ASYNCHRNOUS ARCHITECTURE FOR SENSOR NETWORK NODES

Aurélien BUHRIG, Marc RENAUDIN, Dominique BARTHEL
TIMA Laboratory, CIS group, Grenoble, France {Aurélien.Buhrig, Marc.Renaudin}@imag.fr
France Telecom R&D - Meylan – France – Dominique.Barthel@francetelecom.com

Abstract: We present an asynchronous software and hardware architecture specifically suited for wireless sensor network nodes. To reduce power consumption and/or increase performances, some blocks go into hardware. The whole system is modelled using a unique asynchronous HDL before being partitioned. The software part that is executed on an asynchronous processor is then scheduled using a quasi-static scheduling and operates in an event-driven way with reactive hardware through an interface controller. We use an asynchronous analog to digital converter combined to a new approach in the non-uniform signal processing theory to obtain an entire event-driven platform. The use of asynchronous hardware allows to efficiently design a fine-grained dynamic power consumption control mechanism controlling V_{dd} (digital voltage scaling) and V_{bb} (bulk biasing) in order to manage the speed/power consumption trade-off and to go in a low-power idle mode state with very few static leakage. Finally, to increase the lifetime of the nodes, some scavenging techniques are added.

Key words: sensor network node, architecture, asynchronous, low-power, event-driven, high level description, modeling.

1. INTRODUCTION

Ambient intelligence is getting more and more importance in nowadays life and addresses a large panel of applications. This intelligence is closely linked to the sensing of the environment that leads to the design of sensor platforms that can be connected into wireless ad hoc networks. The possible

applications of sensor networks are numerous. Among them: home humidity and temperature monitoring, seismic sensing in harsh environment, movement detection and localization for the consumer electronic and military purpose, analysis of chemical substance in a natural environment, and so on.

Ad hoc networks, due to their intended support of "no-limit" infrastructure-less communication, pose many significant new challenges in comparison with traditional wireless networks. The improvements in microelectronic technologies have made possible the design of small and less consuming systems on chip, but the autonomy of such devices is still a key issue. Indeed, these nodes are expected to sense the environment, compute data and perform wireless communications while consuming as little energy as possible, potentially under some application-imposed real-time constraints. They have a limited embedded energy that consists in a traditional battery to which it is possible to add a renewable energy source (coming from vibrations[1], solar radiations [2] or RF power [3]).

In this context, energy saving is critical to operate the sensor network for a long period of time. This work aims at minimizing this power consumption at the node-level and we present in this paper our vision of such an ultra low-power platform in which both software and hardware are designed in an asynchronous and totally event-driven way to maximize the lifetime of the nodes. As there is no switching activity when the circuit does not require to perform any operation and since they allow easier dynamic power management and are able to respond very quickly, asynchronous circuits are ideal for sensor networks in which most of the time is spent waiting for events on the radio interface, sensor or even timeouts from a standalone timer.

In this paper, we give a new vision of the co-design of the hardware/software architecture of sensor network nodes. This vision lies in the adoption of a fully asynchronous event-driven system, especially well suited to design for ultra low power. The nodes of the sensor network are asynchronous systems (clock-less) in which all the processing chain, software and hardware processes are event-driven.

The whole system is modeled using a single language or at least a single representation. This common description, which is at the moment Petri Nets, can be simulated and analyzed to allow an efficient co-design and an efficient generation of the interfaces.

In order to save even more energy, a fine-grained power consumption management is adopted. For confidentiality purpose, a low-power cryptography operator can be used. Finally, if some applications require dynamic hardware reconfiguration, the use of asynchronous FPGA is considered.

2. CLASSICAL SENSOR NETWORK OS COMPARISON AND LIMITATION

2.1 TinyOS vs general-purpose OS

An important problem to cope with is to make code execution efficient by minimizing the operating system (OS) overhead without removing traditional wireless-sensor network features. Traditional general-purpose multitasking embedded OS were originally developed for the PC platform and have been adapted to embedded systems. Those OS, even the most embedded ones, are too general-purpose to be efficient and the context switching generates expensive overhead that is tolerable for a fast and unlimited energy PC platform or a powerful embedded processor but is not acceptable for ultra low power embedded platforms.

A very interesting way to solve this problem is approached with TinyOS developed at UC Berkeley [4]. TinyOS is a reactive OS that does not target a broad range of general applications but only specific tasks for sensor networks. It can be described as follow: external events that occur on the RF or sensor interface propagate upward from the lowest layers till they are handled by the upper ones in an asynchronous way between the different blocks.

A performance and power consumption comparison between TinyOS and the general-purpose operating system eCOS [5] is reported in [6]. The result is that the use of TinyOS drastically reduces power consumption and improves performances. Some features (such as virtual memory, dynamic memory allocation, etc.) that are useless for sensor networks are not implemented.

2.2 Limitations

The TinyOS architecture is good at reducing power consumption. However we here–after highlight some important limitations of TinyOS [6]:

- It would be advantageous for power consumption and/or performance to implement some components of the application into hardware
- Those components could execute specific tasks whereas in TinyOS and traditional operating systems all tasks go into software
- There is no way to dispatch software tasks onto different resources
- There is no global power-control mechanism; implementing dynamic voltage scaling is a good way to reduce power consumption while meeting the performance constraint but is hardly possible with TinyOS.

- Some events can be lost during treatment if the event queue is full.

2.3 Approach adopted

2.3.1 Overview

In our approach, we want to keep the advantages of TinyOS without its drawbacks. In particular, we think that a major feature is to implement some parts of the system into hardware. For example, if a CRC or a Viterbi operator is needed to encode and decode messages, it is advantageously integrated into hardware in order to reduce the power consumption that would be engendered by the numerous computations performed by the processor to execute equivalent software.

Furthermore, due to the reactive nature of the desired operating system and applications, one expects the hardware part to be event-driven. Therefore, we implement it using the asynchronous technology [7], also known for its low-power consumption. The choice for this technology is detailed in the next section and requires dedicated hardware/software interfaces.

Finally, we add a global power-control mechanism, supported by a power management unit implemented in software and hardware. This power management is flexible enough to be applied at different granularity levels according to the hardware processes and the node architectures.

2.3.2 Asynchronous technology

As mentioned previously, it is advantageous to implement some parts of the system in hardware. We have chosen the asynchronous technology for two main reasons.

Firstly, the asynchronous technologies allow an important power consumption reduction. Indeed, asynchronous systems do not use any clock and the synchronization between the different asynchronous operators is performed by a request/acknowledgment protocol (handshaking protocol) implemented locally as shown in Figure 1. Consequently, only the parts of the circuit performing an operation have an activity. The rest of the circuit consumes very little energy (only static leakage) and are immediately woken-up when an event occurs on its inputs.

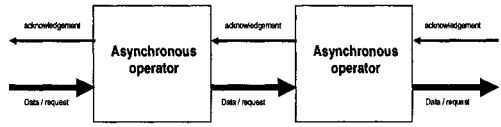

Figure 1. Asynchronous principle

Secondly, due to the reactive nature of the software, it is interesting to go further with the asynchronous architecture so that the whole system behaves in a totally event-driven way. With such an architecture, an event (from the radio interface for instance) will propagate between different asynchronous blocks regardless of the software or hardware nature of the crossed blocks. This communication type requires specifying a communication interface between software and hardware. This interface is not as complex as an interface between synchronous hardware and event-driven software.

2.3.3 Communications

The synchronous and asynchronous notions can be tricky and depends on the context. Indeed, these notions apply at different levels (hardware technology, communication specification, communication implementation). At hardware technology level, an asynchronous system is a clock-less system where the synchronization between two asynchronous blocks is performed as mentioned previously. Figure 2 shows the four phases protocol.

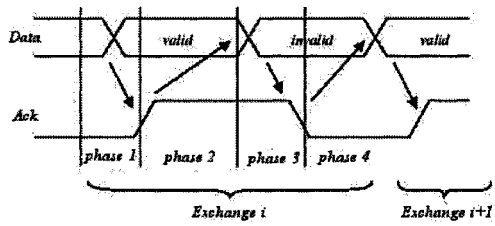

Figure 2. Four phases protocol

Therefore, in asynchronous technology, since both communicating processes exchange information using requests and acknowledgements, both are synchronized and hence the asynchronous hardware implements a synchronous communication.

3. SOFTWARE ARCHITECTURE

3.1 Modelling of the whole system and partitioning

Both software and hardware have an identical asynchronous event-driven architecture and this is very interesting to model the whole system and to abstract the implementation of the processes. At top level, the system is designed using a single concurrent description model. At the moment, this description is done with an asynchronous HDL, the CHP (Communicating Hardware Processes [8], derived from the CSP [9]). This HDL being not very convenient to describe software and another description language capable of being translated into Petri Nets will be used in the future.

The concurrent description of the system is then automatically translated into Petri Nets [10] and each concurrent process of the high level description is translated into a Petri Net.

The complete description of the system is obtained by the composition of the Petri Nets of every top level processes are composed at the communication level.

Once the system is described, the partitioning can occur according to some criteria such as performance, power consumption (that can be simulated and analyzed with adequate tools such as TAST [11] using the Petri Net representation of the system) or modularity.

Finally, the software part is scheduled, whereas the hardware parts are synthesized into gates using appropriate tools [11].

3.2 Scheduling

An important part of operating system overheads and hence power consumption comes from the dynamic scheduling of the software tasks. In dedicated software operating systems such as TinyOS [4], or in hardware OS parts implemented in low-power processors such as SNAP [12] or bitSNAP [13], the scheduling is ensured using a FIFO. In our case, we have chosen to implement a static scheduling to cope with this problem. This scheduling is found using an algorithm whose core is based on the algorithm proposed in [14] and applied to the Petri Net model of the whole system. The limitations of the algorithm proposed in [14] is that the communications cannot be probed and are implemented with infinite FIFO. Our method does support probing operation and does not assume infinite memory communication channels.

4. HARDWARE FEATURES

4.1 CPU

As seen before, asynchronous hardware has an event-driven architecture that allows the system to be clock-less and to reduce power consumption. The class of asynchronous circuit we use to design the nodes is Quasi Delay Insensitive (QDI) circuits. A QDI circuit behaves correctly regardless of the delay of the gates and wires under the weak assumption of *isochronic fork*. A fork (a wire that connects a sender to several receivers) is *isochronic* when the delays between the sender and the receivers are about the same. This architecture is data-driven. In other words, the asynchronous block is asleep when no data comes in. As soon as a data is present on its input, the hardware wakes up.

The processing unit is not defined yet. Enhanced versions of the asynchronous 8-bit microcontroller MICA [15] or the 16-bit processor ASPRO [16, 17] can be used as well as an asynchronous 32-bit RISC processor currently under development. The data width can be selected to enable the best power consumption/computational power trade-off for a given sensor network.

4.2 Power management

A different way to make power savings is to use dynamic voltage scaling (DVS) to control power consumption at run time. With synchronous circuits, decreasing the voltage makes the signal transition slower to establish and hence imposes a decrease of the clock frequency. Changing the clock frequency introduces delay overhead due to the synchronization of the *phase lock loop* (PLL) and extra power consumption. With asynchronous circuits, modifying the voltage does not introduce any overhead. The speed of the circuit changes on its own since the circuit behavior is not sensitive to the rising and falling times of the signals (delay insensitivity).

So it is easy to add to the processor a hardware part that smoothly manages power consumption. This power manager is able to increase or decrease the supply voltage (V_{dd}) of one or several parts of the system according to the performances required. With the evolution of the technologies, a more and more important part of the power consumption is due to static leakage. To reduce the static leakage, we use a technique called "back biasing" which consists in biasing the bulk to affect the transistors threshold V_T [18]. The bulk biasing is a feature that is integrated to the

power management unit. Therefore this unit is able to control V_{dd} and V_{bb} to reduce power consumption, and this at different granularity levels. In practice, the software is controlling the power management unit in order to optimize the speed/power

According to the application, it is possible to define different policies. For instance, if the node has to perform real time operation, one chooses the power management to enable the application to meet the real time constraints. On the contrary, if the node must have the longest lifetime as possible, the speed will be chosen according to the power consumption and the remaining battery energy.

4.3 Hardware-Software Interface

A controller is designed to operate correctly between software and hardware tasks. This *Hardware-Software Interface* (HIS, Figure 3) will manage communications between hardware and software in order to prevent the microprocessor from being interrupted by each hardware bloc that would generate software handler calls and extra power consumption. This controller wakes up the microprocessor when a desired hardware event income.

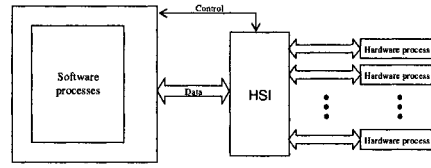

Figure 3. Hardware Software Interface Scheme

4.4 Signal processing chain – AADC

A new approach in the signal processing which leads to a significant energy saving is to use an asynchronous analog to digital converter (AADC) [19]. This one takes samples only if the sensed physical characteristic changes more than a predefined quantum. The samples and the date are saved altogether. This enables a saving on samples. Then a non uniform sampling theory [20] is applied to the samples to take advantage of the AADC. Such a converter is hence totally data driven.

As for the performances of such a method, for the same number of points, the processing of the measures needs more computations than a traditional ADC with classical signal theory. Nevertheless, since these

computations are performed on a smaller set of points, the computational complexity is globally reduced. In comparison with traditional synchronous ADC on a voice signal application, the computational complexity is reduced by one order of magnitude by using asynchronous processing chain with non uniform sampling theory [20].

5. CONCLUSION AND FUTURE WORK

This paper presents an entire event-driven platform for sensor networks. This platform aims at drastically reducing power consumption thanks to the reactivity of every parts of the system, from the analog to digital converter to the radio frequency interface and through a data-driven asynchronous processor executing software processes that communicates with hardware with an interface controller (HSI). This platform is able to operate in a large range of power supply making possible a fine-grained power consumption management by controlling the voltage and the bias of the bulk.

Now, the asynchronous platform is specified and a first prototype is being designed. Our future works will focus on the design of CAD tools in order to describe the whole using system level language or model. Those CAD tools will have to integrate features to help the designers to co-design the system at best, taking into account the required performances and power consumption.

Future work will also focuses on the modeling of performance requirement of the software tasks after the scheduling in order to know statically the speed the system needs at runtime. This aims at controlling dynamically the voltage supply of the CPU according to its orders (expressed in MIPS) using a feedback system [21].

ACKNOWLEDGMENT

This work is partially supported by France-Telecom R&D department.

REFERENCES

1 S. Meninger, J. O. Mur-Miranda, R. Amirtharajah, A. Chandrakasan, and J. Lang, "Vibration-to-electric energy conversion," presented at International Symposium on Low Power Electronics and Design, San Diego, California, USA, 1999.

2 B. A. Warneke, M. D. Schott, B. S. Leibowitz, L. Zhou, C. L. Bellew, J. A. Chediak, J. M. Kahn, B. E. Boser, and K. S. J. Pister, "An autonomous 16 mm3 solar-powered node for

distributed wireless sensor networks," presented at Sensors'02, Orlando, Florida, USA, 2002.
3 A. Bayrashev, A. Parker, W. P. Robbins, and B. Ziaie, "Low frequency wireless powering of microsystems using piezoelectric-magnetostrictive laminate composites," presented at 12th International Conference on Transducers, Solid-State Sensors, Actuators and Microsystems., Boston, USA, 2003.
4 TinyOS Group, "TinyOS tutorial," 2003.
5 RedHat, "eCOS."
6 S. F. Li, R. Sutton, and J. M. Rabaey, "Low Power Operating System for Heterogeneous Wireless Communication Systems," presented at PACT 01, Barcelona, Spain, 2001.
7 M. Renaudin, "Asynchronous Circuits and Systems : A Promising Design Alternative," in *Microelectronic Engineering*, vol. 54, 2000, pp. 133-149.
8 A. Martin, "Programming in VLSI: from communicating processes to delay-insensitive circuits," in *Developments in concurrency and communication*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1991, pp. 1-64.
9 C. A. R. Hoare, "Communicating Sequential processes," *Communication of the ACM*, vol. 21, pp. 666-677, 1978.
10 M. Renaudin and A. Yakovlev, "From Hardware Processes to Asynchronous Circuits via Petri Nets: An Application to Arbiter Design," presented at Workshop on Token Based Computing, Bologna, Italy, 2004.
11 K. Slimani, Y. Rémond, G. Sicard, and M. Renaudin, "TAST profiler and low energy asynchronous design methodology," presented at International Workshop on Power And Timing Modeling Optimization and Simulation, Santorini, Greece, 2004.
12 V. Ekanayake, C. Kelly, and R. Manohar, "An Ultra-low-power Processor for Sensor Networks," presented at Eleventh International Conference on Architectural Support for Programming Languages and Operating Systems, Boston, USA, 2004.
13 V. Ekanayake, C. Kelly IV, and R. Manohar, "BitSNAP: Dynamic Significance Compression For a Low-Energy Sensor Network Asynchronous Processor," presented at Eleventh IEEE International Symposium on Asynchronous Circuits and Systems, New York City, USA, 2005.
14 J. Cortadella, A. Kondratyev, and L. Lavagano, "Quasi-static scheduling for concurrent architectures," presented at Third International Conference on Application of Concurrency to System Design (ACSD'03), Guimarães, Portugal, 2003.
15 A. Abrial, J. Bouvier, M. Renaudin, P. Senn, and P. Vivet, "A New Contactless Smartcard IC using an On-Chip Antenna and an Asynchronous Micro-controller," presented at ESSCIRC'00, Stockholm, Sweden, 2000.
16 M. Renaudin, P. Vivet, and F. Robin, "ASPRO : an Asynchronous 16-bit RISC Microprocessor with DSP Capabilities," presented at ESSCIRC, Duisburg, Germany, 1999.
17 M. Renaudin, P. Vivet, and F. Robin, "ASPRO-216: A Standard-Cell Q.D.I. 16-Bit RISC Asynchronous Microprocessor," presented at 4th International Symposium on Advanced Research in Asynchronous Circuits and Systems, San Diego, California, USA, 1998.
18 E. Labonne, G. Sicard, and M. Renaudin, "Dynamic Voltage Scaling and Adaptive Body Biasing Study for Asynchronous Design." Grenoble: TIMA - INPG, 2004.
19 E. Allier, G. Sicard, L. Fesquet, and M. Renaudin, "A new class of Asynchronous A/D Converters Based on Time Quantization," presented at ASYNC'03, Vancouver, 2003.
20 F. Aeschlimann, E. Allier, L. Fesquet, and M. Renaudin, "Asynchrounous FIR filters: Towards a new digital procession Chain," presented at ASYNC'04, 2004.
21 D. Rios, A. Buhrig, and M. Renaudin, "Power Consumption Reduction using dynamic control of Microprocessor performance", presented at PATMOS, Leuven, Belgium, 2005.