

Design of A Development Platform for HW/SW Codesign of Wireless Integrated Sensor Nodes

Kashif Virk,
Jan Madsen,
Andreas Vad Lorentzen
System-on-Chip Group
Technical University of Denmark
email: {virk, jan}@imm.dtu.dk

Martin Leopold,
Phillipe Bonnet
Distributed Systems Laboratory
Institute of Computing Science
Copenhagen University, Denmark
email: {leopold, bonnet}@diku.dk

Martin Hansen
Valby Design Center
IO Technologies A/S
email: martin@iotech.dk

Abstract

Wireless integrated sensor networks are a new class of embedded computer systems which have been made possible mainly by the recent advances in the micro and the nano technology. In order to efficiently utilize the limited resources available on a sensor node, we need to optimize its key design parameters which is only possible by making system-level design decisions about its hardware and software (operating system and applications) architecture. In this paper, we present the design of a sensor node development platform in relation to an application of wireless integrated sensor networks for sow monitoring. We also discuss the related hardware/software codesign tradeoffs.

1. Introduction

Wireless integrated sensor networks are a class of networked embedded systems that combine sensing, computation, and communication in an inexpensive and very small form factor device with limited energy. They are meant to act as a bridge between the physical and the virtual worlds. Apart from myriads of applications being proposed for them, their low cost and size, ease of deployment, and autonomous operation make them a viable and non-intrusive solution for livestock monitoring applications.

In the Hogthrob [26] project, we are aiming to develop a sensor network infrastructure for sow monitoring. A part of the project consists of developing sensor nodes that can be tagged onto the sows (in replacement of the RFID tags they

wear today). Such sensor nodes must be low-cost (costing no more than a couple of Euros), small-sized (small enough, when packaged, to be worn as an ear tag) and low-energy (a few months' autonomy is a minimum). In order to conform to the above-mentioned requirements, we have decided to design a sensor node on a chip.

Today's Danish farms for pig production are using RFID tags for sow identification and controlling their food consumption. However, these tags have proven to be quite impractical to locate sows in large pens. Moreover, they are not flexible enough to be useful in contexts other than controlling the food consumption. For example, the pig farmers have to manually monitor the key aspects of a sow's lifecycle such as the onset of estrus¹ or farrowing - the phenomena that have a profound effect on pig production.

Numerous sensor network research projects have designed sensor nodes with microprocessors from Atmel, Texas Instruments, Intel, etc. for similar purposes, notably [16, 22]. However, none of the sensor node architectures, reported so far in the literature, approaches the sensor node design from a hardware/software codesign perspective (except in [6], but to a limited extent). We have designed a sensor node development platform in order to explore the design-space both in terms of hardware and software and to end up with a complete sensor node implemented on a single chip. Hence, a key component of our sensor node development platform is an FPGA which has enabled us to

¹Estrus or Heat Period is the period when a sow can be bred and it lasts for a short time only. If a sow is not bred during its first estrus, it is considered unproductive from the commercial point of view since it normally returns to estrus about 3 weeks later and needs to be fed and housed meanwhile.

explore various hardware/software tradeoffs.

An important design consideration while designing our sensor node development platform, called the Hogthrob platform, has been to reduce the overall cost of prototyping by using COTS (Common Off-the-Shelf) components. Another consideration has been to have the capability to experiment with various combinations of sensors, radio transceivers (e.g., Bluetooth, Zigbee, Wi-Fi, UWB, etc.), and microprocessors (e.g., Atmel, Intel, ARM, Texas Instruments, Microchip, etc.) to select the optimal combination. To achieve these objectives, we have adapted a modular design strategy so that we can swap sensors and radio transceivers with the ones resulting in more efficient energy and system performance. For trying different microprocessors and/or to perform hardware acceleration, we needed some form of reconfigurable logic on the sensor node development platform so that we can configure the sensor node with various microprocessor cores. Of course, low power, small form factor, and robust packaging were the necessary features as well because the sensor nodes have to be mounted on sows.

The Hogthrob platform has also been designed with a view to explore the tradeoffs of implementing application functionality either in software (on the embedded processor) or hardware (on the reconfigurable logic), without being constrained by the initial design choices as was the case in [20]. As an initial design step, all the application functionality has been placed on the embedded processor and is gradually being moved to the FPGA. At the current stage of software development, the radio transceiver and other peripherals are being controlled by the software running on the embedded processor but, eventually, the embedded processor will only initialize the FPGA and function as an external timer and an A/D converter for the FPGA.

2. Sensor Node Hardware Architecture

The Hogthrob platform architecture consists of four closely-interacting subsystems (please refer to Figure 2). These subsystems are: the sensing subsystem, the computing subsystem, the communication subsystem, and the power-supply subsystem. The platform has been designed using a modular design approach and comprises one mother board (8.5cm x 7cm) which comprises the computing and the power-supply subsystems, one daughter board for the communication subsystem (4cm x 5cm) and another daughter board for the sensing subsystem. The mother board has been further divided into the analog and the digital sections with the analog section mostly occupied by the power-supply subsystem and the computing subsystem comprising the digital section.

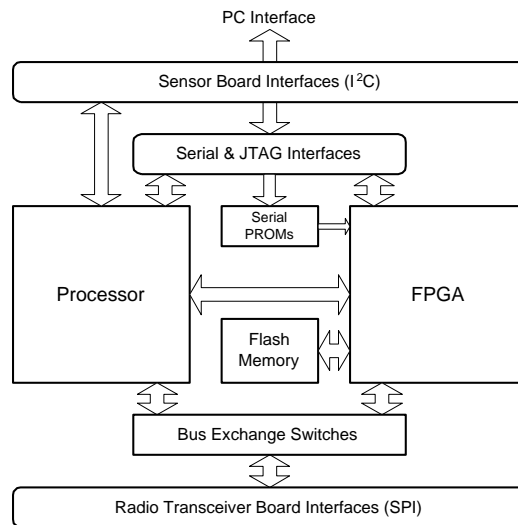


Figure 1. Hogthrob Sensor Node Development Platform Interfaces

2.1 Sensing Subsystem

The sensing subsystem can support an assortment of analog and digital sensing devices. As the Hogthrob project is still in its infancy, the requirements for monitoring the various parameters that can be associated with the phenomenon of the onset of estrus in sows might change. However, preliminary studies have indicated a direct correlation between the movement of a sow and the onset of estrus [10]. Therefore, at present, we are looking into the use of a MEMS-based, three-axis accelerometer with a digital output (possibly, the most recently-released, LIS3L02D, single-chip device from ST Microelectronics [24]). In future, we might have to use temperature and acoustic sensors which can have analog outputs. The analog outputs from these sensors can be processed by the 10-bit A/D converter available on the ATmega128L either directly or, if necessary, one analog input can be routed via an on-chip comparator for signal conditioning. The A/D converter supports 8 analog input channels. The ATmega128L also supports an I²C interface which is commonly available on most of the sensors.

2.2 Computing Subsystem

The computing subsystem is centered around the Atmel ATmega128L microcontroller running at a clock frequency of 8 MHz at 3.0V and the Xilinx Spartan3 series XC3S400 FPGA (please see Table 1) running at the clock frequencies of 48MHz and 4MHz at 2.5V for the peripherals and 1.2V for the core. The primary function of the computing subsystem is to execute the sow monitoring application and to

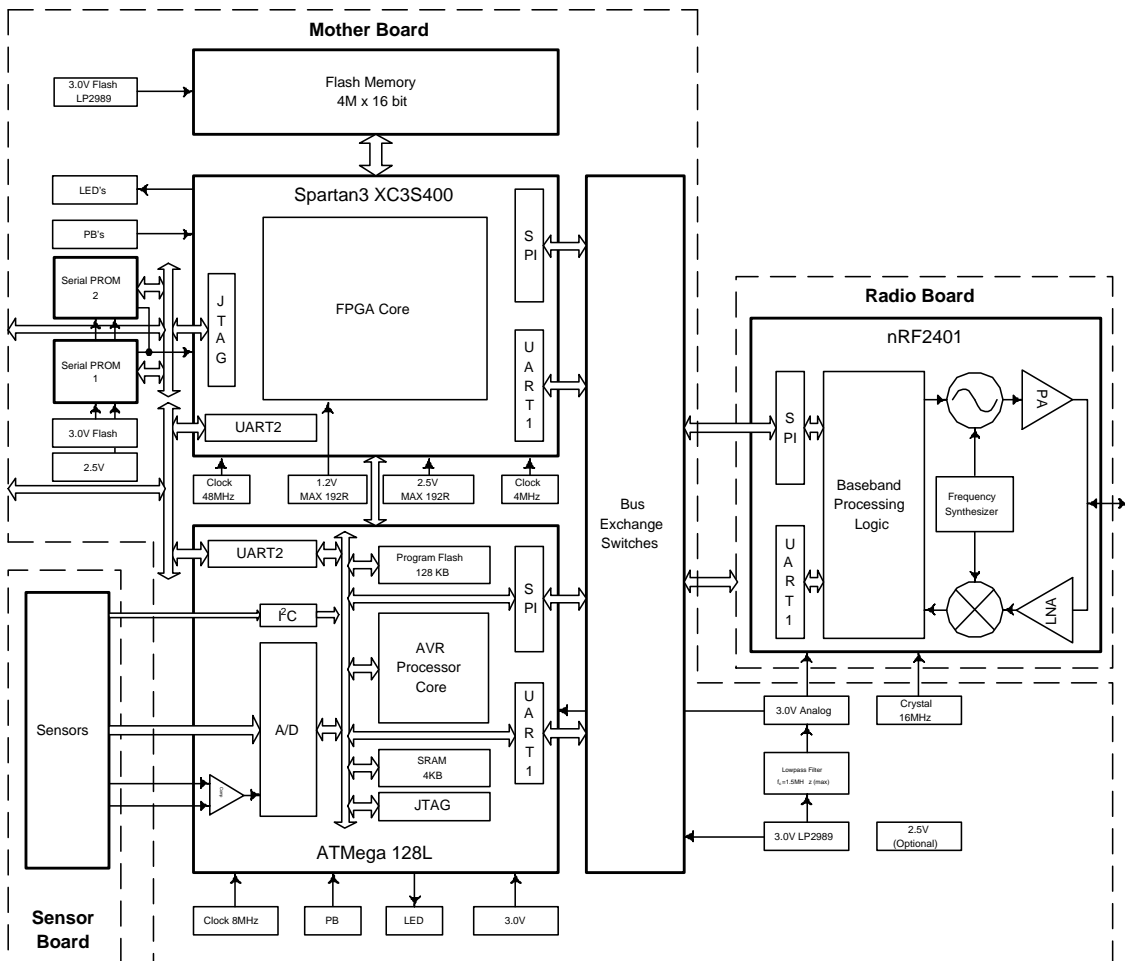


Figure 2. Hogthrob Sensor Node Development Platform Architecture

coordinate the functions of the sensor node. The operating system running on the sensor nodes forms the core of the software running on the computing subsystem which is responsible for the task scheduling operations and resource management. We are presently running the TinyOS which is an event-based embedded operating system developed at the University of California, Berkeley [27] (please see Section 4).

There are a number of interfaces supported by the AT-Mega128L which are also supported by the mother board (please refer to Figure 1). These interfaces include the two-wire (I²C) interface for the sensing subsystem, the three-wire (SPI) interface for the communication subsystem, the JTAG interface for in-system programmability and debugging, and the serial (RS-232) interface for interaction with the PC.

The Spartan3-series FPGA has been included to act either as a hardware accelerator for the ATmega128L or

it can be configured with a stand-alone microprocessor core working independently of the on-board ATmega128L. This will allow us to experiment with various microprocessor/microcontroller cores without redesigning the mother board.

The FPGA supports the same interfaces as the AT-Mega128L [18]. The access of either the ATmega128L or the FPGA to the radio transceiver is controlled by the bus exchange switches. The interface between the AT-Mega128L and the FPGA consists of parallel multiplexed address and data buses which can be demultiplexed by implementing an address latch in the FPGA and using the ALE (Address Latch Enable) signal. The FPGA uses an in-system programmable Flash memory (4M x 16 bits). There are two serial configuration PROM's provided with the FPGA which are controlled by the ATmega128L and the configuration data can be downloaded to them through the JTAG port. A number of LED's and push-buttons have

ATMega128L Resources		Spartan3 Resources		nRF2401 Characteristics	
EEPROM	4KB	Gates	400K	Carrier Freq.	2.4 GHz
RAM	4KB	CLB Size	32x28	Modulation	GFSK
I/O's	53	Slices	3,584	Data Rate	0-1 Mbps
8-Bit Timers	2	Logic Cells	8,064	Sensitivity	-90 dBm
16-Bit Timers	2	CLB FF's	7,168	Voltage	1.9-3.6V
10-Bit ADC Chan.	8	Dist. RAM	56K Bits	Current: TX	10.5 mA
SPI Interface	1	RAM Blocks	16	Current: RX	18 mA
I ² C Interface	1	Block RAM	288K Bits	Current: PD	1 μ A
JTAG Interface	1	Config. PROM	1.7M Bits	Channels	25
UART Interface	2	Max. I/O	264	Sec. Mod.	FH-SS

Table 1. Salient Features of the Hogthrob Platform

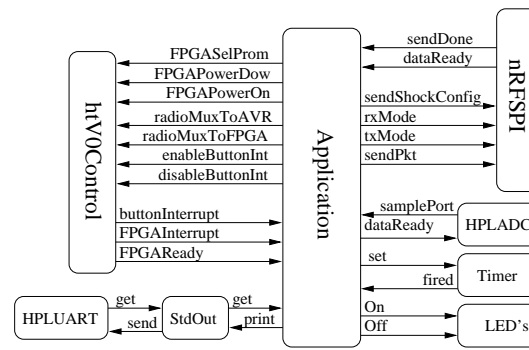


Figure 3. TinyOS Components

been provided at the mother board for easy test and debugging.

2.3 Communication Subsystem

The communication subsystem manages the data transfer and signaling (beaconing) between the sensor nodes. It includes the network protocols and the radio transceiver. The radio transceiver is a Nordic VLSI nRF2401 (please see Table 1) that can deliver a maximum data rate of 1 Mbps. It consists of a fully-integrated frequency synthesizer which can generate a frequency range of 2.4-2.5 GHz in the ISM² band, a power amplifier, a crystal oscillator (it uses a 16MHz external crystal), and a GFSK³ modulator. The output power and the frequency channels of the radio transceiver are fully programmable through the 3-wire (SPI) interface. The antenna used for radio transmission and reception is an omnidirectional stub antenna. The transmission range of the radio transceiver has been measured to be 80 meters under ideal conditions.

2.4 Power Supply Subsystem

The power-supply subsystem comprises 3 or 4 AAA-sized batteries and a collection of DC-DC converters to service the entire sensor node. The voltages generated by the DC-DC converters are: 3.0V, 3.0V-Flash, 2.5V, and 1.2V. The 3.0V supplied to the radio transceiver is filtered through a low-pass passive LC filter ($f_c=1.5\text{MHz}$) to generate 3.0V-Analog which is also used by the ATMega128L for its analog I/O. The two serial configuration PROM's for the FPGA use two supplies: 3.0V-Flash and 2.5V. An optional 2.5V is also reserved for the radio transceiver for future use.

²ISM stands for Industrial, Scientific, and Medical

³GFSK stands for Gaussian-filtered Frequency-Shift Keying

3 Sensor Node Software

The sensor node is limited in a number of ways, memory, computational power, etc. However, the most limited resource is the energy. The energy performance of a sensor node is greatly influenced by the software running on it. The sensor node control software (or the operating system) has to be designed to efficiently utilize the limited resources and, especially, the power-conserving features of the sensor node platform and to incur low computation and communication overhead. Furthermore, the application software has to take advantage of the spatial and temporal characteristics of the targeted application; it must *model* the application as realistically as possible. In the Hogthrob project, the focus of our application is the accurate and reliable detection of the estrus of sows.

As mentioned in Section 3, we are running TinyOS – an event-based embedded operating system, on the Hogthrob platform. The extremely modular and flexible design of TinyOS is very well-suited for exploring the boundary between hardware and software [21]. TinyOS is a programming environment rather than an operating system in the traditional sense and is closely tied to the nesC language which is an extension of the C-language [9]. The TinyOS programs comprise a number of *components* interconnected by *interfaces*. A component implements an interface, and can serve either as a software module or as a wrapper for a hardware block. All the components of a TinyOS program are compiled into a single static image which is uploaded to the target platform. Compiling a static image allows optimizations that are otherwise troublesome in a traditional operating system, such as whole program analysis, compile-time data race analysis and more detailed dead-code detection. TinyOS was originally developed for the Mica sensor node platform [14], and has been, successfully ported to the Hogthrob platform.

3.1 Porting TinyOS to the Hogthrob Platform

The core of TinyOS has no platform-dependent components [13], therefore, the process of porting TinyOS to the Hogthrob platform has been fairly straightforward. In addition, we have implemented two TinyOS components, `nRFSPI` and `htV0Control`, to access the radio transceiver, the FPGA, the bus-exchange switches, and the push-buttons (please see Figure 3):

nRFSPI: The Hogthrob platform software is a work in progress and is currently limited to simple connectionless unreliable communication. Although the nRF2401 radio transceiver implements major parts of the physical access protocol, it does not handle collisions and retransmissions. The nRFSPI component functions as a wrapper for this functionality and provides a *packet-level* interface to the *byte-level* SPI peripheral of the ATmega.

htV0Control: This component implements the methods and the events closely tied to the Hogthrob platform – selecting the PROM for the FPGA configuration, booting the FPGA, setting up the FPGA communication, setting the bus-exchange switches and the push-button interrupts.

As the configuration of the Hogthrob platform is quite similar to the Mica platform [14], we can use most of the other hardware components of TinyOS (e.g., the A/D Converter, UART's, LED's, Timer's) with minor modifications.

3.2 The Sow Monitoring Application

In collaboration with KVL⁴ (which is our partner university in the Hogthrob project), we are studying the behavior of a herd of sows before and during their estrus. Thus far, we have identified the following characteristic behavioral features which are relevant to the application software running on the sensor nodes.

1. During the night, the sows rest (sleep) for long periods (4-8 hours at a stretch)
2. The heat period occurs infrequently but regularly suggesting different *duty cycles* of operation – not in heat period (low sample rate); might be in heat period (high sample rate).

With these observations, it seems appropriate to power down a sensor node while a sow is sleeping and to duty-cycle the sensor nodes according to the sow activity. Our initial approach is to formulate a model (based on the Markov Model or the Finite State Automata) and associate a duty cycle with each state (e.g., sleeping - one sample per hour, active - one sample per minute, close to a boar - 4 samples per second).

⁴The Royal Veterinary and Agricultural University, Copenhagen, Denmark

As mentioned earlier, it has been shown that there is a correlation between the movement of a sow and the onset of its heat period [10]. We are now conducting experiments to get initial time series characterizing the movement of sows using accelerometers. Along with KVL, we will develop an application model based on these time series.

4 Hardware-Software Codesign of the Sensor Node

In order to explore various options available for designing the computing subsystem of our sensor node SoC, we have designed and implemented a custom microprocessor core on the FPGA. The design of low-power microprocessors for portable systems is an ongoing research subject [3, 4], however, our focus in this project is not to advance the field of low-power microprocessor design. Therefore, we have made our design decisions by selecting from relatively well-understood options.

As mentioned above, for our sensor network application, the microprocessor will spend most of its time in sleep mode – not actively executing instructions. Thus, our main design focus has to be on reducing startup times and providing the right low-power states and hardware accelerators and not on executing the program instructions efficiently.

Furthermore, a key issue is the availability of design automation tool chain. Having a suite of design automation tools available while developing an actual application is a must. As an example, consider the Freescale evaluation board [8] based on the Motorola HCS08 microcontroller, which is not supported by the GNU GCC compiler. Porting TinyOS to this platform not only involves rewriting the hardware drivers, but also the tool chain differences. These considerations have led us to implement an AVR-instruction set compatible processor, which we call Nimbus after a classic Danish motorcycle.

4.1 Nimbus Processor Core

The Nimbus processor core (please see Figure 4) implements a subset of the Atmel ATmega103 functionality with no hardware multiply unit. It is based on the AVR processor core from OpenCores⁵ which has been debugged and modified to support different power modes. The serial flash is used for program storage. However, for simplicity, the programs are stored in the SRAM memory blocks of the FPGA while testing.

To get a feel of how the Nimbus processor core performs, we have simulated the Nimbus core using the Synopsys Power Compiler and compared it with the Atmel ATmega128L.

⁵www.opencores.org

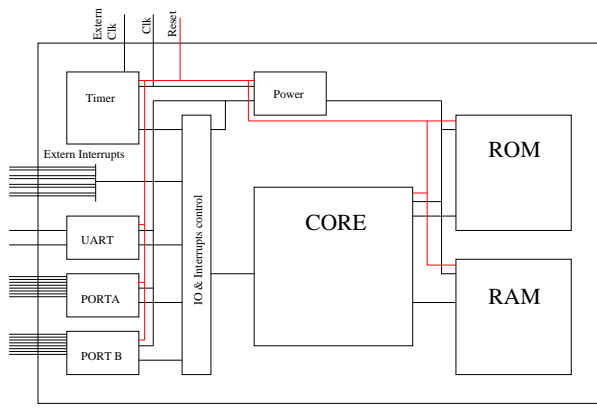


Figure 4. Nimbus Microprocessor Core

For comparing the two processor cores, we have written a few benchmarks and have run them in the simulated environment of the Nimbus processor core and on the Atmel AT-Mega128L on a BTNode2 [15]. Each of these benchmarks exercise different features of each processor core. To allow us to measure just the power consumption of the processor core and no additional component, we have slightly modified⁶ the BTNode2 board. The Atmel AT-Mega128L, on the BTNode2, runs at 7.35 MHz and is powered with a 3.3 V power supply while the Nimbus processor core is simulated to be running at 7.0 MHz with a 1.32 V power supply.

In order to explore and compare the architectural advantages of the Nimbus processor core over the Atmel AT-Mega128L, the two technologies have to be the same. However, the actual technology of the Atmel AT-Mega128L is not published. Therefore, we have made an assumption that it is manufactured using a 0.25 micron technology.

We summarize the comparison results in Table 2. Although the simulated operating voltage of the Nimbus core is lower than that of the Atmel AT-Mega128L, but even with this simple comparison, with this relatively simple implementation of the Nimbus processor core, it is evident that lowering the power consumption of the microprocessor in the computing subsystem is possible.

The results are summarized in Table 2. It is clear from the results that, using this comparison, the Nimbus core outperforms the Atmel AT-Mega128L. While this is promising, our conclusion might be biased given the fact that we are using a lower operating voltage for the Nimbus core and if our assumption that the Atmel AT-Mega128L is manufactured using a 0.25 micron technology is wrong. The numbers are, obviously, not comparable in that case.

⁶on the BTNode2 board, there is a 0 Ohm resistor that can easily be replaced by two wires, allowing the use of an ammeter

Benchmark	Nimbus	ATMega	Description
nop	2.26 mW	47.5 mW	tight loop of no operation instr.
idle	1.00 μ W	17.0 mW	idle mode of the ATMega
power-save	1.22 μ W	38.6 μ W	power-save mode of the ATMega
power-down	0.59 μ W	39.0 μ W	power-down mode of the ATMega
add	1.38 mW	30.1 mW	tight loop of add instr. stored in registers
add-mem	1.90 mW	31.9 mW	tight loop of add instr. stored in memory
hamming	1.76 mW	32.3 mW	Hamming encoding and decoding

Table 2. Comparison of the Nimbus Microprocessor Core with the AT-Mega128L Microprocessor

5 Work in Progress

To further explore the HW/SW codesign options and to enable platform tuning [7], we are busy capturing the SNAP [5] and BitSNAP [28] processor designs in GEZEL [11]. We are also exploring various sensor node platform modeling approaches in Java [1, 2, 19], GME [12], and SystemC [25].

The prototype sensor node is presently undergoing field trials and, once successful, we plan to build a second prototype which will be more compact and more customized. We will also compare the synchronous AVR core with its asynchronous counterpart. The ultimate goal is to shrink the sensor node to a single system-on-chip costing less than a couple of Euros. We are also planning to exploit a number of energy scavenging solutions being made available by the ongoing research in the fields of micro and nano technology [23, 17].

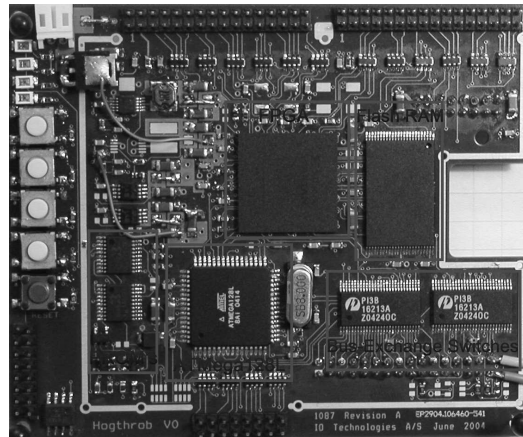


Figure 5. Motherboard of the Hogthrob Sensor Node Hardware

Acknowledgements

The work presented in this paper has been funded by the Danish National Research Council (STVF) project titled "Hogthrob – Networked On-a-Chip Nodes for Sow Monitoring" (STVF 2059-03-0027).

References

- [1] Ben Titzer and Daniel K. Lee and Jens Palsberg. Avrora: Scalable Sensor Network Simulation with Precise Timing. In *Proceedings of Fourth International Conference on Information Processing in Sensor Networks, Joint IPSN 2005 Main Track and Special Track on Platform Tools and Design Methods for Networked Embedded Sensors (SPOTS), Sunset Village, UCLA, Los Angeles, USA, April 25-27, 2005*.
- [2] Ben Titzer and Jens Palsberg. Nonintrusive Precision Instrumentation of Microcontroller Software. In *Proceedings of Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES 2005), Chicago, Illinois, USA, June 15-17, 2005*.
- [3] T. D. Burd and R. W. Brodersen. Energy-Efficient CMOS Microprocessor Design. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS'95), 1995*.
- [4] T. D. Burd and R. W. Brodersen. Processor Design for Portable Systems. *Journal of VLSI Signal Processing Systems - Special issue on Technologies for Wireless Computing*, 13:203–221, 1996.
- [5] Clinton Kelly IV and Virantha N. Ekanayake and Rajit Manohar. SNAP: A Sensor-Network Asynchronous Processor. In *Proceedings of 9th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2003), Vancouver, Canada, pages 24–35. IEEE Computer Society, 12-16 May 2003*.
- [6] Enz, C.C.; El-Hoiydi, A.; Decotignie, J.-D.; Peiris, V. WiseNET: An Ultra Low-Power Wireless Sensor Network Solution. *IEEE Computer*, 37(8):62–70, August, 2004.
- [7] Frank Vahid and Tony Givargis. Platform Tuning for Embedded Systems Design. *IEEE Computer*, 34(2):112–114, 2001.
- [8] Freescale Semiconductor, Inc. MC13192 Evaluation Board Reference Manual, Revision 1.0. <http://www.freescale.com>, September 2004.
- [9] D. Gay, P. Levis, R. von Behren, M. Welsh, E. Brewer, and D. Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. In *Proceedings of Programming Language Design and Implementation (PLDI)*, June 2003.
- [10] R. Geers, S. Janssens, J. Spoorenberg, V. Goedseels, J. Noordhuizen, H. Ville, and J. Jourquin. Automated Oesterus Detection of Sows With Sensors for Body Temperature and Physical Activity. In *Proceedings of ARBIP95, Kobe, Japan, 1995*.
- [11] GEZEL. <http://www.ucla.edu/schaum/gezel>.
- [12] GME. <http://www.isis.vanderbilt.edu/Projects/gme>.
- [13] V. Handziski, J. Polastre, J.-H. Hauer, C. Sharp, A. Wolisz, and D. Culler. Flexible Hardware Abstraction for Wireless Sensor Networks. In *Proceedings of the Second European Workshop on Wireless Sensor Networks (EWSN 2005)*, 31 January - 2 February 2005.
- [14] J. L. Hill and D. E. Culler. MICA: A Wireless Platform for Deeply Embedded Networks. *IEEE Micro*, 22(6):12–24, November 2002.
- [15] J. Beutel and O. Kasten and M. Ringwald. Prototyping Wireless Sensor Networks with BTNodes. In *Proceedings of First European Workshop on Wireless Sensor Networks (EWSN 2004)*, January 2004.
- [16] P. Juang, H. Oki, Y. Wang, M. Martonosi, L. S. Peh, and D. Rubenstein. Energy-Efficient Computing for Wildlife Tracking: Design Tradeoffs and Early Experiences with ZebraNet. In *ASPLOS X*, pages 96–107. ACM Press, 2002.
- [17] A. Kansal and M. B. Srivastava. An Environmental Energy Harvesting Framework for Sensor Networks. In *Proceedings of the 2003 International Symposium on Low-Power Electronics and Design*, pages 481–486, 2003.
- [18] Kashif Virk. Testing FPGA Interfaces on Hogthrob Motherboard. Technical Report, Computer Science & Engineering, Informatics & Mathematical Modeling, Technical University of Denmark, December 2004.
- [19] O. Landsiedel, K. Wehrle, B. L. Titzer, and J. Palsberg. Enabling detailed modeling and analysis of sensor networks. *Praxis der Informationsverarbeitung und Kommunikation (PIK Journal) - Special Issue on Sensor Networks*, 2005. In press.
- [20] M. Leopold, M. Dydensborg, and P. Bonnet. Bluetooth and Sensor Networks: A Reality Check. In *Proceedings of the First International Conference on Embedded Networked Sensor Systems*, pages 103–113, November 2003.
- [21] P. Levis, N. Lee, M. Welsh, and D. Culler. TOSSIM: Accurate and Scalable Simulation of Entire TinyOS Applications. In *SenSys'03*, November 2003.
- [22] A. Mainwaring, J. Polastre, R. Szewczyk, D. Culler, and J. Anderson. Wireless Sensor Networks for Habitat Monitoring. In *WSNA'02*, pages 88–97. ACM Press, September 2002.
- [23] S. Roundy, P. K. Wright, and J. Rabaey. A Study of Low-Level Vibrations as a Power Source for Wireless Sensor Nodes. *Computer Communications*, 26:1131–1144, 2003.
- [24] ST Microelectronics, Geneva. Micro Electro-Mechanical Systems. <http://www.st.com/mems>.
- [25] SystemC Workgroup. <http://www.systemc.org>.
- [26] The Hogthrob Project. <http://www.hogthrob.dk>.
- [27] University of California, Berkeley. TinyOS Community Forum. <http://www.tinyos.net>.
- [28] Virantha N. Ekanayake and Clinton Kelly IV and Rajit Manohar. BitSNAP: Dynamic Significance Compression for a Low-Energy Sensor Network Asynchronous Processor. In *Proceedings of 11th International Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC 2005), New York, USA, pages 144–154. IEEE Computer Society, 14-16 March 2005*.