Low Power TinyOS Tuned Processor Platform for Wireless Sensor Network Motes

R.K. RAVAL, C.H. FERNANDEZ, and C.J. BLEAKLEY
University College Dublin, Ireland.

In this article we describe a low power processor platform for use in Wireless Sensor Network (WSN) nodes (motes). WSN motes are small, battery-powered devices comprised of a processor, sensors, and a Radio Frequency transceiver. It is expected that WSNs consisting of large numbers of motes will offer long-term, distributed monitoring, and control of real-world equipment and phenomena. A key requirement for these applications is long battery life. We investigate a processor platform architecture based on an application-specific programmable processor core, System-On-Chip bus, and a hardware accelerator. The architecture improves on the energy consumption of a conventional microprocessor design by tuning the architecture for a suite of TinyOS based WSN applications. The tuning method used minimizes changes to the Instruction Set Architecture facilitating rapid software migration to the new platform. The processor platform was implemented and validated in an FPGA-based WSN mote. The benefits of the approach in terms of energy consumption are estimated to be a reduction of 48% for ASIC implementation relative to a conventional programmable processor for a typical TinyOS application suite without use of voltage scaling.

Categories and Subject Descriptors: C.3 **[Computer Systems Organization]**: Special-Purpose and Application-Based-Systems - *Microprocessor/microcomputer applications, real time, and embedded systems*

General Terms: Design, Performance

Additional Key Words and Phrases: embedded system design, hardware-software co-design, Wireless Sensor Network, low power processor

ACM File Format:
RAVAL, R.K., FERNANDEZ, C.H., AND BLEAKLEY, C.J. 2010. Low power TinyOS tuned processor platform for wireless sensor network motes. *ACM Trans. Des. Autom. Electron. Syst. x, x, Article x (Xxxx 200x), x pages,* DOI = xxxx http://doi.acm.org/xxx

1. INTRODUCTION

Wireless Sensor Networks (WSNs) consist of large numbers of small network devices or motes [Hill, 2003]. Each battery-powered mote consists of a processor, sensors, and a Radio Frequency (RF) transmitter. It is envisaged that large numbers of motes will provide long-term monitoring, and control of real-world equipment and phenomena. Previously proposed applications include early detection of wild fire, building monitoring, structural monitoring, tracking wild animals, military surveillance, and medical research [Akyildiz, 2002]. One of the limiting factors in the deployment of WSN technology is battery lifetime. Currently, research is being conducted on all aspects of WSNs in order to meet the stringent requirements of small device size and long term, un-tethered operation.

Authors' addresses: RK. Raval, C.H. Fernandez, and C.J. Bleakley, UCD Complex and Adaptive Systems Laboratory (CASL), UCD School of Computer Science and Informatics (CSI), University College Dublin, Belfield, Dublin 4, Ireland; E-mail: rajkumar.raval@ucd.ie, carloshdez@gmail.com, chris.bleakley@ucd.ie.

Previous work on reducing the power consumption of WSN processors has focused on full custom processor development and circuit level optimization for low power, including the use of asynchronous logic and voltage scaling. In contrast, in this work, we investigate application-specific tuning of the processor Instruction Set Architecture (ISA) combined with offloading of computationally complex Operating System (OS) functions to hardware accelerators. The tuning method aims to reduce energy consumption while minimizing changes to the ISA such that software migration to the new platform is straightforward. The proposed tuned BlueDot ISA is based on the conventional Atmel ATmega128L ISA but frequently used instructions are accelerated, infrequently used instructions are decelerated, commonly used instruction pairs are combined, RAM accesses are pipelined, and a commonly-used function is offloaded. In this way, the cycle count of TinyOS applications is significantly reduced, with a corresponding reduction in energy consumption. Software binary compatibility with the ATmega128L is maintained by means of a post-compilation software tool that re-maps existing machine instructions to new machine instructions.

To the authors' knowledge, this tuning approach has not been applied previously to WSN processors. The advantage of the approach is that significant energy savings can be achieved with minimal changes to the application software and tool flow.

The processor platform was implemented on an FPGA layer that was incorporated in a WSN mote. Verification and validation were significantly accelerated by implementation of the processor platform on the FPGA. The processor platform was synthesized for ASIC and the power consumption estimated by gate-level simulation. It was found that the tuning process reduced energy consumption by 48%, on average, over a range of TinyOS applications.

The remainder of the article is structured as follows. Section 2 describes the background to the work covering previous WSN processor designs and details the characteristics of TinyOS. Section 3 presents an analysis of the power consumption of a typical WSN application scenario. Section 4 presents the BlueDot architecture. Section 5 describes implementation of the processor platform. Section 6 gives the results of implementation in terms of clock frequency, area, and energy consumption. Finally, Section 7 concludes the article.

2. BACKGROUND

2.1 WSN Processors

The first WSN motes, developed at UC Berkeley, were based on Atmel AVR general-purpose 8-bit microcontrollers [Atmel, 2004]. The Atmel processor was selected due to its low power consumption, its support for Analog to Digital Converters and Real-Time Clocks and the availability of programming tool support. The ATmega128L is an 8-bit Harvard RISC architecture with 16-bit instructions, 32x8 bit general-purpose registers, peripheral control registers, and a 2-cycle multiplier. Currently, the most popular Commercial Off-The-Shelf (COTS) processors used in WSN motes are the Atmel ATmega128L and the Texas Instruments MSP430 [Texas Instruments, 2004]. Both are general-purpose microcontrollers. As such, their ISAs are not optimized for WSN applications [Polastre et al., 2005]. Further information on the performance and power consumption of these and other commercially available WSN processors can be found in Lynch et al. [2005].

Due to the need for lower power consumption and longer battery life, research is ongoing on developing lower power Application-Specific Processors (ASP) for WSN motes [Hempstead et al., 2008].

SNAP is a 16-bit fully asynchronous processor with an event driven architecture [Ekanayake, 2004]. It has a FIFO task dispatcher with message and timer co-processors. An asynchronous Atmel AVR clone was described in Necchi et al. [2006]. These processors achieve low power consumption. However, it can be difficult to integrate asynchronous logic into conventional, commercial synchronous design flows for low cost System-on-Chip (SoC) solutions. For this reason, asynchronous logic is considered unattractive for many applications.

Sub-threshold sensor processors, primarily for biomedical sensor network applications, are described in Nazhandali et al. [2005b] and Hanson [2008]. These designs achieve low power consumption. However, as described in Hanson [2008], sub-threshold logic is highly susceptible to temperature and process variations. In addition, due to the low voltage swing, noise arising from other on-chip SoC components could be an issue for commercial WSN applications.

In contrast, Spec is an 8-bit synchronous WSN mote processor with 16-bit instructions [Hill, 2003]. The ALU operations are single cycle but the memory operations take two cycles. The ISA contains 32 general-purpose registers and can address a 16-bit address space. The processor has a special bank of context switching registers to improve interrupt-handling efficiency. A number of functions are offloaded to hardware accelerators. However, the design of the RF hardware accelerator is such that it needs the processor to handle transmission and reception. As a result, the frequency of processor interrupts to handle communication is nearly the same as for the conventional Mica processor.

The Smart Dust microcontroller is a 12-bit synchronous processor that runs at 1V and employs component level clock gating and guarded ALU inputs [Warneke and Piste, 2004]. Power optimization focuses mainly on circuit-level optimizations to reduce switching activity.

The Nimbus processor was developed by optimization of an Atmel AVR microprocessor clone for low power [Lorentzen, 2004; Leopold, 2004]. Power reduction was achieved by restricting the design to a subset of the AVR ISA, by using low power circuit design and by scaling the supply voltage. The project team also investigated an asynchronous variant of the Nimbus processor, called Disa.

The Hempstead platform consists of an event processor and a secondary general-purpose non-pipelined microcontroller [Hempstead et al., 2005; Hempstead et al., 2009]. Low power optimizations focus mainly on voltage scaling and the use of hardware accelerators. System bus, timer, radio interface, and message processor sub-systems are included. Circuit level optimizations are considered in further work by the same authors [Hempstead et al., 2006].

μAMPS consists of a 16-bit synchronous Digital Signal Processor (DSP) and a Fast Fourier Transform (FFT) coprocessor connected via a shared bus [Finchelstein, 2005]. The architecture includes voltage scaling and has a DMA engine.

Of these processors, only the Spec has extensive programming tool support, due to the fact that it is compatible with the Atmel AVR ISA. The other processors have custom ISAs. To the authors' knowledge, TinyOS has not been ported to the other processors, making software migration a time consuming task. None of the previous work considers optimization of the Atmel microprocessor ISA, other than the simple sub-setting

approach used in the Nimbus processor. The application-specific ISA tuning approach described herein may be combined with circuit level optimizations, such as sub-threshold voltage and asynchronous logic to further reduce power consumption.

2.2 WSN Processor Workload

TinyOS [Hill et al., 2000] is an application specific operating system for WSN motes originally developed by UC Berkeley. TinyOS has become the de-facto standard in the WSN research community due to its open source availability, small footprint, portability, and development tools [Hill et al., 2004; Levis et al., 2005, Gay et al., 2007],

TinyOS has a component-based architecture and uses a C-like programming language - NesC. TinyOS components interact with each other in one of three ways - via commands, events or tasks. A command is a request to perform some service, such as to initialize a sensor, while an event signals completion of that service. An event is a notification of a condition from a lower-level module to a higher-level handling module. Rather than performing a computation immediately, command and event handlers post tasks to a scheduler. The function is then scheduled and initiated later by TinyOS.

Simulators for TinyOS are described in [Shnayder et al., 2004]. Avrora, a fast cycle accurate instruction-level simulator for WSN motes, was used in this work [Titzer, 2005].

To the authors' knowledge, there are no publically available software benchmarks for WSN applications. The authors of Hempstead et al. [2004] proposed the creation of TinyBench, a standardized benchmark suite for TinyOS. However, the benchmark suite, although proposed, was not actually released. The authors of Nazhandali [2005a] proposed a set of C applications representing real time workload applications of a WSN processor. Again, the software is not publically available. A WiSeNBench workload was proposed and assessed on the ARM processor in Mysore et al. [2008]. The work did not go on to propose or assess the effect of ISA optimizations.

The TinyOS distribution itself includes a comprehensive set of applications [TinyOS, 2009], a subset of which is used in this research for workload analysis in place of a formal benchmark.

3. WORKLOAD ANALYSIS

In order to determine the most effective processor platform architecture, a workload analysis was conducted for a reasonable, computationally tractable, WSN application scenario. A 3x3 grid of 9 nodes was considered, as shown in Figure 1. All nodes are Mica2 motes with the Atmel ATmega128L processor on-board. All nodes run TinyOS based application software. Three leaf nodes, nodes 5, 7 and 8, sense light and send measurements to the intermediate routing nodes (Surge) to be forwarded to the base node, node 0. A 4 tap simple Finite Impulse Response (FIR) averaging filter application was developed to represent the workload of a typical WSN processor intensive application. Node 8 was programmed with this application. It filters the ambient light samples read by the ADC and sends the data via RF. Nodes 1, 3 and 4 sense light, send their data to the base node and forward packets coming from leaf nodes. Node 2 was programmed to generate regular radio traffic by broadcasting counter values. Node 6 monitors the activity in the network by displaying any packet received on the Light Emitting Diodes (LEDs). Node 0 executes the TransparentBase application that sends packets from the host PC to the WSN and receives packets from the WSN and forwards them to the host PC via a UART.

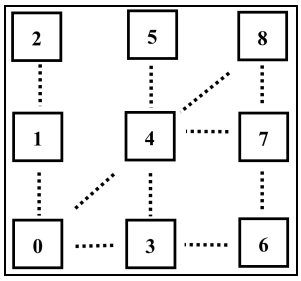

Fig. 1. Network topology.

Table I. Functions with the largest proportion of workload per node.

Node	Application	Function	Clock Cycles (%)
1,3,4	Surge	Interrupt Handler for RF-SPI interface	38
		Main (OS kernel and scheduler)	15
		nesC atomic	6.5
		Timer Interrupt	1.5
5,7	SenseToRfm	Interrupt Handler for RF-SPI interface	34
		Main (OS kernel and scheduler)	13
		ADC related interrupt	2.5
		Timer Interrupt	1.3
8	FirToRfm	Interrupt Handler for RF-SPI interface	18.7
		Main (OS kernel and scheduler)	8
		ADC related interrupt	4.5
		nesC atomic	4
0	TransparentBase	Interrupt Handler for RF-SPI interface	44
		Main (OS kernel and scheduler)	16
		Timer Interrupt	1
		ADC	1
6	RfmToLeds	Interrupt Handler for RF-SPI interface	43
		Main (OS kernel and scheduler)	16
		Timer Interrupt	0.8
		ADC	0.7
2	CntToRfm	Interrupt Handler for RF-SPI interface	38
		Main (OS kernel and scheduler)	13
		Timer Interrupt	1.3
		ADC	1

An automated software system was developed for profiling the workload. An XML configuration was used to store the configuration parameters. The executable was generated from the nesC source using the avr-gcc compiler and input to Avrora to obtain the instruction trace and the function trace. A further program post-processed and summarized the log files. The network was simulated for 300 seconds of virtual time. Use of a larger network or longer simulation time was desirable but the log files generated become impractical to process.

The instruction execution trace was sorted and analyzed to identify frequently used instructions and pairs of instructions. In addition, the function execution trace was analyzed to identify frequently used functions, their hierarchy and the number of instructions executed as part of them.

Due to the fact that TinyOS is an event driven OS, most of its functionality is contained in interrupt routines. Table I summarizes the TinyOS functions with the greatest workload for each node. The interrupt handler routine for the SPI-RF interface dominates, consuming 30-50% of active processor cycles. Main, the OS kernel function, consumes 10-20% of the total workload across all of the applications. The other dominant functions are nesC-atomic, Timer and ADC related interrupt routines.

Table II lists the 10 most frequently used instructions across all applications. These instructions constitute 40-70% of the total instruction workload, depending on application. The most frequently used are the stack manipulation instructions, push and pop. Results show that 60-80% of accesses to data memory are to the stack, i.e. the bottom 20-25 locations in the Atmel ATmega128L address space.

Table III lists the most frequently occurring consecutive pairs of instructions. These instruction pairs constitute 25-35% of all of the pairs encountered in each application trace. The most frequently occurring instruction pairs are pop-pop and push-push.

Table II. Most frequently used instructions, all applications.

Instruction	**Description**	**Instruction usage (%)**
PUSH	Stack manipulation	19
POP	Stack manipulation	19
LDS	Load direct from SRAM	6
BRNE	Conditional Branch	5
RJMP	Relative jump	4.5
RET	Return from subroutine	4
CALL	Call subroutine	4
CPI	Compare immediate	3
CPC	Compare with carry	3
MOV	Move between registers	3
Total		70.5

Table III. Most frequently occurring instruction pairs, all applications.

Instruction pair	**Instruction pair usage (%)**
POP & POP	6
PUSH & PUSH	6
CPI & CPC	3.7
MOV & EOR	2.5
CPC & BRNE	2.3
LDS & MOV	1.9
PUSH & IN	1.7
BRNE & CPI	1.6
AND & BRNE	1.4
SUBI & SBCI	1.2
Total	28.3

4. PROPOSED ARCHITECTURE

4.1 Processor Platform

Herein, we propose a processor platform, called BlueDot, which supports energy efficient execution of TinyOS based WSN applications. The processor platform is intended to replace the conventional processor in a WSN node. Since replacing the conventional processor with a BlueDot processor platform does not alter the functionality of a node, a node containing the new platform can be used together with conventional nodes in the same network.

The BlueDot processor platform architecture consists of an application-specific programmable processor core and a hardware accelerator interconnected via a System on Chip (SoC) Bus.

4.2 Processor Core

The BlueDot processor core ISA was determined by tuning the conventional ATmega128L ISA based on the results of the workload analysis. The following optimizations were applied:

- Frequently used multi-cycle instructions were modified to only use one clock cycle
- Frequently used pairs of instructions were merged into single instructions
- Interrupt handling time was significantly reduced
- Instructions were added to enhance communication with hardware accelerators
- In some cases, Data RAM accesses were advanced to speed up execution

In addition, to facilitate software compatibility the conventional Atmel microcontroller, two modes of operation are supported:

- Compatible Mode: ATMega128L binary compatible
- Advanced Mode: close to ATMega128L binary compatible

Software compiled for the Atmel ATMega128L will execute without modification on the BlueDot processor core when in Atmel Compatible Mode. Taking advantage of the optimizations available in Advanced Mode requires that compiled ATMega128L machine code be post-processed, either manually or automatically, to convert some ATMega128L instructions to optimized BlueDot instructions. This post-processing was done after compilation using a script to substitute BlueDot accelerated equivalents for ATMega128L machine instructions.

Table IV lists the accelerated instructions and gives the speed-up achieved in both Compatible and Advanced Mode. Compatible Mode accelerates commonly used existing instructions. Some instructions cannot be accelerated due to Data RAM access limitations. Others cannot be accelerated due to the length of their opcodes. For example, an instruction with a 32-bit opcode, which requires a 2 cycle fetch, takes up two slots in the pipeline regardless of the speed of the execution step. In Advanced Mode, opcodes can be altered. Hence some frequently used instructions can be further accelerated. However, due to the limited number of opcodes available, some 16-bit instructions had to be decelerated, i.e. adjusted to 32-bit opcodes, to achieve this. RCALL and ANDI were decelerated for this reason. The new compound instructions added to Advanced Mode are described in more detail in Table V. Overall, Advanced Mode provides acceleration over Compatible Mode due to the inclusion of these new compound instructions, even though a small number of instructions are actually decelerated relative to Compatible Mode.

Three additional instructions were added to allow for direct communication between the processor core and the SoC Bus. These are listed in Table VI.

Table IV. Accelerated instructions.

Instruction	Cycles Atmel	Cycles BlueCore (compatible)	Cycles BlueCore (advanced)	Savings (normal)	Savings (advanced)
Single Instruction Optimizations					
PUSH	2	1	1	1	1
POP	2	1	1	1	1
INT HANDLING	4	2	2	2	2
RET	4	3	3	1	1
RETI	4	3	3	1	1
CALL	4	3	3	1	1
RCALL	3	2	3	1	0
ICALL	3	2	2	1	1
SBI	2	1	1	1	1
CBI	2	1	1	1	1
ST(INDIRECT)	2	1	1	1	1
STD	2	1	1	1	1
ADIW	2	1	1	1	1
SBIW	2	1	1	1	1
ANDI	1	1	2	0	-1
Instruction Pair Optimizations					
INPUSH	3	-	1	-	2
POPOUT	3	-	1	-	2
SGBCLR	2	1	1	1	1
MOVCALL	5	-	3	-	2
CLIMOV	2	-	1	-	1
LDSCPSE	3/4/5	-	2/3/4	-	1
CLRLDS	3	-	2	-	1
NOP & SBIW	-	-	-	-	-
INC & SEC	2	1	1	1	1
NOPM	-	-	-	-	-

The RAM memory used for the processor is a synchronous memory with combinational inputs. In pipelined execution, writes can be performed in one cycle. Reading takes two cycles - the first to set the address and the second to obtain the data, see Figure 2(a). This arrangement minimizes the critical path of the circuit. Instructions such as POP, RET, RETI, LD and LDS must access the RAM in read mode. The performance impact is significant, given that these instructions are among the most frequently used. The solution adopted herein is to bring the read process forward one cycle, as shown in Figure 2(b), putting the address on the address bus while still executing the previous instruction. This solution works well when the previous instruction does not make use of the address bus to access the memory, otherwise a resource conflict occurs. If a conflict is detected, the read is not brought forward which resolves the conflict. The processor core switches between forward and normal read mode depending on the instructions that are being executed. Table VII lists all possible resource conflicts. If any of the instructions in the second column are executed after any

of the instructions in the first column then the forward read process is disabled and the instruction in the second column executes the read process normally.

Table V. New Advanced Mode instructions.

Instructions	Description
INPUSH	Move the contents of an I/O register to a general-purpose register and to the top of the stack.
POPOUT	Move the value at the top of the stack to a general-purpose register and to an I/O register.
SGBCLR	Move the Status Register contents to the specified general-purpose register and clear a bit in the Status Register.
MOVCALL	Move the contents of one general-purpose register to another, store the Program Counter to the Stack, and call a subroutine.
CLIMOV	Move the contents of a general-purpose register to another and clear the global interrupt flag (I) in the Status Register.
LDSCPSE	Load a byte from data memory into a general-purpose register, compare the value to that of another general-purpose register, and skip the next instruction if the values are the same.
CLRLDS	Clear a general-purpose register, load one byte from data memory, and store it in a general-purpose register.
NOPSBIW	Perform the SBIW operation in 2 cycles.
INCSEC	Increment the value stored in a general-purpose register and set the carry flag in the Status Register.

Table VI. SoC Bus communication instructions.

Instruction	Description
WDIF	Move an immediate value to the output of the FIFO RAM in the SoC Bus Interface.
WRIF	Move the contents of one general-purpose register to the output of the FIFO RAM in the SoC Bus Interface.
RRIF	Read a value from the input of the FIFO RAM in the SoC Bus Interface and store it in a general-purpose register.

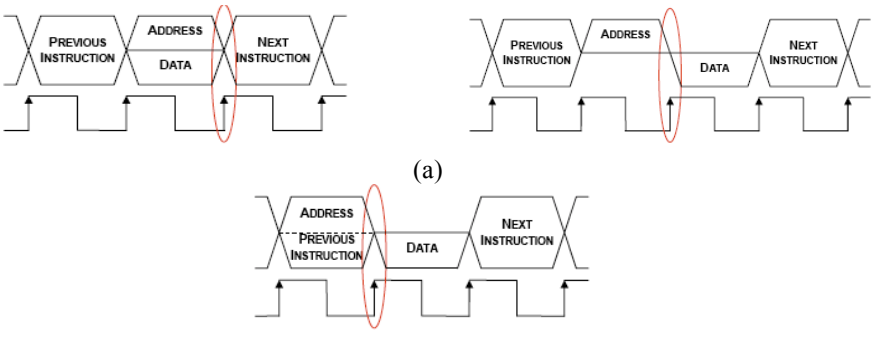

Fig. 2. (a) RAM data write in one cycle and data read in two cycles, (b) RAM read with look ahead.

Table VII. Instruction combinations giving rise to resource conflicts.

First Instruction	Second Instruction
RCALL	RET
ICALL	RETI
CALL	POP
ST	POPOUT
PUSH	
INPUSH	

4.3 SoC Bus

The SoC Bus was designed to provide low power communication between the Processing Units (PUs) in the design. A shared 8-bit Bus was chosen to provide for simple layout. To allow for Dynamic Voltage and Frequency Scaling (DFVS), the Bus clock operates asynchronously to the PU clocks and is re-timed in the Bus interface module of each PU. Delay tolerance was designed into the Bus to allow for PUs in deep sleep mode [Hu, 2004]. Messages are buffered on an as-needed basis.

The Bus logic is comprised of three main hardware blocks: the Arbiter, the Scheduler and the Interfaces. The Arbiter controls the Bus, the Scheduler is used to store messages missed by asleep or busy PUs and the Interfaces connect the PUs to the bus. The Bus and the Bus side of the Interface modules are synchronized to a clock generated by the Arbiter. The PUs themselves have independent clocks. There is typically no need to segment the Bus into smaller sections, although this can be done. Transmission requests are signaled to the Arbiter via a dedicated line per PU. In the case of a large number of PUs, encoders and decoders may be used to reduce the number of lines. The other Bus signals are shared.

The Interface module is divided into two separate blocks - reception and transmission. The two parts of the module operate independently. The Bus side of the blocks is clocked using the SoC Bus clock. The PU side is clocked with the PU clock. A handshaking protocol operating via a shared RAM is used to re-time and queue messages.

When the PU requests that the transmit block sends a message, the block requests a Bus grant from the Arbiter using the dedicated line. Once the Arbiter grants permission, the transmit block starts sending the message via the Bus, indicating this to the other PUs by setting the ‘message being sent’ signal. The PU clears the request signal as soon as the transmission has begun. The message consists of the destination PU Id and a series of data bytes. The data bytes are sent synchronously to the Bus clock. The receiving module clocks the data into the Interface component if the PU Id on the Bus matches its own, hard-wired PU Id. The transmission finishes when the last byte is sent, which is indicated with a ‘bus last byte’ signal. At this moment, the transmitter clears the message being sent to indicate that the Bus is available to transmit a new message.

The SoC Bus protocol and architecture are described in more detail in Fernandez et al. [2007].

4.4 Hardware Accelerator

As described in Section 3, the interrupt handler routine for the Radio-SPI component is the most frequently used function in the WSN workload. This routine initializes the radio chip and transmits and receives packets over the radio interface. Communication with the radio chip is performed using a SPI interface. In the Mica2 mote, the SPI interface

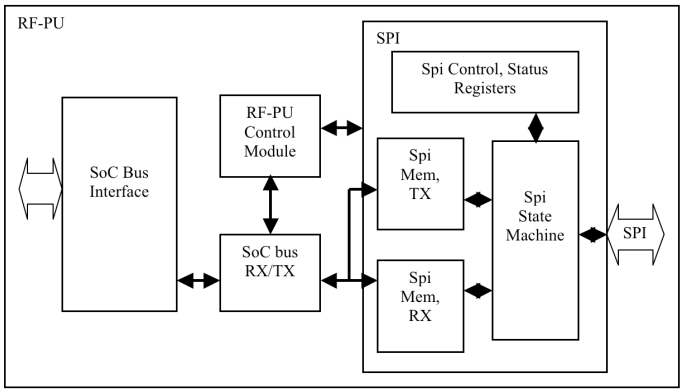

Fig. 3. Radio-SPI processing unit.

interrupts the Atmel processor whenever a byte is received or transmitted, which keeps the processor active and awake for long periods of time with little activity. Porting this functionality to dedicated hardware can save energy, as it can reduce processor activity and speed up execution. The processor itself is only interrupted when a complete frame is received, rather than every time a byte is received. Similarly, the processor sends the complete frame to the dedicated hardware block and switches to sleep mode, leaving the handling of the transmission to the PU.

A functional block diagram of the Radio-SPI PU is shown in Figure 3. The Control Module state machine is the main controller for the PU. It decodes the commands on the SoC Bus, accesses the internal registers and controls the transmission and reception of the frames via the two interfaces (the Radio and SoC Bus). To communicate with the radio chip, the necessary SPI signals are generated. This is done by means of the SPI state machine, which sends these signals and implements internal control, status and extended status registers. There are two blocks of memory for transmission and reception of messages over the SPI interface. These are used in a FIFO manner. Interrupts to the processor are also generated in this block.

The SPI protocol uses four wires to synchronously communicate with the radio, with the Radio-SPI processing unit acting as Master. A set of registers is used to configure communication with the radio chip.

Further hardware accelerators can be added to the architecture without modify the existing PUs. New PUs are simply interfaced to the SoC Bus and allocated a new PU Id.

5. IMPLEMENTATION

The BlueDot processor platform was implemented in Verilog. Depending on the software, the processor core within the platform could run in one of three modes. In Baseline Mode it operates as a clone of the ATmega128L with exactly the same cycle count. In Compatible Mode it executes the same ATmega128L machine code but some instructions are accelerated, reducing the cycle count. In Advanced Mode, the machine code is modified to allow for use on the new compound instructions, further reducing cycle count.

Verification was performed at the block and module level using ModelSim. The processor platform was validated using a Tyndall WSN node [O'Flynn et al., 2005]. The node consisted of a number of PCB layers and a coin battery. The layers include an

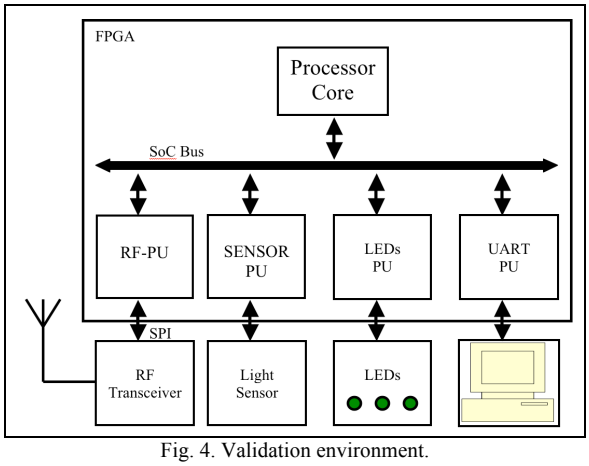

Fig. 4. Validation environment.

ATmega128L processor, light, and temperature sensors, memory, Analog to Digital Converters (ADCs), Xilinx Spartan FPGA, and T.I. CC2420 radio chip. The processor and FPGA are inter-connected so as to allow communication between the processor and FPGA and to allow control of the node hardware by either the processor or FPGA. This enables rapid development by allowing initial software implementation on the processor and incremental offload of functionality to the processor platform on the FPGA. The test environment is illustrated in Figure 4. The UART provides for monitoring of SoC Bus traffic and manual transmission of Bus messages via a PC terminal window application. This feature allows manual control of the system and significantly accelerated debugging of the design. A simple RF transmission and reception application was built to validate the design. Sensor and LED control PUs were constructed to test the functionality of the SoC Bus and to assess the flexibility of the functional offload concept.

To evaluate the performance of the design, the cycle count figures obtained from ModelSim instruction simulations were compared with those of the conventional ATmega128L. To obtain estimates of area, power and timing, the processor was synthesized for ASIC using Synopsys DesignCompiler with a Faraday-UMC 0.13µm low leakage CMOS technology library. The memory access time was assumed to be 4.86ns, worst case. The Synopsys PrimePower tool was used for power estimation. The area figures provided exclude interconnect and scan test. The power and energy estimates were obtained pre-layout but include manual estimates for memory access and clock tree energy, leakage is included.

6. RESULTS

The ASIC area and maximum clock frequency of the various BlueDot components are provided in Table VIII. The average energy/instruction of the processor core was 26.42pJ. The average energy per cycle measured for the processor core was 16.10pJ. The number of cycles and energy required for execution of the TinyOS applications was assessed for the Baseline processor only and for the full BlueDot processor platform including Advanced Mode processor core and RF-PU. The results are presented in Table IX. The full BlueDot platform provides significant energy savings of up to 56% over the Baseline processor. Speed-ups in terms of cycle count of up to 61% were achieved, with an average of 55%.

Table VIII. ASIC area and maximum clock frequency.

Module	Area (G.E.)	Max. Freq. (MHz)	Area (%)
BlueCore	14,825	59	47.65
BlueCore-SRAM (2KB)	9,339	--	30.01
RFPU	1,764	~ 200	5.67
RFPU-SRAM (64B)	292	--	0.94
Scheduler	1,952	~ 200	6.27
Scheduler-SRAM (576B)	2,626	--	8.44
Arbiter	317	~ 200	1.02
Total Area	31,114	--	100

Table IX. Cycle count and total energy per application.

Benchmark	Baseline Processor Only		BlueDot Processor Platform		Savings	
	Cycles	Energy (mJ)	Cycles	Energy (mJ)	Cycles (%)	Energy (%)
Surge	15,5411,318	2.56	65,423,405	1.20	57.9	53.1
SenseToRfm	169,652,976	2.83	71,970,712	1.35	57.6	52.3
TransparentBase	133,221,340	2.19	51,472,876	0.96	61.4	56.2
CntToRfm	153,625,633	2.53	66,858,237	1.23	56.5	51.4
RfmToLeds	135,800,032	2.24	52,544,705	0.99	61.3	55.8
FirToRfm	348,171,711	5.66	216,855,729	3.64	37.7	35.7
Average	182,647,168	3.00	87,520,944	1.56	55.4	50.7

Table X. Energy consumption of architectural instances averaged over all applications.

Architecture Instance	Baseline Processor	Compatible Mode Processor	Advanced Mode Processor	RF-PU	Average Energy Cons. (mJ)	Average Energy Cons., (%)
1	√	-	-	-	3.00	100
2	-	√	-	-	2.39	80
3	-	-	√	-	2.28	76
4	√	-	-	√	2.29	76
5	-	√	-	√	1.68	56
6	-	-	√	√	1.56	52

The number of bytes transferred between the ATmega128L and the CC2420 radio transceiver in the SenseToRfm application was 719,745 bytes, estimated over 300 virtual seconds of simulation of the 9 node sensor network. On the BlueDot platform, the energy/bit to communicate this data over the SoC Bus was 5pJ per bit measured over a data length of 16 bytes.

The gate-level estimated energy consumption averaged over all applications in the workload for various architectural instances is listed in Table X and shown in Figure 5. All three processor core variants are considered. All of the processor variants were assessed with and without the RF-PU. Adding the RF-PU to the Baseline processor (instance 1 to instance 4) provides the largest power savings (-24%). Moving from the

Baseline processor to the BlueDot processor in Compatible Mode (instance 1 to 2) provides comparable savings (-20%). Switching from Compatible to Advanced Mode (instance 2 to 3) provides a small energy saving (-4%). The full BlueDot platform with the processor operating in Advanced Mode provides an energy saving of 48%.

The memory footprint improvement of the Advanced Mode BlueDot processor over the Atmel ATmega128L is 3.0 – 5.75%, depending on application.

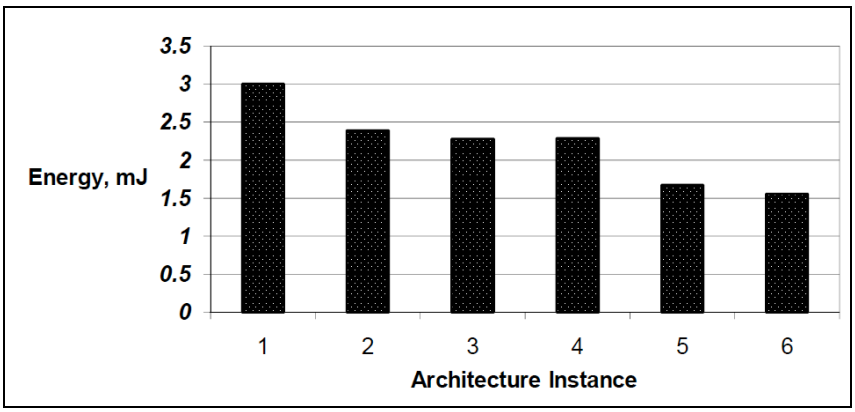

Fig. 5. Energy consumption of various architectural instances averaged over all applications.

7. CONCLUSION

This article described an application-specific processor platform for TinyOS based WSN nodes. The platform provides significant energy savings but maintains near binary compatibility with the conventional ATmega128L microcontroller. The article describes a workload analysis of WSN applications, the processor platform architecture, an application-specific processor core, the SoC Bus interconnect and the RF interface hardware accelerator. The platform was validated in an FPGA embedded in a WSN mote. The energy savings due to the various energy optimizations were assessed and compared with conventional implementation on a microprocessor with an ATmega128L ISA.

The BlueDot platform consumes 48% less energy than the Baseline ATmega128L-equivalent processor when executing the same WSN application suite. The work shows that minor changes to the ISA and offload of a small, but frequently used function, can provide significant energy savings. Making minor changes to the ISA has the significant advantage of ensuring near binary compatibility, which eases adoption of the new platform.

The speed-ups introduced in this work may allow for further energy savings in other mote components. For example, faster interfacing may allow the radio to be switched off more quickly, thus saving energy consumption in the radio. The authors intend to investigate the impact of these processor optimizations on higher layer workloads. The authors also plan to extend the work to investigate the effect of circuit-level optimizations. The ISA tuning approach used herein is generally applicable; we plan to study its use in other application areas.

ACKNOWLEDGEMENTS

This work was supported by Enterprise Ireland (ref: PC/2005/175) and by Science Foundation Ireland (ref: NAP63).

REFERENCES

AKYILDIZ, I.F., SU, W., SANKARASUBRAMANIAM, Y., AND CAYIRCI, E. 2002. A survey on sensor networks. *IEEE Communications Magazine*, 40(8), 102-114.

ATMEL CORPORATION. 2004. *8-bit AVR Microcontroller with 128K Bytes In-System Programmable Flash.*

EKANAYAKE, V., KELLY, C., AND MANOHAR, R. 2004. An ultra low-power processor for sensor networks. *ACM SIGOPS Operating Systems Review*, 38(5), 27-36.

FERNANDEZ C. H., RAVAL R. K., AND BLEAKLEY C. J. 2007. GALS SOC interconnect bus for wireless sensor network processor platforms. In *Proceedings of the 17th Great Lakes Symposium on VLSI* (*GLSVLSI*), Stresa-Lago Maggiore, Italy, 132-137.

FINCHELSTEIN, D. F. 2005. *Low-power Digital Processor for Wireless Sensor Networks*. Master's thesis, Massachusetts Institute of Technology.

GAY, D., LEVIS, P. AND CULLER, D. 2007. Software design patterns for TinyOS. In *ACM Trans. on Embedded Computing Systems*, 6(4), 22.

HANSON, S., ZHAI, B., SEOK, M., CLINE, B., ZHOU, K., SINGHAL, M., MINUTH, M., OLSON, J., NAZHANDALI, L., AUSTIN, T., SYLVESTER, D., AND BLAAUW, D. 2008. Exploring variability and performance in a sub-200-mV processor. In *IEEE Journal of Solid-State Circuits*, 43(4), 881-891.

HEMPSTEAD, M., WELSH, M., AND BROOKS, D. 2004. Tinybench: The case for a standardized benchmark suite for TinyOS based wireless sensor network devices. In *Proceedings of the 29th IEEE International Conference on Local Computer Networks* (*LCN*), Washington, DC, 585-586.

HEMPSTEAD, M., TRIPATHI, N., MAURO, P., GU-YEON WEI AND BROOKS, D. 2005. An ultra low power system architecture for sensor network applications. In *Proceedings of the 32nd International Symposium on Computer Architecture*, 208-219.

HEMPSTEAD, M., WEI, G.Y., AND BROOKS, D. 2006. Architecture and circuit techniques for low-throughput, energy-constrained systems across technology generations. In *Proceedings of the International Conference on Compilers, Architecture and Synthesis for Embedded Systems* (CASES), 368-378.

HEMPSTEAD, M., LYONS, M.J., BROOKS, D. AND WEI, G.Y. 2008. Survey of hardware systems for wireless sensor networks. In *Journal of Low Power Electronics*, 4(1), 11-20.

HEMPSTEAD, M., WEI, G.Y. AND BROOKS, D. 2009. An accelerator-based wireless sensor network processor in 130nm CMOS. In *Proceedings of the 2009 Int. Conf. on Compilers*, 215-222.

HILL, J. 2003. *System Architecture for Wireless Sensor Networks*. PhD thesis, Department of Computer Science, University of California, Berkeley.

HILL, J., HORTON, M., KING, R., AND KRISHNAMURTHY. 2003. The platforms enabling wireless sensor networks, In *Communications of the ACM*, 47(6), 41-46.

HILL, J., SZEWCZYK, R., WOO, A., HOLLAR, S., CULLER, D. AND PISTER, K. 2000. System architecture directions for networked sensors. In *Proceedings of the 9th International Conference on Architectural Support for Programming Languages and Operating Systems* (*ASPLOS-IX*), New York, NY, 93-104.

HU, Z., BUYUKTOSUNOGLU, A., SRINIVASAN, V., ZYUBAN, V., JACOBSON, H., AND BOSE, P. 2004. Microarchitectural techniques for power gating of execution units. In Proceedings of the International Symposium on Low Power Electronics and Design (ISLPED), Newport Beach, CA, 32-37.

LANDSIEDEL, O., WEHRLE, K., AND G□OTZ, S. 2005. Accurate prediction of power consumption in sensor networks. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors* (*EmNetS*), Sydney, Australia, 37- 44.

LEOPOLD M. 2004. *Power Estimation using the Hogthrob Prototype Platform*. Master's thesis, University of Copenhagen, Denmark.

LEVIS, P., MADDEN, S., POLASTRE, J., SZEWCZYK R., WHITEHOUSE, K., WOO, A., GAY, D., HILL, J., WELSH, M., BREWER, E., AND CULLER. D. 2005. Tinyos: An operating system for wireless sensor networks. In Weber, W., Rabaey, J., Aarts, E. (Eds.) *Ambient Intelligence*. Springer-Verlag, Berlin.

LORENTZEN, A. V. 2004. *Low-power Processors for the Hogthrob Project*. Master's thesis, Technical University of Denmark, Denmark.

LYNCH, C., AND O'REILLY, F. 2005. Processor choice for wireless sensor networks. In *Workshop on Real-World Wireless Sensor Networks* (*REALWSN'05*), Stockholm, Sweden, 1-5.

MUTTERSBACH, J., VILLIGER, T., AND FICHTNER, W. 2000. Practical design of globally asynchronous locally-synchronous systems. In *Proc. of the 6th International Symposium On Advanced Research in Asynchronous Circuits and Systems* (*ASYNC*), Washington, DC, 52-59.

MYSORE, S., AGRAWAL, B., CHONG, F.T., AND SHERWOOD, T. 2008. Exploring the processor and ISA design for wireless sensor network Applications. In *Proceedings 21st International Conference on VLSI Design*, Hyderabad, India, 59-64.

NECCHI, L., LAVAGNO, L., PANDINI, D., AND VANZAGO, L. 2006. An ultra-low energy asynchronous processor for wireless sensor networks. In Proceedings of 12th IEEE Symposium on Asynchronous Circuits and Systems, 80-85.

NAZHANDALI, L., MINUTH, M., AND AUSTIN, T. 2005a. Sensebench: Toward an accurate evaluation of sensor network processors. In *Proceedings of the IEEE International Workload Characterization Symposium* (*ISWC*), 197-203.

NAZHANDALI, L., MINUTH, M., ZHAI, B., OLSON, J., AUSTIN, T. AND BLAAUW, D. 2005b. A second-generation sensor network processor with application-driven memory optimizations and out-of-order execution. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems* (*CASES*), San Francisco, CA, 249-256.

O'FLYNN, B., BELLIS, S., DELANEY, K., BARTON, J., O'MATHUNA, S.C., BARROSO, A.M., BENSON, J., ROEDIG, U., AND SREENAN, C. 2005. The development of a novel minaturized modular platform for wireless sensor networks. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks* (*IPSN*), 370-375

POLASTRE, J., SZEWCZYK, R., AND CULLER, D. 2005. Telos: enabling ultra-low power wireless research. In *Proceedings of the* 4*th International Symposium on Information Processing in Sensor Networks*, Los Angeles, CA, Article 48.

SHNAYDER, V., HEMPSTEAD, M., CHEN, B., ALLEN, G. W., AND WELSH, M. 2004. Simulating the power consumption of large-scale sensor network applications. In *Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems* (*SenSys*), New York, NY, 188-200.

TEXAS INSTRUMENTS INC. 2004. *Datasheet MSP430 Mixed Signal Microcontrollers*.

TINYOS. 2009. Open source distribution and community. Available at http://www.tinyos.net.

TITZER, B.L., LEE, D.K., AND PALSBERG, J. 2005. Avrora: Scalable sensor network simulation with precise timing. In *Proceedings of the 4th International Symposium on Information Processing in Sensor Networks*, Article 67.

WARNEKE, B.A., AND PISTER, K.S.J. 2004. An ultra-low energy microcontroller for smart dust wireless sensor networks. In *Digest of Technical Papers IEEE International Solid-State Circuits Conference* (*ISSCC*), 1, 31.

Received July 2009; accepted March 2010