Enhancing Workload-dependent Voltage Scaling for Energy-efficient Ultra-low-power Embedded Systems

Veni Mohan
University of Minnesota Twin Cities
Minneapolis, Minnesota
mohan073@umn.edu

Akhilesh Iyer
University of Minnesota Twin Cities
Minneapolis, Minnesota
iyerx108@umn.edu

John Sartori
University of Minnesota Twin Cities
Minneapolis, Minnesota
jsartori@umn.edu

ABSTRACT

Ultra-low-power (ULP) chipsets are in higher demand than ever due to the proliferation of ULP embedded systems to support growing applications like the Internet of Things (IoT), wearables and sensor networks. Since ULP systems are also cost constrained, they tend to employ general purpose processors (GPPs) rather than more energy-efficient ASICs, even though they typically run a single application for the lifetime of the system. Prior work showed that it is possible to reduce the operating voltage and thus the power of such systems without reducing the frequency, since the fixed software stack of a system typically only exercises a subset of a processor's paths, and unexercised paths need not meet timing constraints for the system to work correctly. In this context, we find additional scope for power reduction by intelligently optimizing the processor design based on the system's application-specific activity characteristics to allow an even lower safe operating voltage. We demonstrate automated techniques that maximize the application-specific voltage reduction for a system, resulting in 35% additional power savings, on average, compared to the application-specific minimum voltage before optimization and 48% total power savings compared to the original design at nominal voltage.

KEYWORDS

dynamic timing slack, ultra-low-power, embedded systems, internet-of-things

ACM Reference Format:
Veni Mohan, Akhilesh Iyer, and John Sartori. 2018. Enhancing Workload-dependent Voltage Scaling for Energy-efficient Ultra-low-power Embedded Systems. In *DAC '18: DAC '18: The 55th Annual Design Automation Conference 2018, June 24–29, 2018, San Francisco, CA, USA.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3195970.3196046

1 INTRODUCTION

Rapid growth in application areas such as the internet-of-things (IoT), wearables, and embedded sensor networks has driven demand for ultra-low-power (ULP) embedded systems. These systems typically have small form factors, are powered by batteries and energy harvesters, and are characterized by ultra-low power and energy budgets [1–5]. ULP systems are also characterized by a limited software stack – typically one application – that runs repeatedly over a device's lifetime. Given their unique constraints and characteristics, ASICs seem like a natural fit for ULP systems. However, the ubiquitous deployment of these systems means that they are also cost constrained. Thus, most ULP systems are built around general-purpose processors (GPPs), despite their inferior power and performance characteristics. A GPP can handle a wide range of applications, thus resulting in faster time to market and a higher return on investment (RoI). Although ASICs are highly optimized for energy efficiency, they have a substantially lower RoI in most cases due to their lack of generality.

In ULP systems, power and energy requirements are the primary factors that determine critical system parameters such as size, weight, cost, and lifetime [6, 7]. Thus, reducing power and energy requirements can have a significant impact. Since a GPP is designed to handle a large variety of applications, individual applications may only exercise a fraction of a GPP's logic [1, 4, 5]. In fact, not every application exercises the timing-critical paths of a processor that determine its minimum safe operating voltage for a particular timing constraint [8–10]. This presents a unique opportunity wherein some applications can be run at a lower-than-nominal voltage that is guaranteed to be safe, due to workload-dependent dynamic timing slack (DTS) exhibited by the application [9]. The insight behind this opportunity is that paths may safely be allowed to fail timing as long as they are not exercised by an application, since they will continue to produce the same constant output values, despite their inability to propagate a transition within the chip's timing constraints.

The amount of available DTS that can be exploited for an application depends on which paths are exercised by the application and how much timing slack they have. In this paper, we observe that it is possible to increase DTS for a particular system by identifying the paths that the system software can exercise and increasing their timing slack (e.g., through cell upsizing and Vt swapping). This provides additional scope for voltage and power reduction. However, such optimizations also incur an overhead. Thus, they must be performed intelligently in order to minimize the overall power of the optimized design.

In this paper we make the following contributions.

- We propose workload-dependent voltage scaling (WDVS), which extends the idea of exploiting DTS for power savings through the use of application-specific design optimizations that reduce the safe operating voltage of a processor without affecting its performance.
- We provide an automated technique that takes the hardware and software specifications of an arbitrary system, identifies the paths that are exercisable by the application, and applies optimizations on them to enhance DTS.
- We evaluate WDVS for a set of embedded applications on a common embedded processor and show that WDVS can improve power savings by 35%, on average, compared to exploiting DTS in an unmodified design. Average power savings for WDVS are 48% with respect to the original design at nominal voltage.

DAC '18, June, 2018, San Fran., CA, USA

ACM ISBN 978-1-4503-5700-5/18/06…$15.00
https://doi.org/10.1145/3195970.3196046

2 RELATED WORK

Dynamic Timing Slack: In this work, we build upon the previous work of Cherupalli et al. [11], who showed that application-specific dynamic timing slack (DTS) can be exploited to reduce the voltage of a processor running an application without reducing the processor's frequency, while still guaranteeing correct operation of the application. The insight in [11] was to allow the paths that cannot be exercised by an application to fail timing constraints as voltage is scaled down, allowing only the exercisable paths to determine the minimum operating voltage for the application. In our work, we apply focused optimizations on the exercisable paths to increase the amount of DTS, and consequently, the power savings available from exploiting DTS. In Section 5, we compare the benefits of our approach against those of the approach in [11].

Activity-aware slack distribution: To increase the amount of DTS for an application or set of applications, we target a subset of exercisable paths in the design for special optimizations. In their work on recovery-driven design, Kahng et al. [12] performed optimizations on the most exercised critical paths in a design with the goal of reducing the operating voltage of the design. Specifically, their goal was to reshape the design's slack distribution to allow more timing speculative voltage scaling for a particular target error rate that could be tolerated by an error resilience mechanism, such as Razor [13, 14]. Unlike in timing speculative works, our optimizations and voltage scaling are completely non-speculative; an application is guaranteed to execute error-free after scaling to the minimum voltage determined by our approach.

3 WORKLOAD-DEPENDENT VOLTAGE SCALING

Given the ultra-low-power requirements of emerging computing devices like those used in the IoT and the fact that such devices typically have a single captive application, our goal in this work is to minimize the power of a ULP device used in a limited-application setting. We do so by aggressively scaling down the voltage, such that the paths that are not exercisable by a device's captive application are allowed to fail timing constraints. Since the paths will still produce the correct (constant) values for the application, the application is guaranteed to still compute correct outputs.

Reducing voltage reduces power but increases delay. It is possible to provide more headroom for voltage scaling by replacing standard cells in a design's critical paths with faster, leakier versions. By applying such optimizations only on the application-specific dynamic critical paths [9] of a design, we can increase DTS and scale voltage further; however, this optimization increases power and may increase area. Our approach for workload-dependent voltage scaling (WDVS) finds the optimum balance between these opposing techniques to minimize power for a ULP system. It is an automated technique to reduce the operating voltage by optimizing the design for an application, or a set of frequently-exercised applications, without affecting the performance of other applications that may be run on the chip.

The first step is to disable timing constraint checks for paths that can never be exercised by an application by declaring them as false paths. Identification of unexercisable logic in a processor is performed through an input-independent symbolic simulation of hardware and software that propagates unknown logic value symbols for all application inputs (from input ports and memory), thus evaluating all possible executions of the application for all possible inputs and identifying all logic in the processor that can possibly be exercised by the application. This procedure is described in detail in prior literature [11, 15]. Once unexercisable paths are ignored, the standard cells in the remaining (exercisable) timing-critical paths are over-optimized, either by Vt swapping (which replaces a cell with a version that is faster but leakier) or by up-sizing them (which increases drive strength, I_{ds}, the drain current of the transistor) to increase timing slack of these paths. The result is a design that can run the target application(s) at a higher frequency for the same minimum safe operating voltage. Alternatively, it can deliver the original target performance at a lower voltage.

Effectively, we are leveraging the additional timing slack generated in the application-specific critical paths (DTS) to reduce the on-chip operating voltage. This reduces power across the entire chip – both the exercised and unexercised parts. Note that this technique does not negatively impact the performance or minimum operating voltage for non-targeted applications, though there may be some power overhead for such applications (we investigate this in Section 5). However, these considerations may not be important in most ULP systems, since they typically only run one application.

To ensure that the optimized design retains the mapping between each gate and its activity profile obtained from symbolic co-simulation, our optimization tool only performs Vt swaps and/or standard cell up-sizing, as opposed to equally-popular techniques such as net splitting / load splitting or logic decomposition. Such optimizations could be integrated into our tool flow, but for clarity, we relegate them to future work (Section 6). Since the power and timing analysis tool (Synopsys Primetime, in our case) is instructed to only perform cell swaps (Vt swaps and upsizing) during optimization, there is not much difference in the placement of the cells after optimization. As a result, the interconnect RC (resistance and capacitance) values in the design do not change significantly. We take advantage of this similarity and use the same interconnect RC for timing and power calculation during WDVS, which significantly reduces algorithm runtime. To validate the accuracy of this approximation, we have generated an interconnect RC file (using Cadence Encounter after implementing the Engineering Change Order (ECO) obtained from Primetime) for a design netlist optimized for the `mult` benchmark and used this file for slack and minimum power calculation at the minimum operating voltage obtained from WDVS. They each differ by only about 1.5% from the values obtained by using the original interconnect RC file for analysis, which confirms the validity of our approximation.

Optimization Procedure:

To minimize the power of a design for a particular application, WDVS makes explorations at increasingly-aggressive performance targets by first increasing DTS on the dynamic critical paths in a design as much as possible, then reducing the operating voltage until one of the exercisable paths violates the original design timing constraints. The goal is to determine the performance target and corresponding minimum safe voltage that minimize total power. Figure 1 illustrates the WDVS algorithm.

The WDVS algorithm first (a) optimizes the performance of exercisable logic in the design for a tighter timing constraint to generate additional DTS then (b) reduces voltage, while still maintaining non-negative slack. Part (a) increases the power of the design while part (b) reduces power. Thus, power may either increase or decrease from one performance step of the algorithm to the next, depending on whether the optimizations performed enabled a large enough

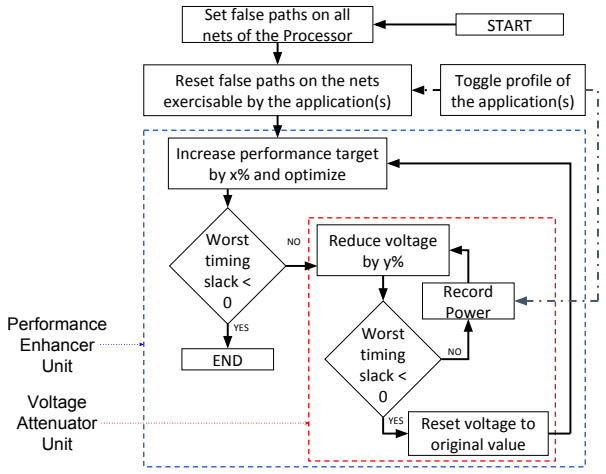

Figure 1: Our WDVS procedure minimizes the power of a design for a particular workload through alternating performance enhancement and voltage attenuation explorations.

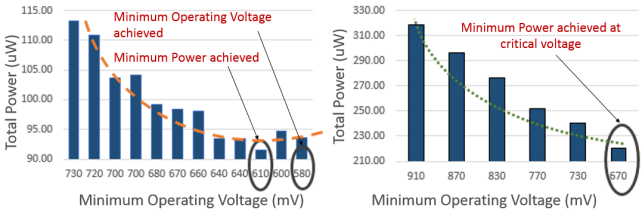

Figure 2: Performance enhancement and voltage attenuation find the sweet spot that minimizes total power. Left and right sub-figures correspond to performance targets of 50 MHz and 100 MHz, respectively.

voltage reduction to offset their associated overhead. Empirical evaluations show that there exists a sweet spot for the operating voltage, at which overall power is minimum, and beyond this point any further attempt to optimize the design and reduce voltage leads to an increase in power. Hence, the algorithm alternates between performance enhancement and voltage reduction to find the performance target and voltage at which power is minimized.

Figure 2 shows how power changes as the processor is iteratively optimized to minimize power for the `mult` benchmark. The left and right sub-figures correspond to designs optimized for 50 MHz and 100 MHz, respectively. The figure shows that the minimum achievable voltage does not necessarily correspond to the minimum power. For example, in the design optimized for 50 MHz, the minimum achievable operating voltage is 580 mV, but the minimum power consumption (91.5 μW) is observed at 610 mV. Since delay increases roughly exponentially with reduction in voltage, optimizations, which increase leakage current and load capacitance, must fight harder to overcome the delay increase of voltage reduction at lower voltages. While our optimizations suffer less from this effect, since we only optimize the exercisable paths in a design, the overall result is that optimizing the design to enable WDVS reduces power to a point, after which the optimization cost of further voltage reductions is too high. In the design optimized for 100 MHz, due to the tighter performance constraint, the minimum achievable voltage was hit before the delay increase resulting from voltage reduction became too steep. Thus, there is no parabolic upswing in the power vs. voltage curve for that performance target.

The detailed pseudocode for WDVS is presented in Algorithm 1. The inputs to the algorithm are the processor netlist and the application-specific activity profile(s) for the target application(s), obtained from symbolic co-simulation [11]. Initially, the processor netlist is designed to operate at the nominal voltage (V_{Nom}), at which all applications can safely operate on the processor and meet performance constraints. To focus on the exercisable paths in the design, all paths that include an application-specific unexercisable net are declared as false paths. In case of optimization for multiple applications, the intersection of the sets of unexercisable nets are used. Next, a performance enhancement phase begins; the target performance is raised by a certain percentage (Δ), and the design is optimized to meet this constraint at the nominal voltage. If the worst timing slack after this step is non-negative, a voltage attenuation phase begins; the operating voltage is iteratively lowered until the worst timing slack is negative, and the minimum safe voltage and corresponding power are recorded for the target application(s) along with the optimized netlist. Once the voltage attenuation phase is complete, control returns to the performance enhancement phase, and an exploration is made at a higher performance target. This process continues until the performance enhancer fails to optimize the design for the selected performance target, in which case the algorithm terminates. Upon termination, the algorithm selects the optimized netlist with the minimum power for the target application(s) among all the candidates. When a target application is run on the design, the voltage is set to the application's minimum safe voltage.

Algorithm 1 WDVS Design Optimization

1. **Procedure** *WDVS(design_netlist N, app_activity_profiles P, app_list A)*
2. **for all** app $\in A$ **do**
3. Parse app_activity_profile to extract the list of exercisable nets, a.push(app.toggled_nets)
4. **end for**
5. $f \leftarrow$ Target frequency
6. $V_{Nom} \leftarrow$ Nominal voltage
7. set_false_paths(design_netlist)
8. reset_false_paths(design_netlist, a) // remove the false path attribute on the toggled nets in a
9. $s \leftarrow$ report_worst_slack(design_netlist)
10. **while** s >= 0 **do**
11. $f \leftarrow f * (1 + \Delta)$
12. optimize_design(design_netlist, f, v_{Nom}) // optimize the design for f at nominal voltage
13. $s \leftarrow$ report_worst_slack(design_netlist) // get the worst slack in the optimized design
14. **if** s >= 0 **then**
15. write_netlist(design_netlist) // dump the optimized netlist
16. **for all** app $\in A$ **do**
17. $v \leftarrow$ find_min_voltage(design_netlist) // calculate the min operating voltage for the optimized netlist
18. $p \leftarrow$ report_power(design_netlist, v) // calculate the power in the optimized netlist at voltage v
19. M_app.push([v, p, design_netlist]) // Create a different stack for each application
20. **end for**
21. **end if**
22. **end while**
23. find_min_power(M_app) // get the netlist with minimum power. In case of multiple applications, get the netlist with minimum average power across all applications.

4 EXPERIMENTAL METHODOLOGY

Timing and power simulations are run on a post-routed netlist for the openMSP430 processor [16], aggressively optimized for a frequency target of 100 MHZ at a nominal voltage of 1V. MSP430 is one of the most popular ULP processors used today [17, 18]. The processor is synthesized, placed, and routed with TSMC 65GP (65nm) technology using Synopsys Design Compiler [19] and Cadence EDI System [20]. We generated libraries for each voltage between 1.0V and 0.5V at 0.01V intervals using Cadence Library

Table 1: Application-specific minimum voltage and minimum power for Nominal, DTS [11] and WDVS.

Benchmark	DTS [11] Minimum Voltage (mV)	WDVS Minimum Voltage (mV)	Nominal Power @ 1V (μW)	DTS [11] Power (μW)	WDVS Power (μW)
mult	950	670	392.7	340.3	220.3
FFT	890	660	498.4	384.4	247.8
autocorr1	950	670	510.2	454.1	287.5
binSearch	950	670	510.2	454.1	287.5
div	890	660	524.0	404.6	266.7
inSort	890	690	484.2	373.3	254.1
intAVG	890	650	483.6	372.9	235.1
intFilt	890	650	483.6	372.9	235.1
rle	890	660	513.3	396.0	258.1
tHold	890	660	448.3	345.1	233.0
tea8	890	660	508.7	392.5	259.1

Table 2: Minimum power (μW) when each benchmark is run on a design optimized for one or more applications.

Benchmark	mult	FFT	autocorr1	inSort	intAVG	rle	tHold
mult	220.3	307.3	307.3	319.3	318.1	335.4	335.4
FFT	434.3	247.8	561.8	329.3	274.8	368.3	249.3
autocorr1	231.0	280.0	287.5	274.7	264.0	288.5	246.7
inSort	413.1	274.3	539.4	254.1	282.8	300.4	247.3
intAVG	434.7	255.3	561.3	302.4	235.1	346.2	230.4
rle	441.9	250.6	569.8	261.8	251.7	258.1	226.3
tHold	445.0	264.7	576.6	295.9	274.9	320.3	223.3
mult+rle	223.1	271.7	318.4	266.1	256.0	279.9	238.9
Intersection	270.8	319.5	346.0	330.8	322.1	339.1	302.0
Union	230.6	279.8	287.1	274.5	263.6	288.7	246.5

Characterizer. Power and timing analyses are performed using Synopsys Primetime [21]. We show results for all benchmarks from [22] and for a selection of EEMBC [23] benchmarks that fit in the memory of the processor. Benchmark applications are chosen to represent emerging ultra-low-power application domains such as wearables, IoT, and sensor networks [22]. Activity profiles for the benchmarks are generated using the input-independent application analysis algorithm described in [11] and [15]. The WDVS tool is implemented in Cshell and TCL. Experiments were performed on a server housing two Intel Xeon E-2640 processors (8-cores each, 2GHz operating frequency, 64GB RAM).

5 RESULTS

To evaluate the benefits of our WDVS approach for ULP systems, we compare the minimum voltage and power achieved by WDVS against the approach for exploiting DTS described in [11]. Table 1 shows the minimum safe voltage for each benchmark application, as well as the corresponding minimum power consumption achieved by both techniques. Optimizing the processor to expose more DTS allows WDVS to achieve significantly higher power savings. Figure 3 compares power savings for WDVS against the DTS baseline and against operation at the nominal voltage. On average, WDVS enables a 27% lower operating voltage, resulting in a 35% reduction in power consumption with respect to baseline DTS exploitation. Compared to operation at nominal voltage, the power of a WDVS design is, on average, nearly cut in half. Power savings are up to 37% compared to the DTS baseline, and up to 51% compared to operation at the nominal voltage, with a corresponding voltage reduction of 350mV. Since WDVS can use cells with higher drive strength to optimize dynamic critical paths, it may increase area. Average area overhead across all benchmark applications is 5%; however, power savings are significant despite this small amount of area overhead.

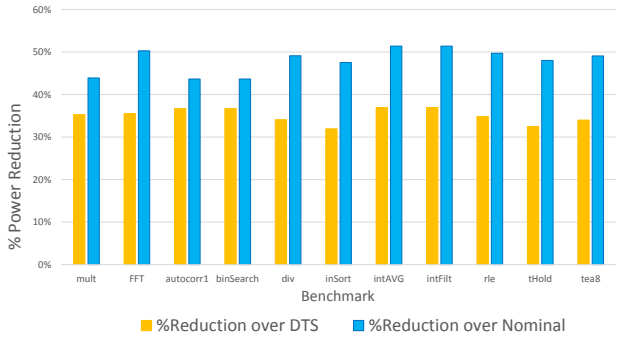

Figure 3: By optimizing a design to exploit DTS, WDVS reduces power significantly with respect to a baseline design that operates at nominal voltage, and even with respect to an aggressive baseline that exploits DTS.

Optimizing a design to increase DTS results in significantly lower power. In the future, market forces such as the IoT revolution, which demands billions of application-specific ULP processors [24–26], and technologies such as printed electronics [27, 28], which offers the possibility of cheaply printing a processor optimized for a particular application, may make the scenario of optimizing a general purpose IP design for particular target applications [15] more common. However, even general purpose processors are optimized for a typical workload of benchmark applications, at least at the architecture level. Our approach enables additional design-level optimization for the target applications for a design and is particularly relevant for ULP designs that are expected to support a small number of applications or common computational kernels. Notably, our design still retains its general-purpose nature, maintains the same functionality as the original design, and does not degrade performance for other applications run on the design, even after optimization. As such, it is important to explore the impact of application-specific optimizations on the power characteristics of other applications that may be run on the chip. Table 2 shows the minimum power for a processor that is optimized for the application denoted by the row label when it runs the application denoted by the column label. For each minimum power value, we used the voltage attenuation procedure described in Section 3 to determine the minimum safe voltage for the application.

Inspection of the main diagonal of the table verifies that power is minimized for an application when the processor is optimized for that application alone. Further inspection of the table reveals other interesting insights. For instance, optimizing for one application may reduce power for other applications as well. As an example, a processor optimized for `mult`, `binSearch`, or `autocorr1` also reduces power when running any of the other benchmark applications. This is evident in Figure 4, which shows the percentage change in power for each of the benchmarks running on each of the designs optimized for a specific application. The reason for this synergistic behavior is that when two applications have dynamic critical paths in common, optimizing the dynamic critical paths of one application can allow both applications to achieve a lower operating voltage, essentially "killing two birds with one stone". For example, `mult` and `FFT` both use the processor's hardware multiplier, which contains several dynamic critical paths for the applications. Thus, a processor optimized for `mult` exposes significantly more DTS for FFT as well.

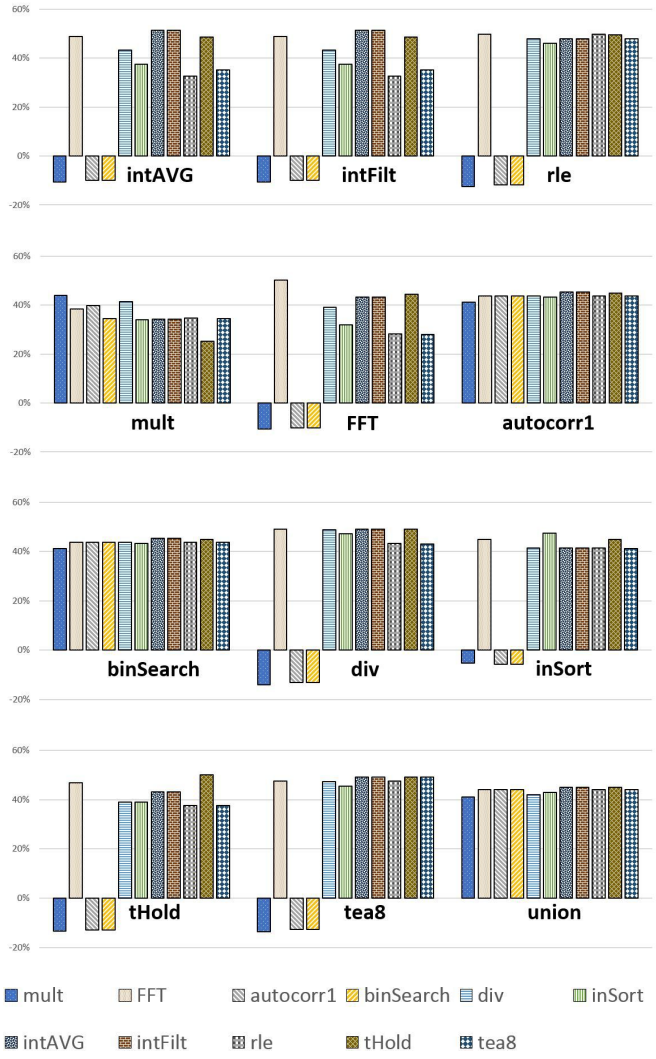

Figure 4: The sub-figures show the power reduction (%) when each application runs at its minimum safe voltage on a design optimized for the application denoted in the sub-figure. In some cases, running an application on a design optimized for a different application results in negative power savings.

As another example, the `autocorr1` application has the most exercisable gates of any application in our benchmark set. In fact, the sets of exercisable gates for all other benchmarks are close to being subsets of the exercisable gates for `autocorr1`. Thus, optimizing DTS for `autocorr1` also optimizes most of the dynamic critical paths for all other applications. As such, all applications achieve power reductions on the processor optimized for `autocorr1`. This can also be seen in Figure 5, which shows Venn diagrams of the exercisable gates for several application pairs. As seen in the figure, the exercisable gates for `autocorr1` almost completely encompass the exercisable gates for `mult` and `rle`, covering over 99% of the exercisable gates for `mult` and 96% for `rle`. The situation is similar for the other applications.

It can also be seen in Figure 4 that optimizing for one application can have a negative impact for another application. Interestingly, although optimizing for `mult` helped `FFT` (as noted above), optimizing for `FFT` actually hurts `mult`. This is because `mult` exercises longer critical paths in the hardware multiplier than `FFT`, so optimizing the dynamic critical paths for `FFT` adds overhead to the design, increasing the power, but fails to optimize the paths necessary for `mult` to a achieve a lower voltage. This behavior can be explained further with a Venn diagram in Figure 5. About 90% of the exercisable gates for `FFT` are also exercisable by `mult`, including the gates belonging to `FFT`'s dynamic critical paths. However, there are many exercisable gates for `mult` that are not exercisable by `FFT`, including gates that belong to the dynamic critical paths for `mult`. As can be seen in Table 2, this also means that optimizing for `mult`, while beneficial for `FFT`, is not as good as optimizing for `FFT` itself, since optimizing for `mult` performs many optimizations that are unnecessary for `FFT`.

The last three rows of Table 2 correspond to designs that are optimized for multiple benchmarks. For instance, the design labeled '`mult+rle`' is expected to provide maximum power benefits for systems running these two applications for a majority of their lifetime. This is validated in Table 3. The designs optimized for benchmarks listed in the first column each run `mult` and `rle` for equal amounts of time. Thus, average power is calculated as the average of the power values for each application.[1] The design optimized for both `mult` and `rle` has lower power than the designs optimized for either application alone. While optimizing to maximize DTS for multiple applications improves average power, it degrades the minimum power for the applications when considered individually, since the design is over-optimized for the individual applications, which exercise only a subset of the gates in the union set. This also explains why WDVS has benefits over exploiting DTS on the original design. Since the original design considers all gates as equals when optimizing the design, the resulting design is over-optimized for any individual application that only exercises a subset of the gates in the design.

The last two rows of Table 2 represent designs that are optimized for all the applications in our benchmark set. One design optimizes the intersection of exercisable gates, and the other exercises the union. These designs are expected to have good power characteristics for a processor that runs all the applications roughly equally. The intersection design fails in this regard, because for every application, it fails to optimize gates that are in the application's critical paths. Thus, the applications have limited DTS. The union design, on the other hand, provides significant benefits for all applications, as can be seen in the last sub-figure of Figure 4. This result suggests that even enhancing DTS for a large set of applications (e.g., the entire benchmark set used to characterize a general-purpose processor during its design) can have significant power benefits over exploiting DTS on the unoptimized design. Since our optimizations only incur a relatively small area overhead and do not negatively impact performance for any application, this result may motivate the use of our WDVS optimization approach even for applications that are expected to run a large number of applications.

6 CONCLUSION AND FUTURE WORK

Ultra-low-power chipsets are in higher demand than ever due to applications like IoT, wearables, and sensor networks. In this work,

[1] In the event that there are multiple applications run unequally on the chip, the design can be optimized using a power metric that sums the power of each application, weighted by its fraction of the total runtime.

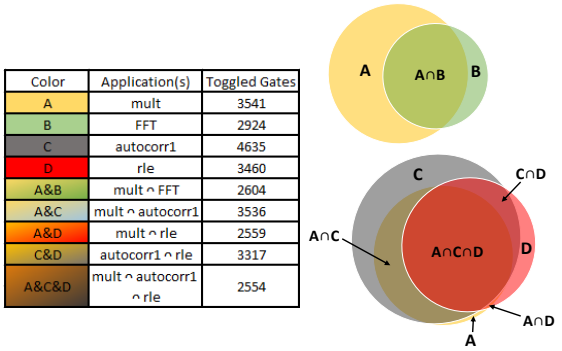

Figure 5: Venn diagrams show the intersection of exercisable gates for different applications. The extent to which optimizing for one workload reduces power for another depends on the number of common and unique exercisable gates and the timing-criticality of the paths to which they belong.

Table 3: Minimum power (μW) for designs that run `mult` and `rle` for equal amounts of time.

Benchmark	mult	rle	Average Power=(mult+rle)/2
mult	220.3	335.4	277.9
rle	441.9	258.1	350.0
mult+rle	223.1	279.9	251.5

we proposed design optimizations that target the workload-specific critical paths in a design to reduce the power of ULP systems by exposing more DTS and allowing them to operate at lower voltages for the same performance, while still guaranteeing correct application behavior. Compared to exploiting DTS without our optimizations, we demonstrated 35% additional power savings, on average, and power savings are up to 51% compared to the baseline openMSP430 design operating at nominal voltage.

There is considerable scope to build upon the work done in this paper, including the following future work directions.

• **De-optimization of the paths not used by any application**: This would further reduce leakage power and also reduce area, potentially resulting in area reduction rather than overhead. However, the general-purpose nature of the design would be somewhat degraded, as performance may be degraded for non-target applications.

• **WDVS-aware compiler optimizations**: Compiler optimizations can tune software to avoid activity on dynamic critical paths and expose more DTS.

• **WDVS-aware architecture-level optimizations**: Many parameters of the processor, such as cache memory, network bus width, pipeline depth, etc. can be optimized to leverage more control over dynamic critical paths and improve DTS for the target set of applications.

• **Load splitting/net splitting**: Net splitting would lead to introduction of new cells in the design, which would in turn lead to netlist-activity profile mismatch. However, it may do a better job at recovering power than the other two optimizations used in this paper. Optimizations that introduce new cells can be supported by regenerating the activity profile of the processor each time new cells are introduced, potentially reducing minimum power at the expense of increased simulation runtime.

REFERENCES

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

[2] Ross Yu and Thomas Watteyne. Reliable low power wireless sensor networks for the internet of things: Making wireless sensors as accessible as web servers.

[3] A. Dunkels, J. Eriksson, N. Finne, F. ÃŰsterlind, N. Tsiftes, J. AbeillÃĪ, and M. Durvy. Low-power ipv6 for the internet of things. In *2012 Ninth International Conference on Networked Sensing (INSS)*, pages 1–6, 2012.

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

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

[6] B. H. Calhoun, S. Khanna, Y. Zhang, J. Ryan, and B. Otis. System design principles combining sub-threshold circuit and architectures with energy scavenging mechanisms. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 269–272, May 2010.

[7] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Determining application-specific peak power and energy requirements for ultra-low power processors. *SIGARCH Comput. Archit. News*, 45(1):3–16, April 2017.

[8] Giang Hoang, Robby Bruce Findler, and Russ Joseph. Exploring circuit timing-aware language and compilation. In *Proceedings of the Sixteenth International Conference on Architectural Support for Programming Languages and Operating Systems ASPLOS XVI*, pages 345–356, 2011.

[9] J. Sartori and R. Kumar. Compiling for energy efficiency on timing speculative processors. In *DAC Design Automation Conference 2012*, pages 1297–1304, June 2012.

[10] J. Constantin, L. Wang, G. Karakonstantis, A. Chattopadhyay, and A. Burg. Exploiting dynamic timing margins in microprocessors for frequency-over-scaling with instruction-based clock adjustment. In *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 381–386, March 2015.

[11] H. Cherupalli, R. Kumar, and J. Sartori. Exploiting dynamic timing slack for energy efficiency in ultra-low-power embedded systems. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 671–681, June 2016.

[12] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori. Slack redistribution for graceful degradation under voltage overscaling. In *2010 15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 825–831, Jan 2010.

[13] D. Ernst, Nam Sung Kim, S. Das, S. Pant, R. Rao, Toan Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: a low-power pipeline based on circuit-level timing speculation. In *Proceedings. 36th Annual IEEE/ACM International Symposium on Microarchitecture, 2003. MICRO-36.*, pages 7–18, Dec 2003.

[14] D. Blaauw, S. Kalaiselvan, K. Lai, W. H. Ma, S. Pant, C. Tokunaga, S. Das, and D. Bull. Razor ii: In situ error detection and correction for pvt and ser tolerance. In *2008 IEEE International Solid-State Circuits Conference - Digest of Technical Papers*, pages 400–622, Feb 2008.

[15] Hari Cherupalli, Henry Duwe, Weidong Ye, Rakesh Kumar, and John Sartori. Bespoke processors for applications with ultra-low area and power constraints. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 41–54, New York, NY, USA, 2017. ACM.

[16] O Girard. Openmsp430 project, 2013.

[17] Wikipedia. List of wireless sensor nodes, 2016. [Online; accessed 7-April-2016].

[18] Jacob Borgeson. Ultra-low-power pioneers: TI slashes total MCU power by 50% with new "Wolverine" MCU platform. *Texas Instruments White Paper*, 2012.

[19] Synopsys. Synopsys design compiler user guide.

[20] Cadence. Encounter digital implementation user guide.

[21] Synopsys. Synopsys primetime user guide.

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

[23] Embedded Microprocessor Benchmark Consortium. EEMBC. http://www.eembc.org, 2017.

[24] Henry Blodget, Marcelo Ballve, Tony Danova, Cooper Smith, John Heggestuen, Mark Hoelzel, Emily Adler, Cale Weissman, Hope King, Nicholas Quah, John Greenough, and Jessica Smith. The internet of everything: 2015. *BI Intelligence*, 2014.

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

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

[27] K. Myny, E. van Veenendaal, G. H. Gelinck, J. Genoe, W. Dehaene, and P. Heremans. An 8b organic microprocessor on plastic foil. In *2011 IEEE International Solid-State Circuits Conference*, pages 322–324, Feb 2011.

[28] BK Charlotte Kjellander, Wiljan TT Smaal, Kris Myny, Jan Genoe, Wim Dehaene, Paul Heremans, and Gerwin H Gelinck. Optimized circuit design for flexible 8-bit rfid transponders with active layer of ink-jet printed small molecule semiconductors. *Organic Electronics*, 14(3):768–774, 2013.