A Second-Generation Sensor Network Processor with Application-Driven Memory Optimizations and Out-of-Order Execution

Leyla Nazhandali, Michael Minuth, Bo Zhai, Javin Olson,
Todd Austin and David Blaauw

The University of Michigan
1301 Beal Ave, Ann Arbor, MI 48109
subliminal@eecs.umich.edu

ABSTRACT

In this paper we present a second-generation sensor network processor which consumes less than one picoJoule per instruction (typical processors use 100s to 1000s of picoJoules per instruction). As in our first-generation design effort, we strive to build microarchitectures that minimize area to reduce leakage, maximize transistor utility to reduce the energy-optimal voltage, and optimize CPI for efficient processing. The new design builds on our previous work to develop a low-power subthreshold-voltage sensor processor, this time improving the design by focusing on ISA, memory system design, and microarchitectural optimizations that reduce overall design size and improve energy-per-instruction. The new design employs 8-bit datapaths and an ultra-compact 12-bit wide RISC instruction set architecture, which enables high code density via micro-operations and flexible operand modes. The design also features a unique memory architecture with prefetch buffer and predecoded address bits, which permits both faster access to the memory and smaller instructions due to few address bits. To achieve efficient processing, the design incorporates branch speculation and out-of-order execution, but in a simplified form for reduced area and leakage-energy overheads. Using SPICE-level timing and power simulation, we find that these optimizations produce a number of Pareto-optimal designs with varied performance-energy tradeoffs. Our most efficient design is capable of running at 142 kHz (0.1 MIPS) while consuming only 600 fJ/instruction, allowing the processor to run continuously for 41 years on the energy stored in a miniature 1g lithium-ion battery. Work is ongoing to incorporate this design into an intra-ocular pressure sensor.

Categories and Subject Descriptors

B.5.1 [**Design**]: Arithmetic and logic units, Control design, Datapath design, Memory design

General Terms

Design

Keywords

Sensor network, Microprocessor, Energy efficiency, Memory organization

CASES'05, September 24–27, 2005, San Francisco, California, USA.

1. INTRODUCTION

The size scaling trends in computer design have seen supercomputers shrink into minicomputers, then desktops, then handhelds, and most recently into sensor processors. With each reduction in size, systems have enjoyed decreased cost and power requirements and new computing applications. Sensor processors represent a new level of compact and portable computing. These small processing systems reside in the environment they monitor, combining sensing, computation, storage, communication, and power supplies into small form factors. Sensor processors encompass a vast array of applications, ranging from medical monitoring, to environmental sensing, to industrial inspection, and military surveillance. The untethered nature of sensor processors requires that they survive for long periods of time on stored or scavenged energy, making energy efficiency the primary design factor. Fortunately, this goal is well served by their low performance requirements, as environmental sensing data rates are typically low, permitting designs on the order of 100s of KIPS to a few MIPS. Table 1 shows the benchmarks used in this study, which includes a variety of data processing, communication and sensing algorithms. These programs represent a broad slice of the types of applications one could expect to see on an ultra-low energy sensor network processor platform. Also shown in the table are the data rate requirements for a number of low to mid-level ambient data sources. Additional details on these benchmarks and data sources can be found in [13, 3, 4]. As shown in Table 2, processing demands are quite low, all data rates permit processing of samples to take on the order of milliseconds, with some applications permitting even seconds between data samples.

Table 1: Sensor Network Applications.

Application	Description
adRout	Ad-hoc router control algorithm [14]
CompRLE	Run-length encoded compressor [5]
TEA	TEA encryption algorithm [16]
CRC8	Cyclic redundancy code generator
intFilt	4-tap signed FIR filter
tHold	Threshold detector w/ noise margin
insertSort	In-place insertion sort
binSearch	Binary search
runAvg	Running average
Hist4	4-bin histogram generator
maxFinder	Maximum value search

Table 2: Sensor Network Data Sources

Phenomena	**Sample Rate** (Hz)
Atmospheric temperature	0.017 - 1
Barometric pressure	0.017 - 1
Body temperature	0.1 - 1
Natural seismic vibration	0.2 - 100
Blood pressure	50 - 100
Engine temperature/pressure	100 - 150
ECG (electro-cardiogram)	100 - 250

Figure 1 illustrates the performance of these benchmarks on a variety of commercial microcontrollers (the figure is from [13]). For each processor, we show the performance of the low and mid-bandwidth benchmark sets (i.e., at or below 100 Hz data rates). Each processor is implemented in a 130nm CMOS process technology running at the listed voltage. The results are shown as an xRT (times-real-time) rating. This value indicates how many times faster than real-time the processor can handle the worst-case mid/low-bandwidth benchmark data stream rate. For example, the ARM720T [1] at 1.2V with a 100 MHz clock is able to process worst case mid-bandwidth data stream 2965 times faster than real-time. Or in other words, the ARM720T in the example would have a duty cycle of 1/2965 for the worst-case benchmark. Clearly, the commercial microprocessor offerings (i.e., the first four designs shown in Figure 1) are much more powerful than necessary for these sensor processor applications.

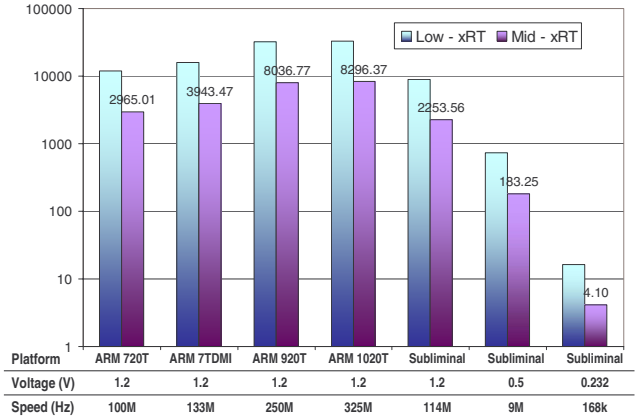


Figure 1: Performance Analysis on Varied Designs.

Given the performance headroom for current designs, it becomes possible to gain energy efficiency by reducing supply voltage. A reduced supply voltage, while requiring commensurate reductions in clock frequency, will render quadratic reductions in dynamic energy demands. Figure 1 also shows, in the last three bars, the performance of our previously proposed (firstgeneration) sensor processor design, running at a nominal 1.2V, the lowest super-threshold voltage of 0.5V, and an energy-optimal subthreshold voltage of 0.232V. It is certainly the case that the low performance requirements of these applications enable significant voltage reduction. The 0.5V design still runs the mid-bandwidth benchmarks more than 100 times faster than necessary. Consequently, we must consider the possibility of running designs at subthreshold levels. The first-generation design at 0.232V still performs four times faster than real-time. Given the performance headroom of these designs, even at subthreshold voltages, it is clearly the case that the primary concern for the design of sensor processors is not performance, but rather energy per unit effort (e.g., instructions).

In this context, our first-generation subthreshold-voltage sensor processor made the contribution of showing the link between microarchitectural design and energy-efficiency in the subthreshold voltage domain. There are a number of implications of subthreshold voltage operation, in particular, the preclusion of differential signaling (e.g. differential bit-lines and sense amplifiers), tri-state buffers and dynamic logic. From an architectural perspective, the most crucial aspect of subthreshold-voltage design is the existence of an energy minimal design point. This is in direct contrast to super-threshold design where, without regard for performance, it is always possible to reduce energy requirements by lowering voltage. In contrast, subthreshold voltage levels exist where reducing voltage can result in slower operation and larger energy requirements. The reason for this energy minimal point is illustrated in Figure 2. As shown in the figure, a decreasing supply voltage results in quadratic reductions in the active (dynamic) energy component. At the same, leakage currents decrease linearly. However, this reduction in leakage current leads to an exponential increase in circuit latency, and ultimately an exponential increase in leakage (static) energy. As shown in Figure 2, these two contrasting energy trends create a minimal-energy voltage, Vmin [17].

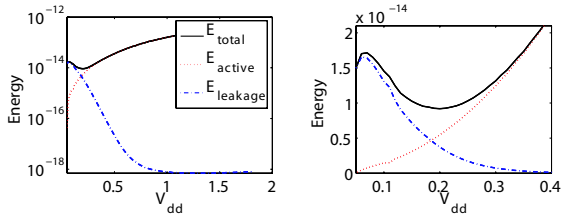


Figure 2: Energy vs. voltage.

In [13], we designed a simple architecture with variable-length instructions, capable of reaching energy-efficiency levels of 1.38 pJ/instruction. In addition, through the analysis of a wide array of microarchitectures, we found that to optimize energy efficiency:

- *Area must be minimized* as it is a critical energy factor due to the dominance of leakage energy at subthreshold voltages.
- *Transistor utility must be maximized* because effective transistor computation offsets static leakage power, which permits a lower operating voltage and lower overall energy consumption for the design.
- *CPI must be minimized* at the same time, otherwise, gains through small area and high transistor utility are squandered on inefficient computation.

The baseline design was implemented in an IBM 130nm CMOS fabrication technology, and the previously presented simulated results and our simulation environment were validated against the actual hardware measurements [12].

In this paper, we take the baseline design of [13] and concentrate on optimizing the instruction set and microarchitecture to reduce overall design size and improve performance and energy per instruction. The new designs presented utilize 8-bit datapaths and a highly compact 12-bit RISC instruction set architecture. To further enhance code density and reduce code size requirements, we incorporate micro-operations into our ISA, by fusing memory operations to ALU operations. In addition, our ISA incorporates flexible operand handling, permitting low-overhead updating of condition codes and pointer values. Furthermore, we introduce a number of application specific instructions targeted at improving the performance of sensor network applications. At the microarchitectural level, we incorporate a prefetch mechanism and a novel

memory architecture that utilizes row-address predecode. Program instructions specify a memory page, which sets up the row-address decoders prior to memory accesses. All memory accesses to the currently active memory page are completed in a single cycle; all remaining accesses take two cycles. Additionally, our microarchitecture incorporates a number of latency tolerance features to ensure a low CPI and high energy efficiency. We incorporate branch speculation and out-of-order execution, but in a simplified form to reduce area and leakage overheads. Using SPICElevel timing and power simulation, we find that our new design features produce a number of pareto-optimal designs that demonstrate a variety of performance and energy trade-offs. Our most efficient design is capable of running at 142 kHz (0.1 MIPS) while consuming only 600 fJ/instruction, allowing the design to run continuously for 41 years on the energy stored in a 1g lithium-ion battery.

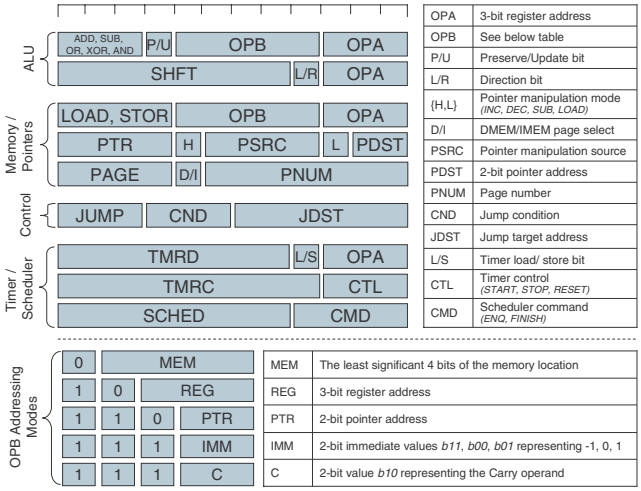
The remainder of this paper is organized as follows. Section 2 details the ISA optimizations implemented to improve code density and execution performance. Section 3 describes our application-driven memory optimizations, and Section 4 details microarchitectural optimizations, including branch speculation and support for out-of-order execution. We evaluate our proposed design features with SPICE-level simulation in Section 5, demonstrating the energy and performance tradeoffs for a variety of designs. Finally, Section 6 details related work, and Section 7 draws conclusions and suggests future research directions.

2. INSTRUCTION SET ARCHITECTURE OPTIMIZATIONS

Minimizing leakage currents is imperative for a subthreshold sensor processor that must survive extended periods of time on stored or scavenged energy. Since memory comprises the single largest factor of leakage energy, efficient designs must reduce memory storage requirements. To this end, we focus on the development of instruction set architectures (ISAs) that improve code density and simplify decode logic, thereby resulting in small implementation area and minimal leakage. In the following subsections, we present these ISA optimizations we made for our second-generation design.

2.1 RISC Encoding

Our second-generation design is based on a RISC architecture, which replaces the 4-bit variable length CISC ISA used in the first-generation design. CISC-style architectures allow instructions with multiple widths, thereby providing the ability to create dense code that has few unused bits, especially if the granularity of the instruction units is small (like the 4-bit nibbles used in our first-generation). On the other hand, RISC-style architectures typically have more unused bits (due to the fixed instruction size) which results in a larger application footprint, but the simplicity of instruction decoding allows for much simpler and smaller decoder logic. Consequently, these two ISA design styles generate competing trends between code density and decoder logic complexity, both of which have a direct impact on the energy consumption of the sensor processor. In this second-generation design, we exploit the best of both worlds by developing a RISC-style architecture that yields simple, small control logic and dense code through the use of a highly compact 12-bit instruction encoding. In fact, for our 12-bit instruction encoding, 3942 of the 4096 possible instruction encodings are defined as valid instructions, resulting in an encoding efficiency of 96.2%. Our new RISC-style ISA encoding is summarized in Figure 3.



OPA	3-bit register address
OPB	See below table
P/U	Preserve/Update bit
L/R	Direction bit
{H,L}	Pointer manipulation mode *(INC, DEC, SUB, LOAD)*
D/I	DMEM/IMEM page select
PSRC	Pointer manipulation source
PDST	2-bit pointer address
PNUM	Page number
CND	Jump condition
JDST	Jump target address
L/S	Timer load/ store bit
CTL	Timer control *(START, STOP, RESET)*
CMD	Scheduler command *(ENQ, FINISH)*

MEM	The least significant 4 bits of the memory location
REG	3-bit register address
PTR	2-bit pointer address
IMM	2-bit immediate values *b11*, *b00*, *b01* representing -1, 0, 1
C	2-bit value *b10* representing the Carry operand

Figure 3: ISA Organization.

2.2 Two-Operand Format with Memory Operands

One critical decision in designing an ISA is the number and variety of operands to support. The major options are stack-based, accumulator-based, register-memory and load-store architectures. Our analysis shows that stack-based ISAs yield the simplest control logic, but it results in large code sizes as most applications require three to four instructions per ALU operation (two PUSH, one POP and the ALU instruction itself). A second alternative, an accumulator-based architecture, was utilized in our first-generation design. This ISA style, which writes all results to a single accumulator register, still has relatively simple decoder logic, but, again, requires as many as two to three instructions per ALU operation. Only very special applications, such as TEA, where the result of one operation is again used for the second operation, benefit from accumulator-style architectures. In our second-generation design, we are using a load-store architecture with an 8-entry register file. We choose this architecture over the other alternative, register-memory, because it is simpler and less expensive to pipeline. The drawback of a load-store architecture with a small register file is that it may result in inefficient code which spends time loading the registers and storing them back to memory, rather than doing useful work. We address these deficiencies with the use of micro-operations, as described later in this section.

Having adopted a load-store style architecture, we evaluated the implication of utilizing a two vs. a three operand encoding. In a two-operand architecture, one of the operands is both a source and destination, while in a 3-operand architecture the destination may be different from both sources. A two-operand architecture results in shorter instructions, but in cases where the source should not be destroyed, extra copying of the register is required. Therefore, the code-density benefits of a two-operand encoding are dependent on the application domain. We choose a 2-operand architecture as our study shows that for the representative set of applications, the 3-operand choice generates code that is about 10% larger than the 2-operand one. However, we use a simple technique to optionally prevent destruction of the source operand: we include a preserve/update bit (P/U) in ALU instructions to indicate whether the source operand should be preserved. In this case, the result is written to register R0 instead of the specified source register, and it can later be accessed by subsequent instructions. This facility is especially useful in two common scenarios. The first scenario is when the result is not required, e.g., only condition bits are set after

the operation. The second scenario is when intermediate results are calculated, and R0 suffices as a scratch register.

For addressing modes, we include both direct and indirect memory access, along with 2-bit immediate values and direct register operands.

2.3 Micro-operations

To further improve code density and reduce register pressure we include support for microoperations. Micro-operations can improve code density by combining two processor operations into a single instruction encoding. Simple control logic converts instructions with memory operands into a load micro-operation followed by an ALU micro-operation. This scheme allows the simplicity of a load-store architecture along with code density of a register-memory architecture. Moreover, since the data arriving from memory to the load micro-operation can be stored in a temporary register (i.e., a latch), it reduces register pressure on the register file

2.4 Application-Specific Instructions

Efficient Pointer and Carry Manipulation: In our first-generation design, we showed the usefulness of having instructions for loading, incrementing and decrementing pointers. In our second-generation design, we augment this facility with additional pointer manipulation capabilities. This instruction compares the pointer with a value in memory which holds the address of the end of the array, to check if the pointer is within bounds. It involves two micro operations: a load to the temporary register and a subtraction between pointer and register, which only affects the Carry and Zero bits.

In addition, a few sensor network applications (such as error correction and encryption algorithms) require more precision than 8 bits (the precision of our architecture). Larger data widths such as 16 and 32 bits are necessary to make relevant use of these applications. Shifting and arithmetic operations are common in these applications, so providing a mechanism to handle bit traversal across 8-bit boundaries will reduce computation time and energy demands. We allow for the use of the previously computed carry bit as an additional option for operand B. This special carry operand may be used in all arithmetic and load instructions.

Event Scheduler Control: Our design includes a hardware-based event scheduler that can be controlled using either external interrupts or software commands. The scheduler is a 4-entry circular queue that contains partial pointers (3 bits each) to the highest eight lines of data memory. These lines of data memory contain event handler pointers and are used to dispatch events to handler code. The event scheduler has two modes: active and idle. In idle mode, the scheduler waits for a function to complete (indicated with the software command SCH FINISH). In active mode, the scheduler is waiting for a new event to enter the queue (either through an interrupt or software command SCH ENQ). When the queue is non-empty, the event handler is dispatched by fetching the event handler PC from memory, and the scheduler enters idle mode. For example, an averaging function can push a threshold detection function onto the scheduler queue just before it finishes. Once the scheduler becomes active, it will start the threshold detection function. This allows smaller functions to be chained together to form a more complex data manipulation routine.

Timer Control: Our design includes a hardware-based, software-controlled 32-bit timer. The timer has basic control functionality including start, stop, and reset (TMRC [START — STOP — RESET]). It can be read and written in 8-bit segments (through LOAD and STOR). With a clock frequency of 50kHz, the timer can achieve a maximum unique time range of 24 hours.

2.5 Code Density Analysis

Figure 4 shows the code density benefits of each of the proposed ISA enhancements. No Optimizations indicates the relative code size of our second-generation RISC ISA, without any of the extensions described in this section. The bar CISC uArch is the relative size of our firstgeneration CISC ISA, described in [13]. The remaining bars show the benefits of the optimizations described in this section. W/Preserve shows the code size improvements with source operand preservation. W/Carry Operand and W/PTR Sub shows the benefits of carry and pointer manipulation extensions, for extended precision operations and array traversals, respectively. W/uOperation shows the benefits of micro-operations. The W/PAGE optimization is related to memory architecture, which is discussed in Section 3. W/All optimizations shows the combined benefit of all ISA optimizations. All optimizations combined provide nearly a 30% reduction in the RISC-style code size. Compared to our first-generation nibble-size variable length ISA, our new RISC-style ISA still achieves a 17% reduction in overall code size. Combined with a 36% reduction in decode logic size, the new RISC-style ISA provides a significant opportunity to reduce design size and leakage energy.

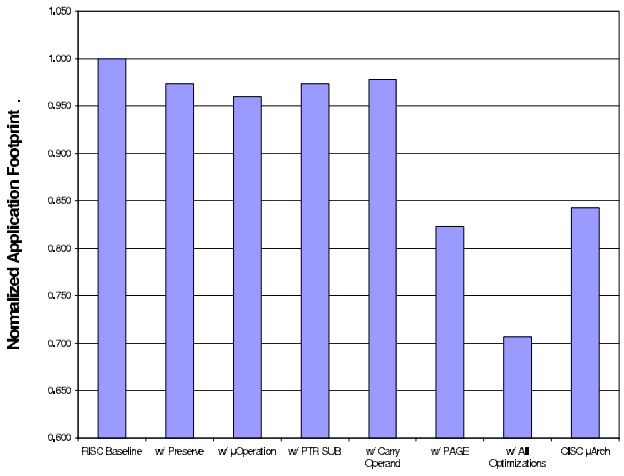


Figure 4: Code Density Analysis.

3. APPLICATION-DRIVEN MEMORY OPTIMIZATIONS

Our second-generation sensor processor employs an optimized memory architecture which improves code density and CPI at the same time. This architecture takes advantage of spatial locality through the use of a row-address pre-decode stage, without use of a data cache.

As illustrated in Figure 5, the data memory architecture is divided into 16-entry pages. A PAGE instruction is used to specify the current page in the memory. Having chosen the current page, only four bits are needed within a LOAD instruction to directly address a memory operand. Moreover, the upper bits of the address, supplied by the PAGE instruction, are used to implement row-address pre-decode, by pre-selecting the 16-byte memory before execution of the LOAD instruction. The use of address pre-decode reduces the latency of memory operations from two to one cycle. The only exception to this scenario is accessing memory through pointers, i.e. indirect memory accesses. As there is no restriction on the content of pointers, they can specify an access outside the specified page. In such case, it will take two cycles to perform the access, and the pipeline must be stalled to accommodate this latency.

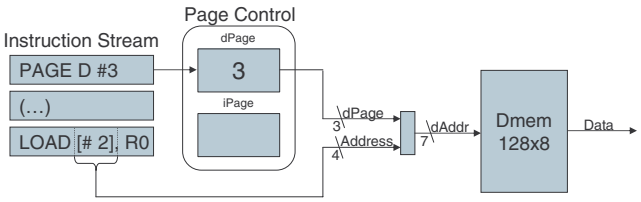


Figure 5: Data Memory Pre-decode Architecture.

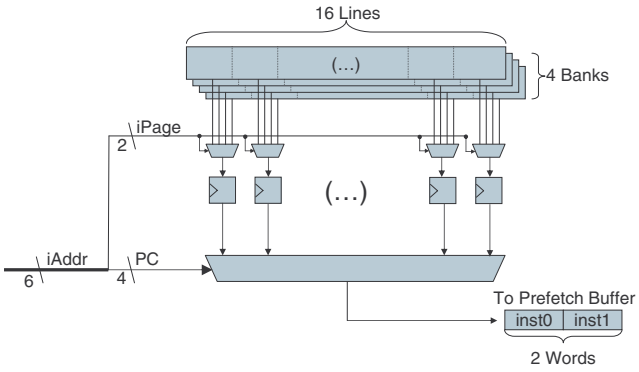


Figure 6: Instruction Pre-decode Architecture.

Similarly, as shown in Figure 6, the instruction memory is divided into 16-entry banks and a PAGE instruction is used to specify the working page. The main difference is that in the instruction memory, the page is automatically incremented as the program counter reaches the end of the page, whereas in the data memory, the data page is only changed manually. Another characteristic of the instruction memory is that it has double bandwidth to enable fetching more instructions in less number of cycles.

4. MICROARCHITECTURAL OPTIMIZATIONS

The second generation processor design is based on a 3-stage pipeline as shown in Figure 7. The number of stages is chosen based on the observation that energy-efficient designs in subthreshold operation need to strike a balance between area and transistor utilization. Increasing the number of pipeline stages potentially increases the transistor utilization because transistors residing on the shorter logic paths will switch more often. However, the more the stages, the greater the number of flip-flops needed to store intermediate values. We explored a 4-stage pipeline solution for this architecture and our primary analysis showed that the area overhead due to extra flipflops overshadows the gains from higher transistor utilization. On the other hand, a 2-stage design resulted in a poorly balanced solution as registers and memory accesses must be serialized in the second stage. Consequently, we found that the 3-stage pipeline solution strikes the best balance between design area and transistor utilization.

The first pipeline stage includes instruction fetch and decode logic. There is a small 4-entry instruction prefetch buffer in this stage, which enables single-cycle access to the next instruction. The logic that drives the memory control signals is also generated in this stage. The second stage performs memory accesses and ALU operations. Finally, the last stage is the write-back stage, where the result, either from ALU or memory is written back to the register file. The front end of the base pipeline performs a simple Branch Not Taken speculation. In case of misprediction, which is detected in the second stage, the pipeline is flushed with a 2-cycle penalty. As mentioned previously, the baseline pipeline is stalled for one cycle when accessing memory through pointers. Two different optimizations, namely, out-of-order execution and Branch Taken speculation, that are frequently used in super-threshold designs to alleviate the penalties incurred from misprediction and true data dependencies, are implemented on top of our base processor. In the remainder of this section, we study the effect of these optimizations on instruction latency and energy-per-instruction.

4.1 Out-of-Order Execution

The purpose of this optimization is to fill the wasted cycle in a micro-operation between a load using a pointer and a dependent ALU operation. Our approach to out-of-order execution is illustrated in Figure 8. If the LOAD does not access the current memory page, it takes two cycles to finish the LOAD operation, thus the ALU operation needs to be stalled before the data is ready. The wasted cycle before the LOAD finishes could be filled by an independent instruction, however, since the load and ALU operation are combined in a single instruction, this cannot be accomplished by the programmer/compiler. To facilitate latency tolerance of LOAD operations, we implement a limited out-of-order execution feature, which monitors just one instruction ahead in the instruction stream (in the prefetch buffer) and if it is independent, it is fed into the pipeline before the dependent ALU operation.

4.2 Taken Branch Speculation

Branch speculation is a well-known technique to tolerate the latency of branch dependencies, however, these techniques typically rely on costly logic and storage components, such as branch target buffers, return stack buffers, history tables, etc. In a sensor processor with a limited energy budget and an acute sensitivity to leakage, these devices consume too much energy to provide value. In order to avoid such overheads, our first-generation sensor processor implements a basic Not-Taken branch speculation mechanism. However, while inexpensive, this facility has limited benefits as the vast majority of branches in our sensor processor workload are taken branches. To further improve CPI, we have implemented a static Taken speculation mechanism. As the branch target is directly specified in the instruction, it is possible to predict the target by just looking one instruction ahead in the instruction prefetch buffer.

5. ENERGY-PERFORMANCE ANALYSIS

5.1 Experimental Framework

To evaluate our design solutions, we start by implementing each of the analyzed designs in synthesizable Verilog. Then we use Design Compiler, a Synopsys tool, to synthesize the design to Artisan standard cells for IBM 130 nm CMOS technology. The exceptions to this design flow are storage devices, such as the register file and prefetch buffer, which are manually designed at the gate-level to achieve energy-efficient latch-based storage. Subsequently, Sedsm, by Cadence, is used to place and route the design. The wire-load and capacitance information extracted from the placed and routed design is then used to accurately simulate the design and extract power usage for each application. On the other hand, SPICE simulations are used to characterize standard cells at different voltages. This data can be used to estimate how frequency, leakage power, and active power are scaled by changing voltage. Based on this information, the optimal energy voltage point, along with the instruction latency and energy consumption per instruction is calculated. For the experiments in this section, we set the voltage for all experiments to the energy-optimal voltage of the baseline RISC-style

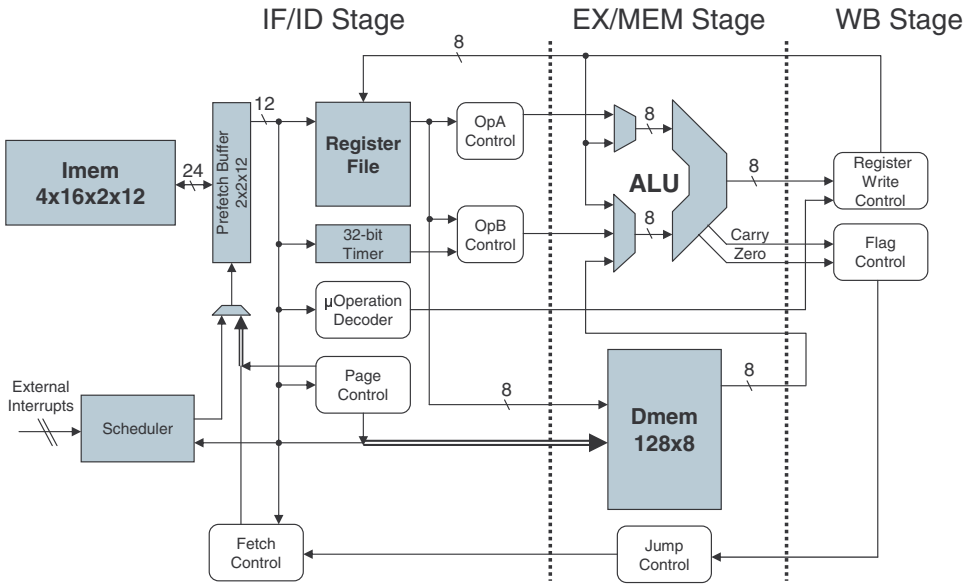


Figure 7: Microarchitectural Organization.

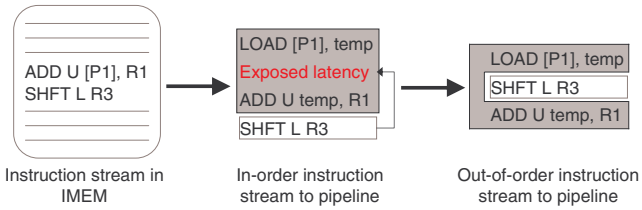


Figure 8: Limited Out-of-Order Execution.

sensor processor design, which is 200 mV. Finally, our simulation environment has been validated against the actual hardware measurements from our first generation design.

5.2 Simulated Results

Figure 9 shows the energy-performance tradeoffs for 13 design variants, based on our first and second-generation sensor processors. Each design was synthesized to an IBM 130 nm CMOS fabrication process. The first-generation designs are labeled to indicate: i) the number of pipeline stages (1s, 2s, or 3s), ii) the number of memories (v - single memory, h - I and D memory), iii) the datapath width (8w, 16w, or 32w), and iv) the existence (r) of explicit registers (designs without explicit registers store register values in memory). The second-generation designs (shown in the blown-up graph) are as follows: Ideal is an ideal un-implementable design that has single-cycle access for all data memory accesses. Base is a design that requires two cycles for indirect memory accesses. Sched is our base design with the scheduler and timer added, this design exposes the energy requirements of timer and scheduler. Spec is a design that incorporates only taken branch speculation, and OoO incorporates out-of-order execution and all other optimizations including branch speculation and ISA extensions.

In order to compare the performance and energy consumption of the first and second generation, we normalized the energy per instruction and instruction latency of the first generation. Our analysis of the set of representative applications shows that each second generation RISC instruction is worth an equivalent of 1.9 instructions in the first generation CISC ISA, the low density can be primarily attributed to the accumulator-based architecture of the first generation design. Thus, energy per instruction and instruction latency of the first generation designs are scaled by a factor of 1.9.

In Figure 9, designs that lie closer to the origin are faster and more energy-efficient than designs further away. The designs that are closest to the left and bottom of the graph are pareto-optimal designs, as they represent the best designs developed, with varying energy and performance trade-offs. All other designs are not worth implementing because at least one of the pareto-optimal designs is both faster and more energy efficient. Clearly, this is precisely the case for all our second-generation processors compared to the first-generation designs. Moreover, a number of second-generation designs are pareto-optimal. Some solutions that do not include all of the optimizations described also fall into pareto-optimal configurations because they improve energy demands while reducing performance.

Figure 10 validates the circuit-level analysis presented in [13]. For the five studied designs, the CPI shows a correlation with the minimum-energy voltage, especially for the three middle designs. Here, the higher the CPI, the higher is the minimum energy voltage. The reason is that if the total number of transistor switching required to complete an instruction is similar between two designs, the one with higher CPI has lower switching activity in a given time period, therefore a low activity rate. This in turn results in higher minimum energy voltage. The first and last designs in this graph do not show the same correlation with Vmin, as the first one is an ideal design where no penalties are assumed for indirect access memory, which makes its CPI better than the other designs without an architectural penalty. The last design does not follow the general trend because it is a variant of Base, with only the scheduler added which does not improve CPI.

Figure 11 presents the energy distribution of different components of the base processor at two different voltages, nominal voltage and energy-optimal voltage. The darker (blue) slices represent stages of the pipeline, whereas the lighter (yellow) slices represent storage units, register file and prefetch buffer. As shown, the energy consumed by the register file and prefetch buffer increases from 34% to 45% when moving from nominal voltage to minimum-energy voltage. In general, the storage devices have a more significant impact on energy consumption at subthreshold voltages compared to super-threshold because they usually require less activity than other components, and thus their leakage energy prevails.

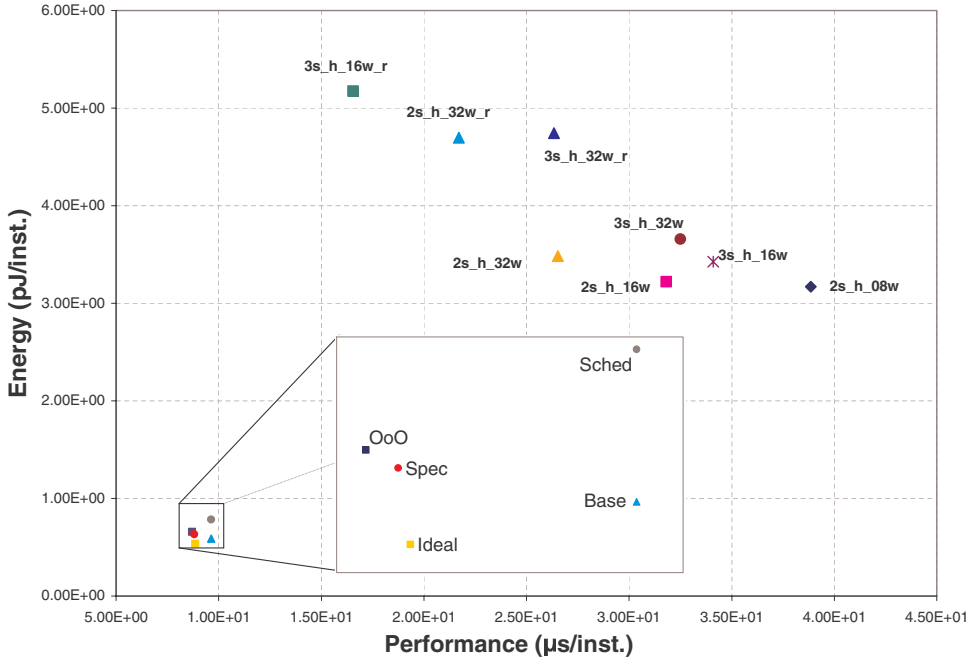


Figure 9: Pareto Analysis of First and Second-Generation Designs.

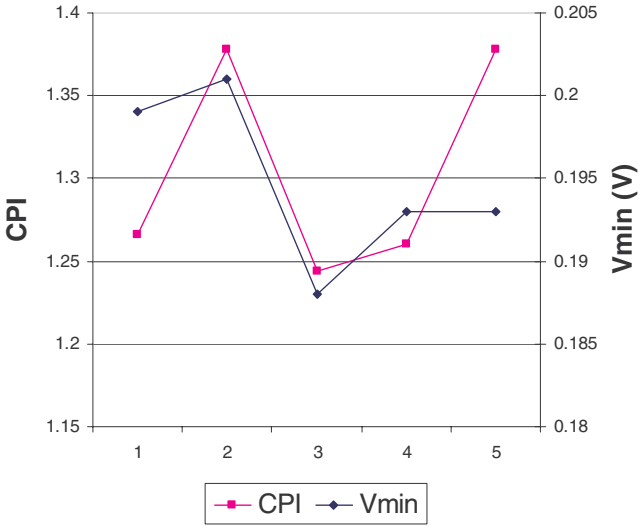


Figure 10: CPI and Vmin Correlation.

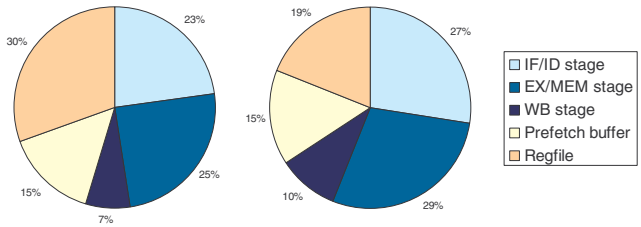


Figure 11: Energy Distribution of the base processor at (a) minimum-energy voltage of 0.2V and (b) nominal voltage of 1.2V.

6. RELATED WORK

Although numerous studies have been done in the area of sensor network system design, research on energy-efficient sensor processors is fairly new and the number of studies on this topic is limited. Most sensor network testbeds [11, 15, 6] use off-the-shelf microcontrollers such as such as the 8-bit ATMega128L operating between 4MHz and 8MHz and consuming about 1.5nJ/instruction (more than three orders of magnitude more energy than the processors studied in this work). Some of the high-end sensor network platforms [7, 2] use Intel StrongArm/XScale processor, consuming 1nJ/instruction.

Burd et al., presented some of the early work on energy-efficient processor designs in [8]. They acknowledge the fact that prior to their work, traditional architectural design methodologies for microprocessor systems had focused primarily on performance, with energy consumption being an afterthought. They demonstrate a top-down processor system design methodology for reducing energy consumption, with performance requirements of portable devices being the focus. However, their study is mostly concentrated on circuit-level optimizations with little focus on the processor core architecture.

In the sensor processing domain, Clever Dust 1 and 2 [15], developed as part of the Smart Dust project at UC-Berkeley, are two microcontrollers specifically designed for Dust sensor motes with the objective of reducing energy consumption. Clever Dust 2, inspired by the low-power CoolRISC microprocessor, is a load-store RISC Harvard architecture and no pipelining. It reportedly consumes 12 pJ per instruction to execute general instructions excluding memory operations and the energy consumption for instruction fetch. SNAP/LE is a sensor node processor based on an asynchronous data-driven 16-bit RISC core with a low-power idle state, and low-latency wakeup response [9]. SNAP/LE has an in-order, single-issue core that does not perform any speculation. It has a 16-bit datapath consuming 24 pJ per instruction. An application-driven architecture, which offloads immediate event handling from the general purpose component and leaves it idle, has been presented by Brooks, et. al [10]. With a 10% duty cycle (i.e. system being idle 90% of the time), this system consumes 2W at 100kHz,

which, assuming a CPI of 1, results in about 20 pJ per instruction. Finally, our previous work in sensor processor design involved a CISC based 8-bit processor, which consumes 1.34pJ per instruction.

The vast majority of work in sensor processor design has been concentrated on super-threshold voltages, resulting in higher energy consumption and delivering more-than-needed performance. In our subthreshold design the power consumed while the processor is active is usually lower than the idle power consumption of other sensor processors. The processors presented in this paper represent a new level of energy efficiency for sensor network processors. Our second generation solution is nearly three times more energy-efficient than our previously proposed design and 25 times more energy-efficient than the most energy-efficient design we had found in the literature.

7. INSIGHTS AND FUTURE WORK

We have presented a second-generation sensor processor that includes ISA optimizations, application-specific memory optimizations, and microarchitectural optimizations. The design is based on observations from a first-generation sensor processor design. Combined together, our optimizations result in a 54% energy savings, with our best design running at 600 fJ/instruction. To our knowledge, this design represents the most energy efficient computing design ever proposed, surpassing its predecessors by a factor of three. From another perspective, our most energy-efficient processor would run continuously for 312 years on the energy of a single AAA battery (7.6g of Lithium-Ion with a 3160 J energy payload).

Our ISA optimizations focus on compact encodings for reduced memory requirement. Reduced memory sizes are crucial to minimize system leakage. We achieve this by using a compact 12-bit RISC encoding with a two-operand format that supports memory operands. In addition, we introduce micro-operations and present application-specific instructions added to support fast pointer manipulation, explicit carry handling, and ISA support for schedulers and timers. Given all these ISA optimizations, we achieve about a 17% compaction in code size and a 36% reduction in the size of decoder logic, compared to our first-generation CISC ISA design.

We highlight a number of successful microarchitectural optimizations that improve CPI and energy-per-instruction. We introduce a memory system architecture with row-address predecode, and include ISA extensions that permit the programmer to specify the most likely storage addresses for future accesses. Accesses to this active page result in faster memory access and lower energy demands. In addition, we introduce instruction prefetching, branch speculation, and limited out-of-order execution that allows most memory access latencies to be tolerated.

We simulated our design using a SPICE-level simulation environment developed for our firstgeneration design, and subsequently validated against our manufactured first-generation design. We find that the design features introduced in this paper compose a variety of design solutions that realize a number of pareto-optimal design. Our lowest energy design requires 600 fJ/instruction, our fastest design performance level is at 0.115 MIPS with a 660 fJ/instruction energy demand.

We are in the process of taping-out this design to be manufactured by IBM in their 130nm CMOS fabrication technology. The manufactured design includes on-chip solar cells as power sources. In addition, it will include a selection of processor and memory implementations built with a variety of standard cell designs, ranging from low-Vth commercial cells to high-Vth experimental cells optimized for subthreshold operation. The test memories range from standard memories, implemented with MUX-combined SRAM arrays to experimental memories with 4- and 3-transistor low-leakage SRAM cells. Finally, the test harness provides a SCAN interface between the outside world and all the state (memory and registers) contained on the test chip. In addition, the SCAN interface can be used to reset and restart any processor or test harness contained on the test chip. We will report on the results of this test chip in future work.

Finally, work is ongoing to incorporate our second generation sensor processor into an intraocular pressure sensor. The system is designed to grip the inner surface (vitreous) of the eyeball. The system will be installed via out-patient surgery, and it will provide patients with real-time feedback on the interior eye pressure. Recent medical studies have shown that careful monitoring and subsequent control of intra-ocular pressure can delay the onset of blindness in glaucoma and diabetes patients. The intra-ocular pressure measurement system includes a subthreshold sensor processor, 384 bytes of memory, 1024 bytes of ROM, a MEMOS-based pressure sensor, a Peltier-based energy scavenging mechanism (which utilizes temperature gradients within the eyeball to produce electricity), and a communication system based on inductive coupling. The entire system is currently under design, but a paper prototype has been shown to be less than 2 cubic-millimeters. The work in this paper significantly reduces the energy requirements of the sensor processor and memory system, permitting the system to be powered by energy scavenged from the human body. This effort is an ongoing collaboration with the Kellogg Eye Center at the University of Michigan.

Acknowledgements. This work is supported by grants from NSF and the Gigascale Systems Research Center.

8. REFERENCES

[1] ARM, Ltd. website. *http://www.arm.com*.

[2] Intel mote research project website.

[3] Monitoring in anaesthesia. In *http://www.liv.ac.uk/-afgt/Phys_Meas_Notes.pdf*.

[4] Multilog, multilogpro and ecolog weather stations. *www.fourier-sys.com/pdfs/data_loggers/*.

[5] Online resource for information on data compression. *http://www.data-compression.info/Algorithms/RLE*.

[6] University of california, los angeles, wireless integrated network sensors website. *http://wins.ucla.edu*.

[7] Wireless sensing networks project at rockwell scientific website. *http://wins.rsc.rockwell.com*.

[8] T. D. Burd. Subthreshold leakage modeling and reduction techniques. In *PhD thesis, University of California, Berkeley*, 2001.

[9] V. Ekanayake, C. Kelly, and R. Manohar. An ultra low-power processor for sensor networks. In *Proc. International Conference on Architectural Support for Programming Languages and OS*, 2004.

[10] M. Hempstead, N. Tripathi, P. Mauro, G.-Y. Wei, and D. Brooks. An ultra low power system architecture for sensor network applications. In *International Symposium on Computer Architecture*, 2005.

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. E. Culler, and K. S. J. Pister. System architecture directions for networked sensors. In *Architectural Support for Programming Languages and Operating Systems*, pages 93–104, 2000.

[12] L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, T. Austin, and D. Blaauw. A 180 khz, 185 mv sensor network processor. In *in prepration*.

[13] L. Nazhandali, B. Zhai, J. Olson, A. Reeves, M. Minuth, R. Helfand, S. Pant, T. Austin, and D. Blaauw. Energy optimization of subthreshold-voltage sensor network processors. In *International Symposium on Computer Architecture*, 2005.

[14] C. Schurgers and M. B. Srivastava. Energy efficient routing in wireless sensor networks. *MILCOM*, Oct. 2001.

[15] B. A. Warneke and K. S. Pister. An ultra-low energy microcontroller for smart dust wireless sensor networks. In *Proc. International Solid-State Circuits Conference*, 2004.

[16] D. J. Wheeler and R. M. Needham. TEA, a tiny encryption algorithm. *Lecture Notes in Computer Science*, 1008, 1995.

[17] B. Zhai, D. Blaauw, D. Sylvester, and K. Flautner. Theoretical and practical limits of dynamic voltage scaling. In *Proc. Design Automation Conference*, 2004.