Toward Ultra Low-Power Hardware Specialization of a Wireless Sensor Network Node

Abstract— **Research in micro-electro-mechanical systems (MEMS) technology, wireless communications, and digital electronics has enabled the future emergence of Wireless Sensor Networks (WSN). These systems consist of low-cost and low-power sensor nodes that communicate efficiently over short distances. It has been shown that power consumption is the biggest design constraint for such systems. WSN nodes are being designed using low-power micro-controllers such as the MSP430. However, their power dissipation is still orders of magnitude too high. In this paper, we propose an approach to hardware specialization that uses the power-gated distributed hardware tasks. We target the control-oriented tasks running on WSN nodes and present, as a case study, a temperature monitoring application. Our approach is validated experimentally and shows prominent power gains over software implementation on a low-power micro-controllers such as the MSP430.**

I. Introduction

Recent advancements in micro-electro-mechanical-systems (MEMS) technology, wireless communications, and digital electronics have facilitated the development of low-cost, low-power, multi-functional sensor nodes that are small in size and communicate efficiently over short distances. Systems of 1000s or even 10,000s of such nodes are anticipated and can revolutionize the way we live and work. A Wireless Sensor Network (WSN) is composed of a large number of sensor nodes, that are densely deployed either inside a region of interest or very close to it. Each node consists of processing capability (one or more micro-controllers and/or DSP chips) and may contain different types of memory (RAM, ROM or flash). A node also has an RF transceiver and a power source (e.g. batteries or solar cells), and can accommodate various sensors and actuators.

Power consumption has been realized as the biggest constraint in the design of a WSN node. It is not possible to attach a huge source of energy with the WSN nodes due to the fact that the nodes must be low-cost and relatively smaller in size as compared to the traditional sensor nodes [1]. To make the situation worse, WSN nodes may have to work unattended for long durations due to difficult access to them or a huge number of nodes. As a result, they must survive with self-harvested or non-replenishing sources of energy. All these restrictions toward the energy retrieval make the power consumption the most important design parameter.

In the recent years, WSN nodes have been designed using low-power micro-controllers such as the MSP430 [2] from Texas Instrument or CoolRISC [3] from EM Microelectronic. These programmable processors share common characteristics such as a reasonable processing power with low power consumption at a very low cost. However, the power dissipation of these devices is still orders of magnitude too high for application domains such as WSN, since these systems expect sensor nodes to operate with extremely limited energy resources for very long time periods (months if not years). Worse, because these nodes remain idle during most of their lifetime, their static power consumption plays a major role in their actual energy budget.

In such situations, the only way to further improve the energy efficiency of such a system is to customize its design to the application at hand. An approach to specialization has been proposed recently which consists in implementing each task of a control-oriented application graph on a specialized hardware architecture [4]. This architecture is in the form of a minimalistic datapath controlled by a custom FSM and is being automatically generated from a task specification in C, by using an ASIP-like retargeted design environment.

In this paper, we are investigating the application of this hardware specialization approach from WSN perspective. We propose, as a case study, a temperature-monitoring WSN in which the monitoring nodes sense the temperature of different regions of interest and convey the results toward the base-station where this information is processed to make sure that the environmental conditions are normal in the regions. Such type of WSN can be used for wild fire monitoring system.

The main contribution of this article lies in further investigating the power benefits of the *power-gated hardware tasks* based specialization approach. A simple yet realistic case study of a WSN example also serves as an experimental validation that the approach is conceivable for real-life WSN applications.

Our experiments show that dynamic power savings of two orders of magnitude can be obtained for different control-oriented tasks of our application (w.r.t. a low-power MCU such as the MSP430). Moreover, since the tasks are power-gated, their static power consumption will be virtually zero when the WSN nodes will be in sleep mode.

This rest of this paper is organized as follows. We start by presenting the related works in Section II and describe thoroughly our proposed case study in Section III. In Section IV, we present experimental results which confirm the validity of the approach. Finally, conclusion and future research directions are drawn in Section V.

II. Related Works

In the last decade, a wide range of applications for sensor network have been developed. Some of the application areas are environment, military, health, and security. WSN may

consist of many different types of sensors such as seismic, low sampling rate magnetic, thermal, visual, infrared, acoustic and radar. These sensors are able to monitor a wide variety of ambient conditions such as temperature, humidity, lightning, pressure, and vehicular movements etc [5]. This section details the literature study of some of such WSN applications. Later in the section, we highlight some application benchmarks that have been proposed for WSN. Finally, we present some power optimization efforts done at micro-architectural and operating system level in context of the WSN.

A. Important WSN applications

Environmental monitoring is an important application of WSN. A lot of research work has been done on the different environmental aspects. In reference [6], a habitat monitoring system is discussed which includes the habitats of birds, animals and insects. Similarly, forest fire detection and prevention [7], strength monitoring of the civil infrastructures [8], and detection of volcanic eruptions [9] are some other examples of environment-monitoring WSN systems.

WSN can also be used as an integral part of military command, control, communication, computing, intelligence, surveillance, reconnaissance and targeting (*C4ISRT*) systems [10]. The rapid deployment, self-organization and fault tolerance are some characteristics that make WSN a very promising sensing technique for military *C4ISRT* systems. Similarly, VigilNet is also a good example of an integrated wireless sensor node for military surveillance application [11]. VigilNet acquires and verifies information about enemy capabilities and positions of hostile targets.

In addition, the benefits of WSN have also been proved in other domains of human life such as health and home applications ([12], [13]).

B. WSN application benchmarks

We have seen in this section that WSN applications consist of a heterogeneous nature as they are pretty different in their overall goals to be achieved. However, the basic micro-tasks performed in a WSN node are quite similar. These tasks are: sensing a certain phenomenon, gathering its relevant data and forwarding it to a base-station in a pre/post-processed state. Several attempts have been conceived to profile the workload of a generic WSN node. Two of the recent application benchmarks for WSN are SenseBench [14] and WiSeNBench [15]. Both of them have tried very well to cover the general applications and algorithms that can be run on a typical WSN node.

C. Low-power MCUs and operating systems

As far as power optimization of WSN domain is concerned, many research efforts have been made. These works cover all the design aspects of a WSN from application layer of the communication stack to the physical layer (e.g. efficient routing algorithms, low-power medium access control (MAC) protocols etc.). However, since the focus of our research work is the micro-architectural level, we try to summarize the characteristics of low-power micro-controllers (such as the MSP430 and the CoolRisc) that have been developed for low-power applications and the light-weight operating systems running on them. The common characteristics of such MCUs are: a simple datapath (8/16-bit wide), a reduced number of instructions (only 27 instructions for the MSP430), and several power saving modes that allow the system to select the best compromise between power saving and reactivity (i.e. wake-up time).

Most of such MCU packages comprise a limited amount of RAM (only a few hundred to a few kilo bytes) and non-volatile flash memory. This limited amount of storage resources poses great challenges to the software designers since both the user application and operating system must work with this very small amount of memory.

As a consequence, there have been several attempts to reduce the complexity of the operating system (OS) on these devices. In particular, many approaches have been proposed to reduce the overhead caused by dynamic scheduling of the threads by using alternative concurrency computational models. For example, the TinyOS [16] is built upon an event-driven approach, without explicit thread management , and Contiki [17] proposes a simplified thread execution model (named *protothread*), in which preemption can only occur at specific points in the task control flow.

III. Proposed Case Study

In this work, we target control-oriented WSN application tasks that can be represented as a control task graph. In this type of graph, a task execution is generally triggered either by another task or by a combination of external events. We restrict ourselves to such multitasking system in which preemption can only occur at certain specific steps of the program, as in the case of *protothread* construct available in the Contiki's ultra light-weight OS.

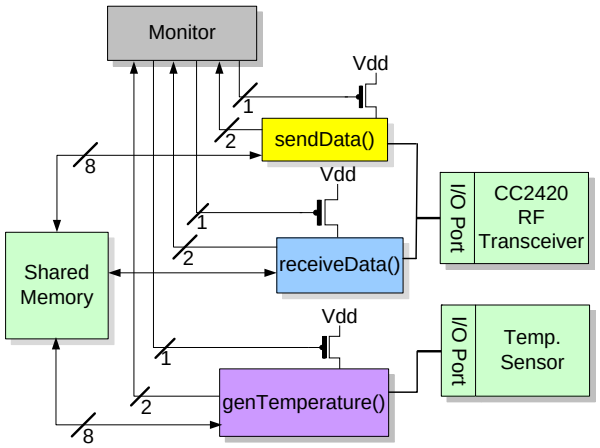

Fig. 1. System-level application mapping of a generic WSN node

This section details the important tasks of our temperature monitoring WSN system. We highlight the important algorithms that are run on the temperature-sensing, intermediate

and the base-station nodes. These nodes will be denoted as **source**, **relay** and **sink** nodes respectively in future discussion. Fig. 1 shows the system-level block diagram of a generic WSN node in our proposed system.

In reference [4], the author has also mentioned the notion of a system-level controlling FSM, called *Monitor*. *Monitor* is responsible for the activation and deactivation of the individual hardware tasks. For the sake of completeness, we will also outline the control signal interaction between the individual hardware tasks and *Monitor*. But we will not go into the details of its working as it is out of scope for this paper.

A. Application tasks running on a source node

Here are the basic control tasks running on a working WSN source node of our case study:

- Temperature sensing: The principle function of a source node in this WSN example is to monitor the temperature of a particular region of interest. So, in order to simulate this sensing task, we have written a C function, called `genTemperature()`, that randomly generates a temperature value.
- Neighbor calculation: After sensing, the next task is to convey the monitored temperature value to the base-station. Since the transmission (TX) power increases exponentially with the distance between source and destination nodes, a multi-hop strategy increases over all power efficiency of the system by reducing the long range communications. We use a simplified version of geographical routing used by another WSN system, PowWow [18]. This protocol is chosen because it is very simple and does not need extra-communications to route a message. The algorithm is implemented in `calcNeighbor()` function, that calculates the nearest neighbor in the direction of the base-station by calculating its linear distance from the source node.
- Waking up the neighbor: After selecting the neighbor, the source node sends it a wake-up beacon using `sendBeacon()`. After receiving the acknowledgment from the neighbor in due time, we proceed to the next task. In case of a time-out, we repeat the procedure of sending beacon.
- Sending data to the neighbor: The next task is data transmission that is described in `sendData()` function. This function basically calls its sub-functions that are responsible for sending data to the physical interface of the radio transceiver. These sub-functions are `controllerWrite()` and `sendData2SPI()` that basically write data to the serial peripheral interface (SPI) bus of the radio transceiver. In our example, we are using CC2420 radio chip from Texas Instrument [19] as RF transceiver.
- Shutting down the source node: After successful data transmission, the source node will go to sleep mode until the next temperature sensing and forwarding task is scheduled.

Fig. 2 shows the task flow graph of the application running on a source node.

B. Application tasks running on a sink node

The basic control tasks running on a WSN sink node of our case study are as follows:

- Receiving a wake-up beacon: The sink node will be in sleep mode until it receives a wake-up beacon from a source or relay node. This application task is described in `receiveBeacon()` function.
- Analyzing the beacon: After receiving the beacon, the sink node analyzes that whether the beacon is destined for it or not (done by `analyzeBeacon()` function). In former case, it moves toward the next task to be performed while in latter case, it again goes to idle mode by calling `shutDown()` function.
- Sending acknowledgment: If the beacon is destined for the sink node, it generates an acknowledgment for the originating (source or relay) node by calling `sendAck()` function. Then it remains ready for the data reception from the originating node.
- Receiving data: The next task is data reception that is described in `receiveFrame()` function and calls its sub-functions `controllerRead()` and `receiveFrameFromSPI()`. These functions are used to read data from the SPI bus of the radio transceiver.
- Analyzing the data: After successful data reception, the sink node will analyze the temperature data sent to it by calling the `analyzeData()` function. The data processing can be of different nature depending on the application at hand. In our case, a temperature value greater than 50 degree Celsius will be registered as an alarming event in the base-station database.
- Shutting down the sink node: After data analysis, the sink node will go to sleep mode until the next wake-up beacon is received from a source or relay node.

Fig. 3 shows the task flow graph of the application running on a sink node.

C. Application tasks running on a relay node

The task flow graph of a relay node is a hybrid of both the generic source and sink nodes. A relay node has to perform as a source node while interacting with base-station (sink) node. Whereas, it will perform certain functions of a sink node when it interacts with a temperature-sensing (source) node. Fig. 4 shows the task flow graph of a relay node application.

D. Control interface between Monitor and hardware tasks

Fig. 5, shows the control signals interchanged between *Monitor* and the hardware tasks. The unidirectional control line from *Monitor* to each hardware task consists of 1-bit which is the **start** signal for the hardware task. Upon reception of a valid value on this line, the hardware task is activated performing the micro-coded task at hand. Similarly, unidirectional control line from a hardware task to *Monitor* consists of 2-bits. One bit is used to signal out the termination

of the hardware task; whereas the other bit it used to signal out an event declaring which hardware task is to be activated next. Only, single bit is sufficient for this purpose since there are at most two potential application tasks to be activated at the termination of a task presented in our case study application graph (fig. 2, 3 and 4).

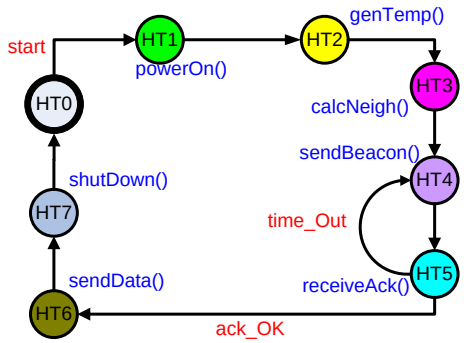

Fig. 2. System-level task flow graph of a source node

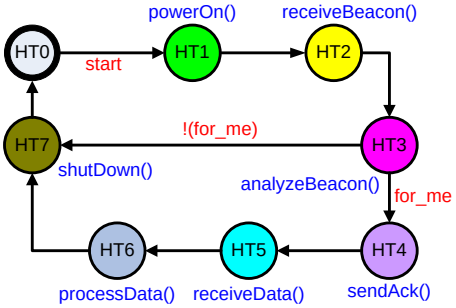

Fig. 3. System-level task flow graph of a base-station node

IV. Experimental results

Our compilation flow (fig. 6) is based on the GECOS compiler infrastructure [20], a retargetable C compiler framework, from which we use the BURG instruction selector that is retargeted to our simplified datapath model. This low-level program representation is then used to generate VHDL descriptions of (i) a custom datapath which implements the minimum required set of operations for the task at hand, and (ii) an FSM that will control the execution units of our datapath.

The hardware task VHDL designs have been synthesized for 130 nm CMOS technology using Synopsys's *Design Compiler*. We used these synthesis results to extract gate-level static and dynamic power estimations assuming a 100 MHz clock frequency. For the sake of comparison, with a software implementation, we used as baseline the MSP430F21x1 dissipation of 44 mW normalized at 100 MHz (the data sheet indicates a dissipation of 440 μW at 1 MHz).

The results are given in table I where it can be observed that, for the hardware tasks of a temperature monitoring example, power benefits of two orders of magnitude can be gained.

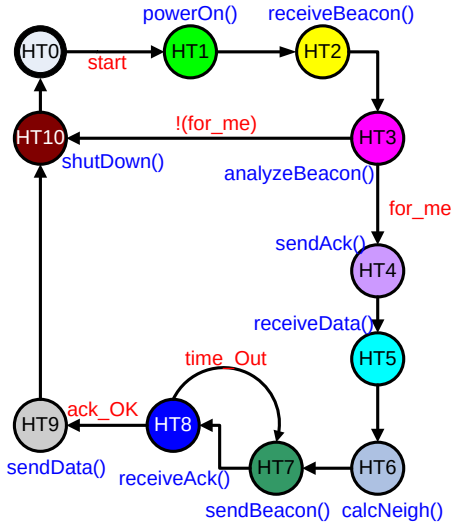

Fig. 4. System-level task flow graph of a relay node

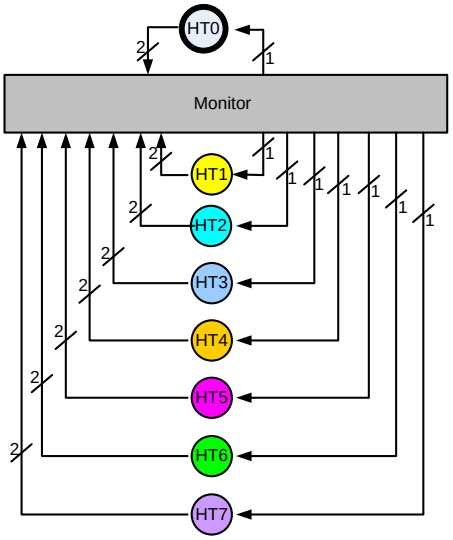

Fig. 5. Control signal interface between *Monitor* and hardware tasks of a source node

The last column of the table I summarizes the surface areas consumed by the hardware tasks. We have also synthesized an MSP430-like MCU core and the early experiments give us a surface area of around 55,000 μm^2. So, several hardware tasks can be placed in the same area as is consumed by an MSP430-like MCU.

We have also managed to synthesize the early versions of *Monitor* for the **source**, **sink** and **relay** nodes that consume 17.2 μW, 17.9 μW and 20.04 μW respectively. Since *Monitor* is the only active part of our WSN node in standby mode, we compare its power consumption with that of the MSP430F21x1 in standby mode that is 110 μW (the datasheet indicates a dissipation of 1.1 μW at 1 MHz). So, we gain at least a power saving of one order of magnitude as compared to the MSP430 in standby mode.

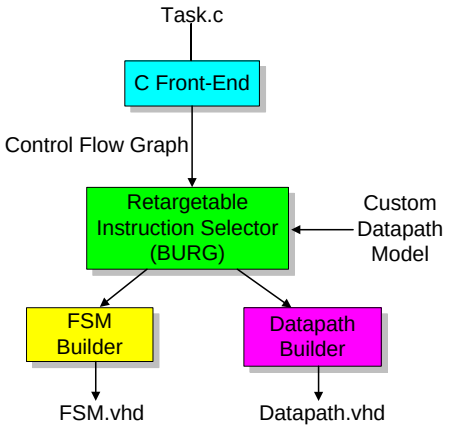

Fig. 6. Software design flow for hardware task generation

TABLE I

DYNAMIC POWER CONSUMPTION FOR VARIOUS CONTROL TASKS AND DATA MEMORY BLOCK.

Hardware Task				
Name	No. States in FSM	Power (μW)	Gain (x)	Area (μm^2)
sendData	43	209	210	4782
sendBeacon	43	209	210	4782
receiveFrame	27	211	209	4714
receiveFrameFromSPI	598	328.40	134	19034
genTemperature	69	208	210	4633
controllerWrite	83	230.77	190	6105
controllerRead	175	254	173.2	8700
calcNeighbor	376	264.26	166.5	9565

Data memory		
Size (Bytes)	Power (μW)	Area (mm^2)
512	480	0.040

For the sake of completeness, second part of table I shows the size and power estimation obtained for a memory block used to store the global and local variables involved in our case study. The memory has been synthesized for 130 nm CMOS technology by using Faraday's *Memory Compiler*.

V. CONCLUSION

In this paper, we have proposed an original approach for the ultra low-power implementation of control-oriented application tasks of a WSN application. Our approach is based on *power-gated hardware tasks* that are implemented as specialized hardware blocks. We presented as a case study a WSN system implementing a temperature monitoring application.

The synthesis results for the hardware tasks of the case study application graph show that, compared with the MSP430 micro-controller and under a very conservative assumption, power reductions by two orders of magnitude are possible.

We envision two directions for our future work. We first aim at studying and developing a system-level model a WSN node based on hardware tasks. We would also like to evaluate the feasibility of our approach on control-oriented reconfigurable structures, which would provide support for small grain power-gating techniques [21].

REFERENCES

[1] U. Berkeley, "Smart dust." [Online]. Available: http://robotics.eecs.berkeley.edu/ pister/SmartDust/

[2] T. Instruments, "MSP430 User's Guide," Texas Instruments, Tech. Rep., 2006.

[3] E. Electronic, "EM6812, Ultra Low Power 8-bit FLASH Mircro-Controller," EM Electronic, Tech. Rep., 2005.

[4] "Reference omitted for the blind review process."

[5] D. Estrin, R. Govindan, J. Heidemann, and S. Kumar, "Next Century Challenges: Scalable Coordination in Sensor Networks," in *MobiCom '99: Proceedings of the 5th annual ACM/IEEE international conference on Mobile computing and networking*. New York, NY, USA: ACM, 1999, pp. 263–270.

[6] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless Sensor Networks for Habitat Monitoring," in *WSNA '02: Proceedings of the 1st ACM international workshop on Wireless sensor networks and applications*. New York, NY, USA: ACM, 2002, pp. 88–97.

[7] Y. Li, Z. Wang, and Y. Song, "Wireless Sensor Network Design for Wildfire Monitoring," *The Sixth World Congress on Intelligent Control and Automation, WCICA 2006.*, vol. 1, pp. 109–113, 2006.

[8] S. Kim, S. Pakzad, D. Culler, J. Demmel, G. Fenves, S. Glaser, and M. Turon, "Health Monitoring of Civil Infrastructures using Wireless Sensor Networks," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*. New York, NY, USA: ACM, 2007, pp. 254–263.

[9] G. Werner-Allen, J. Johnson, M. Ruiz, J. Lees, and M. Welsh, "Monitoring Volcanic Eruptions with A Wireless Sensor Network," *Proceeedings of the Second European Workshop on Wireless Sensor Networks, 2005.*, pp. 108–120, Jan.-2 Feb. 2005.

[10] I. A. W. S. Y. S. E. Cayirci, "Wireless Sensor Networks:A Survey," *Computer Networks*, vol. 38, no. 4, pp. 393–422, 15 March 2002.

[11] T. He, S. Krishnamurthy, L. Luo, T. Yan, L. Gu, R. Stoleru, G. Zhou, Q. Cao, P. Vicaire, J. A. Stankovic, T. F. Abdelzaher, J. Hui, and B. Krogh, "VigilNet: An Integrated Sensor Network System for Energy-Efficient Surveillance," *ACM Trans. Sen. Netw.*, vol. 2, no. 1, pp. 1–38, 2006.

[12] D. Malan, T. Fulford-jones, M. Welsh, and S. Moulton, "CodeBlue: An Ad Hoc Sensor Network Infrastructure for Emergency Medical Care," in *In International Workshop on Wearable and Implantable Body Sensor Networks*, 2004.

[13] C. Herring and S. Kaplan, "Component-Based Software Systems for Smart Environments," *Personal Communications, IEEE [see also IEEE Wireless Communications]*, vol. 7, no. 5, pp. 60–61, Oct 2000.

[14] L. Nazhandali, M. Minuth, and T. Austin, "Sensebench: toward an accurate evaluation of sensor network processors," Oct. 2005, pp. 197–203.

[15] S. Mysore, B. Agrawal, F. Chong, and T. Sherwood, "Exploring the processor and isa design for wireless sensor network applications," in *VLSI Design, 2008. VLSID 2008. 21st International Conference on*, Jan. 2008, pp. 59–64.

[16] P. Levis, M. S., P. J., S. R., and W. K., "TinyOS: An Operating System for Sensor Networks," in *Ambient Intelligence*, 2005, pp. 115–148.

[17] A. Dunkels, B. Gronvall, and T. Voigt, "Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors," *29th Annual IEEE International Conference on Local Computer Networks, 2004.*, pp. 455–462, Nov. 2004.

[18] I. CAIRN, "Powwow, protocol for low power wireless sensor network." [Online]. Available: http://powwow.gforge.inria.fr/

[19] T. Instruments, "Single-Chip 2.4 GHz IEEE 802.15.4 Compliant and ZigBee Ready RF Transceiver." [Online]. Available: http://focus.ti.com/docs/prod/folders/print/cc2420.html

[20] L. L'Hours, "Generating Efficient Custom FPGA Soft-Cores for Control-Dominated Applications," in *ASAP '05: Proceedings of the 2005 IEEE International Conference on Application-Specific Systems, Architecture Processors*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 127–133.

[21] A. Rahman, S. Das, T. Tuan, and S. Trimberger, "Determination of Power Gating Granularity for FPGA Fabric," *Conference 2006, IEEE Custom Integrated Circuits*, pp. 9–12, Sept. 2006.