ENERGY-EFFICIENT HETEROGENEOUS RECONFIGURABLE SENSOR NODE FOR DISTRIBUTED STRUCTURAL HEALTH MONITORING

Andreas Engel, Björn Liebig and Andreas Koch

Technische Universität Darmstadt
Embedded Systems and Applications Group
Email: {engel,liebig,koch}@esa.cs.tu-darmstadt.de

ABSTRACT

Distributed structural health monitoring (SHM) using wireless sensor nodes (WSN) requires frugal spending of the limited energy budget. We propose a reconfigurable heterogeneous architecture, combining a low-power micro-controller (MCU) with a Field-Programmable Gate Array (FPGA), as a means for energy-efficient in-sensor processing. Details covered include a generic communication interface between both computing units and several clock-management schemes for energy efficiency. We evaluate the architecture on the use-case of a Random Decrement (RD) algorithm and also consider additional pre-filtering to reduce the volume of wirelessly transmitted data. Compared to conventional low-power sensor nodes, we can reduce the energy required for data processing by up to 81 %.

***Index Terms*—** wireless sensor network, reconfigurable computing, structural health monitoring, energy efficiency, random decrement technique

1. INTRODUCTION

SHM attempts to automatically detect damage to large structures such as bridges or wind turbines, reducing maintenance costs by allowing longer intervals between manual inspections. One SHM algorithm is the RD technique (see Section 3), which determines the free-decay response of the monitored structure for later modal identification. By relying on natural random excitations, the RD technique does not require additional actors for explicitly stimulating the structure. This allows its implementation on energy-constrained wireless sensor nodes.

However, accurately capturing the vibration of large structures requires sampling frequencies on the order of 100 Hz. This poses two problems considering the wireless sensor nodes' limited energy budget: First, the transmission of the detailed vibration data from the sensor to a central node actually performing the analysis is energetically expensive. Instead, local in-sensor preprocessing should be used to aggregate the data. To this end, even a low-power sensor node has to have sufficient computational capabilities. Second, when aiming for shorter sampling intervals, which in turn require more frequent wake-ups of the node from its deep sleep state, the transition times between sleep and active states becomes significant.

We propose a heterogeneous node architecture, combining a radio system on chip (RF-SoC) for the WSN functionality, with an FPGA for the energy-efficient realization of complex computations, to address both of these issues. Using a proof-of-concept prototype, we will describe how the RD technique can be implemented on the target architecture and show under which operating conditions the heterogeneous platform outperforms a solely MCU-based sensor node. In addition, we will investigate sophisticated clock generation schemes to further reduce the overhead induced by the platform power management infrastructure.

In the following section, we will briefly discuss related work on wireless sensor nodes in structural health monitoring, followed by an introduction into the RD technique. In Section 4, we introduce our heterogeneous node architecture and provide details of the RD implementation. Section 5 presents detailed results of the platform energy requirements compared to an RD implementation on an MSP430 MCU, a processor commonly used in low-power WSNs. Finally, we conclude and look out towards future improvements in Section 6.

2. RELATED WORK

The monitoring of large structures using wireless sensors is the subject of many current research efforts [1–4]. As an example of a concrete algorithm for such structural health monitoring, the RD technique is used to extract the modal parameters of a monitored structure [5–7].

Commercially available wireless sensor nodes are often based on low-power MCUs aiming for the lowest quiescent current [8]. Zimmermann et al. proposed an RD implementation on such a platform using an 8 bit Atmel MCU and recognized the necessity to improve the energy efficiency of even that low-power system [7].

Reconfigurable compute units (RCU) using FPGAs can perform complex computations more efficiently then MCUs

and digital signal processors (DSP) [9]. In time-critical applications, the use of RCUs often permits computations that cannot be performed by MCUs or DSPs at all under the given constraints. FPGAs have thus been employed for mid- to high-performance computing applications, recent work includes [10–12]. However, all of these studies utilized FPGAs relying on static memory (SRAM) for their configuration storage. While flexible, the use of SRAM precludes the use of deep-sleep modes powering down most of the device (which would lose the configuration information).

For low-power applications, FPGAs using the inherently non-volatile Flash memory are more suitable [13]. To this end, Vera-Salas et al. [14] combined a Flash-based Microsemi Igloo nano FPGA with a wireless transceiver. However, despite the power advantages of Flash configuration storage, this architecture is still sub-optimal in that the FPGA cannot enter device-wide deep sleep since it has to continue to perform low-intensity management tasks such as time-keeping and -synchronization.

Our HaLoMote [9] architecture avoids this problem by combining a Flash-based FPGA not only with a wireless transceiver, but with a complete RF-SoC that also encompasses a low-power processor core. This core has sufficient performance to handle low-intensity administrative tasks, even when the RCU is sleeping. The HaLOEWEn reconfigurable WSN [15] is the first implementation of the HaLoMote architecture. In this work, we will analyze its power characteristics and show improvements over the initial version on the use-case of the RD algorithm for SHM.

3. THE RANDOM DECREMENT TECHNIQUE

The RD technique was developed for damage detection in the late 1960s [16]. It estimates the free-decay response of large, potentially damaged structures in the form of so-called RD signatures $D_{xx} : \{0, \ldots, m-1\} \mapsto \mathbb{R}$. These signatures are defined as the average of n time sequences of a sensor's vibration data $(x_i)_{i\in\mathbb{N}}$, each sequence having the length m and starting with the same value a (the *trigger level*):

$$D_{xx}(k) = \frac{1}{n}\sum_{i=1}^{n} x_{i+k}, \quad 0 \le k < m, \quad x_i = a \qquad (1)$$

The measured displacement x_{i+k} is composed of the structure's response to the known initial displacement $x_i = a$, the structure's response to the unknown initial velocity $\dot{x}_i$, and a random component caused by external excitations. Due to its zero mean, the random component will be extinguished in D_{xx} for large n. Assuming a randomly distributed $\dot{x}_i$, which requires triggering on rising and falling signal edges, the velocity response will also be eliminated. Thus, D_{xx} converges toward the structure's displacement response as indicated in Figure 1.

As D_{xx} is proportional to the auto correlation of the signal x [17], it is called the *auto RD signature*. Furthermore,

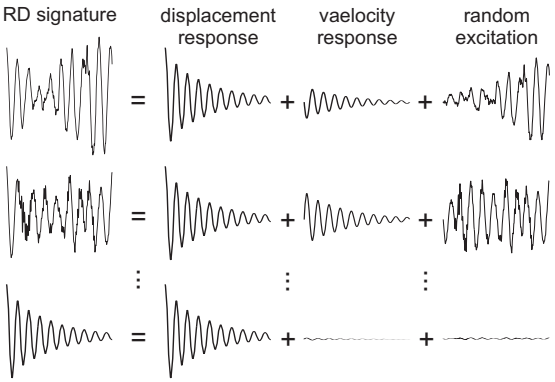


Fig. 1. Elimination of the random signal fractions by averaging of structural vibration data [2]

the RD technique can be extended to multi-channel measurements [5] by computing the *cross RD signature*

$$D_{yx}(k) = \frac{1}{n}\sum_{i=1}^{n} y_{i+k}, \quad 0 \le k < m, \quad x_i = a \qquad (2)$$

which is proportional to the cross correlation of the reference signal x and the response signal y.

The implementation of the RD technique requires selecting the sampling frequency, the number of averaging steps, the length of the RD signatures, and the trigger level. For the latter, $a = \sqrt{2}\cdot\sigma_x$ is suggested [5], with σ_x being the standard deviation of x.

4. PROPOSED SOLUTION

4.1. Heterogeneous reconfigurable sensor node

To efficiently execute the RD technique on an energy-constrained embedded system, we propose an improved version of the Hardware Accelerated Low Energy Wireless Embedded Sensor Node (HaLOEWEn) illustrated in Figure 2. This heterogeneous WSN platform is the first implementation of the HaLoMote architecture described in [9] and an enhancement of the work presented in [15]. The architecture combines a TI CC2530 RF-SoC as its MCU for wireless communication and time-management with an Microsemi Igloo M1AGL1000 FPGA as its RCU for hardware-accelerated computations. The RCU also controls a single channel $16\ \mathrm{bit}$ ADC over a serial interface for data sampling and a $64\mathrm{k} \times 16\ \mathrm{bit}$ SRAM over a parallel interface to store the RD signatures. Both peripheral components were chosen due to their low power draw in standby mode. While the RCU core operates at a supply voltage of $1.2\ \mathrm{V}$, the other components require $2.5\ \mathrm{V}$. Both voltage rails are derived from a $3.7\ \mathrm{V}$ source by two LTC3388-1 switching step-down regulators. This device was specifically chosen for its high efficiency at small loads to reduce the losses in standby mode.

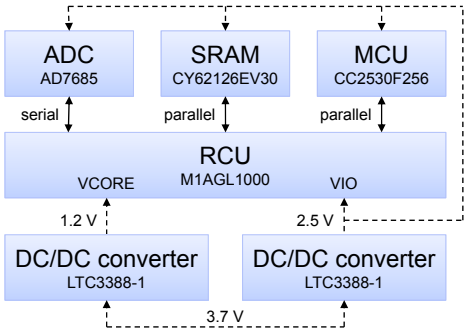


Fig. 2. HaLOEWEn Implementation of the HaLoMote architecture

The reduction of the voltage conversion losses achieved in this manner is an important improvement compared to the original HaLOEWEn implementation discussed in [15].

4.2. API for accelerating hardware kernels

In a heterogeneous architecture, the on-chip communication between the different computing units affects the system-level performance. As the CC2530 does not expose its memory bus to the I/O ports, we cannot embed RCU registers into the MCU memory space or vice versa. An obvious alternative would utilize the MCU USART controller for serial communication as proposed in [15]. We avoided the overhead of serial communications by explicitly controlling ("bit banging") general-purpose I/O pins to form a parallel bus. However, due to the limited number of available MCU pins, the bus is restricted to an 8 bit *data*, a 3 bit *cmd* and a *clock* line as shown in Figure 3. The bidirectional *data* line is driven by either of the two computing units (MCU or RCU) depending on the current *cmd* selected by the MCU. The bus transfers are synchronous to the *clock*. An additional *shutdown* signal is used by the MCU to shutdown (deep sleep) and wake-up the RCU.

Based upon this physical communication layer, we implemented an application-independent API permitting the MCU

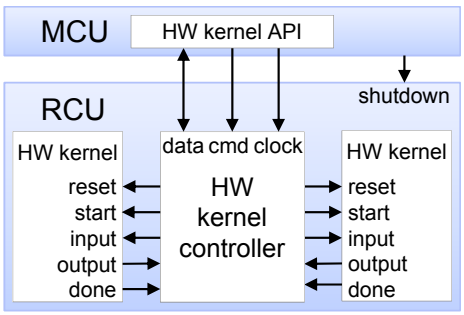


Fig. 3. Controller for HW kernel execution

cmd	*data* driver	*data* interpretation
SWITCH	MCU	index of HW kernel to select for data transfers and configuration
START	MCU	index of HW kernel to start
RESET	MCU	index of HW kernel to reset
OBSERVE	RCU	the running states of the HW kernels
READ	RCU	next sequence of the output data
WRITE	MCU	next sequence of the input data
CONFIG	MCU	auto-(re)start configuration bits
SWRSTN	MCU	ignored (command is used to completely reset the RCU)

Table 1. HW kernel API for on-chip communication

to control the execution of hardware (HW) kernels on the RCU. Each HW kernel may have an input, an output and an internal state. Its data processing is triggered by a start pulse, completion is signaled by a done pulse. Thus, the typical usage of a HW kernel would be writing its input data from the MCU to the RCU, generating the start pulse, waiting for the done pulse, and then reading the HW kernel output data back to the MCU. In a multi-kernel scenario, each data transfer must be associated with the index of the targeted HW kernel. We propose to separate the HW kernel selection from the data transfers (similar to virtual circuit switching) to reduce the communication overhead in applications with differing kernel execution frequencies. Another optimization deals with the automatic start pulse generation. Most HW kernels may be started when their inputs were completely written. On the other hand, a HW kernel without explicit inputs (e.g., the ADC control module) may be restarted when its output has been completely read. To reduce the overall communication demand, we therefore introduced a runtime-configurable auto-(re)start behavior for each HW kernel.

A HW kernel controller connects the parallel communications interface with the HW kernels by interpreting the *cmd* signal as described in Table 1. It manages the sequencing of the HW kernel inputs and outputs to the *data* line, generates the start flags and keeps track of executing HW kernels. The latter information is used to delay an RCU shutdown possibly requested by the MCU until all HW kernels actually finish their computations.

4.3. HaLOEWEn Implementation of the RD technique

To monitor large structures with the RD technique described in Section 3, reference and response nodes sampling the reference and response signals have to be distributed all over the structure. The accumulations of the response signals to the corresponding cross RD signatures are initiated by events representing a reference signal crossing its trigger level. In a wireless multi-hop network, the complete dissemination

of such events from the originating reference node to all response nodes within a single sampling period cannot be guaranteed due to possible packet losses and a transmission time dependent on the number of hops. This event propagation scheme would not properly scale with the size of the network and also increase the sensor nodes power consumption as they would have to continuously listen for events.

As an alternative, we use a *delayed* distribution of events over the monitoring network: Each event is represented by the ID of its originating reference node and the age of the event. The latter starts at zero during event generation and is incremented in every sampling period in all nodes that have received and stored the event. The bundled event/age information is broadcast over a single hop in every sampling period. When an event reaches a response node, its age information is used to properly calculate the cross RD signature. The details of the network protocol lie outside the scope of this work and will be discussed in a later article. Here, we focus on the hardware required to support the delayed event reception.

The core data structures are a sample shift-register and an event set as depicted in Figure 4. In every sampling period, the ADC output is inserted into the shift-register, displacing the oldest buffered sample. All events wirelessly received by the MCU are inserted into the event set. In every sampling cycle, the ages of all events are incremented. When the age of an event reaches or exceeds the sample shift-register depth $|Q|$, the associated RD signature data in the external memory is incremented by the current output of the shift-register. The corresponding memory address is derived as $\text{event}_{\text{id}} \cdot m + \text{event}_{\text{age}} - |Q|$. When the age of an event reaches $|Q| + m$, it is removed from the event set.

This computation synchronizes local samples with remote trigger events delayed by wireless network transmission. Assuming the reference signal x crossed the trigger-level threshold a in sampling period i, then the event $(x, 0)$ is generated at the remote reference node x. At the same time, the response signal sample y_i is inserted into the sample shift-register of the local response node y. The event, originating in node x and distributed through the wireless network, reaches node

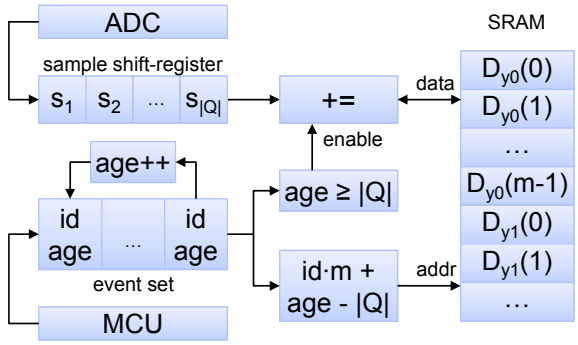


Fig. 4. Hardware modules for delayed event processing

y after a variable transmission delay and is inserted into the event set of the local node. y_i reaches the output of the shift-register in the sampling period $i + |Q|$, and is now synchronized with the matching event $(x, |Q|)$ held in the response node event set. The sample y_i is then accumulated to the RD sequence sample $D_{yx}(0)$ stored at memory address $x \cdot m$. In the following $k < m$ periods, the event is updated to the ages $(x, |Q| + k)$ and leads to the accumulation of y_{i+k} to $D_{yx}(k)$ located at the memory addresses $x \cdot m + k$. After the response node has received and processed n events from each reference node, the individual RD signatures $D_{y0}, D_{y1}, \ldots$ have been stored in contiguous memory locations. In order to satisfy Equation 2, the signature values then have to be divided by n, which is done efficiently by ensuring that n is a power-of-two.

In this manner, $|Q|$ determines the maximum number of sampling periods an event may be delayed by the network. The RD technique can thus be scaled to larger networks just by increasing sample shift-register depth of the response nodes.

To efficiently implement this scheme on the HaLOEWEn platform, we identified several hardware kernels. The trigger-level threshold-crossing check, the update of the sample queue, and incrementing of the event ages with the corresponding accumulations to the RD signatures are independent operations and can thus be parallelized. By delaying the ADC output for one sampling period, the input channel sensing can also be performed in parallel. This additional delay cycle is compensated by proper handling of the event age information. As all these tasks have to be performed in every sampling cycle, they are handled by a single HW kernel K_{sample} without input and the trigger-level threshold-crossing check as its output. By using the auto-restart–on–read functionality of the HW kernel API, the MCU-RCU communication for controlling K_{sample} can be reduced to a single read command.

Another HW kernel K_{insert} handles the insertion of new events into the event set. This is necessary once an event has been received from the network or generated locally. Typically, K_{insert} is executed less frequently than K_{sample} and we can take advantage of the virtual channel-based communication mechanism instead of providing each data transfer with a separate destination kernel ID (which would be similar to packet-switching).

The resulting RD signature $D_{y0}, D_{y1}, \ldots$ must be transmitted to a central node for subsequent analysis. If the centralized damage detection algorithms (not discussed here) require only parts of the spectral information contained in the RD signatures, we can locally perform the transformation to the frequency domain and drop (pre-filter) unnecessary data before it is transmitted. To this end, an integer-in-place-FFT has been implemented as an additional HW kernel. It starts with the 18 most significant bits of the RD signature and performs an automatic scaling by $\sqrt{N}$ for overflow avoidance.

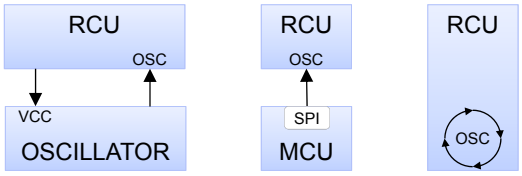
4.4. Efficient generation of the RCU clock signal

Each computing unit needs a periodic clock signal to drive its computations. The CC2530 RF-SoC can be clocked efficiently by an internal RC-oscillator which is automatically turned off when the MCU is put to sleep. In contrast, the M1AGL1000 FPGA generally depends on external components to generate its clock. Commonly, an external oscillator IC is used for this purpose. Unfortunately, when active, this IC often has a significantly higher power draw than the standby power of the RCU. Thus, it also has to be power-managed by being shut-down when the RCU sleeps, and restarted before the RCU is woken up. The power management of the RCU and its clock generation does affect the entire scheduling of the RD algorithm on the heterogeneous platform. We have examined three different RCU clocking schemes (Figure 5) and their impact on the RD algorithm schedule (Figure 6).

When utilizing an external oscillator IC, we suggest to directly control it by the RCU (Figure 5a), as this spares the MCU from observing whether the RCU is currently active. The M1AGL1000 FPGA supports this scheme as it can pull an I/O pin to ground when entering deep sleep mode ("flash freeze"), and to the supply voltage of the I/O banks when waking up even without a clock signal being present. The lengthy start-up time of the oscillator IC is the major disadvantage of this scheme: If the wake-up time of the RCU cannot be spent performing useful computations on the MCU, the MCU must be stalled waiting for the RCU clock to stabilize, leading to reduced energy efficiency.

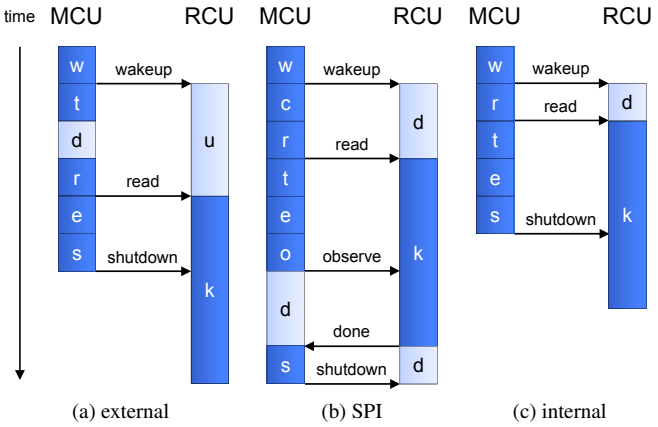
Instead of using an additional external oscillator IC, the serial clock generated by the MCU SPI module can be used to clock the RCU (Figure 5b). In this set-up, the SPI output register of the CC2530 is repeatedly filled by a DMA module to keep the serial clock running (even though no SPI transfers actually take place). The major disadvantage of this scheme is the necessity to keep the MCU active until the RCU finished its computations. As before, the MCU must be stalled unless useful computations can actually be issued. Thus, the approach using the serial clock can only do better then the external oscillator IC if the HW kernel execution time is *shorter* than the oscillator IC startup time.

Finally, a ring of inverters (Figure 5c) can be used to generate the clock signal for the RCU inside of the RCU itself



(a) external (b) SPI (c) internal

Fig. 5. Proposed RCU clocking schemes



(a) external (b) SPI (c) internal

w	timer IRQ wakes up MCU, MCU wakes up RCU
t	update timer for next sampling cycle
u	external oscillator start-up time
d	stall computation
r	read result of K_{sample} from last sampling period
e	check for new event to insert into the event set (for simplicity, the graphics assume a failed check)
k	K_{sample} execution
s	MCU asserts *shutdown* and sleeps
c	start DMA for SPI clock generation
o	observe whether RCU is currently active

Fig. 6. Scheduling of the RD algorithm on the HaLOEWEn platform for the different RCU clocking schemes

[18]. The frequency of this oscillator is controllable by the length of the inverter ring, but the actual frequency is also sensitive to the RCU core voltage and operating temperature. If potential frequency inaccuracies are acceptable, this scheme eliminates the flaws of the previous ones, as the MCU is no longer involved in the clock management.

5. EXPERIMENTAL EVALUATION

Clocking scheme	With FFT	Logic Cells	Block RAMs	Maximum Frequency
external	no	1861 (8 %)	3 (9 %)	21.8 MHz
SPI	no	1906 (8 %)	3 (9 %)	21.3 MHz
internal	no	1974 (9 %)	3 (9 %)	21.8 MHz
internal	yes	11023 (45 %)	19 (59 %)	18.9 MHz

Table 2. Hardware synthesis results for Microsemi M1AGL1000V2 FPGA

Table 2 summarizes the synthesis results of the four hardware configurations we used to evaluate the HaLOEWEn architecture. To obtain these results, we used the Synopsys Syn-

plify Premier DP F-2011.09-1 synthesis tool configured for auto-constrained clock frequency, resource sharing, and retiming. Without the optional FFT HW kernel, less than 10 % of the RCU area is used. Thus, for even greater energy savings, a smaller FPGA device from the Microsemi Igloo family could be used in this case. On the other hand, we will show that spending more logic for the FFT implementation is an efficient use of area and energy if the damage detection algorithm profits from frequency-based pre-filtering.

To compare the power consumption of the HaLOEWEn platform with a typical WSN mote, we also implemented the RD technique purely in software on two TI MSP430 MCU-based architectures. The first reference implementation (MSP430-1) utilizes the same ADC and parallel SRAM as the HaLOEWEn platform. However, most MCUs do not expose sufficient programmable I/O pins to interface parallel memories next to other peripherals. Thus, peripherals are usually connected serially to a single SPI bus. To reflect this, we replaced the external SRAM by a low-power serial $32\mathrm{k} \times 8$ bit SRAM (23A256) for the second reference implementation (MSP430-2). This also allows us to operate the entire reference system at just 1.8 V, making it more competitive with the HaLOEWEn mix of 1.2 V and 2.5 V supplies.

The power consumption of the architectures depends on the computational load induced by the RD implementation. This load is characterized by the number of accumulations within each sampling cycle and is thus heavily depending on the vibration data observed at the reference nodes. In a multi-channel scenario, this load would have to be multiplied by the number of reference signals the response node has to handle.

For reproducibility of results and evaluating the scalability of the HaLOEWEn and MSP430 platforms to increasing loads, we will be using synthetic input data. However, the synthetic data was modeled based on an actual 30 s vibration input signal recorded from a pedestrian bridge while a person was crossing. Choosing the signal-specific optimum trigger level according to [5] for this data resulted in a peak load of 30 accumulations per sampling cycle. Our synthetic data will thus cover a range of zero to 90 events per sampling cycle, representative of a response node processing three real data reference signals.

Power measurements were performed using an Agilent 34411A multimeter configured at 100 mA range and an integration time of 100 NPLC (2 s) to average the current spikes into the switching regulators. For both platforms, we measured the current flowing into the 3.7 V rails.

The MSP430-1 (MSP430-2) was operated at 20 MHz (8 MHz), which is its maximum clock at 2.5 V (1.8 V) supply voltage. Its firmware was built with TI Code Composer Studio 5.1.0.09000 optimizing for speed. The CC2530 firmware was built with the IAR 8051 Embedded Workbench 7.60, also optimized for execution speed. The CC2530 was driven by its 16 MHz RC-Oscillator, thus limiting the maximum serial clock to 1 MHz. We used an LTC6930-8.00 as external 8 MHz oscillator due to its fast start-up time of about 50 µs. For a fair comparison, the RCU internal oscillator was also configured to 8 MHz. All RD implementations used a sampling frequency of 128 Hz and 32 bit accumulations to avoid overflows.

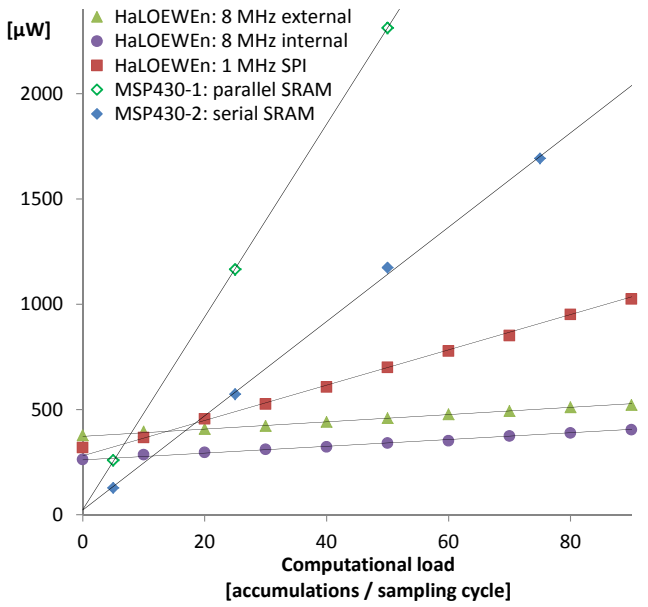
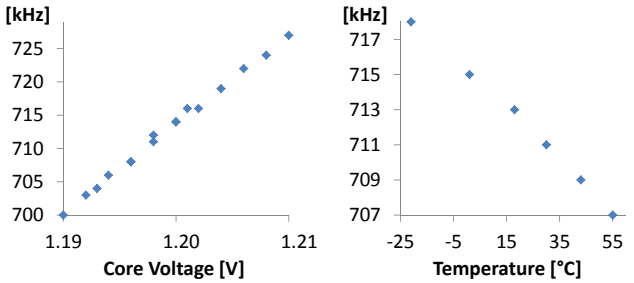


Fig. 7. Load-dependent power draw of the RD implementations

Figure 7 illustrates the resulting load-dependent power consumption for the MSP430 references and the HaLOEWEn platforms with the three proposed clocking schemes. The MSP430-2 clearly outperforms the MSP430-1 mainly due to the reduced core voltage and operating frequency. Furthermore, the MSP430-2 can utilize the dedicated USCI hardware module for SRAM communication while the MSP430-1 relies on extensive manipulation of general-purpose I/O. Thus, the parallel memory is no feasible option for the reference implementation and will be ignored for further discussion.

With an increasing load (more accumulations per sampling cycle), the HaLOEWEn architecture scales better then the MSP430 platform. In practice, the MSP430-2 is not even able to perform more than 100 accumulations in a single sampling cycle, and is thus unusable for scenarios with more then three reference signals. On the other hand, the MSP430 draws less power when only very few accumulations have to be performed. However, the break even-point of power drawn between MSP430-2 and HaLOEWEn lies at just 11…19 accumulations, which is even less than the real bridge signal would induce in a single-node system. For these realistic cases (especially involving multiple nodes), HaLOEWEn will be more power efficient.

Looking at the different RCU clocking schemes, the serial



(a) temperature fixed at 20 °C (b) core voltage fixed at 1.2 V

Fig. 8. Frequency stability of on-chip oscillator consisting of a ring of 400 inverters for variable supply voltage and environmental temperature

clock can outperform the external oscillator for cases with less then 14 accumulations per sampling cycle. In this case, the corresponding runtime of the accumulation kernel clocked at 1 MHz is 49 µs, as each accumulation takes 3.5 cycles on average. This matches the oscillator IC start-up time, thus confirming the trade-off discussion from Section 4.4.

As expected, the internal oscillator outperforms both other clocking schemes as it eliminates the necessity to stall the MCU (see Figure 6). On the other hand, the usage of an internal oscillator is quite unconventional as its frequency depends on the operating temperature and the RCU core voltage. As shown in Figure 8a, a ±10 mV core voltage variation translates into a ±13 kHz frequency variation for a ring of 400 inverters. Since the LTC3388-1 voltage regulator used on both platforms is specified to be stable within ±60 mV on its output voltage, the voltage-dependent frequency variation will be limited to ±78 kHz or 11 % of the target frequency. This far exceeds the temperature-dependent frequency variation, which we observed to be just ±6 kHz or 0.8 % over a ±45 °C temperature range (Figure 8b). Thus, the internal oscillator is a feasible power-saving design choice if its target frequency is conservatively configured to be about 15 % to 20 % less than the maximum clock frequency supported by the specific hardware design (ensuring correct operation even if voltage variations lead to a faster clock).

For future improvements of the HaLOEWEn platform, we also measured the per-component current flowing into the ADC, the SRAM, the MCU, the external oscillator, the RCU core, and its I/O banks at two different load conditions (idle and fully loaded). The resulting breakdown of the platforms power consumption is illustrated in Figure 9. The main insight gained from this data is the inefficiency of the 8051 MCU in the CC2530 RF-SoC: Even when not computing, it is the major power consumer in the system.

We then examined the execution profile of the 8051 MCU more closely, specifically for the common case of using an external clock oscillator IC (Figure 6a). As shown in Table 3,

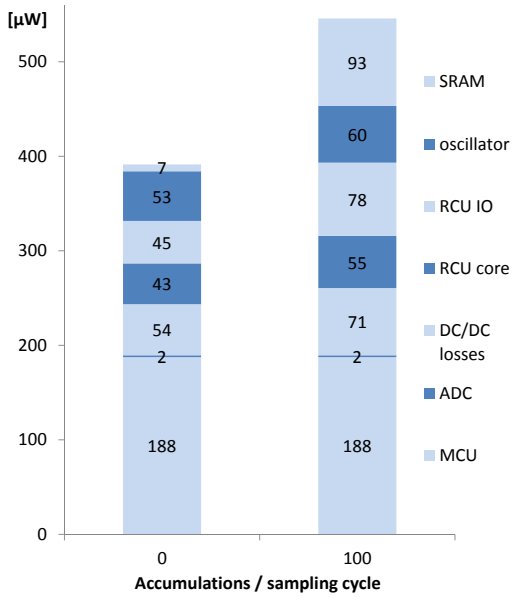


Fig. 9. Per-component breakdown of the HaLOEWEn power consumption at different loads

the MCU is awake for a total of 57.9 µs. However, less than 20 % of the execution time is actually required for controlling the HW kernel (time interval **r**), but more than half the total time is spent waiting for the external oscillator IC to stabilize its clock signal (time interval **d**). If it were possible to shut the MCU down for that time, significant power savings could be realized. In practice, this will not be achievable, since most MCUs do not support sleep periods that short.

For higher computational loads, the SRAM also begins to consume a significant part of the power budget. On the other hand, the power draw of the RCU is almost independent of the computational load. Thus, future optimization will focus on the MCU and SRAM for further power reduction.

Finally, we evaluated the possible energy savings of the optional in-sensor frequency transformation for data prefiltering. Using the FFT HW kernel, the MCU can sleep while the RCU computes the 2048 point FFT. When driven by the internal 8 MHz oscillator, this computation requires 69 ms and consumes a total energy of 505 µJ. The resulting frequency spectrum consists of 1024 36 bit complex numbers. Transmitting this data in its entirety would require 52 mJ. If only 1 % of the spectrum could be discarded as being irrelevant for the central damage detection algorithm (a

computation	**w**	**t**	**d**	**r**	**e**	**s**	total
duration [µs]	3.7	6.9	31	11.2	1.2	3.9	57.9

Table 3. MCU execution times for computations of Figure 6a

savings of 520 µJ), the energy expended on the FFT would already have been more than recovered.

The MSP430 does not carry sufficient internal RAM for the full 2048 point FFT, and using the serial external SRAM for the FFT calculation would not be a fair comparison. To determine the capabilities of the platform, we implemented a 256 point FFT which did fit in the internal RAM, utilizing its integrated hardware multiplier with the FFT twiddle factors defined as constant integer array. This FFT calculation requires 124 ms of execution time and consumes 259 µJ of energy. This is more than four times the energy the RCU would have required for the 256 point FFT.

6. CONCLUSION AND FUTURE WORK

The experimental results confirm the high potential of the proposed heterogeneous architecture for low-power applications such as the distributed structural health monitoring. Compared to a typical MSP430-based WSN, we have reduced the energy required to compute RD sequences by up to 81 %. Furthermore, we demonstrated how the energy-efficient computation realizable on the RCU can further reduce the energy required by shrinking the transmitted data volume via in-sensor preprocessing.

The detailed analysis of the power consumed by the HaLOEWEn node identified the CC2530 as a major power sink. To eliminate this problem, we are considering to substitute the 8051 MCU by an up-to-date alternative, such as the MSP430 Wolverine. Beyond that, we will focus on the hardware acceleration of network protocol-specific functionality, e.g., the time synchronization necessary for simultaneous sampling of the reference and the response signals.

7. ACKNOWLEDGMENT

This work is part of the LOEWE program funded by the Hessian Ministry of Higher Education, Research and Arts.

8. REFERENCES

[1] E. Sazonov, K. Janoyan, and R. Jha, "Wireless intelligent sensor network for autonomous structural health monitoring," 2004, pp. 305–314.

[2] M. Koch, H. Bollenbacher, et al., "Entwicklung und Umsetzung von verteilten Systemen zur Vibrationsanalyse und Strukturidentifikation," Diplomarbeit, Fraunhofer LBF, 2009.

[3] S. Kim, S. Pakzad, et al., "Wireless sensor networks for structural health monitoring," in *Proceedings of the 4th international conference on Embedded networked sensor systems*, 2006, pp. 427–428.

[4] S. N. Pakzad, G. L. Fenves, et al., "Design and implementation of scalable wireless sensor network for structural monitoring," *Journal of Infrastructure Systems*, vol. 14, pp. 89–101, 2008.

[5] J. Asmussen, *Modal Analysis Based on the Random Decrement Technique*, Aalborg University. Department of Mechanical Engineering, 1997.

[6] T. Dornbusch, M. Nottbeck, et al., "Experimental investigation of a random decrement based modals estimation on a pedestrian bridge," in *Conference Proceedings 14th Intternational Adaptronic Congress 2011*, 2011.

[7] A. T. Zimmerman, M. Shiraishi, et al., "Automated modal parameter estimation by parallel processing within wireless monitoring systems," *Journal of Infrastructure Systems*, vol. 14, pp. 102–113, 2008.

[8] R. Bischoff, J. Meyer, and G. Feltrin, "Wireless sensor network platforms," *Encyclopedia of Structural Health Monitoring*, 2009.

[9] A. Engel, B. Liebig, and A. Koch, "Feasibility analysis of reconfigurable computing in low-power wireless sensor applications," in *Reconfigurable Computing: Architectures, Tools and Applications*, pp. 261–268. 2011.

[10] M. Kohvakka, T. Arpinen, et al., "High-performance multi-radio wsn platform," in *Proceedings of the 2nd international workshop on Multi-hop ad hoc networks: from theory to reality*, 2006, pp. 95–97.

[11] C. H. Zhiyong, L. Y. Pan, et al., "A novel fpga-based wireless vision sensor node," in *Proc. IEEE Int. Conf. Automation and Logistics ICAL '09*, 2009, pp. 841–846.

[12] Y. Krasteva, J. Portilla, et al., "Embedded runtime reconfigurable nodes for wireless sensor networks applications," *Sensors Journal, IEEE*, pp. 1800 –1810, 2011.

[13] T. Nylanden, J. Janhunen, et al., "Fpga based application specific processing for sensor nodes," in *Embedded Computer Systems (SAMOS), 2011 International Conference on*, 2011, pp. 118 –123.

[14] L. Vera-Salas, S. Moreno-Tapia, et al., "Reconfigurable node processing unit for a low-power wireless sensor network," in *Reconfigurable Computing and FPGAs (ReConFig), 2010 Int. Conf. on*, 2010, pp. 173 –178.

[15] F. Philipp, F. Samman, and M. Glesner, "Design of an autonomous platform for distributed sensing-actuating systems," in *Rapid System Prototyping (RSP), 2011 22nd IEEE International Symposium on*, pp. 85 –90.

[16] H. A. Cole, *On-line failure detection and damping measurement of aerospace structures by random decrement signatures*, Nielsen Engineering & Research Inc., 1973.

[17] J. K. Vandiver, "A mathematical basis for the random decrement vibration signature analysis technique," *Journal of Mechanical Design*, pp. 303–308, 1982.

[18] Actel, "AC332: Flash*Freeze control using an internal oscillator," Tech. Rep., Actel Corporation, 2009.