignments – one that maximize peak power in all even cycles (left), and one that maximizes peak power in all odd cycles (right). Assuming, for the sake of example, that all gates have equal power consumption and that the $0 \rightarrow 1$ transition consumes more power than the $1 \rightarrow 0$ transition for these gates, the highest possible peak power for this example happens in cycle 6 in the "even" activity trace, when all the gates have a $0 \rightarrow 1$ transition.

3.3 Input-independent Peak Energy Requirements

Our technique generates a per-cycle peak power trace characterizing all possible execution paths of an application. The peak power trace can be used to generate peak energy requirements. Figure 11 shows per-cycle peak power traces sampled from our benchmark applications. Since per-cycle peak power varies significantly over the compute phases of an application, peak energy can be significantly lower than assuming the maximum peak energy (i.e., $peak\ power * clock\ period * number\ of\ cycles$). Instead, the peak energy of an application is bounded by the execution path with the highest sum of per-cycle peak power multiplied by the clock period. To avoid enumerating all execution paths, we

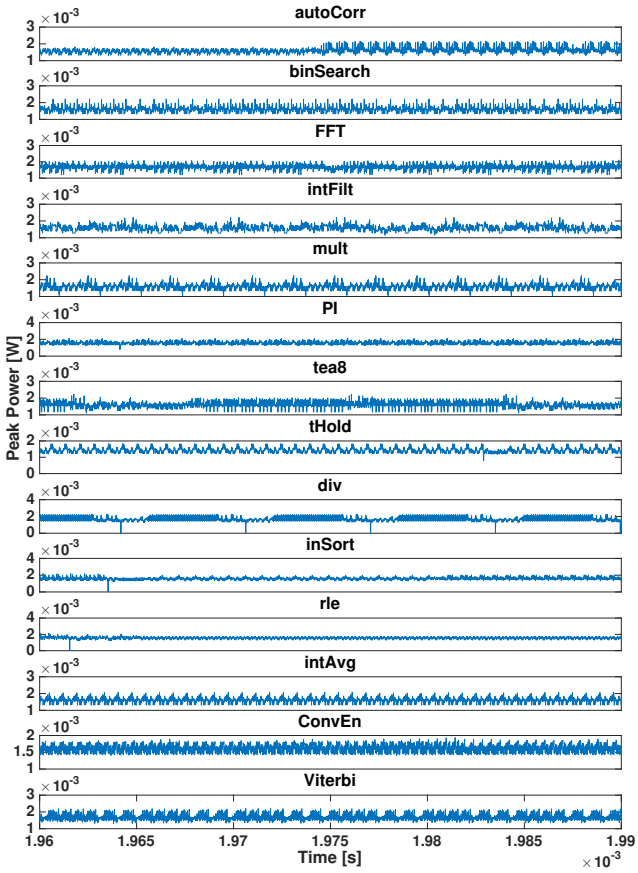


Figure 11: The per-cycle peak power varies significantly over the course of an application, showing that the worst-case average power can be significantly lower than peak power. Therefore, the peak energy can be significantly lower than the product of peak power and application runtime would suggest.

use several techniques. For an input-dependent branch, peak energy is computed by selecting the branch path with higher energy. For a loop whose number of iterations is input-independent, peak energy can be computed as the peak energy of one iteration multiplied by the number of iterations. For cases where the number of iterations is input-dependent, the maximum number of iterations may be determined either by static analysis or user input (as suggested by prior work [27]) [6]. If neither is possible, it may not be possible to compute the peak energy of the application; however, this is uncommon in embedded applications [1].

3.4 Validation of X-based Analysis

To demonstrate that our symbolic execution-based (X-based) activity analysis marks all gates that could possibly be toggled by an application for all possible inputs, we performed a validation check by comparing the sets of gates toggled by input-based simulations for several different input sets against the set of gates marked as potentially-toggled by symbolic simulation. Figure 12 illustrates this comparison for two input-based simulations of the *mult* benchmark with

[5] It is possible that glitching between clock edges can impact the power profile for an application. This impact can be accounted for by Primetime's power analysis [42].

[6] The number of loop iterations is bounded for all evaluated benchmarks. In general, applications with unbounded runtimes are uncommon in embedded domains.

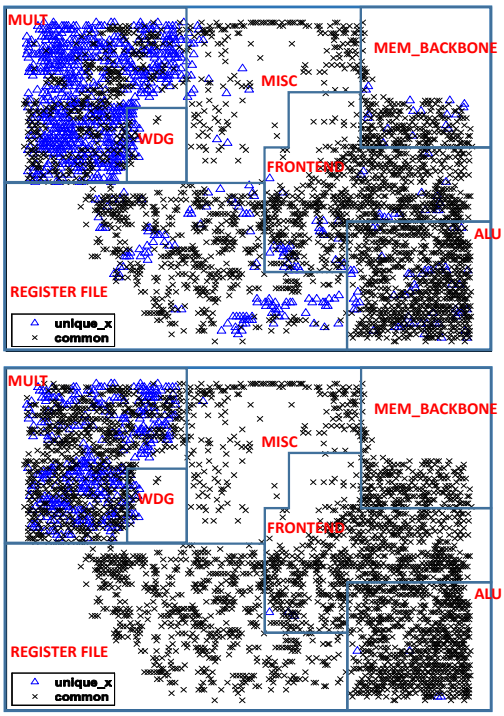


Figure 12: Toggled gates for *mult* with low-activity inputs (top) and high-activity inputs (bottom), compared against potentially-toggled gates identified by X-based analysis. X-based simulation marks all gates that can potentially toggle for an application for all possible inputs. This set of gates (unique_x ∪ common) is a superset of the gates that toggle during an input-based application execution (common).

different input sets – those that have the lowest and highest number of toggled gates. In the figure, toggled gates common to X-based and input-based simulation are shown as Xs, and gates that are exclusively marked by symbolic simulation as potentially-toggled are shown as blue triangles. As expected, there are no gates that are exclusively marked by input-based simulation. Our validation results show that all the gates toggled by input-based simulation are also marked as potentially-toggled by X-based symbolic simulation, validating the correctness of our approach for characterizing toggle activity.

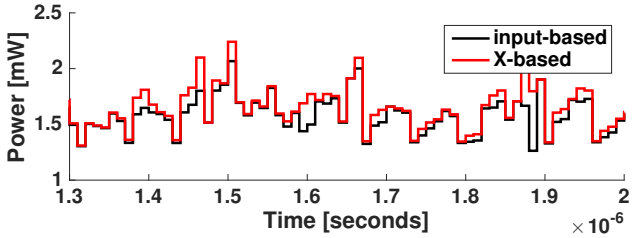


Figure 13: The X-based peak power trace generated by our technique for an application provides an upper bound on all possible input-based power traces for the application. (result shown for $mult$)

We perform a second validation of our technique by comparing the peak power traces generated for benchmarks by our technique against power traces generated by input-based execution of the benchmarks. The validation results confirm that our peak power trace always provides an upper bound on the power of any input-based power trace. Figure 13 shows an example; the X-based peak power trace for the $mult$ application is always higher than the input-based power trace. These validation results also show that the X-based peak power trace closely matches the input-based trace, indicating that the peak power and energy requirements generated by our technique are not overly conservative.

3.5 Enabling Peak Power Optimizations

Since our technique is able to associate the input-independent peak power consumption of a processor with the particular instructions that are in the pipeline during a spike in peak power, we can use our tool to identify which instructions or instruction sequences cause spikes in peak power. Our technique can also provide a power breakdown that shows the power consumption of the microarchitectural modules that are exercised by the instructions. These analyses can be combined to identify which instructions executing in which modules cause power spikes. After identifying the cause of a spike, we can use software optimizations to target the instruction sequences that cause peaks and replace them with alternative sequences that generates less instantaneous activity and power while maintaining the same functionality. After optimizing software to reduce a spike in peak power, we can re-run our peak power analysis technique to determine the impact of optimizations on peak power. Guided by our technique, we can choose to apply only the optimizations that are guaranteed to reduce peak power.

Figure 14 shows an example where our technique identifies peak power spikes in cycles 146 and 150. Our technique also reports the instructions in each stage of the pipeline during those cycles of interest (COIs), as well as the per-module power breakdown for those cycles, which identifies the modules that are consuming the most power. This information can be used to guide optimizations that replace the instructions with different instruction sequences that induce less activity and power in the modules that consume the most power. Since software optimizations can impact performance as well as peak power, we will discuss optimizations that reduce peak power and their impact on performance and energy in Section 5.1.

4. Methodology

4.1 Simulation Infrastructure and Benchmarks

We verify our technique on a silicon-proven processor – openMSP430 [20], an open-source version of one of the most popular ULP processors [8, 46]. The processor is synthesized, placed, and routed in TSMC 65GP technology (65nm) for an operating point of 1V and 100 MHz using Synopsys Design Compiler [41] and Cadence EDI System [12]. Gate-level simulations are performed by running full benchmark applications on the placed and routed processor using a custom gate-level simulator that efficiently traverses the control flow graph of an application and captures input-independent activity profiles (Section 3). We show results for all benchmarks from [48] and all EEMBC benchmarks that fit in the program memory of the processor. These benchmarks are chosen to be representative of emerging ultra-low-power application domains such as wearables, internet of things, and sensor networks [48]. The IPC of these benchmarks on our processor varies from 1.25 to 1.39, with an average of 1.29. Power analysis is performed using Syn-

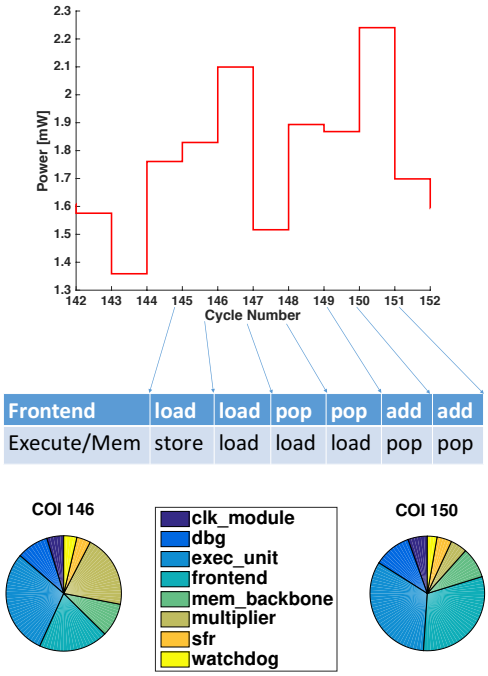


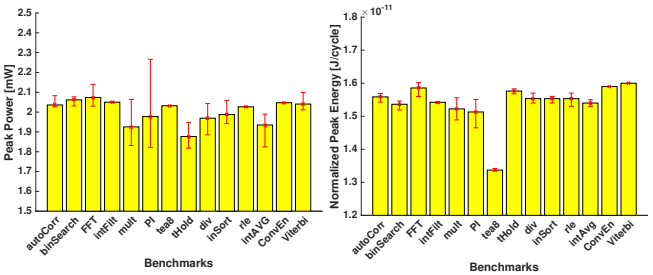
Figure 14: A snapshot of instantaneous power profiles for *mult* at two different COIs where peaks occur. Our technique analyzes the instructions in the pipeline (top) to find each COI's culprit instructions that cause the peak power in each pipeline stage along with the per-module peak power breakdown (bottom) to identify which instructions in which microarchitectural modules are responsible for a peak.

opsys Primetime [42]. Experiments were performed on a server housing two Intel Xeon E-2640 processors (8-cores each, 2GHz operating frequency, 64GB RAM).

Section 2 shows measured data for an MSP430F1610 processor that demonstrate that different applications have different peak power and energy requirements, and the requirements of an application can vary significantly for different inputs. The results motivate an application-specific input-independent technique for determining the peak power and energy requirements for ULP processors. For the results in Section 5, we perform evaluations on the open source openMSP430 processor [20]. Figures 15a and 15b confirm that the peak power and energy requirements of openMSP430 also depend on the application and application inputs. Note that the results in Figure 7 and Figure 15 differ because they are for different implementations of the MSP430 architecture (MSP430F1610 and openMSP430), with different process technology (130 nm vs 65 nm) and operating frequencies (8MHz vs 100 MHz).

4.2 Baselines

For baselines, we compare against conventional techniques for determining the peak power and energy requirements of processors. An overview of the baseline techniques can be found in Figure 4. The design specification-based baseline (*design tool*) is determined by performing power and energy analysis of the design using the default input toggle rate used by our design tools [42]. The stressmark-based



(a) Peak Power **(b)** NPE

Figure 15: Different applications and different input sets for the same application have different peak power and peak energy requirements. (results for openMSP430)

Table 3: Benchmarks

Embedded Sensor Benchmarks [48]
mult, binSearch, tea8, intFilt, tHold, div, inSort, rle, intAVG
EEMBC Embedded Benchmarks [1]
Autocorr, FFT, ConvEn, Viterbi
Control Systems Benchmark
Proportional Integral Controller (PI)

baselines (*GB input-based*) use stressmarks that target peak instantaneous power and average power. Kim et al. used a genetic algorithm to automatically generate stressmarks that target maximum di/dt-induced voltage droop for a microprocessor [28]. We modified their framework to generate stressmarks that target peak instantaneous power and average power for openMSP430. The profiling-based baseline (*input-based*) is generated by performing input-based power and energy profiling for several input sets and applying a guardbanding factor of $4/3$ to the peak power and energy observed during profiling. The guardbanding factor is the same as in prior studies [4, 30] and is appropriate for the input-dependent peak power variability exhibited by our benchmarks (Figure 7a).

5. Results

We use our technique described in Section 3 to determine peak power and energy requirements for a ULP processor for different benchmark applications. Figure 16 compares the peak power requirements reported by our technique against the conventional techniques for determining peak power requirments, described in Section 4.2. The results show that the peak power requirements reported by our X-based technique are higher than the highest input-based application-specific peak power for all applications, confirming that our technique provides a bound on peak power. The results also show that our technique provides the most accurate bound on peak power, compared to conventional techniques for determining peak power requirements. For example, the peak power requirements reported by our technique are only 1% higher than the highest observed input-based peak power for the benchmark applications, on average. Other techniques for determining peak power and energy requirements are significantly less accurate, which can lead to inefficiency in critical system parameters such as size and weight (see Section 1).

Our technique is more accurate than application-oblivious techniques such as determining peak power require-

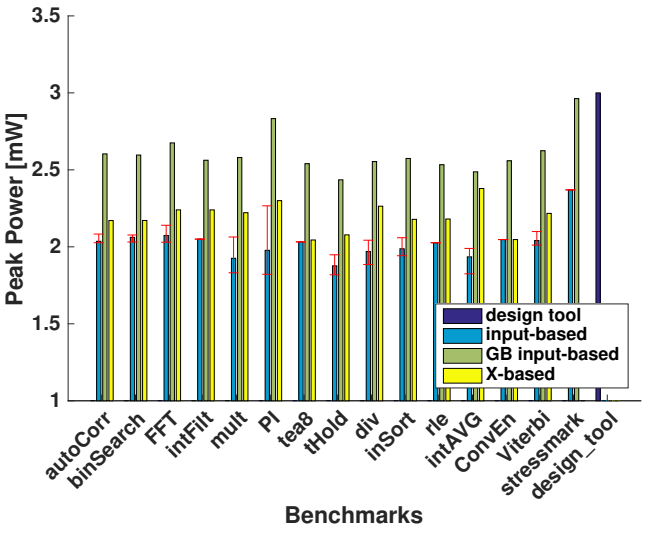


Figure 16: Our X-based technique for determining peak power requirements provides the most accurate (least conservative) guaranteed bound on peak power.

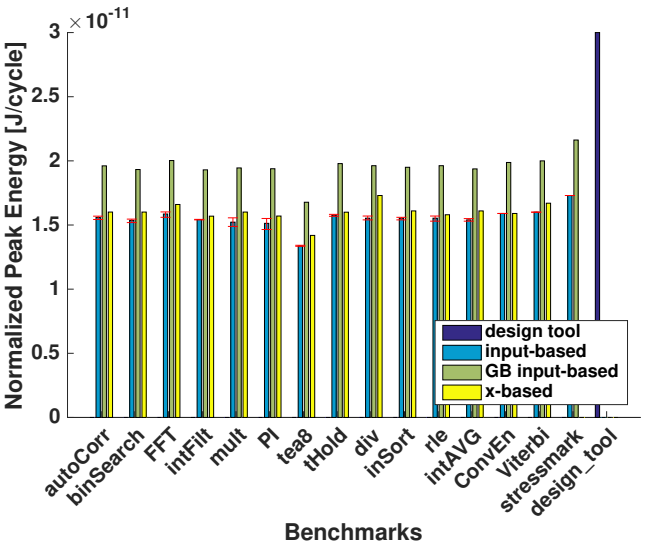


Figure 17: Our X-based technique for determining peak energy requirement (normalized to application run-time in cycles, i.e., the peak average power) is more accurate than existing conventional techniques.

ments from a stressmark or design specification, because an application constrains which parts of the processor can be exercised in a particular cycle. Our technique also provides a more accurate bound than a guardbanded input-based peak power requirement, because it does not require a guardband to account for the non-determinism of input-based profiling (shown in Figure 16 as error bars). By accounting for all possible inputs using symbolic simulation, our technique can bound peak power and energy for all possible application executions without guardbanding. The peak power requirements reported by our technique are 15% lower than guardbanded application-specific requirements, 26% lower than guardbanded stressmark-based requirements, and 27% lower than design specification-based requirements, on average.

Since our technique is application-specific and does not require guardbands, one question is, "Why is the bound provided by X-based analysis more conservative for some applications than others?" The answer is that since X-based analysis provides a bound on power for all possible inputs, it becomes more conservative when there is greater possibility for input-dependent variation in power. For example, the multiplier is a relatively large, high-power module, with high potential for input-dependent variation in power consumption. For some inputs (e.g., $X * 0$), power consumed by the multiplier is minimal, since there are no partial products to compute. For other inputs (e.g., two very large numbers), the power consumed by the multiplier is much larger. Since our symbolic simulation technique assumes Xs for inputs, we always assume the highest possible power for a multiply instruction. Therefore, X-based peak power requirements for applications that contain a large number of multiplications may be more conservative than X-based requirements for other applications.

Conversely, the tea8 application, which performs encryption, only uses low-power ALU modules – shift register and XOR – that have significantly less potential for input-induced power variation. As a result, X-based analysis closely matches input-based profiling results for this application. For all applications, even those with more potential for input-induced power variation, our X-based analysis technique provides a peak power bound that is more accurate than those provided by conventional techniques.

Our technique also provides more accurate bounds on peak energy than conventional techniques, partly because of the reasons mentioned above, and also because our technique is able to characterize the peak energy consumption in each cycle of execution, generating a peak energy trace that accounts for dynamic variations in energy consumption. Using a design specification to determine peak energy is particularly inaccurate, since it does not consider dynamic variations in the energy requirements of an application. The guardbanded input-based technique, which does consider dynamic variations, provides a more accurate peak energy bound than the design specification for all benchmarks. However, it does not always provide a more accurate bound than the design specification for peak power, since peak power is an instantaneous phenomenon that is less dependent on dynamic variations. Figure 17 presents peak energy of different benchmarks, normalized to application runtime in cycles, i.e., peak average power, which characterizes the maximum rate at which the application can consume energy. In Figure 17, the peak energy requirements reported by our technique are 17% lower than guardbanded application-specific requirements, 26% lower than guardbanded stressmark-based requirements, and 47% lower than design specification-based requirements, on average. As expected, application-specific normalized peak energy (Figure 17) varies less than peak power (Figure 16), since peak energy characterizes average peak power over the entire execution of an application, whereas peak power corresponds to one instant in the application's execution.

As described in Section 1, more accurate peak power and energy requirements can be leveraged to reduce critical ULP system parameters like size and weight. For example, reduction in a Type 1 system's peak power requirements al-

Table 4: Percentage reduction in harvester area compared to different baseline techniques, averaged over all benchmarks, for different percentage contributions of the processor peak power to the system peak power.

Baseline	10%	25%	50%	75%	90%	100%
GB-Input	1.49	3.73	7.47	11.21	13.45	14.94
GB-Stress	2.60	6.47	12.95	19.42	23.31	25.90
Design Tool	2.68	6.70	13.41	20.12	24.14	26.82

Table 5: Percentage reduction in battery volume compared to different baseline techniques, averaged over all benchmarks, for different percentage contributions of the processor energy to the overall energy of the system.

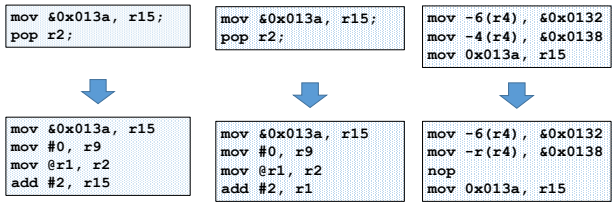
Baseline	10%	25%	50%	75%	90%	100%
GB-Input	1.74	4.37	8.74	13.11	15.73	17.48
GB-Stress	2.59	6.49	12.98	19.48	23.37	25.97
Design Tool	4.66	11.66	23.32	34.98	41.97	46.64

lows a smaller energy harvester to be used. System size is roughly proportional to harvester size in Type 1 systems. In Type 2 systems, it is the peak energy requirement that determines the harvester size; reduction in peak energy requirement reduces system size roughly proportionally. Since required battery capacity depends on a system's peak energy requirement, and effective battery capacity depends on the peak power requirement, reductions in peak power and energy requirements both reduce battery size for Type 2 and 3 systems.

A ULP system may contain other components, such as transmitter/receiver, ADC, DAC, and sensor(s), along with the processor. All of these components may contribute to the system's peak power and energy, and hence, the sizing of the harvester and battery. Tables 4 and 5 show the percentage reduction in the harvester size and battery size, respectively, from our technique for different fractions representing the processor's contribution to the system's peak power and energy. For a real system such as the one shown in Figure 2, which has a harvester area of $32.6cm^2$ and a battery volume of $6.95mm^3$, the area reduction of the harvester is 4.87, 8.44, or $8.75cm^2$ if the system is designed using guardbanded input-based profiling, guardbanded stressmark, or design tool, respectively, for estimating the peak power of the processor. Similarly, the volume reduction of the battery is 0.42, 0.63, or $1.12mm^3$, respectively.[7] As expected, savings from our technique are higher when the processor is the dominant consumer of power and energy in the overall system.[8]

5.1 Optimizations

As discussed in Section 3.5, our technique can be used to guide application-level optimizations that reduce peak power. Here, we discuss three software optimizations, suggested by our technique, that we applied to the benchmark applications to reduce peak power. The optimizations were derived by analyzing the processor's behavior during the cycles of peak power consumption. This analysis involves (a) identifying instructions in the pipeline at the peak, and (b) identifying the power contributions of the microarchitectural modules to the peak power to determine which modules contribute the most.



(a) OPT 1 **(b)** OPT 2 **(c)** OPT 3

Figure 18: Instruction optimization transforms.

The first optimization aims to reduce a peak by "spreading out" the power consumed in a peak cycle over multiple cycles. This is accomplished by replacing a complex instruction that induces a lot of activity in one cycle with a sequence of simpler instructions that spread the activity out over several cycles.

The second optimization aims to reduce the instantaneous activity in a peak cycle by delaying the activation of one or more modules, previously activated in a peak cycle, until a later cycle. For this optimization, we focus on the `POP` instruction, since it generates peaks in some benchmarks. The peaks are caused since a `POP` instruction generates high activity on the data and address buses and simultaneously uses the incrementer logic to update the stack pointer. To reduce the peak, we break down the `POP` instruction into two instructions – one that moves data from the stack, and one that increments the stack pointer.

The third optimization is based on the observation that for some applications, peak power is caused by the multiplier (a high-power peripheral module) being active simultaneously with the processor core. To reduce peak power in such scenarios, we insert a `NOP` into the pipeline during the cycle in which the multiplier is active.

The three optimizations we applied to our benchmarks to reduce peak power are summarized below. The optimizations are shown in Figure 18.

- **Register-Indexed Loads (OPT 1):** A load instruction (`MOV`) that references the memory by computing the address as an offset to a register's value involves several micro-operations – source address generation, source read, and execute. Breaking the micro-operations into separate instructions can reduce the instantaneous power of the load instruction. The ISA already provides a register indirect load operation where the value of the register is directly used as the memory address instead of as an offset. Using another instruction (such as an `ADD` or `SUB`), we can compute the correct address and store it into another register. We then use the second register to execute the load in register indirect mode.
- **POP instructions (OPT 2):** The micro-operations of a `POP` instruction are (a) read value from address pointed to by the stack pointer, and (b) increment the stack pointer by two. `POP` is emulated using `MOV @SP+, dst`. This can be broken down to two instructions –
`MOV @SP, dst` and `ADD #2, SP`.

[7] The battery is a thin film battery of dimensions 5.7mm × 6.1mm × 200 μm (area of $34.7mm^2$). Assuming the height of the battery doesn't change, the corresponding savings in battery area are 6.07, 9.01, and $16.18mm^2$, respectively.

[8] ITRS 2015 projections show that the microcontroller will be the dominant consumer of power in future IoT and IoE systems [2].

• **Multiply (OPT 3):** The multiplier is a peripheral in openMSP430. Data is `MOV`ed to the inputs of the multiplier and then the output is `MOV`ed back to the processor. For a 2-cycle multiplier, all moving of data can be done consecutively without any waiting. However, this involves a high power draw, since there will be a cycle when both the multiplier and the processor are active. This can be avoided by adding a `NOP` between writing to and reading from the multiplier.

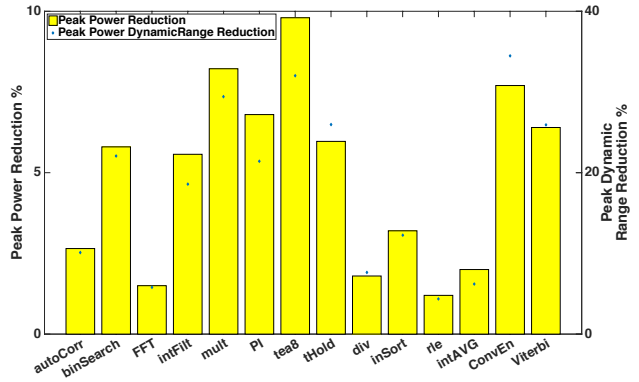


Figure 19: Peak power reduction (left axis) and peak power dynamic range reduction (right axis) achieved by optimizations. These reductions are enabled by our analysis tool and provide further reduction in energy harvester size.

Figure 19 shows the reduction in peak power achieved by applying the optimizations motivated by our technique. Results are quantified in terms of peak power reduction, as well as reduction in peak power dynamic range, which quantifies the difference between peak and average power. Peak power dynamic range decreases as peaks are reduced closer to the range of average power. Reduction in peak power dynamic range can improve battery lifetime in Type 2 and 3 systems, and reduction in peak power requirements can be leveraged to reduce harvester size in Type 1 systems (see Section 1). Our results show that peak power can be reduced by up to 10%, and 5% on average. Peak power dynamic range can be reduced by up to 34%, and 18% on average. Figure 20 shows the peak power traces for an example application before and after optimization, demonstrating that optimization can reduce the peak power requirements for an application.

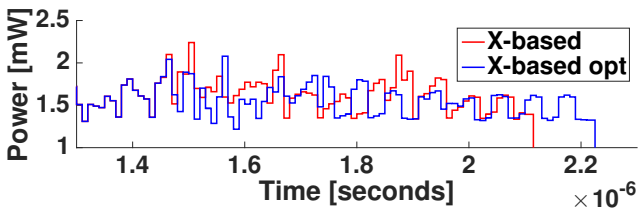


Figure 20: A snapshot of instantaneous power profiles for mult before and after optimization.

Since optimizations that reduce peak power can increase the number of instructions executed by an application, we evaluated the performance and energy impact of the optimizations. Figure 21 shows the results. Applying the optimizations suggested by our technique degrades performance by up to 5% for one application, and by 1% on average. On average, the optimizations increase energy by 3%. Although the optimizations increase energy slightly, they can still enable reduction in size for Type 1 systems, in which harvester size is dictated by peak power, and may also reduce the size of Type 2 and 3 systems, where both peak power and energy determine the size of energy storage and harvesting components (see Figure 3).

Table 6: Microarchitectural features in recent embedded processors

Processor	Branch Predictor	Cache
ARM Cortex-M0	no	no
ARM Cortex-M3	yes	no
Atmel ATxmega128A4	no	no
Freescale/NXP MC13224v	no	no
Intel Quark-D1000	yes	yes
Jennic/NXP JN5169	no	no
SiLab Si2012	no	no
TI MSP430	no	no

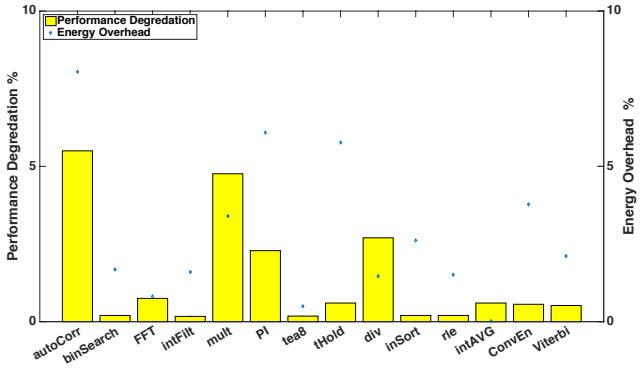


Figure 21: Performance degradation and energy overhead introduced by peak power optimizations is small(average: 1%).

6. Generality and Limitations

We applied our techniques in the context of ULP processors that are already the most widely-used type of processor and are also expected to power a large number of emerging applications [16, 32, 36, 43, 47]. Such processors also tend to be simple, run relatively simple applications, and do not support non-determinism (no branch prediction and caching; for example, see Table 6). This makes our symbolic simulation-based technique a good fit for such processors. Below, we discuss how our technique may scale for complex processors and applications, if necessary.

More complex processors contain more performance-enhancing features such as large caches, prediction or speculation mechanisms, and out-of-order execution, that introduce non-determinism into the instruction stream. Co-analysis is capable of handling this added non-determinism at the expense of analysis tool runtime. For example, by injecting an X as the result of a tag check, both the cache hit and miss paths will be explored in the memory hierarchy. Similarly, since co-analysis already explores taken and not-taken paths for input-dependent branches, it can be adapted to handle branch prediction. In an out-of-order processor, the ordering of instructions is based on the dependence pattern between instructions. Thus, extending input-independent CFG exploration to also explore the data flow graph (DFG) may allow analysis of out-of-order execution.

In other application domains, there exist applications with more complex CFGs. For more complex applications,

heuristic techniques may be used to improve scalability of hardware-software co-analysis. While heuristics have been applied to improve scalability in other contexts (e.g., verification) [11, 21], heuristics for hardware-software co-analysis must be conservative to guarantee that no gate is marked as untoggled when it could be toggled. The development of such heuristics is the subject of future work.

In a multi-programmed setting (including systems that support dynamic linking), we take the union of the toggle activities of all applications (caller, callee, and the relevant OS code in case of dynamic linking) to get a conservative peak power value. For self-modifying code, peak power for the processor would be chosen to be the peak of the code version with the highest peak. In case of fine-grained multi-threading, any state that is not maintained as part of a thread's context is assumed to have a value of X when symbolic execution is performed for an instruction belonging to the thread. This leads to a safe guarantee of peak power for the thread, irrespective of the behavior of the other threads.

Our technique naturally handles state machines that run synchronously with the microcontroller. For state machines that run asynchronously (e.g., ADCs, DACs, bus controllers), we assume the worst-case power at any instant by separately analyzing the asynchronous state machine to compute peak power and energy and adding the values to those of the processor. Asynchronous state machines are generally much smaller than the actual processor, allowing us to not be overly conservative.

A similar approach can be used to handle interrupts. I.e., offset the peak power with the worst power consumed during interrupt detection. The effect of an asynchronous interrupt can be characterized by forcing the interrupt pin to always read an X. Since this can potentially cause the PC to be updated with an X, we can force the PC update logic to ignore the interrupt handling logic's output. This is achieved by monitoring a particular net in the design and forcing it to zero every time its value becomes X. Interrupt service routines (ISRs) are regular software routines and can be analyzed with the rest of the code.

7. Related Work

Peak power has been analyzed in several settings in literature. In particular, several techniques have been proposed to estimate the peak power of a design. Hsiao et al. [22, 23] propose a genetic algorithm-based estimation of peak power for a circuit. Wang et al. [45] use an automatic test generation technique to compute lower and upper bounds for maximum power dissipation for a VLSI circuit. Sambamurthy et al. [38] propose a technique that uses a bounded model checker to estimate peak dynamic power at the module-level. The technique is also functionally valid at the processor level. Najeeb et al. [34] propose a technique that converts a circuit behavioral model to an integer constraint model and employs an integer constraint solver to generate a power virus that can be used to estimate the peak power of the processor. To the best of our knowledge, no prior work exists on determining application-specific peak power for a processor based on symbolic simulation.

The above techniques require a low-level description of the processor (behavioral or gate-level). Techniques have also been proposed at the architecture-level to predict when power exceeds the peak power budget or to lower the peak-to-average power variation. Sartori et al. [39] propose the use of DVFS techniques to manage peak power in a multi-core system. Kontorinis et al. [30] proposed a configurable core to meet peak power constraints with minimal impact on performance. Our technique identifies the peak power and energy requirements of a processor through hardware-software co-analysis.

Estimating peak energy of an application has been previously studied as the worst case energy consumption (WCEC) problem [27, 40, 44]. However, prior techniques do not use accurate power models, instead relying on microarchitectural models, which do not consider the detailed state of a processor or input values. As observed by [33], the power of an instruction can differ based on the previous instructions in the pipeline and its operand values. Our peak power computation technique analyzes an application on a gate-level processor netlist, allowing us to account for the fine-grained interaction between instructions and the worst-case operand values. The result is an accurate power model that can be used for WCEC analyses such as the example analysis in Section 5. Prior work on worst-case timing analysis simply identified the timing-critical path through the program. However, the timing-critical path through a program may not be energy-critical [27, 40]. We calculate energy across all paths through gate-level simulation to determine the path with highest energy.

Symbolic simulation has been applied in circuits for logic and timing verification, as well as sequential test generation [9, 18, 26, 29, 31] and determination of application-specific V_{min} [14]. Symbolic simulation has also been applied for software verification [49]. However, to the best of our knowledge, no existing technique has applied symbolic simulation to determine the peak power and energy requirements of an application running on a processor.

8. Conclusion

In this paper, we showed that peak power and energy requirements for an ultra-low power embedded processor can be application-specific as well as input-specific. This renders profiling methods to determine the peak power and energy of ULP processors ineffective, unless conservative guardbands are applied, increasing system size and weight. We presented an automated technique based on symbolic simulation that determines a more aggressive peak power and energy requirement for a ULP processor for a given application. We show that the application-specific peak power and energy requirements determined by our technique are more accurate, and therefore less conservative, than those determined by conventional techniques. On average, the peak power requirements determined by our technique are 27%, 26%, and 15% lower than those generated based on design specifications, a stressmark, and profiling, respectively. Peak energy requirements generated by our technique are 47%, 26%, and 17% lower, on average, than those generated based on design specifications, a stressmark, and profiling, respectively. We also show that our technique can be used to guide optimizations that target and reduce the peak power of a processor. Optimizations suggested by our technique reduce peak power by up to 10% for a set of benchmarks.

9. Acknowledgments

This work was supported in part by NSF, SRC, and CFAR, within STARnet, a Semiconductor Research Corporation program sponsored by MARCO and DARPA. The authors would like to thank anonymous reviewers and Professor Lizy John for their suggestions and feedback.

References

[1] EEMBC, Embedded Microprocessor Benchmark Consortium. http://www.eembc.org.

[2] International Technology Roadmap for Semiconductors 2.0 2015 Edition Executive Report. http://www.semiconductors.org/main/2015_international_technology_roadmap_for_semiconductors_itrs/.

[3] Microcontroller Sales Regain Momentum After Slump. www.icinsights.com/news/bulletins/Microcontroller-Sales-Regian-Momentum-After-Slump.

[4] Intel corporation: Intel pentium 4 processor in the 423-pin package thermal design guidelines, 2000.

[5] Battery energy. http://www.allaboutbatteries.com/Battery-Energy.html, 2015.

[6] ARM. ARM mbed IoT Device Platform. URL https://www.mbed.com/en/.

[7] H. Blodget, M. Ballve, T. Danova, C. Smith, J. Heggestuen, M. Hoelzel, E. Adler, C. Weissman, H. King, N. Quah, J. Greenough, and J. Smith. The internet of everything: 2015. *BI Intelligence*, 2014.

[8] J. Borgeson. Ultra-low-power pioneers: TI slashes total MCU power by 50 percent with new "Wolverine" MCU platform. *Texas Instruments White Paper*, 2012. URL http://www.ti.com/lit/wp/slay019a/slay019a.pdf.

[9] R. E. Bryant. Symbolic Simulation – Techniques and Applications. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 517–521. ACM, 1991.

[10] I. Buchmann. The Secrets of Battery Runtime. *Battery University*, 2016.

[11] C. Cadar and K. Sen. Symbolic execution for software testing: Three decades later. *Commun. ACM*, 56(2):82–90, Feb. 2013. ISSN 0001-0782. doi: 10.1145/2408776.2408795. URL http://doi.acm.org/10.1145/2408776.2408795.

[12] Cadence. *Encounter Digital Implementation User Guide*. URL http://www.cadence.com/.

[13] B. Calhoun, S. Khanna, Y. Zhang, J. Ryan, and B. Otis. System design principles combining sub-threshold circuit and architectures with energy scavenging mechanisms. In *Circuits and Systems (ISCAS), Proceedings of 2010 IEEE International Symposium on*, pages 269–272, May 2010. doi: 10.1109/ISCAS.2010.5537887.

[14] H. Cherupalli, R. Kumar, and J. Sartori. Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems. In *Computer Architecture (ISCA), 2016 43th Annual International Symposium on*. IEEE, 2016.

[15] C. Compiling. Cloud Compiling. URL http://www.cloudcompiling.com/.

[16] A. Dunkels, J. Eriksson, N. Finne, F. Osterlind, N. Tsiftes, J. Abeillé, and M. Durvy. Low-Power IPv6 for the internet of things. In *Networked Sensing Systems (INSS), 2012 Ninth International Conference on*, pages 1–6. IEEE, 2012.

[17] D. Evans. The internet of things: How the next evolution of the internet is changing everything. April 2011.

[18] T. Feng, L. C. Wang, K.-T. Cheng, M. Pandey, and M. S. Abadir. Enhanced symbolic simulation for efficient verification of embedded array systems. In *Design Automation Conference, 2003. Proceedings of the ASP-DAC 2003. Asia and South Pacific*, pages 302–307, Jan 2003. doi: 10.1109/ASPDAC.2003.1195032.

[19] K. Furset and P. Hoffman. High pulse drain impact on CR2032 coin cell battery capacity. *Nordic Semiconductor and Energizer*, 2011.

[20] O. Girard. OpenMSP430 project. *available at opencores.org*, 2013.

[21] K. Hamaguchi. Symbolic simulation heuristics for high-level design descriptions with uninterpreted functions. In *High-Level Design Validation and Test Workshop, 2001. Proceedings. Sixth IEEE International*, pages 25–30, 2001.

[22] M. S. Hsiao. Peak power estimation using genetic spot optimization for large vlsi circuits. In *Proceedings of the conference on Design, automation and test in Europe*, page 38. ACM, 1999.

[23] M. S. Hsiao, E. M. Rudnick, and J. H. Patel. K2: an estimator for peak sustainable power of vlsi circuits. In *Low Power Electronics and Design, 1997. Proceedings., 1997 International Symposium on*, pages 178–183. IEEE.

[24] N. Instruments. Compile Faster with the LabVIEW FPGA Compile Cloud Service. URL http://www.ni.com/white-paper/52328/en/.

[25] T. Instruments. eZ430-RF2500-SEH Solar Energy Harvesting Development Tool User's Guide. 2013.

[26] P. Jain and G. Gopalakrishnan. Efficient symbolic simulation-based verification using the parametric form of boolean expressions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(8):1005–1015, Aug 1994. ISSN 0278-0070. doi: 10.1109/43.298036.

[27] R. Jayaseelan, T. Mitra, and X. Li. Estimating the worst-case energy consumption of embedded software. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 81–90. IEEE, 2006.

[28] Y. Kim, L. K. John, S. Pant, S. Manne, M. Schulte, W. L. Bircher, and M. S. S. Govindan. Audit: Stress testing the automatic way. In *Proceedings of the 2012 45th Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO-45, pages 212–223, Washington, DC, USA, 2012. IEEE Computer Society. ISBN 978-0-7695-4924-8. doi: 10.1109/MICRO.2012.28. URL http://dx.doi.org/10.1109/MICRO.2012.28.

[29] A. Kolbi, J. Kukula, and R. Damiano. Symbolic RTL simulation. In *Design Automation Conference, 2001. Proceedings*, pages 47–52, 2001. doi: 10.1109/DAC.2001.156106.

[30] V. Kontorinis, A. Shayan, D. M. Tullsen, and R. Kumar. Reducing peak power with a table-driven adaptive processor core. In *Proceedings of the 42Nd Annual IEEE/ACM International Symposium on Microarchitecture*, MICRO 42, pages 189–200, New York, NY, USA, 2009. ACM. ISBN 978-1-60558-798-1. doi: 10.1145/1669112.1669137. URL http://doi.acm.org/10.1145/1669112.1669137.

[31] L. Liu and S. Vasudevan. Efficient validation input generation in RTL by hybridized source code analysis. In *Design, Automation Test in Europe Conference Exhibition (DATE), 2011*, pages 1–6, March 2011. doi: 10.1109/DATE.2011.5763253.

[32] M. Magno, L. Benini, C. Spagnol, and E. Popovici. Wearable low power dry surface wireless sensor node for healthcare monitoring application. In *Wireless and Mobile Computing, Networking and Communications (WiMob), 2013 IEEE 9th International Conference on*, pages 189–195. IEEE, 2013.

[33] J. Morse, S. Kerrison, and K. Eder. On the infeasibility of analysing worst-case dynamic energy. *arXiv preprint arXiv:1603.02580*, 2016.

[34] K. Najeeb, V. Vardhan, R. Konda, S. Kumar, S. Hari, V. Kamakoti, and V. M. Vedula. Power virus generation using behavioral models of circuits. In *VLSI Test Symposium, 2007. 25th IEEE*, pages 35–42, May 2007. doi: 10.1109/VTS.2007.49.

[35] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *IEEE Pervasive Computing*, 4(1): 18–27, Jan 2005. doi: 10.1109/MPRV.2005.9.

[36] C. Park, P. H. Chou, Y. Bai, R. Matthews, and A. Hibbs. An ultra-wearable, wireless, low power ECG monitoring system. In *Biomedical Circuits and Systems Conference, 2006. BioCAS 2006. IEEE*, pages 241–244. IEEE, 2006.

[37] G. Press. Internet of Things By The Numbers: Market Estimates And Forecasts. *Forbes*, 2014.

[38] S. Sambamurthy, S. Gurumurthy, R. Vemu, and J. A. Abraham. Functionally valid gate-level peak power estimation for processors. In *Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design*, pages 753–758. IEEE, 2009.

[39] J. Sartori and R. Kumar. Distributed peak power management for many-core architectures. In *Design, Automation Test in Europe Conference Exhibition, 2009. DATE '09.*, pages 1556–1559, April 2009. doi: 10.1109/DATE.2009.5090910.

[40] K. Seth, A. Anantaraman, F. Mueller, and E. Rotenberg. Fast: Frequency-aware static timing analysis. *ACM Transactions on Embedded Computing Systems (TECS)*, 5(1):200–224, 2006.

[41] Synopsys. *Design Compiler User Guide*, . URL http://www.synopsys.com/.

[42] Synopsys. *PrimeTime User Guide*, . URL http://www.synopsys.com/.

[43] R. Tessier, D. Jasinski, A. Maheshwari, A. Natarajan, W. Xu, and W. Burleson. An energy-aware active smart card. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 13(10):1190–1199, 2005.

[44] P. Wägemann, T. Distler, T. Hönig, H. Janker, R. Kapitza, and W. Schröder-Preikschat. Worst-case energy consumption analysis for energy-constrained embedded systems. In *2015 27th Euromicro Conference on Real-Time Systems*, pages 105–114. IEEE, 2015.

[45] C.-Y. Wang and K. Roy. Maximum power estimation for cmos circuits using deterministic and statistical approaches. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 6(1):134–140, 1998.

[46] Wikipedia. List of wireless sensor nodes, 2016. URL https://en.wikipedia.org/wiki/List_of_wireless_sensor_nodes. [Online; accessed 7-April-2016].

[47] R. Yu and T. Watteyne. Reliable, Low Power Wireless Sensor Networks for the Internet of Things: Making Wireless Sensors as Accessible as Web Servers. *Linear Technology*, 2013. URL http://cds.linear.com/docs/en/white-paper/wp003.pdf.

[48] B. Zhai, S. Pant, L. Nazhandali, S. Hanson, J. Olson, A. Reeves, M. Minuth, R. Helfand, T. Austin, D. Sylvester, et al. Energy-efficient subthreshold processor design. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 17(8):1127–1137, 2009.

[49] Y. Zhang, Z. Chen, and J. Wang. Speculative symbolic execution. In *Software Reliability Engineering (ISSRE), 2012 IEEE 23rd International Symposium on*, pages 101–110, 2012. doi: 10.1109/ISSRE.2012.8.