System Level Synthesis for Ultra Low-Power Wireless Sensor Nodes

Muhammad Adeel Pasha, Steven Derrien
IRISA/INRIA, Université de Rennes 1
Campus de Beaulieu
35000 Rennes
steven.derrien|adeel.pasha@irisa.fr

Olivier Sentieys
IRISA/INRIA, ENSSAT Lannion
22 rue de Kerampont
22300 Lannion
olivier.sentieys@irisa.fr

Abstract

Engineering hardware platform for a Wireless Sensor Network (WSN) node is known to be a tough challenge, as the design must enforce many severe constraints, among which energy dissipation is by far the most challenging one. Today, most of the WSN node platforms are based on low-cost and low-power programmable microcontrollers, even if it is acknowledged that their energy efficiency remains limited and hinders the wide-spreading of WSN to new applications. In this paper, we propose a complete system level flow for an alternative approach based on the concept of hardware micro-tasks, which relies on hardware specialization and power gating to dramatically improve the energy efficiency of the computational part of the node. Early estimates show power saving by more than one order of magnitude over MCU-based implementations.

1 Introduction

Wireless Sensor Network (WSN) is a very promising technology with a large number of potential applications, ranging from environmental monitoring to medicine. WSNs are made of a large number of sensor nodes, each of them consisting of a processing component (usually a microcontroller) with wireless communication capabilities (RF transceiver) and various sensors/actuators. It is widely acknowledged that energy consumption is one of the major constraints in WSN infrastructure. As nodes must be low-cost with reduced scale factors [18], they can not benefit from significant power-supply sources. Besides, they often have to work unattended for long durations and therefore must survive with either self-harvested (e.g. solar cells) or non-replenishing (e.g. battery) sources of energy.

As of today, most WSN node hardware platforms are based on low-power microcontroller units (MCU) such as the MSP430 [17] and the ATmega128 [1] which provide a low-power processing solution at an affordable cost. For the task management, they usually rely on ultra light-weight operating systems (OS) [12, 4, 2] which reduce OS complexity to its minimum, while providing user-level abstractions such as threads, event management, etc. (see Section 2). Even though, the energy dissipated by such hardware platforms still remains way too high for many potential applications of WSN and, in that context, the only way to further improve the computational energy efficiency is to use aggressive hardware specialization combined with low-power design.

We, therefore, propose to replace the MCU by a set of application specific *hardware micro-tasks* [13], each of them being tailored to efficiently implement the required functionality (e.g. event-sensing, MAC, data-routing, etc.). By combining this aggressive hardware specialization (to reduce dynamic power) with VLSI power optimization such as *power-gating* (to reduce static power), we can drastically reduce the energy dissipation of the system.

In this paper, we present *LoMiTa (ultra Low-power Micro-Tasking)*, a complete system-level design flow for designing application specific hardware platforms. Following the philosophy of many WSN software frameworks, this flow uses a combination of a textual Domain Specific Language (DSL) for system-level specifications (interactions between tasks, event management, shared resources) and ANSI-C for specifying the behavior of each micro-task. From such a specification, it is then possible to generate a synthesizable VHDL description of the whole architecture, which provides a direct path to ASIC or FPGA implementation.

We want to clarify that the goal of this paper is not to propose a new model of execution/computation for WSN. We rather see our approach as a simple execution model chosen so as to be a good match for what we think is a promising architectural solution for WSN nodes. Indeed, our experiments show that dynamic power savings by more than one order of magnitude can be obtained (w.r.t. a low-power MCU such as the MSP430). Moreover, we show that the silicon area overhead caused by the duplication of hardware

tasks remains limited and does not hinder the viability of the approach.

The rest of this paper is organized as follows. We start by presenting the literature for energy-efficient WSN in Section 2 and describe thoroughly our proposed system execution model in Section 3. Section 4 provides an overview of our software design flow, and in Section 5, we present experimental results which confirm the validity of our approach. Finally, we conclude and draw future research directions in Section 6.

2 Energy-Efficient WSN

It is well known that the main sources of energy dissipation in a WSN node are communication and computation. It is also widely accepted that communication energy cost dominates in the overall budget, and that the focus should be on reducing this communication energy cost. As an example, the use of advanced digital communication techniques (efficient error correction, cooperative MIMO [11]) and network protocols (energy-efficient routing [16] and/or MAC schemes such as S-MAC [20], B-MAC [14] and RICER [7]) have shown to help in improving the energy efficiency for communication.

However, these techniques (e.g. LDPC error correcting codes) may significantly increase the computation workload on the processing component of the system, which in turn (i) impacts the overall energy budget of the system and (ii) may require processing horsepower that would be above the power/energy budget allowed to typical WSN node MCUs.

As a consequence, improving the *computational* energy efficiency of WSN nodes is an important issue, which we propose to address through the use of hardware specialization. Indeed, we believe that power and energy savings obtained through our approach open possibilities for more computationally demanding protocols or algorithms, which would in turn provide better quality-of-service (QoS), lower transmission energy and higher network efficiency.

In the following section, we will provide an overview of existing approaches to improve power and energy efficiency at VLSI (i.e. circuit) and architectural (i.e. processor) level. We will then introduce the concept of hardware *micro-task* on which we base our approach.

2.1 Low-Power VLSI Design

Power dissipation in VLSI devices can be divided into two categories: *dynamic power* caused by the transistor switching (i.e. stage changes) that occurs while the circuit is operating and *static power* caused by leakage current between power supply and ground. In the context of WSN, as the node remains inactive for long periods (MCU duty cycle lower than 1%), the contribution of static power becomes significant and can not be ignored.

There are many approaches to reduce dynamic power in a circuit, which can be applied at various levels of the design flow. However, many of them are poorly suited to WSN nodes as they often significantly increase the total silicon area, and therefore have a negative impact on static power dissipation.

One exception is *power gating*, which consists in turning-off the power supply of inactive circuit components. *Power gating* helps in reducing both dynamic and static power, and is thus very efficient for devices in which components remain idle for long periods.

The technique 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 Figure 1. This sleep transistor allows the supply voltage of the block to be cut off to dramatically reduce leakage currents.

The approach has already been used in the context of high-performance CPUs [6], and FSM (Finite State Machine) implementations [15] where parts of the design are switched on/off according to their activity. The approach helps in reducing the static power dissipation for FPUs of a high-end CPU by up to 28% at the price of a performance loss of 2%, for FSMs the average reported power reduction was also 28%. In the context of WSN, where nodes remain idle most of the time, such a technique has obvious advantages, and is therefore intensively used for implementing the low-power modes of typical node MCUs, which we describe in the following section.

2.2 Low-Power MCU Architectures

Microcontrollers used in WSN platforms (such as MSP430, ATmega128L) share many characteristics: a simple datapath (8/16-bit wide), a reduced number of instructions (only 27 instructions for the MSP430), and several low-power modes allowing the system to select at runtime the best compromise between power saving and reactivity (i.e. wake-up time). These processors are designed for low-power operation across a range of embedded system application settings but are not necessarily well-suited to the event-driven behavior of WSN nodes as they are based on a general purpose, monolithic compute engine.

For example, Mica2 mote [3] has been widely used by the research community. It is a complete WSN node based on a ATmega128L MCU, with I/O peripherals, an RF transceiver and sensor devices. Measurements of Mica2 show that its MCU consumes an average 8 mA of current when active and approximately 15 μA when in low-power mode. The MSP430F1611 used by Hydrowatch platform consumes a nominal 500 μA [5] whereas the latest version of MSP430 (MSP430F21x2) consumes approximately 8.8 mW@16 MHz.

These power consumptions may seem extremely

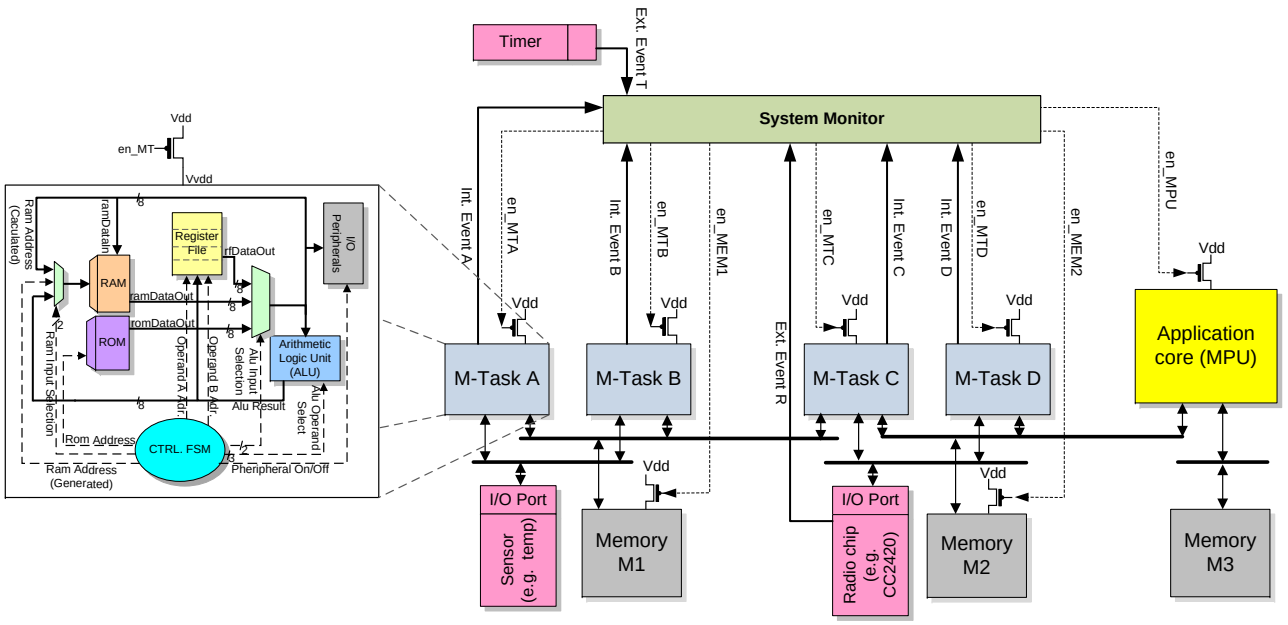

Figure 1. System-level view of a micro-task based WSN node architecture

small compared to that of typical embedded devices, however when looking at energy efficiency metrics such as *Joules per Instruction*, it appears that the proposed architectures (e.g. MSP430) still offer room for improvement. In particular, it is clear that the combination of specialization and parallelism would significantly help in improving energy efficiency. However, such architectural improvements usually come at the price of a significant increase in silicon area, which leads to unacceptable levels of static power for WSN.

In the following, we propose an approach that is able to improve computational energy efficiency, while maintaining the static power at acceptable levels.

2.3 Power-Gated Hardware Micro-Tasks

In the literature, the term *micro-task* has previously been used by Wallner [19] in a different context. In this work, we call a micro-task a specialized hardware structure which executes parts of the WSN node code. Contrary to an instruction-set processor, the program of a micro task is hardwired into an FSM that directly controls a semi-custom datapath. This makes the architecture much more compact (no need of an instruction decoder, nor instruction memory) and allows the size of storage devices (register file and RAM/ROM) as well as the arithmetic logic unit (ALU) functions to be customized to the target application.

Each of these micro-tasks can access a local and/or shared data memory, and can also access peripheral I/O ports (e.g. SPI link to an RF tranceiver).

Left part of Figure 1 shows the micro-architecture for such a micro-task (here with an 8-bit data-path), dotted lines represent control signals generated by the control FSM, whereas solid lines represent data-flow connections between datapath components. Such a drastically simplified architecture allows for obvious dynamic power savings compared to a standard MCU architecture, with a very limited loss in performance since the datapath is tailored to the application at hand. However the approach is only valid for fine-grain tasks, as the more functionalities are handled by the task, the less efficient its specialization will be.

To take advantage of this approach, we therefore propose to distribute the whole WSN node software framework among a set of hardware micro-tasks, so as to maintain a high degree of specialization within each task. For example, a complete WSN communication stack uses approximately 3500 instruction, by distributing the stack functionality onto 7 hardware tasks, we can reach a average task size of 500 instructions, which is a granularity level at which we can expect significant energy improvements. This of course comes at the price of an (limited) increase in area and more important, in static power dissipation.

To tackle this static power, we provide micro-tasks with a *power gating* mechanism, which allows to turn-off the micro-task power supply and drastically reduce the leakage power. We have shown in a previous work that the approach was very promising, and that computational energy efficiency improvements of 8x to 20x were possible compared to state-of-the-art MCUs, even for control-dominated code, where architectural optimizations are known to have lesser impact on power than in pure data-flow [13].

2.4 The Issue of Reprogrammability

Of course, our solution has a drawback: it assumes that the micro-tasks are hard-wired into silicon as ASIC (Application-Specific Integrated Circuit) blocks. This means that the behavior of each micro-task is fixed, making post-production upgrade or bug fixing very costly. This may look like a show stopper, as flexibility is often of a great concern for WSN system designers. However, when looking more carefully to actual design practices, we can observe that the need of flexibility and reprogrammability is essentially geared toward the user application layer, which happens to represent only an extremely small fraction of the WSN node processing workload. Whereas most of the processing workload is almost entirely dedicated to the communication stack.

Besides, in practice, designing a new WSN application usually means adapting a proven existing WSN software framework to a new user application. In other words, the communication stack software is generally reused "as is" and routing algorithms, MAC protocols, device driver layers remain the same (even if their behavior is parameterized). We, therefore, propose to combine the best of both worlds, that is:

- use a very small silicon footprint instruction-set processor (8-bit datapath, minimalistic RISC instruction set) with a power-gating feature to implement the application layer user software,

- use a distributed system of micro-tasks to handle the OS-level services of the WSN node (mostly the communication stack).

Such an approach preserves most of the power/energy savings provided by specialization, while preserving programmability at the user application level.

However, managing/specifying interactions between these tasks raises several other issues which we propose to address in the following section by proposing an associated system-level execution model (and its automated design flow).

3 An Execution Model for Micro-Tasks

Like most event-driven embedded system applications, WSN applications can easily be modeled as Task Flow Graphs (TFG), in which task execution is concurrent and may be triggered either by another task or by a combination of external events be they synchronous (timer) or asynchronous (wake-up radio).

In typical embedded systems, concurrency, event management and shared resource management are handled by a real-time embedded OS, which provides adequate constructs for the programmer (preemptive task scheduling, mutex, etc.). However, such full featured OSs can not be implemented in WSN nodes due to the strong constraints on memory footprint.

In this section, we start by outlining the characteristics of some of the most common OS infrastructures targeted at WSN, and propose an execution model for our proposed approach and detail its specificities and limitations w.r.t to the features of standard WSN OS implementations.

We then highlight the proposed design flow used to generate customized micro-tasks from an application description written in a combination of C (for describing each task behavior) and a custom textual DSL for specifying the interactions between tasks (used to generate a *System Monitor* (SM), see Section 3.2).

3.1 WSN Operating Systems

TinyOS [12] is one of the earliest and the most commonly used OS in WSN platforms. TinyOS provides a component-based event-driven concurrency model, which focuses on optimizing power usage. TinyOS executes tasks without preemption and follows a run to completion semantic. This forces the programmer to split each application functionality into many distinct tasks so as to accommodate the absence of blocking statements, and makes the programs more difficult to write and debug compared to standard *thread-like* constructs.

Contiki [4] is another WSN-specific OS proposed by Dunkels et al. It features an event-driven kernel where multi-threading is not present as an inherent feature but can be implemented as a library that is linked only with programs that explicitly require it. Contiki uses a notion of *protothreads*: a programming abstraction providing a conditional blocking statement to simplify the event-driven programming for memory constrained systems. In Contiki protothreads is stack-less i.e. local variable content is lost whenever the scheduler switches from one protothread to another.

MANTIS Operating System (MOS) [2] uses a traditional multi-threaded approach. It offers a time-sliced approach in which an interleaved concurrency of multi-threading is used to prevent one long-lived task from blocking execution of a second time-sensitive task. However, a larger memory resource is needed for thread-management as task preemption requires that the complete stack of a preempted thread is to be saved.

3.2 Proposed System Model

Figure 1 represents a system level view of a WSN node platform designed according to our proposed approach. Such a system consists of :

- A programmable instruction-set processor, that is used to implement the application level code.

- A set of power-gated hardware micro-tasks accessing shared resources (e.g. peripherals (RF, sensor) and memories).
- A hardware *System Monitor* (SM) that controls the execution of all the micro-tasks along with the application processor.
- Event triggering peripherals (such as wake-up radio, Timer) that can send events to SM.

3.2.1 Events and Commands

In our approach, we use *command* and *event* message structures between SM and micro-tasks similar to those of TinyOS. A *command* is a control signal initiated by SM toward a micro-task enabling the start of its operation. On the other hand, an *event* is a control signal input to SM.

Dotted lines in Figure 1, represent command signals driven by SM to the micro-tasks and shared memories. Solid lines represent events sent back to SM. Events can be of two types:

- An *internal event* which is generated by a micro-task indicating its termination or preemption (in case of a sub-routine call).
- An *external event* which is generated by an external peripheral that can serve as wake-up call from shut-down mode.

3.2.2 Micro-Task Activation/Deactivation

The SM, responsible for activating and/or powering off micro-tasks/memory blocks upon reception of an (a combination of) event(s), is itself implemented as a combinational FSM and a set of status and event registers. The status registers save the activation status of the micro-tasks and are controlled by the FSM whereas the event registers are used to save the incoming events until they are consumed. The FSM is, in turn, activated by periodic (timer event) and/or aperiodic external events (wake-up radio link, push button or sensor triggered events). Figure 2 shows the generic template of an SM.

In our execution model, we restrict ourselves to micro-tasks following a *run-to-completion* semantic, as in the case of TinyOS tasks. This ensures that a given micro-task will never reach a state in which it is active while not executing useful computation (i.e. blocked waiting for some event).

In addition, we make sure that at a given time instant there may not be two active tasks sharing an access to a same shared resource. This property is ensured (in a conservative way) by the SM which makes sure that, prior to activating a candidate task, there are no other active tasks that need access to a resource that *may* also be used by the candidate micro-task.

In the following paragraphs, we review the most noticeable specificities of our execution model. We also compare them whenever possible with those of existing software OS for WSN.

3.2.3 Concurrency Management

Both TinyOS and Contiki allow the programmer to express concurrency in their application through the use of specialized constructs such as *Protothreads* for Contiki, *Tasks* for TinyOS. One of the goals of such non-standard constructs is to help reducing context switching and task scheduling overhead.

In our approach, as we rely on several physically distinct hardware micro-tasks, we provide a natural support for concurrency and task level parallelism, and don't have to pay for any context switching overhead. Similarly, scheduling does not have any execution time overhead, even if we must take into account the extra silicon area required by SM.

Another advantage of the approach, lies in the fact that shared resources such as memory or I/O ports are much easier to handle than in a standard multiprocessor architecture, thanks to the *power gating* feature. Indeed, since no two tasks sharing an access to a same resource can be active (i.e. powered-on) at the same time, we can save the typical extra tri-state (or mux) logic used to share data/address bus lines, which results in savings in terms of power and energy.

3.2.4 Task Hierarchy and Granularity

As mentioned in Section 1 and illustrated in Section 5 the power savings obtained through specialization are sensitive to the task computation granularity, the size of which should remain in the order of a few hundred of assembly level instructions to enable significant savings.

A natural way to control the granularity of a task in an imperative language is through the use of sub-programs. In our execution model, we offer two ways for handling sub-routine calls made within a micro-task specification.

The first one is straightforward and consists in inlining the sub-routine calls, this increases the task granularity, and is acceptable for small sub-program calls. The second one is more complex and consists in generating a child micro-task dedicated to the sub-program execution. In latter case, the parent (i.e. caller) micro-task directly activates its child micro-task and as the child micro-task is also power gated, it only marginally contributes to the power budget, while helping in maintaining a high level of specialization within the parent task.

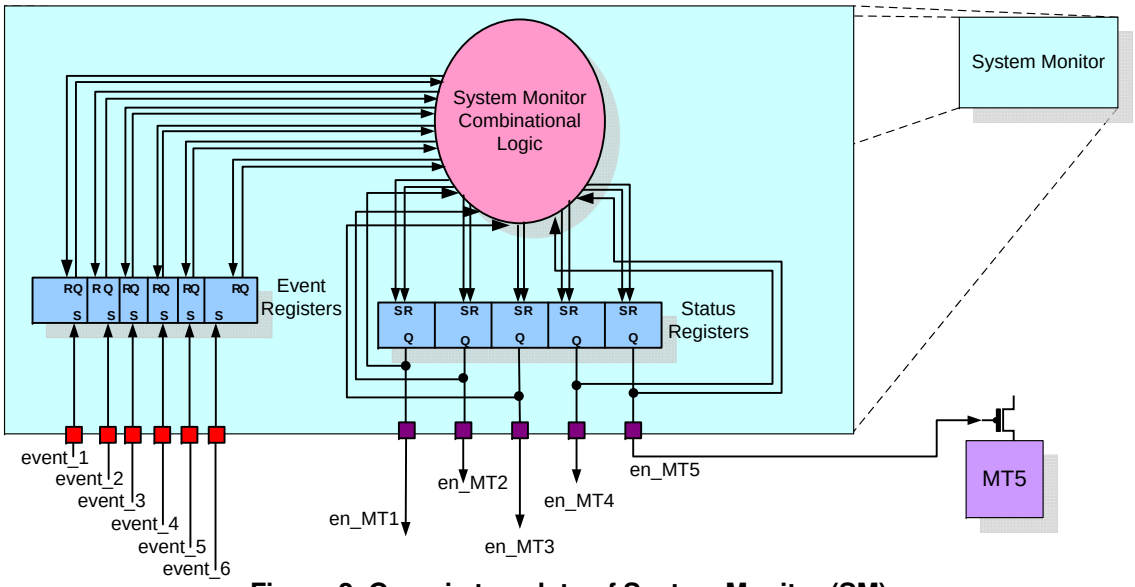

Figure 2. Generic template of System Monitor (SM)

4. Software Design Flow

Of course, even the soundest execution model is useless without a supporting solid software design flow, which allows the programmer to proceed directly from as specification written in a high level language to an executable specification, which in our case consists of an RTL description of the whole hardware system.

This section provides an overview of our software framework for designing micro-task based WSN platforms. It consists of two parts: (i) a customized ANSI-C to hardware compiler which is used to generate the HDL (Hardware Description Language) specification (namely VHDL) of a micro-task, given its behavior in ANSI-C (ii) a DSL, used to specify the system organization and the micro-tasks interactions, that is used to generate the VHDL code for the SM.

Our complete design flow is summarized in Figure 3: we start from the application task descriptions written in ANSI-C and a system-level description of task interactions in DSL to derive the behavior of SM and close the path to hardware generation.

4.1 Micro-task Hardware Synthesis

The generation of the hardware description of a micro-task is done automatically in our flow, starting from a software specification of the task behavior in ANSI-C. Here we will only outline its main characteristics (for details, see [13]).

Our flow builds on the GeCoS compiler infrastructure [8], a retargetable compiler framework for ASIP (Application Specific Instruction Set Processors). It extends typical features of a retargetable compiler by selecting a minimalistic processor datapath template on which all the micro-task instructions are executed. The datapath template is chosen so that to avoid performance degradation in terms of architectural efficiency (in other words the number of cycles required to execute a task with our approach is comparable to that of an MSP430).

Then, instead of producing machine-level code, our flow generates the VHDL descriptions of the micro-task FSM controller and customized datapath.

4.2 System Monitor Synthesis

As far as the system-level description of a WSN node is concerned, we provide a simple Eclipse-based DSL which is used to describe the components of a system: micro-tasks, events, shared memories and peripherals, etc. The textual DSL was developed using Xtext, an MDE (Model Driven Engineering) framework, and generates VHDL description for the SM. Fig. 4 shows a snapshot of a WSN-node system described using our DSL. This system communicates with a radio transceiver through SPI interface.

For each micro-task in the system, we specify its corresponding sub-program name, event configuration under which the task can be activated, read/write-access to shared resources and events produced by the task. Similarly, for each global variable of the application code we specify which memory block is used as storage component. Using all these informations, we derive the guard expression for the activation of each micro-task present in the system.

As an example, a micro-task T_N can only be activated when the following conditions are met:

- The internal and external event signals present in T_N's event configuration are true.

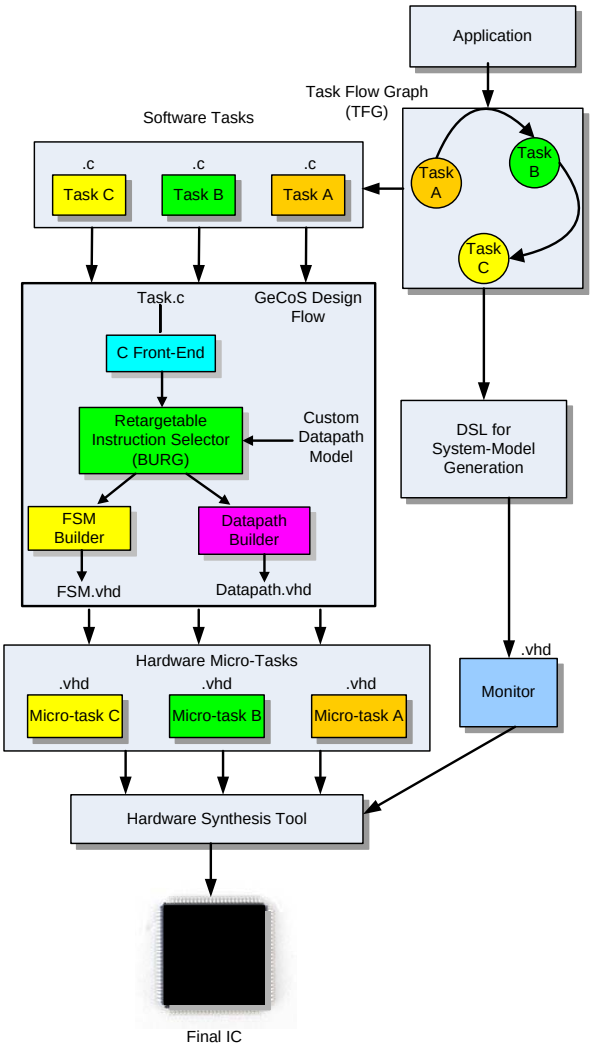

Figure 3. Complete system-level design-flow

- The event signals present in the event configuration of a task T_M are false where T_M is a task sharing a memory or I/O resource with T_N.

Using the above mentioned conditions, we derive the guard condition expression for T_N:

$$C_{G_N} = E_{A_N} \& \forall_{T_M} not(E_{A_M}) \& \forall_{T_I} not(E_{A_I})$$

where E_{A_N} is the event configuration for T_N activation, T_M is a micro-task sharing a memory resource with T_N and T_I is a micro-task sharing an I/O resource with T_N. The status registers holding the activation status of the micro-tasks are controlled by the guard expression derived above.

5 Experimental Validation

Several attempts have been proposed to profile the workload of a generic WSN node. Two of the recent application benchmarks for WSN are SenseBench [10]

```
system send_receive_data {
    include "send_receive.gecos"
    events { extET, En_ReceiveFrame, En_SendFrame, beacon_Sent, data_Received}
    memory memB [gated] {
        contains globals {U1TXBUF};
    };
    memory memC [gated] {
        contains globals {P5OUT};
    };
    io ioA {
        contains ports {port IFG 1, port URXIFG1 2};
    };
    io ioB {
        contains ports {port U1TXBUF 5, port P5OUT 6};
    };

    timer extern period = 100 produces extET;

    task sendFrame2SPI {
        activates With{En_SendFrame}
        produces { beacon_Sent }
        reads ioModule { ioA }
        writes ioModule {ioA }
        reads memory { memB}
        writes memory {memB}

    } ;
    task ReceiveFrameFromSPI {
        activates With { En_ReceiveFrame}
        produces { data_Received }
        reads ioModule { ioB }
        writes ioModule { ioB }
        reads memory { memC}
        writes memory { memC}
    } ;
}
```

Figure 4. A snapshot of the system DSL

and WiSeNBench [9]. We have used our software design flow to generate micro-tasks for several tasks extracted from these benchmarks. This section presents some experimental results obtained for power dissipation of these micro-tasks.

The micro-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 16 MHz clock frequency. These results were compared to the power dissipated by an MSP430F21x2 using (i) the datasheet information (8.8 mW @ 16 MHz in active mode) which includes memory, peripherals and (ii) statistical power estimation for an MSP430 clone ($0.96\,mW$ @ $16\,MHz$), without program memory.

We expect the actual power dissipation of the MSP430 core along with its program memory to lie somewhere in between the two results, and compare our results to both of them.

The results are given in Table 1 where column 2 and 3 show the number of MSP430 instructions and FSM states respectively. Similarly, column 4 and 5 show the power consumption of each micro-task and the power improvements over an MSP430 software implementation using (i) datasheet and (ii) statistical estimation respectively. It can be observed that, for micro-tasks of different benchmark applications, and in the less favorable case, power benefits of at least x20 can be obtained.

In addition, column 6 gives us the silicon area used

Table 1. Power consumption for micro-tasks of various benchmark applications.

Task Name	#MSP430 Instructions	#States in FSM	Power (μW)	Gain (×)	Area (%)
absolute	14	38	32.0	204/29.57	6.8 %
binSearch	43	103	34.0	194/26.88	7.8 %
bpHash	31	76	33.5	197/27.84	7.2 %
crc8	43	75	33.5	197/27.84	7.1 %
tea-Decipher	152	588	29.2	151/20.54	18 %
tea-Encipher	149	582	28.1	156/21.1	16.5 %
factorial	27	71	33.0	199/27.07	6.8 %
findMin	24	62	33.0	200.5/28.8	6.4 %
fir	58	173	34.5	186/26	6.5 %
sumArray	19	51	33.0	203.5/29.56	6.3 %

by each micro-task as a percentage of that of the open-source MSP430-core. Hence, we also show that the silicon area used by most of the micro-tasks is quite small, and is extremely competitive w.r.t to that of an MSP430. This suggest that a system made of 5-7 micro-tasks would have the same foot-print as an MSP430-core.

For the sake of completeness, we also synthesized the VHDL generated for the SM of a data-exchanging WSN node and it shows that it consumes only 12 μW (@ 16 MHz) and area overhead is only 2% of that of an MSP430-core.

6 Conclusion

In this paper, we have proposed an original approach for optimizing the energy efficiency of WSN node platform. Our approach is based on power-gated micro-tasks which are implemented as parallel specialized hardware blocks. The paper details a system-level execution model well suited to the approach and compared its merits with those of currently available OS-based WSN systems.

To show that the approach has significant benefits in terms of energy efficiency, we also presented power estimations for various micro-task specifications adapted from WSN application benchmarks. Synthesis results show that, compared with an MSP430 MCU, energy efficiency improvements by more than one order of magnitude are possible.

Our future work will mainly consist in implementing a complete WSN application following our approach so as to be able to quantify more precisely achievable energy improvements.

References

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

[2] Shah Bhatti, James Carlson, Hui Dai, Jing Deng, Jeff Rose, Anmol Sheth, Brian Shucker, Charles Gruenwald, Adam Torgerson, and Richard Han. MANTIS OS: An Embedded Multithreaded Operating System for Wireless Micro Sensor Platforms. *Mob. Netw. Appl.*, 10(4), 2005.

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

[4] A. Dunkels, B. Gronvall, and T. Voigt. Contiki - A Lightweight and Flexible Operating System for Tiny Networked Sensors. *in LCN'04: 29th Annual IEEE International Conference on Local Computer Networks*, pages 455–462, Nov. 2004.

[5] Rodrigo Fonseca, Prabal Dutta, Philip Levis, and Ion Stoica. Quanto: Tracking Energy in Networked Embedded Systems. In *OSDI'08, 8th USENIX Symposium on Operating Systems Design and Implementation*, 2008.

[6] Zhigang Hu, A. Buyuktosunoglu, V. Srinivasan, V. Zyuban, H. Jacobson, and P. Bose. Microarchitectural Techniques for Power Gating of Execution Units. *International Symposium on Low Power Electronics and Design, 2004. ISLPED '04. Proceedings of the 2004*, pages 32–37, 2004.

[7] E.-Y.A. Lin, J.M. Rabaey, and A. Wolisz. Power-Efficient Rendez-Vous Schemes for dense Wireless Sensor Networks. In *ICC'04: IEEE International Conference on Communications*, volume Vol. 7, pages 3769–3776, June 2004.

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

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

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

[11] T.-D. Nguyen, O. Berder, and O. Sentieys. Cooperative mimo schemes optimal selection for wireless sensor networks. In *VTC'07-Spring: IEEE 65th Vehicular Technology Conference*, pages 85–89, April 2007.

[12] 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.

[13] M. A. Pasha, S. Derrien, and O. Sentieys. A Complete Design-Flow for the Generation of Ultra Low-Power WSN Node Architectures Based on Micro-Tasking. In *DAC'10: Proceedings of Design Automation Conference*, Anaheim, CA, USA, 2010.

[14] Joseph Polastre, Jason Hill, and David Culler. Versatile Low Power Media Access for Wireless Sensor Networks. In *SenSys '04: Proceedings of the 2nd International Conference on Embedded Networked Sensor Systems*, pages 95–107, New York, USA, 2004.

[15] Sambhu Nath Pradhan, M. Tilak Kumar, and Santanu Chattopadhyay. Integrated Power-Gating and State Assignment for Low Power FSM Synthesis. *IEEE Computer Society Annual Symposium on VLSI*, pages 269–274, 2008.

[16] C. Schurgers and M.B. Srivastava. Energy Efficient Routing in Wireless Sensor Networks. In *MILCOM'01: IEEE Military Communications Conference*, volume Vol. 1, pages 357–361, 2001.

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

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

[19] Sebastian Wallner. A configurable system-on-chip architecture for embedded and real-time applications: concepts, design and realization. *Journal of Systems Architecture*, 51(6-7):350 – 367, 2005. Reconfigurable embedded systems: Synthesis, design and application.

[20] Wei Ye, J. Heidemann, and D. Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. *INFOCOM'02: Proceedings of IEEE 21th Annual Joint Conference of the IEEE Computer and Communications Societies*, Vol. 3:1567–1576, 2002.