Heterogeneous Wireless Sensor Nodes that Target the Internet of Things

This article summarizes the architectural design decisions of the Hardware-Accelerated Low-Power Mote (HaLoMote), which combines a field-programmable gate array for energy-efficient data aggregation with a radio system on chip for network management. The authors show HaLoMote's power consumption improvements over typical software processors of wireless sensor networks for two use cases based on general data compression and application-specific feature extraction.

Andreas Engel
Andreas Koch
Technische Universität Darmstadt

The growth of the Internet of Things (IoT) has led to a unification of embedded systems' and systems-of-systems' design spaces, with a focus on cyber-physical systems communicating with each other. While some IoT devices such as mobile systems absolutely require wireless communication, radio transceivers are sometimes employed purely for convenience (for example, to simplify installation or retrofitting), even in stationary settings such as smart homes or industrial process monitoring. In all of these untethered-usage scenarios, decentralized data aggregation in the wireless sensor networks (WSNs) is crucial to keeping the amount of wirelessly transferred data and thus the energy consumed by the transceiver within the limited budget of the devices, which are typically powered by batteries or energy harvesters. Challenges on the embedded processors in the IoT domain are due not only to the ever-growing amount of data to be handled, but are also due to the increasing importance of security and privacy. These force application designers either to ensure that no private information leaves the smart device (that is, all the data processing must be performed locally at the sensors) or to use strong encryption and authentication methods to protect the gathered data. In both cases, the demand for energy-efficient embedded processing capabilities increases.

To address the challenges of computationally intensive distributed applications with limited energy budgets, we proposed the heterogeneous Hardware-Accelerated Low-Power Mote (HaLoMote) as a more energy-efficient approach to the common homogeneous node architectures.[1] Prior attempts at using reconfigurable computing in the WSN domain have seen only limited success (see the "Related Work in Hardware-Accelerated

Published by the IEEE Computer Society

Related Work in Hardware-Accelerated Wireless Sensor Nodes

Most wireless sensor network (WSN) motes are homogeneous architectures based on software processors ranging from small microcontrollers (MCUs) to large digital signal processors (DSPs). This article focuses on hardware-accelerated processing; for a more comprehensive overview of software-processor-based WSN motes, see previous work by Antonio de la Piedra and colleagues.[1]

Field-programmable gate array (FPGA)-based reconfigurable computing units (RCUs) can perform complex computations more efficiently than MCUs and DSPs while providing more flexibility for an iterative design process than application-specific integrated circuits. In real-time applications, the use of RCUs often enables computations that cannot be performed by MCUs or DSPs at all under the given constraints. This makes RCUs attractive for use in sensor nodes performing computationally intensive applications (such as video and image compression).[2,3] However, none of these systems could achieve truly low-power operation: they all relied on FPGAs using static RAM (SRAM) for configuration storage, which thus could not be powered down completely without losing the configuration data itself.

When energy actually becomes a first-order design goal, Flash-based FPGAs are far more suitable for the RCU.[4] Sensor nodes using a combination of a Flash-based Microsemi IGLOO FPGA and a wireless transceiver have already been proposed.[5–7] However, despite the power advantages of Flash configuration storage, these architectures also turn out to be suboptimal: all processing is performed on the RCU (even long-term, low-intensity tasks), and when powered down, the radio transceiver is required to wake up the FPGA again. Thus, at least one of the two power-hungry devices must be enabled all the time.

A better choice is a heterogeneous architecture that combines an RCU and a low-power MCU. The Cookie WSN[8] and the PowWow Mote[9] have joined a small Microsemi IGLOO FPGA with a TI MSP430 MCU and an additional radio transceiver. However, both systems use the FPGA only for low-level handling of radio messages, instead of preprocessing the sensor datastream. Furthermore, the use of discrete MCU and RF components carries the burden of slower communication as well as more complex power management. In the "HaLoMote Hardware Architecture" section in the main article, we examine the usage of a radio frequency system on chip next to a Flash-based FPGA in greater detail.

References

1. A. de la Piedra, A. Braeken, and A. Touhafi, "Sensor Systems Based on FPGAs and Their Applications: A Survey," *Sensors*, vol. 12, no. 9, 2012, pp. 12,235–12,264.
2. C. Zhiyong et al., "A Novel FPGA-Based Wireless Vision Sensor Node," *Proc. IEEE Int'l Conf. Automation and Logistics*, 2009; doi:10.1109/ICAL.2009.5262805.
3. K. Goh et al., "FPGA Based Wireless Sensor Node for Distributed Process Monitoring," *Proc. 7th IEEE Conf. Industrial Electronics and Applications*, 2012; doi:10.1109/ICIEA.2012.6361045.
4. T. Nyländen et al., "FPGA Based Application Specific Processing for Sensor Nodes," *Proc. Int'l Conf. Embedded Computer Systems*, 2011; doi:10.1109/SAMOS.2011.6045452.
5. L. Vera-Salas et al., "Reconfigurable Node Processing Unit for a Low-Power Wireless Sensor Network," *Proc. Int'l Conf. Reconfigurable Computing and FPGAs*, 2010; doi:10.1109/ReConFig.2010.48.
6. B. Stelte, "Toward Development of High Secure Sensor Network Nodes Using an FPGA-Based Architecture," *Proc. 6th Int'l Wireless Communications and Mobile Computing Conf.*, 2010, pp. 539–543.
7. T. Nyländen et al., "Reconfigurable Miniature Sensor Nodes for Condition Monitoring," *Proc. Int'l Conf. Embedded Computer Systems*, 2012; doi:10.1109/SAMOS.2012.6404164.
8. V. Rosello, J. Portilla, and T. Riesgo, "Ultra Low Power FPGA-Based Architecture for Wake-up Radio in Wireless Sensor Networks," *Proc. 37th Ann. Conf. IEEE Industrial Electronics Soc.*, 2011; doi:10.1109/IECON.2011.6119933.
9. O. Berder and O. Sentieys, "PowWow: Power Optimized Hardware/Software Framework for Wireless Motes," *Proc. 23rd Int'l Conf. Architecture of Computing Systems*, 2010, pp. 1–5.

Wireless Sensor Nodes" sidebar). In this article, we summarize the architectural design concepts and power management mechanisms of HaLoMote and its application in two IoT-related domains. This article expands on prior reports with details on HaLoMote's interprocessor communication, basic power management, and performance in different monitoring applications[2,3] by justifying architectural design decisions and detailing more advanced power management strategies.

HaLoMote Hardware Architecture

In this section, we detail the main design choices considered for the heterogeneous sensor node, its basic architecture, and the latest implementation.

Heterogeneous Processing and Communication

To integrate a reconfigurable computing unit (RCU) into a WSN mote, the design options for the computation and communication architecture (shown in Figure 1a to Figure 1d) must be traded off against each other.

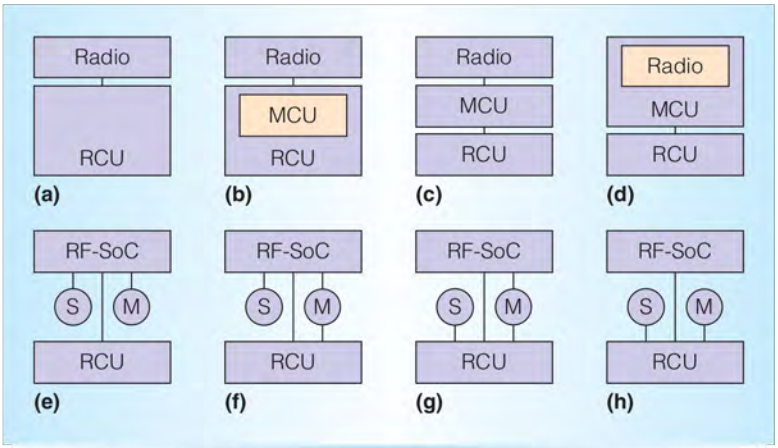

Figure 1. Design options for (a–d) the processing and communication architecture and (e–h) the attachment of sensors (S) and memories (M) to the computing units. (MCU: microcontroller; RCU: reconfigurable computing unit; RF-SoC: radio frequency system on chip.)

Because an RCU typically does not provide wireless communication capabilities, at least a radio transceiver must be attached. Such a transceiver implements the lower communication layers (that is, symbol modulation and medium access control) and provides a limited message buffer that can be accessed by a digital interface. All other radio protocol tasks (such as routing and transport control) must thus be handled by the RCU, if it is directly attached to the transceiver (Figure 1a). Because these are typically rather simple and sequential control-flow-dominated algorithms, waking up the RCU just to handle the radio protocol would not be energy efficient. For the same reason, we should avoid handling the radio stack on a soft-core microcontroller (MCU) inside the RCU, as shown in Figure 1b.

When adding a dedicated MCU (Figure 1c) for all low-priority tasks, such as the radio protocol, basic timekeeping, and the top-level control flow of the application (for example, periodic sensor sampling), the platform can exploit its heterogeneity by selectively shutting down temporarily unused computation and communication units. This basic dynamic power management (DPM) principle also applies when the radio transceiver and the low-power MCU are integrated into a single device (Figure 1d), because these radio frequency systems on chip (RF-SoCs) let us suspend the radio and the MCU separately. Compared to Figure 1c, more of the limited number of the MCU's general-purpose I/O (GPIO) pins can be dedicated for interprocessor communication, because they are not required for communication between the MCU and the transceiver. The combination of an RCU and an RF-SoC thus improves the data throughput between hardware accelerator and wireless transceiver. Therefore, HaLoMote is based on the architecture shown in Figure 1d.

Integration of Peripherals

A second fundamental architectural design choice deals with the integration of sensors and memories, often required to temporarily buffer raw or aggregated sensor data. Attaching both peripherals to the RF-SoC (Figure 1e) lets us collect the number of samples required for the data aggregation algorithm without ever waking up the hardware accelerator. However, the entire stream of sensor data must be transferred over the critical link from the RF-SoC to the RCU to be aggregated. The additional GPIO pins required by the RF-SoC to interface the sensors and the memory further reduces the data throughput between both processing units. The transfer of the raw datastream thus becomes the architecture's bottleneck.

We can mitigate this bottleneck using a shared memory for interprocessor communication (Figures 1f and 1g), either by using a dual-port memory or by synchronizing the memory access of both processors via the remaining direct connections between RCU and RF-SoC. Although a dual-port memory is typically more expensive than a single-port memory (in terms of chip cost and energy consumption), it enables more parallel operations on the heterogeneous platform, such as transmitting one block of the aggregated data while generating the next block. The shared memory approach is most valuable if the sensors are attached to the RF-SoC (Figure 1f) and the data aggregation can be performed on larger blocks of the raw sample data, so the RCU can remain sleeping until the next block is collected.

However, the sensors might need to be attached to the RCU (Figures 1g and 1h) if the RF-SoC does not have enough GPIO pins to control all peripherals (for example, if

the sensors cannot be daisy-chained or connected to a common bus). Furthermore, many event-detection applications require immediate processing of the sensor data to minimize the detection delay. In these cases, only the aggregated datastream must pass the bottleneck from the RCU to the RF-SoC. A dedicated shared memory between both computational units is thus not required, and the architecture of Figure 1g is not useful in those cases.

From the remaining two reasonable architectures (Figures 1f and 1h), we chose the latter as HaLoMote's foundation. With all peripherals being interfaced by the RCU, HaLoMote supports a broad range of applications with different sensor and memory requirements. This flexibility is essential for an exploration of the broad design space of IoT applications.

Implementation of HaLoMote

The design decisions discussed thus far led to the prototype implementation of the HaLoMote architecture shown in Figure 2. The RCU is realized as a discrete field-programmable gate array (FPGA; specifically, Microsemi IGLOO AGL1000) based on nonvolatile memory, allowing for deep sleep modes with fast shutdown and wakeup times, as well as a very low static power draw. External sensors and additional memories connect to the RCU to support the efficient preprocessing of the sampled datastream. Only the aggregated results are transferred to the RF-SoC for transmission into the network. The latter is realized by the Atmel ATmega256RFR2 device.

The power- and area-consuming human-machine interface peripherals are not located on the main printed circuit board, but can be attached for debugging purposes as needed. Most monitoring applications require a significant amount of external memory. We chose four 1-Mbit serial SRAMs instead of a single parallel memory to enable parallel independently addressable memory accesses by the RCU. Furthermore, one or more SRAMs can thus be selectively replaced with pin-compatible nonvolatile ferroelectric RAMs (FRAMs) for even more aggressive power management. By directly integrating the memory on the mainboard, we can use

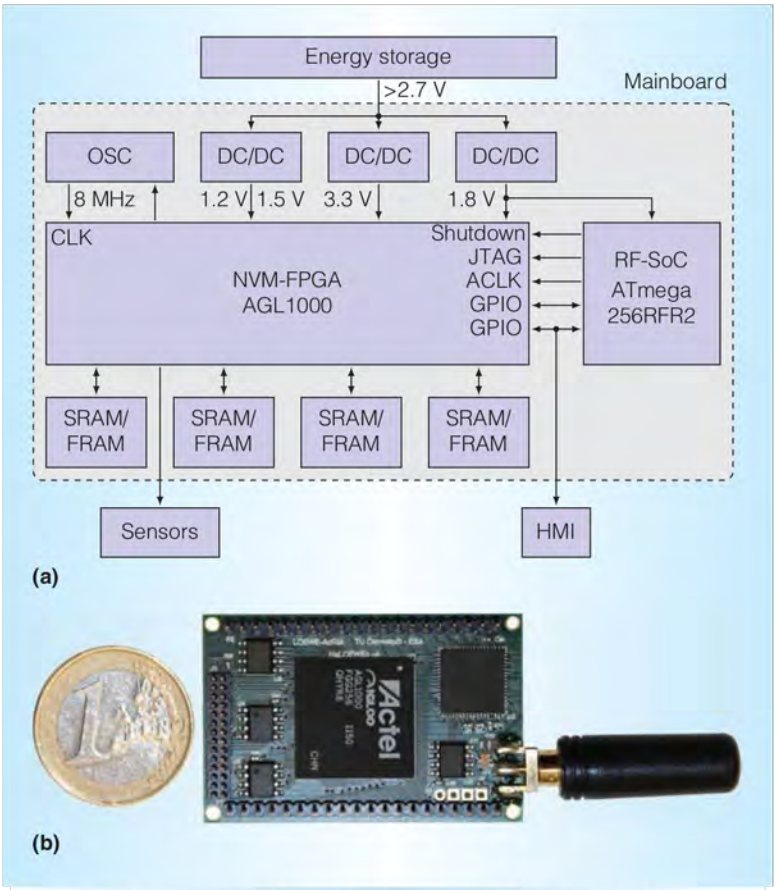

Figure 2. Implementation of Hardware-Accelerated Low-Power Mote (HaLoMote): (a) schematic and (b) 46 mm × 30 mm printed circuit board. (ACLK: auxiliary clock; FRAM: ferroelectric RAM; GPIO: general-purpose I/O; HMI: human-machine interface; JTAG: Joint Test Action Group; NVM-FPGA: nonvolatile memory field-programmable gate array; OSC: oscillator; SRAM: static RAM.)

the remaining 40 pins of the expansion headers to attach application-specific sensors.

Finally, by exposing the FPGA Joint Test Action Group interface to the MCU, HaLoMote supports over-the-air reconfiguration. However, this comes at the cost of an additional 3.3-V regulator responsible for providing the higher programming voltage.

Active Power Management

Because both the RCU and MCU devices support DPM, a proper power-management scheme should be straightforward: The MCU is woken up periodically at the beginning of each sampling interval to handle system management tasks. It retrieves previously computed data from the RCU, offloads new

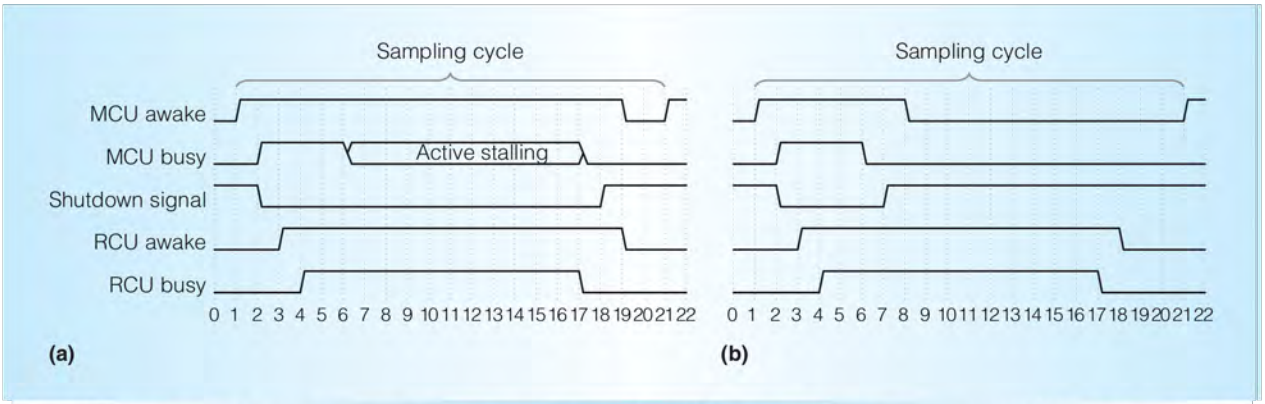

Figure 3. Scheduling of dynamic power management between the MCU and the RCU: (a) stalling the MCU and (b) delayed Flash-Freeze.

computations (if any) to the RCU, and then goes to sleep itself. The RCU is powered up only if it actually has work to do.

The Microsemi IGLOO Flash-Freeze mode preserves the hardware configuration and the content of the on-chip state while reducing the FPGA's power draw to just 53 µW. This sleep mode can be entered and exited within a few microseconds of transition time, which is negligible even when sampling at several hundreds of hertz. The MCU signals the RCU to enter (or exit) the Flash-Freeze mode by asserting (or deasserting) the shutdown signal (see Figure 2a). However, the interdependency between both of the heterogeneous architecture's computing units introduces difficulties for the overall DPM strategy.

Flash-Freeze Control

Figure 3a shows a sample DPM scheduling for a single sampling cycle between time steps 1 and 21. The MCU wakes up at Time 1 and deasserts the shutdown signal at Time 2. After it finishes its low-level management tasks at Time 6, the MCU must wait until the RCU also finishes its computations at Time 17 before asserting the shutdown at Time 18 and falling to sleep at Time 19. Actively stalling the MCU just to wait until the RCU can be shut down is not energy efficient. Although the MCU could fall asleep earlier and use a GPIO interrupt to wake up as soon as the RCU has completed, the additional state transitions of the MCU would increase the mote's energy consumption.

To decouple the shutdown of the RCU from the shutdown of the MCU, a DPM controller inside the RCU delays the actual shutdown request until the RCU is finished. This is achieved by a dedicated control bit inside the IGLOO fabric. Figure 3b shows the same sample schedule as before, now exploiting the improved Flash-Freeze control. The MCU asserts the shutdown signal at Time 7 and falls asleep at Time 8 immediately after finishing its tasks. The RCU, however, can keep running and shuts itself down at Time 18.

The decoupling of the DPM mechanisms of both processors thus simplifies the overall implementation and reduces the time the MCU must be kept awake. We can further reduce the amount of time the RCU spends awake by driving it with an MCU-generated auxiliary clock (such as the serial clock of an SPI module) while the main oscillator is ramped up.[1] The startup of a crystal oscillator can take up to tens of microseconds and is not negligible for intersampling cycle power management for sampling frequencies of several hundred hertz.

Preserving Live Variables in External Memory

Although the IGLOO FPGA can enter and exit the Flash-Freeze mode quickly, the 53 µW static power consumption within this mode is at least an order of magnitude larger than the power drawn by typical sleeping WSN software processors. To reduce the FPGA's power consumption further, its voltage supply must be shut down completely by an appropriate

Table 1. Power consumption and streaming throughput of different 1-Mbit memories operated at 2 V.

	Memory type		
Parameter	**23A1024**	**SST25WF010**	**FM25V10**
Type	SRAM	Flash	Ferroelectric RAM
T_w	20 Mbits/s	0.26 Mbits/s	25 Mbits/s
T_r	20 Mbits/s	20 Mbits/s	25 Mbits/s
P_w	2 mW	12 mW	3 mW
P_r	2 mW	4 mW	3 mW
P_s	2 µW	0 µW	0 µW
t/n	63 µs/bit	3,080 µs/bit	50 µs/bit

power field-effect transistor controlled by the MCU (not shown in Figure 2).

All runtime data held in the volatile (SRAM-based) registers or block RAMs (BRAMs) is lost after each power cycle. Thus, before shutting down the FPGA, any data that will be required again after the power cycle (that is, the live variables) must be preserved in an external memory, from which it can be restored after the FPGA is repowered.

This data transfer requires additional energy for the memory read and write operations in addition to the extra time the FPGA must be kept awake. Thus, the sleep period with reduced power consumption must be sufficiently long to recoup the transfer overhead. To estimate the minimum sleep time per swapped bit, let P_r, P_w, and P_s denote the power drawn by the memory during reading, writing, and shutdown. Furthermore, let T_r and T_w be the achievable throughput for streaming reads and writes, as well as P_a and P_f be the power drawn by the FPGA in active and Flash-Freeze modes. The swapping of n-bit live variables for a sleep period t pays off, if

$$t \cdot P_f > \frac{n}{T_w}(P_a + P_w) + \frac{n}{T_r}(P_a + P_r) + \left(t - \frac{n}{T_w} - \frac{n}{T_r}\right)P_s.$$

Table 1 lists the power draw and throughput specification of three types of memory. Out of all devices that can be mounted on the current HaLoMote implementation (that is, serial memory in an 8-pin small outline integrated circuit package), we selected the most frugal chips with a capacity of 1 Mbit. The relative break-even time t/n depends on the power drawn by the FPGA and is thus application specific. However, assuming $P_a =$ 30 mW for a worst-case analysis has proven reasonable for typical applications. Thus, t/n is the minimum relative sleep time (per bit) that actually amortizes the state save and restore overhead.

With its low write throughput, Flash memory is not suitable for live variable preservation. Preservation in SRAM pays off after 63 µs per bit. It is outperformed by using FRAM, having a delay of just 50 µs/bit. Thus, assuming 100 bytes of live variables, sleep times longer than 40 ms would profit from preservation in FRAM. Note that, even when using FRAM, such state preservation is generally not feasible for applications requiring continuous acquisition and processing with large live-variable sets.

HaLoMote Applications

Lossless data compression is the most generic form of in-sensor preprocessing. RCUs are well-suited for this kind of data aggregation, because they can process multiple independent sensor channels in parallel, and the applied compression algorithm can be adapted to the characteristics of the data source. In previous work,[2] we used a forward-adaptive differential pulse code modulation with a Rice symbol coder to compress vibration data from three microelectromechanical systems sensors to 79 percent of its original size. While HaLoMote's energy-efficient hardware-accelerated encoder reduced the overall energy required for

Table 2. Execution time and energy consumption per sample for structural health monitoring feature extraction on different processing platforms.

	Processing unit					
Parameter	**TI CC2530**	**ATmega256RFR2**	**TI CC430**	**STM32F407**	**TI CC2650**	**AGL1000**
Architecture	8-bit MCU	8-bit MCU	16-bit MCU	32-bit DSP	32-bit MCU	Flash FPGA
Instruction set	8,051	AVR	MSP430	Cortex-M4F	Cortex-M3	Custom
Compiler	SDCC 3.4	AVRGCC 4.3.1	CL430 4.4.3	ARMGCC 4.9.3	ARMGCC 4.9.3	Synplify 2014
Clock speed (MHz)	32	16	20	128	48	8
V_{CC} (V)	2.0	1.8	2.4	1.8	1.8	1.2
Sleep mode	LPM2	POWER-SAVE	LPM3	STOP	STANDBY	Flash-Freeze
t_{active} (µs)	8,056	2,712	1,249	43	149	9
t_{wakeup} (µs)	100	34	150	110	151	1
t_{idle} (µs)	N/A	N/A	1,101	2,347	2,200	2,490
P_{active} (mW)	13	6.7	11	72	5.2	30
P_{idle} (µW)	2	2.7	12.7	156	1.8	53
E_{cycle} (nJ)	N/A	N/A	15,403	11,382	1,564	432

compression and wireless transmission to 81 percent of sending uncompressed data, a comparable software encoder for the TI CC2530 8-bit MCU would have increased the system's overall energy consumption to 134 percent due to the lengthy sequential encoding.

Application-specific feature extraction provides even more potential for data aggregation than lossless data compression. In vibration-based structural health monitoring (SHM), the modal parameters of large infrastructure objects (such as bridges) are derived from cross-correlation functions describing the temporal and spatial relations between different vibration sensors distributed over the structure. The distributed estimation of those cross-correlation functions is a useful data aggregation mechanism, because it significantly reduces the required wireless throughput while relying primarily on simple accumulations of values. However, many of these accumulations must be performed, which can be done in parallel on the RCU.

In previous work,[3] we described a HaLoMote implementation of this SHM-specific preprocessing. Four acceleration sensors are sampled at 400 Hz (as required by the SHM application) and are high-pass filtered in parallel. Eight cross-correlation functions must be computed from the filtered sensor channels on every sensor node. We measured the overall execution time per sampling cycle on different processing platforms, as shown in Table 2. The software processors we used as reference for the HaLoMote platform range from small 8-bit MCUs up to a powerful 32-bit DSP to cover the architectures typically used in WSN and IoT applications. The devices were configured to run at the highest clock speed for the lowest applicable supply voltage, and the software compilers were configured to optimize for execution speed. Most processors execute fixed-point implementations of the SHM algorithm, whereas the STM32F407 executes a floating-point implementation with support of its hardware floating-point unit (FPU). We took the power consumption in both active mode and the deepest appropriate sleep mode (that is, with enabled wake-up clock; see Table 2), as well as the transition time between both states (that is, wakeup), from the software processors' datasheets, whereas we actually measured the FPGA's power consumption. The overall energy consumption per 2.5-ms sampling cycle was derived as

$$\begin{aligned} t_{idle} &= 2.5\ \text{ms} - t_{active} - t_{wakeup} \\ E_{cycle} &= (t_{active} + t_{wakeup}) \cdot P_{active} + t_{idle} \cdot P_{idle}, \end{aligned}$$

thus assuming that ramping up the internal regulators during wakeup is as costly as the regular active mode.

The two 8-bit MCUs could not achieve the required sampling frequency, so Table 2 does not provide t_{idle} and E_{cycle} values for these devices. High-performance DSPs such as the TI C6747 are as fast as the IGLOO FPGA (that is, 9 µs per cycle), but do not provide sufficiently deep sleep states (that is, 60 mW for the C6747 in static mode) and thus are not listed in Table 2. The IGLOO FPGA itself consumed only 28 percent of the energy spent by the best software processor, the TI CC2650. However, for HaLoMote, we also have to take into account the additional overhead of the ATmega256RFR2 MCU. Because of the improved Flash-Freeze control, the MCU spends about 98 percent of each sampling cycle in idle mode, thus causing an overhead of just 342 nJ. In total, the heterogeneous HaLoMote architecture still outperforms the best homogeneous software processor by a factor of two in terms of overall energy consumption.

Note that these figures do not include wireless data transmission. Another 94 mJ are required to transmit the eight correlation functions (with 1 Kbyte each) if a channel utilization of 10 percent can be achieved after packet loss and communication overhead. A 60-second measurement thus requires 113 mJ on HaLoMote for computation and transmission. A comparable WSN-based modal identification system without decentralized data aggregation requires more than 3,000 mJ on that 60-second measurement.[4] The energy efficiency gain of more than 26 times thus justifies the use of a more complex heterogeneous architecture on HaLoMote.

We evaluated the heterogeneous HaLoMote architecture in distributed applications from the two IoT domains of industrial condition monitoring and structural health monitoring. For these cases, we observed energy efficiency improvements of 2 and 26 times, respectively, over recent low-power software processors and prior works. For highly energy-critical use cases (such as long harvester-powered operation), these gains can offset the more complex node architecture. Further improvements currently under investigation include the use of a custom-designed coarse-grained reconfigurable RCU instead of the FPGA. MICRO

References

1. A. Engel, B. Liebig, and A. Koch, "Energy-Efficient Heterogeneous Reconfigurable Sensor Node for Distributed Structural Health Monitoring," *Proc. Conf. Design and Architectures for Signal and Image Processing*, 2012, pp. 43–50.
2. A. Engel and A. Koch, "Hardware-Accelerated Data Compression in Low-Power Wireless Sensor Networks," *Reconfigurable Computing: Architectures, Tools, and Applications*, LNCS 8405, 2014, pp. 167–178.
3. A. Engel, T. Siebel, and A. Koch, "A Heterogeneous System Architecture for Low-Power Wireless Sensor Nodes in Compute-Intensive Distributed Applications," *Proc. 40th IEEE Conf. Local Computer Networks*, 2015, pp. 636–644.
4. M. Bocca et al., "A Synchronized Wireless Sensor Network for Experimental Modal Analysis in Structural Health Monitoring," *Computer-Aided Civil and Infrastructure Eng.*, 2011; doi:10.1111/j.1467-8667.2011.00718.x.

Andreas Engel is a postdoctoral researcher at Technische Universität Darmstadt. His research interests include microcontroller- and FPGA-based wireless sensor networks, energy-management concepts for long-term WSN operation, and coarse-grained reconfigurable array-based hardware accelerators. Engel received a PhD in engineering from Technische Universität Darmstadt. Contact him at engel@esa.cs.tu-darmstadt.de.

Andreas Koch is a full professor and the head of the Embedded Systems and Applications group in the Computer Science Department at Technische Universität Darmstadt. His research interests include application-specific computer architectures and their design and programming tools. Koch received a PhD in engineering from Technische Universität Braunschweig. Contact him at koch@esa.cs.tu-darmstadt.de.