A Complete Design-Flow for the Generation of Ultra Low-Power WSN Node Architectures Based on Micro-Tasking

Muhammad Adeel Pasha, Steven Derrien, Olivier Sentieys
University of Rennes 1, IRISA/INRIA
Campus de Beaulieu
Rennes, France
[adeel.pasha,steven.derrien,olivier.sentieys]@irisa.fr

ABSTRACT

Wireless Sensor Networks (WSN) are a new and very challenging research field for embedded system design automation, as their design must enforce stringent constraints in terms of power and cost. WSN node devices have until now been designed using off-the-shelf low-power microcontroller units (MCUs), even if their power dissipation is still an issue and hinders the wide-spreading of this new technology. In this paper, we propose a new architectural model for WSN nodes (and its complete design-flow from C downto synthesizable VHDL) based on the notion of micro-tasks. Our approach combines hardware specialization and power-gating so as to provide an ultra low-power solution for WSN node design. Our first estimates show that power savings by one to two orders of magnitude are possible w.r.t. MCU-based implementations.

Categories and Subject Descriptors

B.1.2 [**Automatic Synthesis**]: WSN Node Controller

General Terms

Design, Experimentation

Keywords

Hardware Specialization, Low-Power Design, WSN

1. INTRODUCTION

Wireless Sensor Network (WSN) is a fast evolving technology having a number of potential applications in various domains of daily-life, such as structural and environmental monitoring, medicine, military surveillance, robotic explorations etc. A WSN is composed of a large number of sensor nodes that are usually deployed either inside a region of interest or very close to it. WSN nodes are low-power embedded devices consisting of processing and storage components (a processor connected to a RAM and/or flash memory) combined with wireless communication capabilities (RF transceiver) and some sensors/actuators.

Designing a WSN node is a daunting task, since the designers must deal with many stringent design constraints. For example, because these nodes must have small form-factors and limited production cost, it is not possible to provide them with significant power sources [23]. In most cases they must rely on non-replenishing (e.g. battery) or self-sufficient (e.g. solar cells) sources of energy. As WSN nodes may have to work unattended for long durations (month if not years), their energy consumption is often the most critical design parameter.

As far as their design is concerned, WSN nodes have until now been based on low-power MCUs such as MSP430 [22] and ATmega128L [3]. These programmable processors provide a reasonable processing power with low power consumption at a very affordable cost. Most of such MCU-packages also offer a limited amount of RAM (from a few hundred Bytes to a few kilo-Bytes) and non-volatile flash memory.

However, these processors are designed for low-power operation across a range of embedded system application settings. As a consequence, they are not necessarily well-suited to WSN node as they are based on a general purpose, monolithic compute engine. On the software end, WSN nodes generally rely on a light-weight Operating System (OS) layer to provide concurrency management for both external event handling and/or application task management.

Eventually, power dissipation of current low-power MCUs still remains orders of magnitude too high for many potential applications of WSN. We believe that the hardware specialization is an interesting way to further improve energy efficiency: instead of running the application/OS tasks on a programmable processor, we propose to generate an application specific micro-architecture, tailored to each task of the application at hand.

We propose such an approach where a WSN node architecture is made of several *micro-tasks* that are activated on an event-driven basis, each of them being dedicated to a specific task of the system (such as event-sensing, low-power MAC, routing, and data processing etc.). By combining hardware specialization with power reduction techniques such as *power-gating*, we can drastically reduce both dynamic (thanks to specialization) and static power (thanks to power-gating). The impact of the induced loss of flexibility is discussed in Section 3.

The contributions of this work are two-fold:

- We provide an integrated system-level design-flow for micro-task-based WSN architectures. In this flow, the behavior of each micro-task is specified in C and is mapped to an application specific micro-architecture using a modified version of a retargetable compiler infrastructure.
- We use this flow to perform design space exploration

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
DAC '10, June 13-18, Anaheim, California, USA
Copyright 2010 ACM 978-1-4503-0002-5/10/06 ...$10.00.

by exploring the trade-off in power/area that can be obtained by modifying the bitwidth of the generated micro-tasks, and compare the obtained results to those achieved by using an off-the-shelf low-power MCU such as the MSP430.

In particular, our experiments show that dynamic power savings by one to two orders of magnitude can be obtained for different control-oriented tasks of a WSN application (w.r.t. the MSP430). Besides, the static power profile of our architectural model also remains very low (thanks to power-gating).

The rest of this paper is organized as follows. We start by presenting the related work in Section 2 and highlight our proposed system-model and notion of micro-task in Section 3. Section 4, thoroughly covers our complete design-flow and in Section 5, we present experimental results for design space exploration and corresponding power benefits. Finally, we conclude and draw future research directions in Section 6.

2. RELATED WORK

As mentioned in the introduction, power efficiency is one of the most important design parameter for WSN nodes, and consequently a lot of efforts have been made to propose energy efficient implementations of such devices.

To design an energy-aware WSN node, it is mandatory to analyze its power dissipation characteristics. A canonical WSN node consists of four subsystems: i) a computing subsystem having an MCU, ii) a communication subsystem having RF transceiver, iii) a sensing subsystem having the sensor/actuator interfaces and iv) a power Supply subsystem. Among them, communication and computation subsystems consume bulk of the power-budget [20, 7], and it is generally accepted that efforts toward energy reduction should target both of them in particular.

As far as reduction in communication energy is concerned, there have been a lot of efforts put on the communication-stack energy optimization and low power RF technology, including energy-efficient routing algorithms, low-power MAC protocols, efficient error correction codes and power-aware transmission techniques (See [1] for a survey).

On the other hand, low-power micro-architectures and more specifically operating systems (OS) [6, 19] (customized to the WSN domain) have received a great deal of attention in the last few years. Even though an exhaustive survey is out of the scope of this work, we will highlight some of the most significant contributions in this domain.

2.1 WSN-related OS

Task, power and device management are some of the core features of a typical WSN OS. TinyOS [19] is one of the earliest and the most commonly used OS in WSN. It provides a component-based event-driven concurrency model without explicit thread management. Contiki [6] is another frequently used WSN OS that proposes a simplified thread execution model (named *protothread*), in which preemption can only occur at specific points in the Task Control Flow. TinyOS and Contiki have a memory foot-print of around 1 kB and 4 kB respectively, and hence require a significant amount of memory for their operation. Our approach is different since it allows multiple micro-tasks to run in parallel, each one having its own dedicated processing resource.

2.2 Power Analysis of WSN MCUs

As far as low-power micro-controllers (e.g. MSP430 and ATmega128L) are concerned, they share many characteristics: a simple datapath (8/16-bit wide), a reduced number of instructions (only 27 instructions for the MSP430), and several power saving modes which allow the system to select at runtime the best compromise between power saving and reactivity (i.e. wake up time).

Table 1: Actual and normalized power consumption for various low-power MCUs.

WSN MCU	Normalized Power	Actual Power
ATmega103L[2]	66 mW (@ 16 MHz)	5.5 mA (@ 4 MHz, 3.0V)
ATmega128L[5, 7]	48 mW (@ 16 MHz)	8 mA (@ 8 MHz, 3.0V)
MSP430F1611[9]	24 mW (@ 16 MHz)	500 μA (@ 1 MHz, 3.0V)
MSP430F21x2[22]	8.8 mW (@ 16 MHz)	250 μA (@ 1 MHz, 2.2V)

Most of the current WSN nodes are built on these commercial MCUs. For example, Mica2 mote [5] has been widely used by the research community and is based on Atmel's ATmega128L. The same MCU has also been used by the designers of the eXtreme Scale Mote (XSM) [7]. The Hydrowatch platform is built on the MSP430F1611 whereas Texas Instruments has launched a series of MCUs, MSP430F21x2, which is specialized for WSN nodes [22]. Table 1 summarizes power consumptions of these MCUs at a normalized frequency of 16 MHz along with the actual consumptions present in the literature.

It is an acknowledged fact that the power budget of a WSN node that would rely only on energy harvesting technologies is estimated to be around 100 μW [21]. Comparing this constraint with that of current MCUs power consumption profiles clearly drives us toward alternative architectural solutions (e.g. hardware specialization of the system).

2.3 Power Efficient Hardware Synthesis

High-Level Synthesis and retargetable compiler for ASIP (Application Specific Instruction Set Processor) have been a very active research domain for the last 15 years. Even if there are still many open research issues; there now exist several mature academic/commercial tools (e.g. XPilot [14], Spark [12], NISC [4, 11], Catapult-C from Mentor Graphics, Impulse-C, etc.) that are capable of producing specialized hardware description from software specification in C/C++.

Interestingly, all these tools share a common characteristic: they generally see hardware specialization as a mean to improve performance over a standard software implementation. This performance improvement however often comes at the price of an increased area cost (co-processor or instruction-set extension requires additional area). Of course, these specializations also have a significant impact on power efficiency, since they allow for a drastic reduction of dynamic power of the system.

Indeed, except from Fin et al. [8] and L'Hours et al. [16], very few papers have addressed the problem of using processor specialization as a mean to reduce silicon footprint. We believe that in the context of WSN node architecture, where silicon area and ultra low-power are the two main design issues, such an approach deserves attention.

The following section presents an original approach which builds on this idea. More specifically, we propose a new architectural model (along with a complete system-level downto gate-level design-flow) for WSN nodes design, based on the notion of *concurrent* power-gated hardware micro-tasks.

3. PROPOSED APPROACH

WSN applications, being event-driven in nature, can be represented as Tasks Flow Graphs (TFG) where a task execution is triggered by *events*, be they external or produced by another task. Fig.1 shows the TFG of a temperature-sensing application which periodically senses the temperature and may then send a message to its base station depending on the observed value. This application involves many

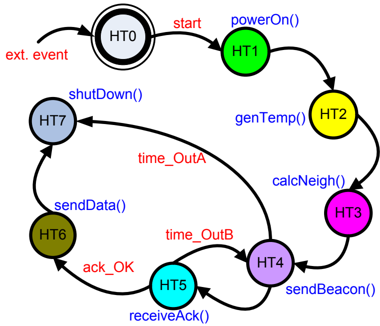

Figure 1: A task flow graph (TFG) presenting the micro-tasks running on a temperature-sensing node.

tasks: updating neighboring nodes information (for multi-hop routing), sending *beacons* to them for starting the data transmission, waiting for their acknowledgment, etc.

All these control-oriented tasks are spread across different layers of the communication stack and involve further sub-tasks associated to them. For instance, calculating the next node in a multi-hop system involves an efficient routing algorithm that is an integral part of network layer of the communication stack. Similarly, sending beacon and data packets involve physical layer functions that exchange data between I/O peripherals of MCU and RF transceiver using SPI-protocol. In typical WSN node, such tasks are handled by an MCU and corresponding OS that provides support for multi-tasking features.

In our approach, we propose to implement each of these concurrent tasks in the form of a set of customized hardware micro-tasks so as to drastically reduce its power dissipation thanks to specialization.

In this section, we briefly discuss our proposed system-level execution model and power-gating technique. Subsequently, we introduce the notion of *micro-task* and highlight the architecture of a WSN node based on micro-tasking.

3.1 System-level Execution Model

Our proposed node is based on an event-centric concurrency model where a hardware monitor turns-on/off each micro-task upon receiving a specific event or a set of events with the help of power-gating technique.

Power-gating is a well-known VLSI circuit-level technique used to reduce both dynamic and static power. It consists in adding a *sleep transistor* between the actual V_{dd} (power supply) rail and the component's V_{dd}, thus creating a *virtual supply voltage* called V_{vdd} as illustrated in Fig. 2. Sleep transistor allows the supply voltage of the block to be cut off to dramatically reduce leakage currents. It is different than clock-gating which only reduces the dynamic power of the circuit.

The basic goal of proposed execution model is to achieve an ultra low-power system with a simpler task-management strategy that best-suits our micro-task-based system architecture. The monitor is an FSM that will commence its execution when certain starting conditions are met and will run to finish. During each execution state, micro-tasks may be activated or deactivated according to events received by the monitor. To simplify the matter, we restrict ourselves to such multi-tasking system in which a micro-task can not be interrupted and runs to completion.

The monitor also ensures that no two micro-tasks having a shared -storage or I/O- resource are active simultaneously at a given time instant. This mutual-exclusiveness allows a drastic simplification of access control logic of shared resources resulting in a more efficient design in terms of area and power. In fact, there will be no need for tri-state or multiplexer logic at shared resource interfaces in our model.

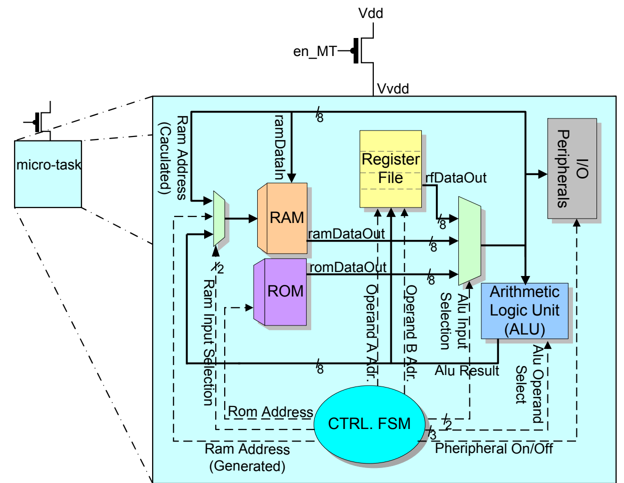

Figure 2: Architecture of a generic micro-task.

3.2 Notion of Micro-task

Each task of a TFG can be mapped onto a specialized hardware structure called a *micro-task*. This hardware can be seen as a small MCU datapath micro-architecture, driven by a control FSM that sequence/executes the micro-code corresponding to the task at hand. It can access some shared memories, and can be directly connected to some of the peripheral I/O ports.

At first glance, our approach may seem similar to the processor specialization of Gorjiara et al. [4] however, there are two significant differences. First, our main goal is to minimize the silicon footprint of the resulting micro-task, improving performance is only a secondary objective. Second, our micro-tasks should be seen as *temporary* computational resource, which can be completely powered off when not-needed (thanks to power-gating).

Fig. 2 shows the template of a micro-task architecture with an 8-bit data-path; dotted lines represent control signals generated by the control FSM whereas solid lines represent the data-flow connections between the various datapath components.

3.3 WSN Node Architecture

Fig. 3 presents the generic architecture of a node designed using micro-tasking. Main entities of the system consist of (i) a hardware monitor, (ii) a set of micro-tasks generated by our tool, and (iii) some external interfaces to I/O peripherals (sensor/actuator or RF-transceiver) that can send events to the monitor. Dotted lines represent command signals from the monitor to micro-tasks whereas solid lines represent external and internal events signals toward the monitor.

There are small locally shared memories used by micro-tasks that can be power-gated once their corresponding micro-tasks are shut down. We must emphasize that a system-level model (see Section 4.3) is used to specify, after a given task firing, which symbols (e.g. arrays) can be "killed" and this information is used to turn the shared memories on/off. This notion of small power-gated locally shared memories, instead of a large global one, will also contribute to the overall reduction in power consumption.

There is also a very small global memory (based on non-volatile flash technology) that is used to store the global data such as the node-ID, node-address, neighborhood table and if there is some potential data to be saved by the micro-tasks, in case of local memory shut-down. Since an *always-ON* memory can be critical from the point of view of static

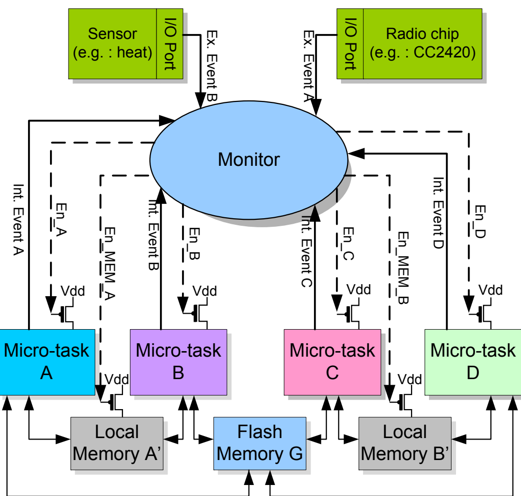

Figure 3: A system level architectural view of a node built using micro-tasks.

power dissipation, we use a non-volatile flash memory to store the needed data and then the whole system can be powered-gated to achieve virtually zero static power.

3.4 The issue of non-reprogrammability

As pointed out in the introduction, our approach assumes that the micro-tasks are hard-wired into silicon as application specific blocks. As a consequence, post-production upgrade or bug fixing becomes very difficult. This is obviously a very important issue, since flexibility is often of a great concern for WSN system designers.

However, the issue of programmability is often more related to the application layer which represents a small fraction of processing requirements. Indeed, most of the communication stack functions such as routing algorithms, MAC protocols and device drivers remain fixed. Hence, providing programmability in the form of a small foot-print programmable core would solve the problem while preserving most of the power saving (thanks to specialization).

Besides, there also exist alternative target technologies, e.g. *Structured ASICs*, which offer very low NRE costs, at a price of 2 to 5 times decrease in power efficiency. Given that our power savings are more than one order of magnitude, this approach would still remain extremely competitive.

The next section covers, in details, the proposed software design-flow used to generate the specialized micro-tasks.

4. SOFTWARE DESIGN FLOW

This section details our proposed software design-flow used to generate these customized micro-tasks from an application description written in C. We also highlight some features of a domain specific language (DSL) developed for system-level execution model description of our proposed WSN node. The generation of a micro-task is done automatically from a software specification of the task behavior in ANSI-C. Our compilation flow is based on a generic retargetable compiler infrastructure.

We use the *intermediate representation (IR)* produced by the compiler *front-end* which is in the form of a CDFG (Control and Data Flow Graph), in which instructions are represented as trees. We then use the retargetable instruction selection framework provided by the compiler to map this *IR* to the micro-architecture of each micro-task.

4.1 Instruction Selection Phase

An *IR* usually represents each basic operation (e.g. memory fetch or store, addition or subtraction, conditional jumps etc.) by a tree node. On the other hand, a real machine instruction often performs several of these basic operations at the same time. Finding the appropriate mapping of machine instructions to a given *IR*-tree is done through an instruction selection phase.

Even if there exist more sophisticated approaches for instruction selection (e.g. instruction selection of DAG [15]), our current implementation uses a simple BURG-based tree-covering algorithm [10]. The originality of our approach comes from the fact that we are not constrained by a pre-existing instruction-set, and we can therefore use a relatively large number of instruction patterns, so as to obtain an efficient covering.

For example, our micro-task uses several relatively complex instruction patterns involving memory operands so as to limit the need for load/store instructions. It also uses wordlength specific instructions (byte, word or long operand) so as to efficiently use the actual bitwidth of the micro-architecture datapath.

4.2 Micro-task Generation

The machine-specific *IR* obtained through instruction selection is then transformed into an FSM, in which each instruction is mapped to a sequence of micro-code (i.e FSM states) used to control the micro-task datapath.

This transformation stage also involves a wordlength conversion step in which instructions operating on 16-bit or 32-bit operands may be transformed into sequential byte-level microcode in order to match the characteristics of the underlying micro-task datapath.

From the set of instruction patterns used in the selection phase, we also derive a template of the micro-task datapath, which is trimmed down so as to provide the minimum-required functionality (types of operators and number of registers) required to execute the task at hand.

Our micro-task generation flow is build on top of *Eclipse Modeling Framework (EMF)*, a Model-Driven Engineering (MDE) framework. More precisely, we formulated a metamodel for micro-task architecture, and used the facilities for code generation provided by the framework to generate a synthesizable VHDL description for the micro-task (FSM + datapath).

4.3 Execution Model Generation

We also developed a Domain Specific Language (DSL) that is used to specify the system-level execution model of a WSN node. Due to space limitation, detailing the features and development of this model is out of scope for this paper, however for the sake of completeness, we outline its main characteristics.

A system-level execution model consists of the micro-tasks of a node and the events that will be generated and consumed by the micro-tasks upon their termination and activation respectively. The DSL was developed using Xtext (another MDE framework) and generates a VHDL description for the monitor.

Fig. 4 shows the complete design-flow for micro-task-based node generation: it starts from the application description modeled as a task graph using the DSL, each task written in C, and goes till the integrated circuit (IC) generation for the node.

5. EXPERIMENTAL VALIDATION

This section by briefly discusses the WSN benchmarks and OS tasks used for design space exploration and micro-task generation and later on, analyzes the experimental results

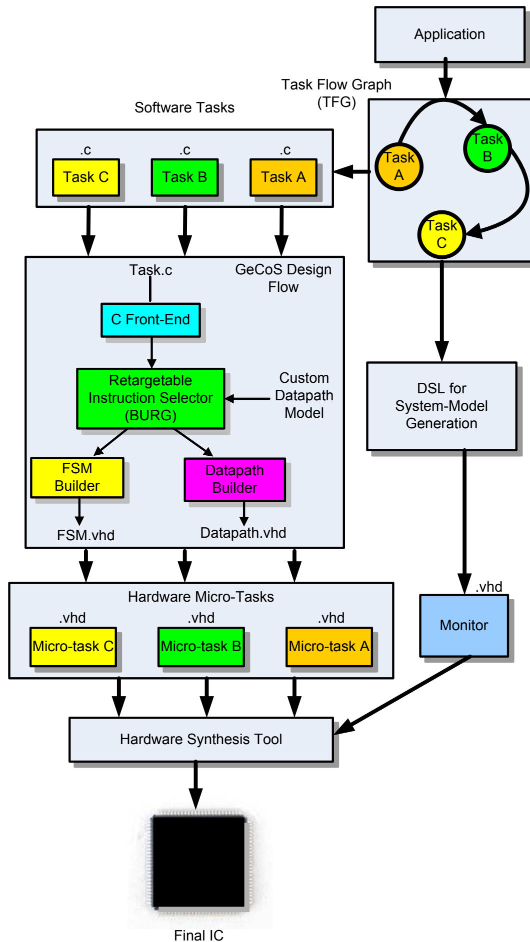

Figure 4: Software design flow for the IC generation and corresponding power savings.

5.1 WSN Benchmarks and OS Tasks

Several attempts have been proposed to profile the workload of a generic WSN node. Two of the recent application benchmarks for WSN are SenseBench [18] and WiSeN-Bench [17], from which we extracted most of our examples.

To cover the OS-task aspect, we also used several of the OS-related control-tasks such as a simplified next-node calculation function used in multi-hop geographical routing algorithm (similar to that was used by PowWow (an open-source WSN platform [13]), and the drivers used to exchange data with the SPI-interface of RF transceiver such as CC2420. All of the above application/OS tasks provide a comprehensive database from real-life WSN applications mostly used by WSN programmers.

5.2 Design Space Exploration

We used the proposed software design-flow to generate micro-tasks having both 8- and 16-bit datapaths to monitor the power savings as compared to commercial MCUs such as the MSP430.

We synthesized the datapaths that have optimized design parameters for different micro-tasks extracted from the above benchmarks. The synthesis was performed for 130 nm CMOS technology using Synopsys's *Design Compiler* and the power and area estimations are given in Table 2.

5.2.1 Power Savings w.r.t. Conventional MCU

The VHDL designs for complete micro-tasks have also

Table 2: Power consumption for datapaths having different design parameters (@ 16 MHz).

Bit Width	Register File Depth	RAM Depth	ROM Depth	ALU Fns.	Power (μW)	Area (μm^2)
8	4	4	4	8	57	7635
8	8	0	8	6	49.7	7038
8	16	0	2	4	69	11163
8	4	2	2	2	42	6160
8	16	2	4	6	93	13098
16	8	0	4	6	99	13184
16	4	4	4	8	116	14423
16	16	0	2	4	139.5	21590
16	4	2	2	2	88	11715

been synthesized for 130 nm CMOS technology. We used these synthesis results to extract gate-level static and dynamic power estimations (@ 16 MHz). These results were compared to the power dissipated by (i) an MSP430F21x2 using the datasheet information (8.8 mW @ 16 MHz in active mode; cf. Table 1) which includes memory, peripherals and (ii) an open-source MSP430 processor core (0.96 mW @ 16 MHz), without program memory and peripherals.

We expect the actual power dissipation of the MSP430 core and its program memory to lie somewhere between the two results, and did a comparison to both of them.

The results are given in Table 3 where columns 2 through 6 show the instruction and cycle count, time taken, power and energy consumption for the two MSP430 MCUs (for the corresponding software implementation). On the other hand, columns 7 through 13 show the power and energy benefits for 8-bit micro-tasks. Similarly, columns 14 through 20 summarize the results for 16-bit micro-tasks. It can be observed that, for micro-tasks of different benchmark applications and OS tasks, power and energy benefits between one to two orders of magnitude can be obtained.

5.2.2 Optimum Bitwidth for Micro-tasks

We generated the micro-tasks for application codes having a variety of wordlengths operations. For example, the application *crc16* mostly uses 16-bit wordlength operations while operations in *tea-decipher* and *tea-encipher* mostly involve 32-bit wordlength data. The rest of the applications under-test use 8-bit wordlength operations.

As expected, for application codes, having wordlengths greater than 8-bit, an 8-bit micro-task has twice the number of FSM states than a 16-bit micro-task. However, interestingly the FSM of a micro-task consumes much lesser power than the datapath and power consumption of even very large FSMs increases in a sub-linear fashion with the number of states (Fig. 5).

As a result, an 8-bit micro-task consumes nearly half the power and silicon area than a 16-bit micro-task, Fig. 6(a) and (b). As far as the energy consumption is concerned, for codes having wordlengths greater than 8-bit, total energy consumption of an 8-bit and 16-bit micro-task is nearly the same. On the other hand, for application codes having 8-bit wordlength, an 8-bit micro-task consumes half of that of a 16-bit micro-task, Fig. 6(c).

Hence, since the datapath's power dominates the FSM's power in our examples, an 8-bit micro-task is a better solution. Nevertheless, for cases where FSMs could be comparatively much larger and consume more power than the datapath, micro-tasks having larger bitwidth would become more suitable.

6. CONCLUSION

In this paper, we have proposed, a novel approach for

Table 3: Power/energy gains for micro-tasks of various benchmark applications and OS tasks (@ 16 MHz).

Task Name	MSP430					8-bit Micro-task							16-bit Micro-task						
	Instr. Count	Clk Cycles	time (μs)	Power (mW)	Energy (nJ)	No. States	time (μs)	Power (μW)	Energy (pJ)	P. Gain (x)	E. Gain (x)	Area (μm^2)	No. States	time (μs)	Power (μW)	Energy (pJ)	P. Gain (x)	E. Gain (x)	Area (μm^2)
crc8	30	81	5.1	8.8/0.96	45/4.9	71	4.4	56.7	249	180/16.9	143/19.6	8864	71	4.4	106.7	469	82.4/9	95/10.5	16049
crc16	27	77	4.8	8.8/0.96	42.2/4.6	103	6.4	57.4	369	153/16.7	114/12.5	9523	73	4.56	106.4	485	82.7/9	91/9.5	16135
tea-decipher	152	441	27.5	8.8/0.96	242/26.4	586	36.6	84.5	3090	104/11.4	78/8.5	19950	308	19.2	152.8	2940	57.6/6.3	82/9	27236
tea-encipher	149	433	27.0	8.8/0.96	238/26	580	36.2	87.3	3160	101/11	75/8.2	20248	306	19.1	152.3	2910	57.8/6.3	81/8.9	27069
fir	58	175	10.9	8.8/0.96	96/10.4	165	10.3	78.2	806	112/12.3	119/12.9	14548	168	10.5	148.7	1560	59/6.5	61/6.7	24975
calcNeigh	110	324	20.2	8.8/0.96	178/19.4	269	16.8	81.4	1370	108/11.8	130/14.2	16873	269	16.8	151	2540	58/6.4	70/7.6	27320
snd2SPI	132	506	31.6	8.8/0.96	278/30.3	672	42	89	3730	99/10.8	74.5/8.1	23351	672	42	159.5	6700	55/6	41/4.5	33778
rcvFromSPI	66	255	15.9	8.8/0.96	140/15.2	332	20.7	81.5	1690	108/11.8	82.8/9	17487	332	20.7	151	3130	58.2/6.4	44.6/4.8	27934

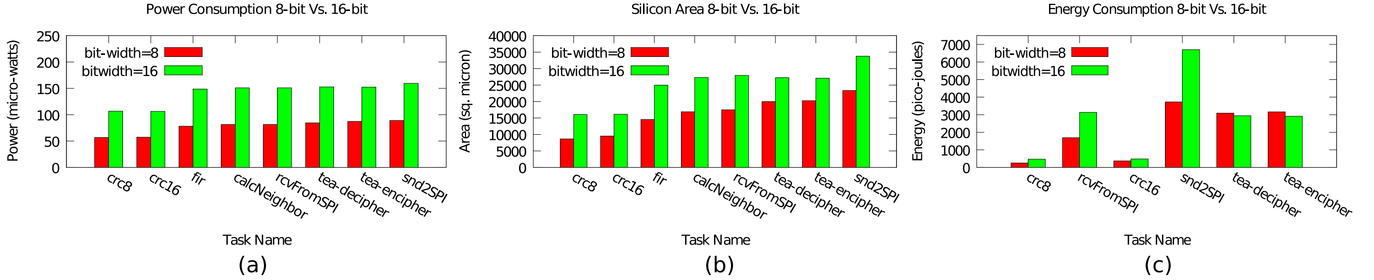

Figure 6: Comparison of power, area and energy consumption for 8- and 16-bit micro-tasks.

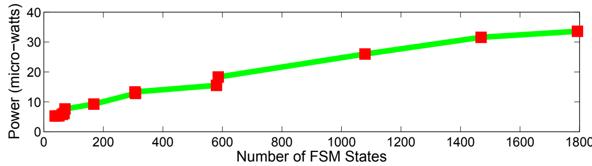

Figure 5: Power consumption vs. number of states of a micro-task FSM.

ultra low-power implementation of control-oriented application tasks of a WSN node. Our approach is based on power-gated micro-tasks that are implemented as specialized hardware blocks. We presented the details of our proposed software design-flow that is used to generate the micro-tasks from a high-level description in ANSI-C.

We presented the design space exploration results for different datapaths optimized for the corresponding applications at hand. We also presented the power dissipation results found for different micro-tasks generated from WSN application benchmarks and OS-specific control-tasks. The synthesis results show that, compared with the MSP430 micro-controller and under a conservative assumption, power reductions by one to two orders of magnitude are possible.

In future, we would also like to explore the feasibility of adding a small power-efficient reconfigurable core to the proposed node in order to provide reprogrammability. We would also like to further customize the micro-tasks for better power and area utilization. For example, finding patterns in FSM and developing more specialized operators in datapath could be interesting for an overall area and power reduction.

7. REFERENCES

[1] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. Wireless Sensor Networks:A Survey. *Computer Networks*, 38(4), March 2002.

[2] Atmel Corporation. ATmega 103L 8-bit AVR Low-Power Microcontroller. Tech. Report, 2007.

[3] Atmel Corporation. ATmega 128L 8-bit AVR Low-Power MCU. Tech. Report, 2009.

[4] B. Gorjiara and D. Gajski. Automatic Architecture Refinement Techniques for Customizing Processing Elements. In *DAC'08*.

[5] Crossbow Technology. Mica2 motes, http://www.xbow.com/.

[6] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. *in LCN'04*, Nov. 2004.

[7] P. Dutta, M. Grimmer, A. Arora, S. Bibyk, and D. Culler. Design of a Wireless Sensor Network Platform for Detecting Rare, Random, and Ephemeral Events. In *Proceedings of IPSN '05*, NJ, USA, 2005.

[8] A. Fin, F. Fummi, and G. Perbellini. Soft-Cores Generation by Instruction Set Analysis. In *Proceedings of ISSS '01*, 2001.

[9] R. Fonseca, P. Dutta, P. Levis, and I. Stoica. Quanto: Tracking Energy in Networked Embedded Systems. In *OSDI'08*, 2008.

[10] C. W. Fraser, R. R. Henry, and T. A. Proebsting. BURG: Fast Optimal Instruction Selection and Tree Parsing. *SIGPLAN Not.*, 27(4), 1992.

[11] B. Gorjiara, M. Reshadi, and D. Gajski. Generic Architecture Description for Retargetable Compilation and Synthesis of Application-Specific Pipelined IPs. In *ICCD'06*, Oct. 2006.

[12] S. Gupta, N. Dutt, R. Gupta, and A. Nicolau. SPARK: A High-Level Synthesis Framework for Applying Parallelizing Compiler Transformations. In *Proceedings of VLSI'03*, 2003.

[13] INRIA, Tech. Project. PowWow, Protocol for Low Power Wireless Sensor Network, http://powwow.gforge.inria.fr/.

[14] J. Cong, G. Han, and W. Jiang. Synthesis of an Application Specific Soft Multiprocessor System. In *Proceedings of FPGA'07*, February 2007.

[15] S. Liao, S. Devadas, K. Keutzer, and S. Tjiang. Instruction Selection using Binate Covering for Code Size Optimization. In *Proceedings of ICCAD'95*, Nov 1995.

[16] L.L'Hours. Generating Efficient Custom FPGA Soft-Cores for Control-Dominated Applications. In *Proceedings of ASAP '05*, Washington, DC, USA, 2005.

[17] S. Mysore, B. Agrawal, F. Chong, and T. Sherwood. Exploring the Processor and ISA Design for Wireless Sensor Network Applications. In *Proceedings of VLSI'08*, Jan. 2008.

[18] L. Nazhandali, M. Minuth, and T. Austin. SenseBench: Toward an Accurate Evaluation of Sensor Network Processors. In *Proceedings of IISWC'05*, Oct. 2005.

[19] P. Levis, S. Madden, J. Polastre, R. Szewczyk, K. Whitehouse, A. Woo, D. Gay, J. Hill, M. Welsh, E. Brewer, D. Culler. *TinyOS: An Operating System for Sensor Networks* . Book Chapter in Ambient Intelligence by Springer, 2005.

[20] V. Raghunathan, C. Schurgers, S. Park, and M. Srivastava. Energy-aware Wireless Microsensor Networks. *IEEE Signal Processing Magazine*, 19(2), Mar 2002.

[21] S. Roundy, P. Wright, and J. Rabaey. *Energy Scavenging for Wireless Sensor Networks: with Special Focus on Vibrations.* Springer, 2004.

[22] Texas Instruments. MSP430 User Guide. Tech. Report, 2009.

[23] University of California, Berkeley. Tech. project: Smart dust.