Bit Serializing a Microprocessor for Ultra-low-power

Matthew Tomei
University of Illinois
tomei2@illinois.edu

Henry Duwe
University of Illinois
duweiii2@illinois.edu

Nam Sung Kim
University of Illinois
nskim@illinois.edu

Rakesh Kumar
University of Illinois
rakeshk@illinois.edu

ABSTRACT

Many emerging sensor applications are powered by energy harvesters that impose strict power constraints. These applications often do not require high performance or energy efficiency. We explore a technique for minimizing power of a microprocessor for power constrained applications: *bit serial computing*. Bit serial computing promises power benefits up to the data width for fully bit serializable logic. We perform a best-effort bit serialization of the openMSP430 microprocessor without making instruction set architecture (ISA) modifications. Although it is very challenging to serialize much of the logic in the microprocessor, we show that power benefits of serialization exceed 42% when the serial and parallel designs synthesized for their maximum operating frequency are running at a low duty cycle. Benefits are expected to be higher when ISA modifications are allowed.

Keywords

bit serial, ultra-low-power

1. INTRODUCTION

Many emerging sensor applications are powered by energy harvesters. Widespread examples of energy harvesting include inductive coupling [7, 9] and solar cells [14, 2]. As long as an energy harvester has access to its external power source (e.g., an RFID is within range of a base station), the harvester effectively acts as an infinite energy source during the application's lifetime. Unfortunately, energy harvesters provide very low wattages, on the order of mili or even micro-watts per square centimeter [12, 14], and power delivered may vary greatly with time.

To cope with these constraints, a new paradigm of *power-neutral computing* has been proposed [3]. Power-neutral computing aims to match the system power consumption, rather than energy consumption, to the power provided. Designing for power neutrality obviates the need for energy storage devices which waste energy through conversion inefficiencies and consume valuable area. On the other hand, the system must tolerate intermittent loss of power. Depending on the power delivery profile, a technique to decrease power consumed can reduce the length of the powered off periods. Since decreasing the power consumption can increase the duty cycle of the system, this class of techniques has the potential to increase the throughput over the lifetime of the system *even at the expense of increased energy per operation and/or decreased performance*.

In addition, previous work has shown that existing low-power processors over-perform on many sensor benchmarks [5]. Even after the processor was made energy optimal, it continued to over-perform. Coupled with the previously discussed opportunity for increasing throughput at the expense of energy and performance, this over-performance implies an opportunity for area and power reduction. While low voltage operation can reduce processor power at the expense of performance, reliability concerns limit the degree of voltage reduction [6].

We consider a technique complementary to low voltage operation: *bit serial computing*. Bit serial computing is defined as computing on a single bit of a datum in each cycle. For a perfectly serializable circuit with 16-bit data width, bit serialization provides 16× power reduction in exchange for up to 16× performance degradation, which may be acceptable for many sensor benchmarks [5]. Actual performance degradation may be much smaller when considering the decreased critical path length of the serial circuit. Similarly, actual area and power benefits may be much smaller, especially for a microprocessor, since not all logic is serializable.

Bit serial functional units have been considered for microprocessors in the context of high-throughput computing [11, 4, 1]. These works exploit the super-linear[1] increase in functionality per unit area or power with decreasing bit width. Similarly, Khanna and Calhoun studied serial adders for ultra-low-power [10]. They found that a serial adder is more energy efficient than a parallel adder given low supply voltage and certain performance constraints. In this paper, we present the first study of bit serialization of a microprocessor in the context of ultra-low-power applications. We investigate how much of the maximum benefits are possible in the context of microprocessors without requiring a change in the ISA. We explore components common to most microprocessors and determine the characteristics that make logic serializable. These characteristics are not limited to the context of microprocessors but are widely applicable to ASIC and other designs as well.

The rest of this paper is organized as follows. In Sections 2 and 3, we describe issues faced in microprocessor serialization and how they limit area savings to 38%. In Section 4, we discuss how we evaluated our design and found that power benefits were limited to 42%. We conclude in Sections 5 and 6 with a discussion of future work and summary of the results.

2. BIT SERIALIZABILITY OF A MICROPROCESSOR

[1]Such benefits area only possible for certain components (e.g., adders)

ISLPED '2016 San Francisco, California USA

DOI: http://dx.doi.org/10.1145/2934583.2934597

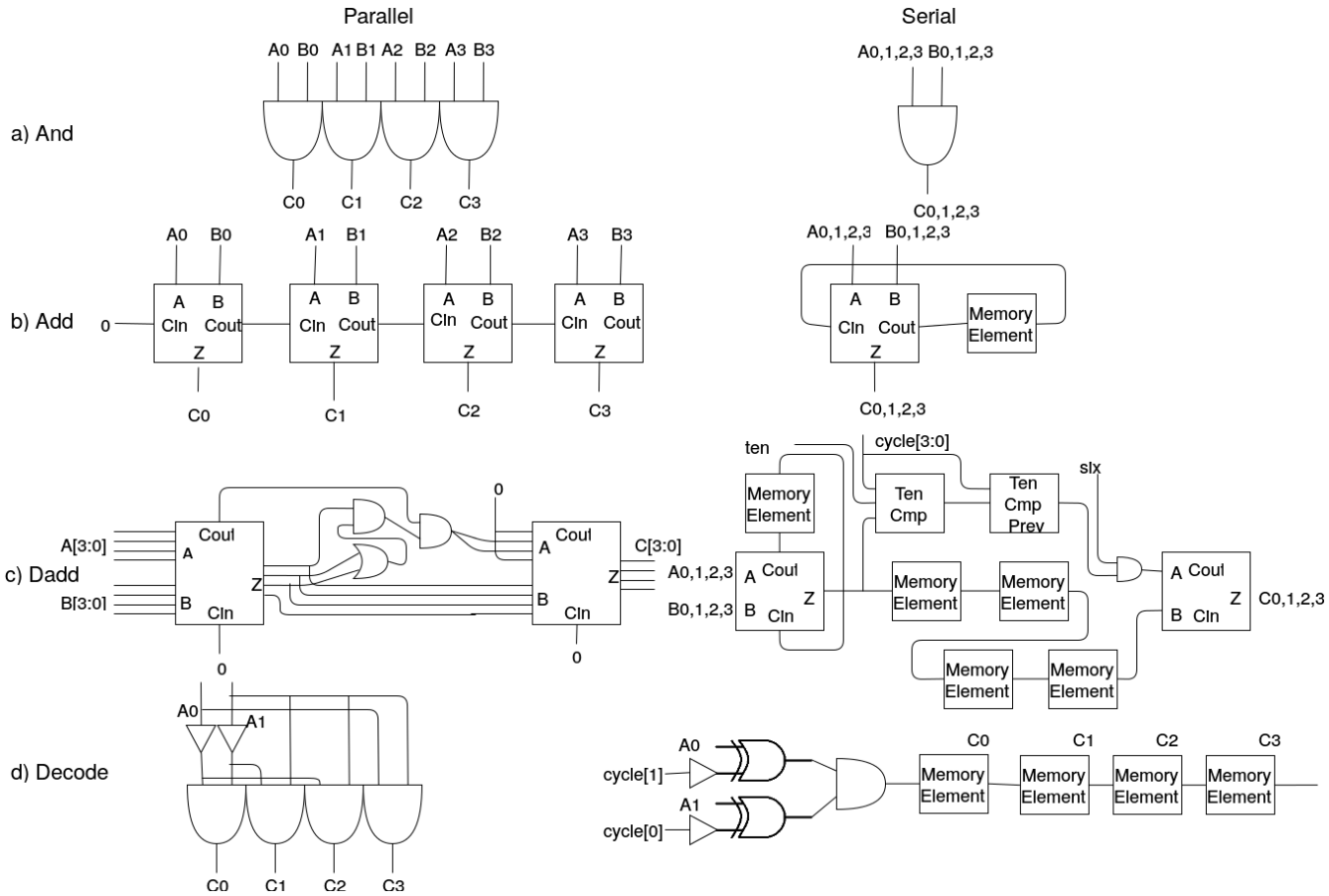

Figure 1: Components with decreasing serializability from top to bottom

Microprocessors support a wide variety of instructions resulting in complex data path and control logic that have varying degrees of serializability. For our work, we define combinational logic to be perfectly bit serializable if the logic corresponding to each bit position is identical and independent; less similarity and independence mean lower serializability. Below we discuss four examples of logic found in a common microprocessor [8] that have different degrees of bit serializability: one perfectly serializable, one with identical logic but dependence between bits, one with neither identical logic nor independence between bits, and one with the maximum possible dependence between bits (every output bit depends on all input bits).

- The first example (Figure 1(a)) is a bit-wise AND computation. This computation is common in microprocessors to support, among other things, a bitwise AND instruction in the ISA[2]. Computation on each bit position is identical and independent, requiring only a single AND gate. The computation can be perfectly serialized using a single AND gate and delivering bits in the operands serially.
- A second example is a ripple-carry adder (Figure 1(b)) supporting, among other things, an ADD instruction in the ISA[3]. The logic operating on each bit, a full adder, is identical, but is not independent due to the carry chain. This introduces a savings-limiting complication: a flip flop must be included in the serial implementation to account for the dependence. As a result, serializing the adder saves only half the area saved from serializing AND.
- A third example is binary coded decimal addition (Figure 1(c)). This computation is commonly supported in microcontrollers since they are often used in applications that involve displaying numbers[4]. Although the per-bit logic in this computation is neither identical nor independent, which severely limits savings, there is still some common logic in the bit-wise additions being performed. In order to serialize, there must be an intermediate shift register which holds a decimal digit until a comparison of the decimal digit to the value ten is complete. After this comparison is complete, six is added serially to the decimal digit if the decimal digit was greater than ten. Area savings of serializing the binary coded decimal adder are about half of the area savings of serializing the binary adder.
- The final example is a 2-to-4 decoder (Figure 1(d)). A 2-to-4 decoder is used whenever a one-hot bit-vector needs to be generated (e.g., while using a 4-to-1 multiplexer at the output of a register file with four physical registers). Each output bit is an AND of both inputs to the decoder. Since every output bit depends on every input bit, there are two possible serializations of the decoder, one with serial output and another with serial input. We show a serial output implementation in Figure 1(d) since it is more efficient than the serial input counterpart due to the shared AND gate. The two XOR gates and one AND gate in this serial implementation consume roughly the same area as the four AND gates in the parallel design. As such, no

[2]The corresponding instruction for MSP430 is AND.

[3]The corresponding instructions in MSP430 are ADD, ADDC, SUB, and SUBC.

[4]The corresponding instruction in MSP430 is DADD.

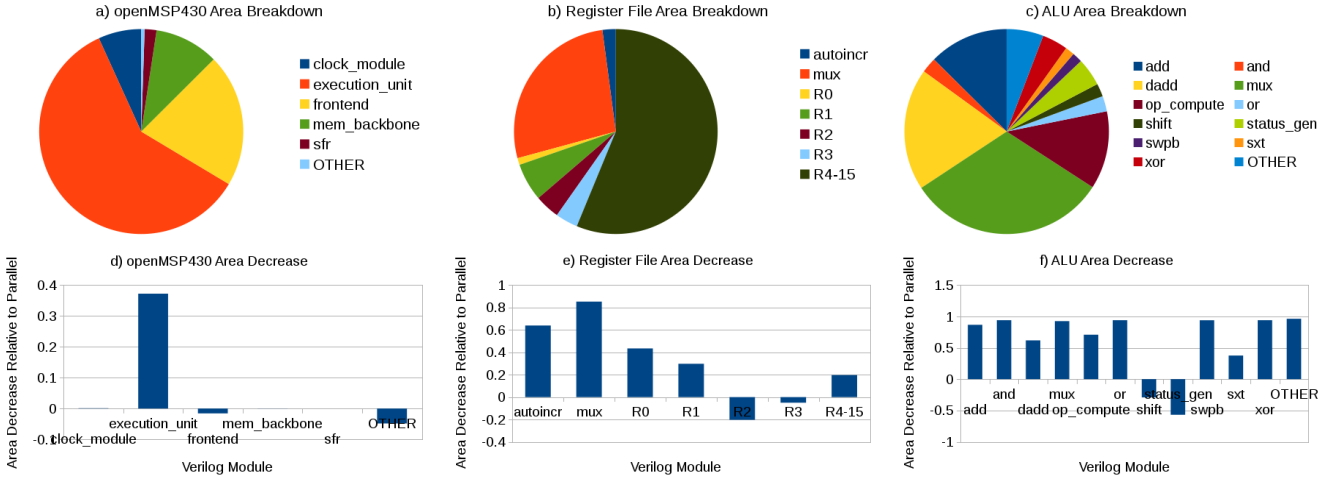

Figure 2: Area breakdowns and savings by Verilog module

area savings are possible.

The above examples show that different logic components in a microprocessor have different degrees of bit serializability. In fact, some logic in microprocessors is unserializable. For example memory cells holding the processor's state (e.g., physical registers and state machines) cannot be shared without destroying the data they store. There are other challenges as well. Consider the memory interface. In order for all bits of an address or datum to be available to memory in the same cycle, a costly shift register is required for deserialization. In general, repeated serialization and deserialization that is needed due to mismatched serializability of different microprocessor components will limit area savings even further.

As such, a microprocessor level study is needed to ascertain benefits from bit serialization in the context of ultra-low-power computing. In this paper, we perform that study for the openMSP430 microprocessor without making any modifications to the ISA. We discuss in Section 5 how ISA changes may allow further benefits.

3. BIT SERIALIZING MSP430

3.1 Base Architecture

We studied the open source Verilog implementation of Texas Instruments' MSP430 micro-controller, openMSP430. An area breakdown by Verilog module is shown in Figure 2(a). openMSP430 consists of three pipeline stages. The first two stages are included in a single Verilog module: front-end. The first pipeline stage fetches the instruction. The second stage decodes the op code, jump condition, and source and destination addressing modes (in memory operands are allowed). The front-end module also includes the state machines controlling the entire pipeline.

The source and destination register decoders operate in the second stage, but are included in the front-end module. The rest of the logic in the second stage is in the execution unit module. The second stage fetches the operands, does the computation, and writes the result. In the next two subsections, we will discuss serializing the front-end and execution unit modules.

There were a few modules in the RTL that we did not serialize. We did not serialize the clock tree because there are no data nets to serialize. We did not serialize the memory backbone, which arbitrates memory requests between the front-end and the execution unit, because serialization and deserialization of data associated with memory requests is done in the front-end and execution unit modules separately. Finally, we did not serialize the special function register because the result would not significantly impact the total area.

3.2 Bit Serializing the Front-end

Most of the front-end has a low degree of serializability since dependence between bits is common (e.g., decode) and much of the logic saves state (e.g., state machines). One opportunity for serialization is in processing instruction operands fetched from instruction memory or provided by a constant generator[5]. Area savings come from serializing the multiplexers that select between these values and output to the execution unit. Serializing the output of these multiplexers also increases the impact of serialization on the execution unit since no parallel-to-serial conversion of the output is required.

To exploit the above opportunity, a cycle counter is required to denote the bit-position corresponding to the current cycle of serial computation. This counter is used to put a lower bound of sixteen cycles on the time spent in each state, allowing for sixteen cycle serial operations. It is also used throughout the execution unit (e.g., to tell adder to disregard saved carry on zeroeth cycle). Unfortunately, the cost of this additional logic outweighs the area benefit of the serialization resulting in a 1.5% area increase shown in Figure 2(d).

3.3 Bit Serializing the Execution Unit

3.3.1 Register File

The MSP430 register file consists of sixteen registers: four function specific registers (R0-R3) and 12 general purpose (R4-R15). In order to support serial execution, we modified R0, R1, R3, and the general purpose registers by making them rotate a single bit per cycle from most significant to least significant bit. Rotation was enabled using the clock gating signal supported by openMSP430. Rotation allows serial access to these registers (read at the least significant bit and written at the most significant bit) using serial input and output multiplexers. Serializing the adder in the register auto-increment module also resulted in large area savings.

The only register not made to rotate was the architecturally visible status register, R2, because multiple status bits are generated in the last cycle of serial execution and thus must be saved in a

[5]The constant generator outputs one of a small set of 16 bit constants based on a selector input. This way, instructions can encode the much smaller selector input instead of the entire constant.

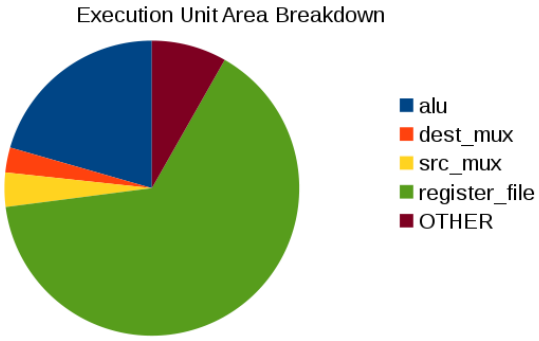

Figure 3: Area breakdown of execution unit by Verilog module

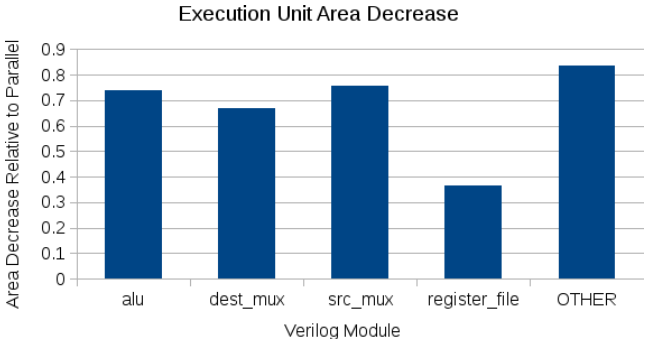

Figure 4: Relative area decrease of Verilog modules in execution unit

single cycle; as in the baseline, the status bits have dedicated read and write ports. Consumption of these bits requires bit-parallel write, which is costly to support for a rotating register (50% area overhead). Since the register is not rotating, special interfaces to the serial input and output of the register file are required. When write to R2 is enabled, we change the value of a given bit if and only if the cycle counter's value corresponds to that bit's position (i.e., a multiplexer is placed on each bit with a comparison to the cycle counter as the selector). When read from R2 is enabled, we select the bit corresponding to the current cycle using a 16-to-1 multiplexer with the cycle counter as the selector.

All the registers that we made rotate decreased in area except R3. Since R3 is the only register that does not support an autoincrement function, there is no multiplexer on the input of the baseline implementation. Serialization of the input multiplexer accounts for the area savings in the other registers.

Figure 2(b) shows the area breakdown of the register file. The largest module is the collection of all general purpose registers, which saw a 20% area decrease (see Figure 2(e)) due to serialization of the input. The second largest module, the output multiplexer, saw the largest relative area decrease due to serialization of the output. For these reasons and the size of the register file module relative to the rest of the processor (Figure 2(a)), serialization of the register file contributes the largest absolute area savings of any module in the design.

3.3.2 *Arithmetic Logic Unit*

Figure 2(f) shows area savings for different modules in the ALU. All logical operations (AND, OR, XOR) saw 16× area savings from serialization since they are perfectly serializable (Section 2). The binary addition (ADD) and binary coded decimal additions (DADD) save 8× and 3× area, respectively, since the computations on bits are not independent and, in the case of binary coded decimal addition, not identical.

Benchmark	Description
rle	Run-length encoded compressor
tea8	TEA encryption algorithm
div	Unsigned integer division
inSort	In-place insertion sort
binSearch	Binary search
intAVG	Signed integer average
intFilt	4-tap signed FIR filter

Table 1: Benchmarks evaluated (subset of Table I in [5])

The three remaining ALU operations are sign extend (SXT), byte swap (SWPB), and right shift by one (shift). We implemented serial byte swap by rotating the destination register for eight cycles and disabling rotation for the next eight cycles. Serial right shift requires a different approach since arithmetic shift is required. Execution of arithmetic shift requires copying the most significant bit to the second most significant place. We do this by waiting for the most significant bit to be available (in the sixteenth cycle of execution) and then waiting for fifteen more cycles until the second most significant place can be written. While this does result in a significant latency increase over the alternative of adding support for the bit copy to the register file, it requires much less area.

All ALU operations send serial streams to a multiplexer at the output of the ALU. This multiplexer is the largest module in the ALU (Figure 2(c)). It is also perfectly serializable and so contributed the highest absolute area savings for the ALU. From Figure 2(f), we see that the ALU has a high average serializability and thus contributes a large percentage of the area decrease even though it is only the second largest module in the execution unit.

3.3.3 *Supporting Logic*

A final opportunity for serialization is in the source and destination operand multiplexers. Since multiplexers are perfectly serializable, one would expect 16× area savings. Figure 2(f) shows that this is not the case for the destination operand multiplexer. In the parallel RTL, two of the inputs to the multiplexer are constants. This allows a design tool to decrease the area of the multiplexer beyond what would be possible if all inputs were dynamic. During serialization, any constant bit-parallel values must be converted to bit serial streams. Therefore, none of the multiplexer inputs in the serial design are constant, and the optimization that was available to the parallel design is not available to the serial design.

In order to meet ISA and memory interface requirements, bit serial requires additional logic. We added an intra-instruction carry bit, which is required to perform additions without clobbering the carry bit in the status register (e.g., computing the memory address of an operand to an add-with-carry instruction). To interface the serial output of the ALU with the parallel memory interface, serial-to-parallel glue logic is required. We added a shift register to deserialize the memory address calculated by the ALU. Finally, the serial implementation also requires parallel-to-serial glue logic on the memory interface. We used a 16-bit register to implement the interface.

3.3.4 *Summary*

A much smaller portion of the execution unit is dedicated to control of the processor than the front-end. Figure 4 shows that all Verilog modules in the execution unit saw an area decrease; the relative size of each module is shown in Figure 3.

4. EVALUATION

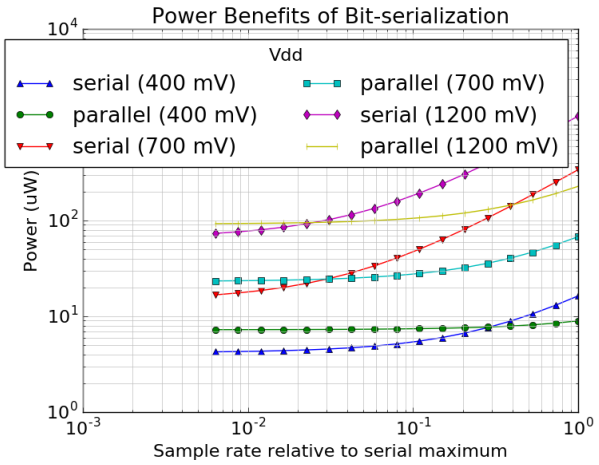

Figure 5: Bit serial has lower power at low sample rates

4.1 Methodology

To synthesize our designs, we used Synopsys Design Compiler with a TSMC 65nm cell library characterized for different operating voltages from 400 mV to 1200 mV. Since our technique targets low operating frequencies, we use high-Vt cells in order to minimize power consumption. We do not evaluate with other cell types because the bit-parallel design consumes less power than the bit-serial design at high frequencies even with the limitation on cell type[6]. We simulated operation of the resulting gate-level netlists using Modelsim. Our test bench included a functional model of a zero-wait-state memory whose power and area we did not evaluate. To measure power consumption of the core given the simulation results, we used Synopsys PrimeTime.

4.2 Results

To evaluate the power and performance impact of bit serialization of openMSP430, we performed gate-level simulations of the sensor benchmarks listed in Table 1. Prior work has shown current embedded processors heavily over-perform for these benchmarks [5]. These benchmarks also represent application settings where power is constrained [12]. Figure 5 shows the power of both the serial and parallel implementations operating at their maximum frequencies (313 MHz and 263 MHz respectively). The designs are clock gated when idle. The sample rate on the x-axis is normalized to the maximum sample rate that can be processed by the bit serial design. The results show that the bit serial design uses up to 38% less power than the bit parallel design for low sample rates. This is due to the fact that static power constitutes a large fraction of overall power at low sampling rates; the serial version has lower static power due to lower area. The results also show that the power benefits of a serial design at a given sample rate increase as voltage is lowered. This is because the static power consumes a higher fraction of overall power at low voltages.

Figure 6(a) shows that the benefits of serial design continue even when the serial and parallel designs are synthesized for every sample rate individually instead of only the maximum. For the parallel design, the design tool takes advantage of the relaxation of the frequency constraint by decreasing the area of the design. This decreases the leakage power, moving the serial-parallel crossover to lower sample rates and decreasing the maximum power savings to 27% at 400 mV.

[6]This also assumes that the bit-parallel design has more unconstrained paths and would therefore benefit more from variable Vt than the bit-serial design.

	Parallel Area %	Serial Area %
Sequential	39.9	55.3
Combinational	59.4	44.2
Buffer	0.689	0.505

Table 2: Composition of designs

	Leakage/Area (400 mV)	Leakage/Area (1200 mV)	Ratio
Sequential	0.338	7.67	22.7
Combinational	0.441	7.54	17.1
Buffer	0.779	15.2	19.5

Table 3: Different types of standard cells have different leakage characteristics

	400 mV	700 mV	1200 mV
Parallel (MHz)	18.9	182	263
Parallel (MIPS)	8.58	82.7	120
Serial (MHz)	25.6	263	313
Serial (MIPS)	0.580	5.24	7.27
Degradation	14.8x	14.4x	16.5x

Table 4: Performance Degradation

Power benefits from bit serialization tend to be higher than the area benefits. For example, when synthesized at the maximum operating frequency, area savings from bit serialization are 38%, but maximum power savings are 42%. This discrepancy is due to the fact that the gate-level netlist corresponding to the serial design has a higher percentage of sequential standard cells than the parallel design (Table 2). Combinational cells leak more per unit area on average than sequential cells (Table 3).

Figure 6(b) shows the power consumed by bit serial design relative to the power consumed by bit-parallel for different benchmarks. For low sample rates, where our design is most useful, power savings are approximately equal to the decrease in leakage of the design and so are benchmark independent. For high sample rates, there is little variation in the serial power consumption because of the simplicity of the design. The small increase in *rle* relative to the other benchmarks is due to a relatively high percentage of instructions with in memory operands. These instructions go through extra states during execution for operand fetch.

The leakage power of the bit serial design decreases faster with voltage than the bit-parallel design, resulting in greater power savings at lower voltages. Figure 6(c) shows this trend. This is not surprising since (a) leakage per area decreases faster with voltage for sequential area than combinational (Table 3), and (b) the serial design has a higher percentage of sequential standard cells (Table 2).

Finally, bit serialization resulted in a performance degradation of 14.4−16.5× (Table 4) when compared to the corresponding parallel designs. However, the absolute performance of the serial designs is between 584 KIPS and 7.24 MIPS. This performance is more than sufficient for a large class of sensor applications [5].

5. DISCUSSION AND FUTURE WORK

Area savings due to bit serialization could be as high as the data width. However, our evaluations show that such savings are impossible to achieve with an existing ISA since a large percentage of the logic is poorly serializable. Since the logic dedicated to control is determined by the ISA, an ISA designed for bit serializability could promote greater serialization.

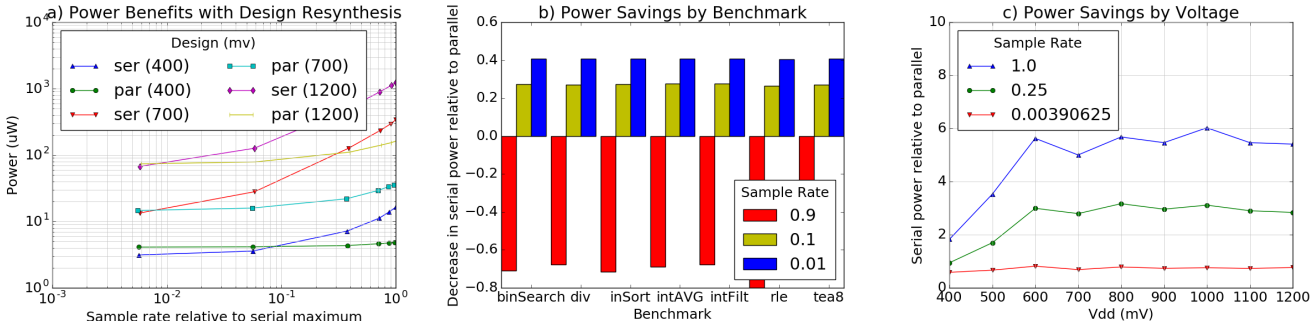

Figure 6: Secondary evaluations with (a)resynthesis, (b)benchmarks, and (c)voltage

As a proof-of-concept, we created a bit serial implementation of an instructional LC-3b core [13]. The LC-3b ISA includes only eight logical registers and much simpler decode due to fewer operations and addressing modes supported. In serializing this design, we saw a 27.5% area decrease when synthesizing for low sample rates (up from 21% area decrease for openMSP430). This shows the feasibility of getting greater benefits from bit serialization when ISA modifications are allowed. Our future work includes designing an ISA amenable to bit serialization and a bit serial processor from the ground up for emerging sensor applications.

As first mentioned in Section 1, a technique that decreases power consumed by an energy harvesting system can increase throughput even if that technique increases energy or decreases performance. Therefore bit-serialization can increase the throughput of performance constrained energy harvesting applications depending on the power-profile. Future work includes evaluating power-profiles of different energy harvesting devices in order to measure the throughput benefit of bit-serialization.

Future work also includes evaluating contexts where a greater percentage of logic is highly serializable without architectural changes. For example, a GPGPU core executes SIMD instructions and therefore has a much higher percentage of highly serializable data processing logic than the MSP430. Another example is a hardware accelerator. Since accelerators are dedicated to a single task, they should have a smaller percentage of logic dedicated to control, which we found difficult to serialize. In addition to decreasing area and power in these contexts, we will explore increasing throughput using the increased throughput per area of serial computation.

6. CONCLUSION

This work performs the first exploration of microprocessor bit serialization for ultra-low-power. We find that for energy harvesting sensors operating at low sample rates, bit serialization can decrease power by up to 42% relative to the bit-parallel implementation. Power savings are limited by the large fraction of the processor that is unserializable. The unserializability is mostly due to saving the processor state or implementing complex control logic. The impact of both control and state saving logic can be decreased by decreasing their size relative to the rest of the processor.

7. ACKNOWLEDGEMENTS

This work was supported in part by NSF (CCF-0953603, CCF 12-55857) and Oracle. Nam Sung Kim has financial interest in AMD and Samsung Electronics.

8. REFERENCES

[1] *Connection Machine Model CM-2 Technical Summary.* Thinking Machines Corporation, 1990.

[2] C. Alippi and C. Galperti. An Adaptive System for Optimal Solar Energy Harvesting in Wireless Sensor Network Nodes. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 55(6):1742–1750, July 2008.

[3] D. Balsamo, A. Das, A. S. Weddell, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Graceful Performance Modulation for Power-Neutral Transient Computing Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(5):738–749, 2016.

[4] K. Batcher. Design of a Massively Parallel Processor. *IEEE Transactions on Computers*, C-29(9):836–840, Sept. 1980.

[5] Bo 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.

[6] R. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. Mudge. Near-Threshold Computing: Reclaiming Moore's Law Through Energy Efficient Integrated Circuits. *Proceedings of the IEEE*, 98(2):253–266, Feb. 2010.

[7] K. Finkenzeller. *RFID handbook: fundamentals & applications in contactless smart cards & identification.* Wiley-Blackwell, Oxford, 2010.

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

[9] A. Juels. RFID security and privacy: a research survey. *IEEE Journal on Selected Areas in Communications*, 24(2):381–394, Feb. 2006.

[10] S. Khanna and B. H. Calhoun. Serial sub-threshold circuits for ultra-low-power systems. In *Proceedings of the 2009 ACM/IEEE international symposium on Low power electronics and design*, pages 27–32. ACM, 2009.

[11] J. Nickolls. The design of the MasPar MP-1: a cost effective massively parallel computer. In *Compcon Spring '90. Intellectual Leverage. Digest of Papers. Thirty-Fifth IEEE Computer Society International Conference.*, pages 25–28, Feb. 1990.

[12] J. A. Paradiso and T. Starner. Energy scavenging for mobile and wireless electronics. *Pervasive Computing, IEEE*, 4(1):18–27, 2005.

[13] Y. Patt and S. Patel. *Introduction to Computing Systems From Bits and Gates to C and Beyond.* McGraw-Hill, 2nd edition, 2003.

[14] V. Raghunathan, A. Kansal, J. Hsu, J. Friedman, and M. Srivastava. Design considerations for solar energy harvesting wireless embedded systems. In *Proceedings of the 4th international symposium on Information processing in sensor networks*, page 64. IEEE Press, 2005.