TinyOS Extensions for a Wireless Sensor Network Node Based on a Dynamically Reconfigurable Processor

Enkhbold Ochirsuren, Heiko Hinkelmann, Leandro Soares Indrusiak, and Manfred Glesner

Abstract Wireless sensor networks (WSNs) present design issues and challenges in both hardware and software platform development. This paper presents the implementation of a hardware-dependent component library that extends TinyOS in order to create an abstraction layer on top of a dynamically reconfigurable hardware architecture. Such hardware architecture is based on a SPARC-compliant processor and it is the core component of a generic sensor node platform targeted for future smart sensor networks. Considered as an application programming interface (API), the components of the implemented library allow the application developer to fully exploit the functionality of the dynamically reconfigurable function unit (RFU). Besides the RFU, the library also provides an interface to other standard system peripherals such as a timer, sensor, and radio transceiver. A simple TinyOS application, which includes gathering data from an attached sensor and wirelessly communicating to other sensor nodes has been demonstrated on the prototype nodes. In addition, a software visualization tool has been developed and integrated to a commercial logic simulator in order to facilitate software debugging during the cycle-accurate simulation of the hardware architecture model, described in the VHDL.

1 Introduction

As being a networked embedded system, WSNs present design issues and challenges in both hardware and software platform development. Generally, the hardware architecture of a wireless sensor node comprises of three components: processor, communication interface and sensors. The previous survey of the state-of-the-art sensor node platforms has shown that the main processing units of currently existing generic sensor node platforms are mostly based on the low-power, 8/16-bit

Enkhbold Ochirsuren · Heiko Hinkelmann · Leandro Soares Indrusiak · Manfred Glesner
Institute of Microelectronic Systems - Darmstadt University of Technology, Karlstrasse 15, 64283 Darmstadt, Germany. e-mail: boldoo, hinkelmann, indrusiak, glesner@mes.tu-darmstadt.de

Please use the following format when citing this chapter:

Ochirsuren, E., et al., 2008, in IFIP International Federation for Information Processing, Volume 271; *Distributed Embedded Systems: Design, Middleware and Resources*; Bernd Kleinjohann, Lisa Kleinjohann, Wayne Wolf; (Boston: Springer), pp. 161–170.

microcontrollers [7]. For the high-performance sensor nodes, the 16/32-bit RISC processor cores are utilized. All these processor-based platforms can provide flexibility so that they can be used for a wide variety of WSN applications. But they can suffer from low performance and low energy efficiency. This is often dealt by using ASICs. Even though ASIC-based, special-purpose sensor nodes can be utilized for some extremely low-power applications, they are not flexible and has limited functionality making it difficult to incorporate new applications. An alternative approach would be to integrate a hardware accelerator in the processing core of a sensor node. This would give us flexibility as well the meet the energy and performance requirements. Such hardware accelerators are often intended to execute specific computation-intensive tasks or tasks with real-time requirements, such as secure high-speed communication protocols, which are inefficient on general-purpose processors.

A new generic sensor node architecture for future smart sensor node platforms has been proposed in [9]. It includes a coarse-grained, domain-specific RFU in a SPARC-compliant processor core to achieve high energy efficiency. Dynamic reconfiguration technique that changes the hardware functionality quickly during runtime is used to reconfigure the RFU.

This paper presents an alternative operating system and its programmability support for a WSN node based on such a dynamically reconfigurable processor. The abstraction of the target sensor node architecture is implemented on top of the TinyOS operating system [8].

The remaining paper is structured as follows. Section 2 gives an overview on the integration of reconfigurable hardware into the generic sensor node architecture and the TinyOS operating system. Section 3 describes the new target architecture for a generic sensor node platform. Section 4 presents TinyOS extensions for the target architecture. The TinyOS portability is evaluated in Section 5, and the paper ends with a conclusion and an outlook on future work.

2 Related work

Although, the processing units of currently existing generic sensor nodes are mostly based on either microcontroller or microprocessor, there have been a few efforts to integrate reconfigurable hardware into a sensor node architecture to increase performance and lifetime of a sensor node.

A conceptual sensor node architecture that includes reconfigurable hardware has been proposed in PicoNode [13]. In this architecture reconfigurable modules have been used in both processing and communication components to provide ultra-low power operation. Modular construction idea of a versatile sensor node for WSNs has been described in [12]. Here, a FPGA coupled with a microprocessor comprises the processing unit of the sensor node and this FPGA is reconfigured to deal with complex signal processing tasks, hence decreasing the load on the microprocessor.

In order to effectively manage limited hardware capabilities and to support concurrent operations, an operating system is often needed for a sensor node. Many WSN applications have been developed on top of TinyOS, an open-source runtime environment designed for sensor network nodes. It is specialized for use on extremely resource constrained embedded systems and employs an event-driven execution model. This execution model allows the underlying hardware to operate relatively longer without battery re-charge by putting it into sleep mode when there no event occurs. Also, component based modular design enables the minimal code size and allows rapid application development by wiring only necessary components. A TinyOS program consists of a scheduler and components that are written in a nesC [6] programming language, which is an extension to the C language.

3 The new target architecture of the generic sensor node platform

The target sensor node architecture that has been used within this work is depicted in Fig. 1. It consists of a processing unit that is based on a RISC processor with a RFU; configuration, instruction and data memories; on-chip peripherals such as an interrupt controller, timer, and UART, generic sensor and transceiver interfaces; a simple bus and its arbiter module.

3.1 The dynamically reconfigurable processor

The processing unit of the architecture consists of a 32-bit LEON2 processor [5], which is the single threaded, SPARC-compliant RISC processor. The RFU is directly added into the processors instruction pipeline so that it can process the predefined computational-intensive tasks with minimumal processor intervention. For efficiency reason, the original processor architecture has been slightly modified. Its

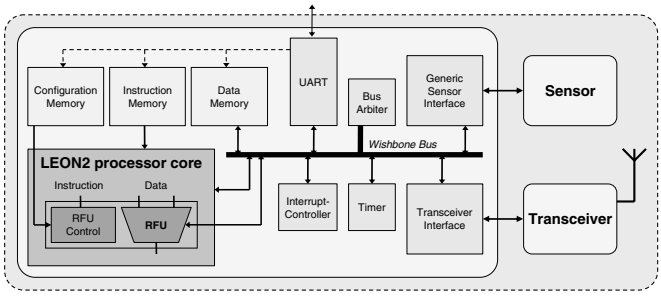

Fig. 1 The architecture of the generic sensor node platform.

caches and unnecessary peripherals were removed and the AMBA bus was replaced by a simpler wishbone bus.

The current implementation of the RFU supports following three functions:

- coding and decoding of the Cyclic Redundancy Check (CRC) with code lengths up to 32 bit
- coding and decoding of Bose-Chaudhuri-Hocquenghem (BCH) codes with code length up to 255 bit and with error correction capabilities of up to 16 errors
- encryption and decryption of the Advanced Encryption Standard (AES) with a data block size of 128 bit and key lengths of 128, 192 and 256 bit

A brief description of the overall RFU control, data path and the configuration mechanism is given in the next subsections.

3.2 The RFU control and data path

The RFU is controlled by three special instructions, which perform single- and multi-cycle operations on the RFU and reconfiguration: single-cycle execution (ESR), multi-cycle execution (EMR), and reconfiguration (CRT). These special instructions have been added to the original SPARC V8 instruction set. The EMR and CRT instructions are executed independently by the RFU, by allowing the processor to execute instructions in parallel with the RFU.

The RFU's data path includes specifically tailored modules for error correction and encryption algorithms, such as a multiply accumulate module (MAC), an inversion module, registers, a local memory and a memory access unit as well as flexible interconnections. The MAC module is based on Galois Field (GF) multiplier and GF adder cells. The inversion module performs GF division operation that is required by the AES algorithm. In order to increase local storage capabilities the registers and the memory modules have been implemented. The memory module can be configured either as look-up table for logic operations or a FIFO buffer with parameterized width and depth. The memory access unit supports block data computation and packet processing by reading or writing data from/to memory consecutively, starting from the given start address.

3.3 The reconfiguration mechanism

A two-layer reconfiguration mechanism is specially developed to allow rapid reconfiguration within a function and between different functions.

In order to provide fast reconfiguration, a number of multi-context configuration tables and a reconfigurable look-up table is utilized. The multi-context configuration tables are responsible for storing several configurations for each module of the data path. A desired configuration can then be selected with a special tag within one cy-

cle. Tag generation is done by the reconfigurable look-up table, which also specifies the sequence of tags for multi-cycle operations and allows jumps and loops. Hence, the RFU can autonomously execute very long computations that comprise of multiple cycles. This tag-driven reconfiguration belongs to the first layer reconfiguration.

Loading of configuration data from an external configuration memory to these tables becomes the second layer reconfiguration. For this purpose, the compressed control information, which is called a reconfiguration profile, is composed and stored in the configuration memory along with complete configuration data. The profile exactly specifies how many entries need to be loaded to each table. In addition, a task manager module monitors reconfiguration process by verifying the current configuration in the tables and can skip unnecessary configuration overhead. As a result, the reconfiguration latency achieved is one cycle to reconfigure within a function, and less than hundred cycles to completely reconfigure an another function.

4 The TinyOS extensions for the target architecture

Even though TinyOS supports several existing sensor node platforms, none of them uses a SPARC-compliant processor as a processing unit. Thus, it requires the preliminary work of porting TinyOS to the target hardware architecture. The whole porting process will be briefly described in the next subsections according to the completion order.

4.1 Platform definitions

In order to define and locate platform specific files, a new subdirectory, called "leon2mote", with associated files has been created in the TinyOS directory tree. These files include: "leon.h", "leon2_hardware.h" and "hardware.h". The "leon.h" file includes details about the LEON2 processor, whereas the "leon2_hardware.h" and "hardware.h" files contain macros for pin assignment, functions for supporting atomic statements and other specific hardware definitions for the sensor node.

4.2 Hardware presentation layers

The TinyOS hardware presentation layers (HPLs) are the lowest level components that directly interact with the underlying hardware. They access the hardware in the usual way, either by memory or by port mapped input/output (I/O). In the reverse direction, the hardware can request services by signalling an interrupt. Using this

communication scheme, the HPLs abstract the details of the hardware and provide a more usable interface for the upper layer components.

Even though each HPL component will be as unique as the underlying hardware, all of them will have a similar general structure. Each HPL component should have:

- commands for initialization, starting and stopping of the hardware that are necessary for effective power management policy
- "get" and "set" commands for the registers that control the operation of the corresponding hardware
- commands with descriptive names for the most frequently used operations
- commands for enabling and disabling interrupts triggered by the hardware
- interrupt handlers for the interrupts that are triggered by the hardware

According to the target architecture description, HPLs for the hardware initialization, LEDs, timer, sensor and radio interfaces, and the RFU have been implemented. With exception of the RFU, the rest for the underlying hardware module are accessed by memory mapped I/O addresses. The timer, sensor and radio interfaces respond to the overlaying HPLs by generating interrupts that are defined in the LEON2 interrupt assignment scheme. Table 1 shows all HPLs and the system components that have been developed within this work.

Table 1 The developed TinyOS components.

Component type	Component name	Description
Hardware specific HPLs	HPLInitM.nc	HPL for the hardware initialization
	HPLTimer.nc	HPL for the timer unit of the LEON2 processor
	HPLLeds.nc	HPL for the LEDs attached to the prototype board
	HPLRfmRx.nc	HPL for the radio interface (receiver mode)
	HPLRfmTx.nc	HPL for the radio interface (transmitter mode)
	HPLPhotoM.nc	HPL for the sensor interface
	HPLRFU_CRC.nc	HPL for the RFU (CRC checksum calculation)
	HPLRFU_AES.nc	HPL for the RFU (AES block (en)-decryption)
System components	LeonTimerM.nc	A system timer component
	LeonSenseToInt.nc	Gathers sensor sample and displays to the LEDs
	LeonIntToRfmM.nc	Sends data to the radio interface
	LeonRfmToIntM.nc	Receives data from the radio interface
	AESM.nc	An AES encryption and decryption component

In case of the RFU, the implementation of associated HPLs is different than others. For each function (cf. Sect. 3.1) that is provided by the RFU, a number of task specific subfunctions are assigned. Each task is associated with one specific function of the RFU with fixed parameters, like 8-bit CRC checksum calculation of a 32-bit data block. Generally, tasks are executed in the RFU in following steps:

- allocate a new task and reconfigure the RFU for that task (CRT instruction)
- load the required parameters for the task (mainly ESR instruction)
- execute the task and return a result (ESR and/or EMR instructions)

Hence, each of these steps is defined as a subfunction and further these subfunctions are used in creating a task specific HPL for the RFU.

Two distinct HPLs have been implemented for the RFU: "HPLRFU_CRC.nc" and "HPLRFU_AES.nc". The former HPL sets up the RFU for 8-bit CRC checksum calculation on 32-bit data block, starts the calculation in the RFU and returns a resulting checksum. The latter HPL reconfigures the RFU for the AES encryption and decryption algorithm with 128-bit key, initiates the algorithm in the RFU and returns a cipher or decrypted data, each having 128-bit block size.

All subfunction definitions are packed in an external library. As stated in Sect. 3.2, the RFU introduces three new instructions for reconfiguration and execution purpose. In order to make these new instructions available to the standard cross-compiler for the LEON2 processor, compiler's source code is modified by including the mnemonics of these new instructions.

4.3 System components

A TinyOS system component provides a concrete system service to the overlying application components. It can be used by any application without modifications.

The typical system service in TinyOS is a system timer that can function as multiple timers, each of which can be managed independently. For this purpose the first timer of the LEON2 timer unit has been chosen. An implementation of the system timer component, "LeonTimerM.nc", has been derived from the original TinyOS timer component "TimerM.nc". The main reason of such a distinct timer component is regarding to the operational mode of the LEON2 processors timer unit. The timer unit decrements its counter value on each timer tick and generates a timer interrupt when the counter value underflows. Therefore, the only difference between them is a modification that reflects such behaviour to the "Timer" interface implementation.

A new block cipher component, "AESM.nc", that performs the AES encryption and decryption has been implemented. The component considers an optimized software implementation of the AES algorithm on 32-bit platforms proposed in [1]. The applied strategy is to restructure the standard algorithm by introducing a transposed version of the state matrix. Due to the transposed state matrix considerable amount of computation is saved by eliminating the rotation operations both from MixColumns and Inverse MixColumns, resulting a performance gain. Moverover, it employs only the nonlinear byte substitution tables (Sbox and Inverse Sbox) as look-up tables to keep the required memory as less as possible.

The list of the developed system components is given in Table 1.

5 The validation and evaluation of TinyOS porting

In order to validate the TinyOS porting and demonstrate a typical sensor node operation, the following TinyOS application was developed. One node periodically gathers samples from a light sensor, displays light intensity on the LEDs and sends the samples wirelessly to other nodes. Other nodes listen to the radio interface until data is received. When the transmitted data is received, they display it on the LEDs and wait for next reception. We have not used any networking mechanism so there is neither addressing nor routing algorithms included. Hence, our applications are not directly similar to the standard "SenseToRfm" and "RfmToLeds" applications.

The validation test that checks TinyOS port is done in the ModelSim logic simulator by simulating a VHDL model of the sensor node architecture programmed with TinyOS application code. In order to track software execution on VHDL model, an additional monitoring tool has been developed and integrated to ModelSim (cf. Fig. 2). During VHDL simulations, it informs component interactions of the currently executing TinyOS application with exact timing information. Besides that it shows the current state of the LEON2 processor core, which includes control/state register contents (%psr, %tbr, %wim), register file contents (%g0-%g7, %l0-%l7, %i0-%i7), and state of all the pipeline stages.

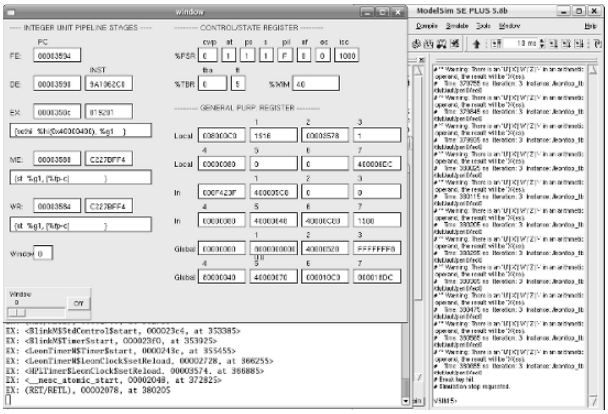

Fig. 2 The GUI of the monitoring tool added to ModelSim.

The benefit of the RFU is proved by the execution time analysis on the example applications. In order to show the impact of the RFU on computing a CRC checksum and encrypting a cipher by the AES algorithm, a couple of example applications are developed for each case: one uses a software (nesC) component and other one uses RFU. It can be obviously seen that the execution of CRC and AES functions in RFU is much faster than if they are executed in software (cf. Table 2). The speed-up factor of up to 93 and 71 can be reached compared to the software implementations.

For demonstration of example applications, prototypes that consist of an FPGA board and an additional board have been used. In the FPGA board the complete sen-

Table 2 Execution time comparison.

Function name	Version	Execution, cycles
CRC	RFU	11
	software (nesC)	1029
AES (encryption)	RFU	238
	software (nesC)	17093

sor node design, including processor, RFU, memories, peripherals, interfaces and bus, is implemented. On the additional board the Xemics DP1203 radio transceiver, a planar antenna and ten LEDs, and extension pins are mounted.

The portability of TinyOS has been evaluated by code size and concurrency potential.

A TinyOS application code size should be as little as possible to fit in resource constraint hardware. Compared to the compiled applications for the Mica mote, code size of our test applications are quite large (cf. Table 3). The reason is that the code includes additional 9Kbytes of code that includes processor boot sequence (1.5KB), trap table (4KB), trap handling routines (2KB), and processor specific constants (1KB). While the amount of required instruction and data memory seems to be large, it is still in the range that most sensor nodes have [2].

Table 3 Code size comparison.

Application name	Mica mote, bytes		LEON2+RFU, bytes	
	ROM	RAM	ROM	RAM
Blink	1652	48	14416	1780
Sense	3602	83	15408	1780
RfmToLeds	7538	280	17072	1804
SenseToRfm	9970	372	17728	1796

The main metric for the concurrency potential is context switching speed. The most expensive switching is related to the hardware interrupt handling. On an average it takes around 75 clock cycles to signal the respective event after a hardware interrupt is triggered.

6 Conclusion and future work

This paper presents the first port of the TinyOS operating system (version 1.x) to a new generic wireless sensor node platform that is based on the customized LEON2 RISC processor with the integrated dynamically reconfigurable function unit (RFU). The most common TinyOS components that can be contained in every application have been implemented and tested by using both the commercial hardware simulator and FPGA-based prototype nodes. The analyses from compilations show that the amount of required memories is still within the range that the most existing

generic sensor node platforms have. From the simulations, it was shown that the RFU significantly speeds up error checking and data (en)-decryption processes.

In order to test the TinyOS porting and track software execution during simulation, an additional monitoring tool has been developed and integrated to the commercial hardware simulator, ModelSim. The monitoring tool facilitates software debugging on cycle-accurate simulation of any LEON2 processor based system-on-chip (SoC) design.

The further work is directed to improve the current extension and add network simulation. Hence, some work will be done towards to support the Active Messaging (AM) protocol stack [3], which is basic communication protocol for the TinyOS applications. Currently, the TinyOS based sensor network models are only simulated in TOSSIM [10], a TinyOS simulator with networking support. Therefore, there is a necessity to implement the TOSSIM components for the target platform and added to the TOSSIM library. The TOSSIM components can later be used in other existing frameworks for modelling wireless systems, such as Viptos [4] and SENSIM [11].

References

1. G. Bertoni, L. Breveglieri, P. Fragneto, M. Macchetti, S. Marchesin, Efficient software implementation of AES on 32-bit platforms, CHES '02: Revised Papers from the 4th International Workshop on Cryptographic Hardware and Embedded Systems, 2003, pp. 159–171.
2. J. Beutel, Metrics for sensor network platforms, REALWSN'06: Proceedings of the ACM Workshop on Real-World Wireless Sensor Networks, 2006, pp. 26–30.
3. P. Buonadonna, J. Hill, D. Culler, Active message communication for tiny networked sensors, Submitted to IEEE INFOCOM 2001, 2001.
4. E. Cheong, E. A. Lee, Y. Zhao, Viptos: A graphical development and simulation environment for TinyOS-based wireless sensor networks, SenSys '05: Proceedings of the 3rd International Conference on Embedded Networked Sensor Systems, 2005, pp. 302–302.
5. J. Gaisler, LEON2 processor users manual, version 1.0.21 XST edition, Available at Gaisler Research. http://www.gaisler.com. Nov. 2003.
6. D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, D. Culler, The nesC language: A holistic approach to networked embedded systems, PLDI '03: Proceedings of the ACM SIGPLAN 2003 Conference on Programming Language Design and Implementation, 2003, pp. 1–11.
7. J. Hill, M. Horton, R. Kling, L. Krishnamurthy, The platforms enabling wireless sensor networks, Commun. ACM, Vol. 47 (2004) No. 6, pp. 41–46.
8. J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, K. Pister, System architecture directions for networked sensors, SIGPLAN Not., Vol. 35 (2000), No. 11, pp. 93–104.
9. H. Hinkelmann, P. Zipf, M. Glesner, A domain-specific dynamically reconfigurable hardware platform for wireless sensor networks, ICFPT '07: International Conference on Field-Programmable Technology, 2007, pp. 313–316.
10. P. Levis et al., TOSSIM: Accurate and scalable simulation of entire TinyOS applications, Proceedings of the First ACM Conference on Embedded Networked Sensor Systems (SenSys 2003), 2003.
11. C. Mallanda, A. Suri, V. Kunchakarra, S. S. Iyengar, A. Durresi, Simulating wireless sensor networks with OMNET++, Submitted to IEEE Computers 2005, 2005.
12. J. Portilla, A. de Castro, E. de la Torre, A modular architecture for nodes in wireless sensor networks, Journal of Universal Computer Science, Vol. 12 (2006), pp. 328–339.
13. J. M. Rabaey, M. J. Ammer, J. L. da Silva, D. Patel, S. Roundy, PicoRadio supports ad hoc ultra-low power wireless networking, Computer, Vol. 33 (2000) No. 7, pp. 42–48