Hardware-Accelerated Data Compression in Low-Power Wireless Sensor Networks

Andreas Engel[1] and Andreas Koch[2]

[1]LOEWE Research Center AdRIA, Darmstadt
[2]Embedded Systems and Applications Group, Technische Universität Darmstadt

Abstract. In wireless sensor networks, the actual transmission of collected data is often the most energy-consuming operation. Frequently, it is worthwhile to spend energy aggregating the raw sensor data on the node to reduce the transmission effort. For many cases, lossless data compression can be employed as a general data aggregation method, as incompressible data (noise) generally does not carry any information worth transmitting. Nevertheless, the energy spent for data compression must be traded-off against the energy saved for transmitting the compressed data. In this work, sensor data of two real-life applications is compressed using a hardware-accelerator of the heterogeneous HaLOEWEn sensor node. The benefits of providing the node with a reconfigurable compute unit is demonstrated by comparing its energy consumption with that of of a purely software-based implementation.

Keywords: reconfigurable computing, wireless sensor network, data compression, heterogeneous architecture, low-power mode

1 Introduction

Wireless Sensor Networks have been the subject of intense research [11]. In these distributed monitoring applications, the data gathered by the sensor nodes usually has to be forwarded to a central base station for final processing or storage. As the radio transceiver is the major power consumer of a wireless sensor node, even computationally intensive decentralized data aggregation methods can result in a reduction of the overall energy consumption of the sensor node. This is a major concern for the typically battery-powered sensor nodes.

In many applications, no specialized high-level data aggregation scheme (e.g., actual feature extraction) can be applied. Instead, all of the sensor data has to be forwarded to the base station. In these cases, lossless data compression can be employed as a more general form of data aggregation. However, efficiency can sometimes be improved by considering an application-specific system model in the compression scheme, e.g., the nature of rotating machinery. Generally, a trade-off between the compression quality and complexity of the underlying data model has to be found. When monitoring slowly changing environmental conditions, differential encoding has often proven useful.

While reducing the communication demands and energy consumption of the sensor nodes, data compression comes at the cost of encoder and decoder complexity. As the decoding is typically performed at the (often mains-powered) base station, the decoder complexity is not a major concern, thus this work focuses on encoding.

To improve the compression quality, many encoders first collect a block of data to analyze its statistical nature before compressing the block with the appropriate settings. This two-pass strategy increases the memory capacity required on the node as well as the latency between data acquisition and transmission. The block size, and thus the gains in compression quality, may be limited by the amount of available memory or real-time requirements in latency-sensitive applications. In addition, both encoder passes require a certain amount of computation time and energy, which must be amortized by the reduced communication effort. Moving operations from software to specialized hardware blocks can often reduce energy consumption and thus offers an attractive option to improve the computation vs. transmission energy balance on a sensor node.

In this work, data acquired by two different monitoring applications is losslessly compressed by a heterogeneous sensor node incorporating a low-power FPGA-based reconfigurable compute unit and a microcontroller-based radio system-on-chip [4]. The energy required for data transmission as well as for software and hardware implementations of the encoding are compared to demonstrate the benefits of hardware-accelerated data compression.

The remainder of this article is organized as follows: Section 2 gives a brief overview of software- and hardware-based data compression in WSN. In Section 3, the sample applications are introduced and a variety of compression schemes is applied to the raw sensor data to find the trade-off between compression quality and encoder complexity. Section 4 details the hardware-accelerated implementation of the most appropriate compression scheme before evaluating the energy reduction of the proposed method in Section 5. Section 6 concludes this work and looks out to further research.

2 Related Work

Fundamentally, we distinguish between generic data compression, applicable to almost all kinds of sensor data, and compressive sensing [3]. The latter assumes highly specific properties in the input signals and is not addressed in this work.

Data compression in WSN was investigated frequently in the last decade [7]. For example, an LZW-compressor was implemented on an MSP430 MCU to analyze the effect of reduced data rates on the end-to-end packet delay in a multi-hop network [2]. The energy savings achievable by an nonlinear adaptive pulse code modulator running on the ARM processor of a Beagle Board were investigated in [6]. However, these authors erroneously considered the compression ratios achieved directly as energy savings, completely ignoring the energy required for the encoding. This gross simplification was not used in [9], where run-length and adaptive Huffman encoding were implemented on the AVR MCU

of a Mica2 mote. The energy for encoding was determined solely by simulations and datasheet-specifications, using just synthetic data streams with guaranteed statistical properties as inputs.

Hardware-accelerated data compression in context of wireless sensor networks focused mainly on lossy image compression in visual surveillance networks, such as the JPEG compression on an Altera EP2C35 FPGA [12], or the identification of relevant image sections using a Xilinx Virtex II FPGA [8]. In addition to not compressing losslessly, these investigations aimed at reducing the required data rate to the throughput limits of the wireless transmission channel, instead of minimizing overall system energy consumption. The acceleration of a second order ADPCM compressor on a Xilinx XC4000 device was proposed in [1], but did not report any energy requirements.

The use of hardware accelerators for lossless data compression under energy constraints, which is the focus of this work, has not been studied extensively before.

3 Characteristics of Monitoring Applications

To investigate the potential and difficulties of compressing sensed data, two different applications were examined. The first one, neural activity in primates, is delay constrained, while the second one, condition monitoring of heavy industrial machinery, is computationally expensive due to multiple parallel data channels.

3.1 Evaluation of Compression Algorithms

As a baseline for our work, we examined the fundamental efficiencies of various compression *algorithms* for the applications, using off-the-shelf software implementations running on a non-energy constrained x86 processor.

In both cases, 8192 samples of each data stream were split into blocks of different size and fed into the compression algorithms listed in Table 1 with their specific run-time options. Static overhead (e.g., file headers) generated by these tools was disregarded when calculating compression ratios.

tool	**options**	**version**	**codec**	**overhead**
bzip2	-9	1.0.6	RLE + BWT + MTF + Huffman	24 B
rar	-m5 -en	5.00	proprietary	55 B
zp	c3	1.00	context modeling + arith. coding	221 B
mp4als	-7e	RM23	adapt. linear prediction + Rice	34 B
flac	-8	1.3.0	adapt. linear prediction + Rice	8292 B
ffmpeg	-acodec *alac* -f u8	0.8.7-6	adapt. linear prediction + Rice	0 B

Table 1. Compression algorithms and options used in further evaluation

In addition to using these off-the-shelf encoders, a custom forward-adaptive differential pulse code modulation (ADPCM-APF) compressor was implemented to allow a fine-grained trade-off between encoder complexity and compression rate. Here, the first M samples of a sample-block $(x_1, \ldots, x_N)$ are transmitted uncompressed. The successive samples x_i are mapped to a prediction error

$$d_i = x_i - \sum_{k=1}^{M} a_k x_{i-k} \qquad M < i \leq N \tag{1}$$

for the linear predictor of order M with coefficients $a_1, \ldots, a_M \in \mathbb{R}$. The classical approach of using an encoder based on a static first-order ($M = 1$) predictor and $a_1 = 1$ is referred to in the following as *dpcm* scheme.

For the forward adaptive predictor, which is referred to as a *adpcm* scheme, the predictor coefficients $a_1, \ldots, a_M$ are not static but fitted to the current sample block by calculating the autocorrelation values for the block:

$$r_k = \tfrac{s_k}{N-k} \qquad \text{with} \qquad s_k = \sum_{i=1}^{N-k} x_i x_{i+k} \qquad 0 \leq k \leq M \tag{2}$$

These values are then used to build a system of linear equations, whose solution results in prediction coefficients that minimize the variance of the prediction error sequence $(d_1, \ldots, d_N)$ [10]. This is important for the downstream Rice encoder, which maps the error value sequence to an actual bit stream, aiming for short bit representations for each error value. We will use Rice encoding in both the *dpcm* and *adpcm* schemes.

As an initial step for Rice encoding, the sequence of signed difference (prediction error) values d_i is converted to a sequence of unsigned values p_i by a simple transformation:

$$p_i = \begin{cases} 2d_i, & \text{for } d_i \geq 0 \\ -2d_i - 1, & \text{for } d_i < 0 \end{cases} \tag{3}$$

The values p_i are then encoded into Rice-form bit sequences

$$R_K(p_i) = \texttt{concat}(U(\tfrac{p_i}{2^K}), B_K(p_i \bmod 2^K)) \tag{4}$$

with K being the Rice parameter that balances the widths of a zero-terminated unary code U and the binary block code B_K in a bit-wise concatenation. Precise predictions ($p_i < 2^K$) can thus be represented by $K+1$ bits. Infrequently occurring larger prediction errors, caused by unforeseen spikes, can still be expressed losslessly by exploiting the variable length unary code. In our experiments, K was statically chosen for each data channel and not adapted to each sample block, in contrast to the audio encoders listed in Table 1.

3.2 Neural Activity in Primates

At the German Primate Center in Göttingen, the neural activities of primates solving different tasks are measured by a micro-electrode inside the probands brains. As the apes have to move freely over a wide testing area, wired instrumentation is impractical and thus the sensor data sampled with 16 bit resolution

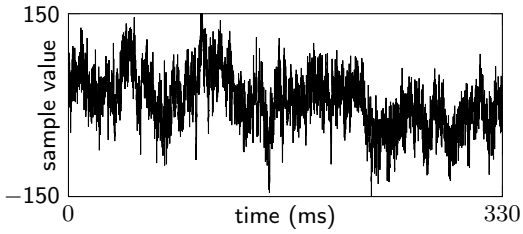


Fig. 1. 8192 samples of neural activity data (min = -155, max = 169, mean = 6.3, stddev = 45.5)

channel	1	2	3
min	-3872	-4156	-22466
max	5203	13104	16307
mean	110	2919	-3339
stddev	1213	2374	13041

Table 2. Statistical characteristics of 8192 samples of machinery condition monitoring data

at a frequency of 24.414 kHz (see Figure 1) has to be transmitted wirelessly. The resulting data rate of 391 kbit/s exceeds the capability of the popular IEEE 802.15.4 protocol, on which many recent low power radio transceivers are based. The captured data stream thus has to be compressed by about 50 % before it can actually be transmitted by an IEEE 802.15.4 transceiver.

Based on the received neural data, the variable penetration depth of the micro-electrode is controlled remotely by an operator at the control station. In order to allow timely interactive manipulation of the probe depth, the maximum end-to-end latency is thus restricted by the human response time of about 100 ms. Thus, the maximum block size used by a two pass encoder may not not exceed 2, 400 samples at 24.414 kHz.

The compression ratios (compressed data size / uncompressed data size) achieved by applying the compression algorithms from Section 3.1 are shown in Figure 2. As expected, the compression ratios of all encoders improve with increasing block sizes. With the exception of *mp4als*, the predictive audio encoders clearly outperform the dictionary-based compression schemes. Given the audio-like characteristics of the neural activity data, this in itself is unsurprising. However, the additional encoder complexity necessary for adapting the higher order linear predictor coefficients in the algorithms used in *mp4als*, *alac* or *flac* does not improve the compression ratios significantly compared to the static

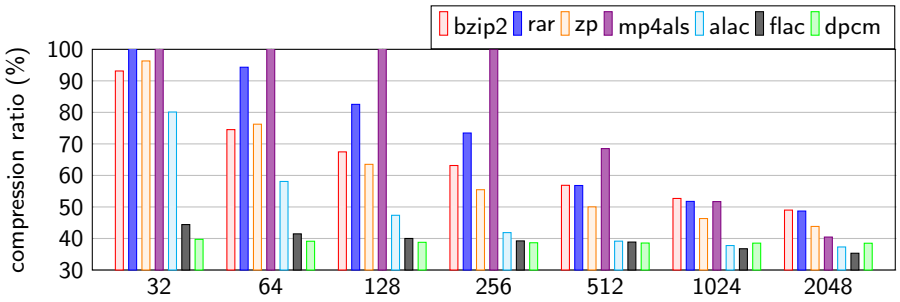


Fig. 2. Reduction of neural activity data achieved by various compression schemes

first-order *dpcm* predictor. For instance, at a block size of 2048 samples, the *flac* encoder achieves only a 3 % improvement in compression ratios at the cost of double the execution time when compared to the *dpcm* encoder running on the same platform.

3.3 Condition Monitoring of Heavy Industrial Machinery

The second application deals with detecting damage or fatigue of the rotating parts of very large industrial machines. This condition monitoring is used to schedule inspection/maintenance intervals before unanticipated major damage leads to high repair costs and downtime of the machinery. Since the monitoring algorithms observe long-term trends in the acquired data, this application is latency insensitive. Note that the sensor nodes are located on heavily vibrating parts of the machine, which would quickly wear out fixed cable connections, thus making low-power wireless communication preferable.

The raw data streams are gathered from a three channel MEMS sensor sampled at 1 kHz with a resolution of 16 bit per channel. Table 2 shows some statistical characteristics of the captured signals[1].

The compression algorithms described in Section 3.1 were applied to each channel separately. The resulting overall compression ratios are shown in Figure 3. Again, the predictive audio codecs are most appropriate. At a block size of 2048 samples, *flac* produces results 6 % smaller than *dpcm*. As in Section 3.2, this improvement comes at the cost of double the execution time for *flac*.

The *adpcm* scheme is examined separately. Figure 4 quantifies the impact of the prediction order. For small blocks, the size of the stored prediction parameters exceeds the benefit of improved compression ratios due to the reduced prediction error variance. For blocks of 2048 samples, the compression ratio strictly decreases with the prediction order and a break even point for best energy efficiency can be derived from the platform specific power draw for computation and transmission. Thus, adaptive prediction should be used for condition monitoring

[1] Actual waveforms cannot be shown here for confidentiality reasons.

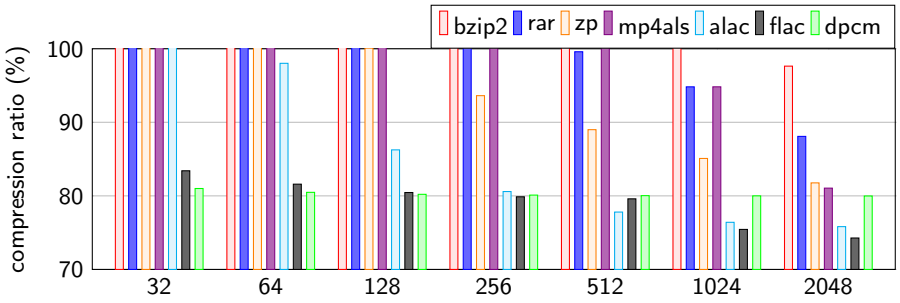


Fig. 3. Reduction of condition monitoring data achieved by various compression algorithms

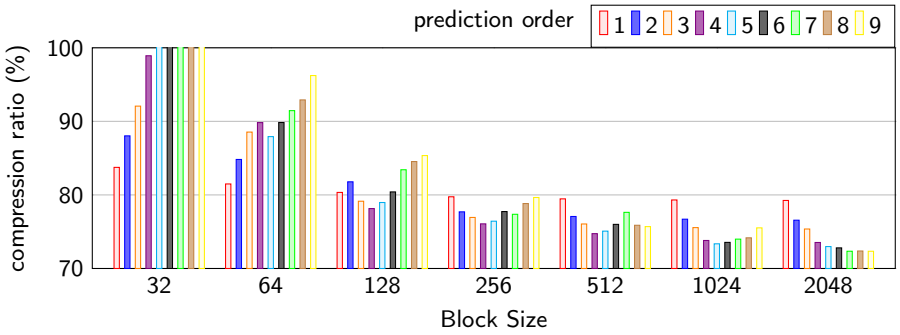


Fig. 4. Impact of prediction order on *adpcm* compression rate

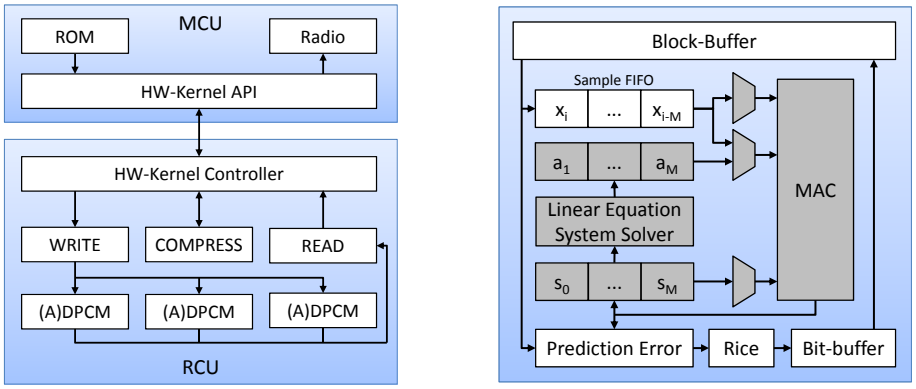
if the target platform supports storing a sufficient amount of samples (recall that there are three data channels, and double-buffering may be necessary for parallel sampling and encoding).

In conclusion, the *flac* scheme reached the best compression ratios for larger sample blocks, while *dpcm* proved superior on small blocks. Thus, for delay-sensitive applications or memory-constrained platforms, the simple *dpcm* scheme should be applied. In all other cases, *flac* can improve the compression rate of *dpcm* by about 5 %. Note that *flac* is just a combination of *adpcm* with an extensive search for the optimal prediction order. For data sources with known characteristics, a static selection of the prediction order may be sufficient. The adaptation of prediction coefficients in *adpcm* only becomes worthwhile, if the further data size reduction of 5 % over *dpcm* can be achieved with less energy than required to transmit the just *dpcm*-compressed data. For this reason, we also examine the energy efficiency of a hardware implementation of the *adpcm* algorithm.

4 Hardware-Accelerated Data Compression

In Section 3, (adaptive) differential pulse code modulation combined with Rice coding was identified as a balanced trade-off between compression quality and run-time effort for the investigated data streams. However, the low-power micro-controllers typically used in wireless sensor nodes may be overburdened even by these simpler algorithms, particularly by the adaptive compression scheme.

In [4], the HaLoMote architecture was proposed to energy-efficiently perform more compute-intensive distributed tasks even on low-power wireless sensor nodes. This heterogeneous architecture combines a micro-controller-based radio system-on-chip (MCU), responsible for handling wireless protocols and system management, with a reconfigurable compute unit (RCU) for the hardware-acceleration of complex computations. The current implementation, called HaLOEWEn, employs a TI CC2531 RF-SoC as MCU and a Microsemi AGL1000 FPGA as RCU. This section describes the hardware implementation of the (*a*)*dpcm* algorithms on the HaLoMote reconfigurable architecture.



(a) System overview (b) Compression Module

Fig. 5. Hardware-software interaction of the heterogeneous sensor node (a) and hardware-accelerated data compression (b)

Figure 5a gives an overview of the proposed implementation. The RCU can hold multiple hardware (HW) kernels performing different algorithmic functions. A communications API allows the MCU to interact with individual kernels [5]. The HW-Kernel controller starts the execution of HW-Kernels, manages data input and output, and reports execution completion. It is driven by the MCU via a bit-parallel interface through the application-independent API. For the data compression application, three HW-Kernels were implemented. The *WRITE* kernel distributes a sample stream to the compression modules, each processing one data channel. In practice, this sample stream will be generated by sensors attached to the RCU. For better reproducibility of results, our experiments below will process a prerecorded sample stream, read from the MCU code memory.

The compressed data stream generated by the compression modules is then sequentialized by the *READ* kernel and passed back to the MCU, which transmits it wirelessly. The HaLOEWEn radio stack allows parallelizing the radio transmission with the data transfer from the RCU to the MCU. Thus, the MCU duty cycle is not stretched by the communication task. This is important for the energy efficiency of the system (shorter duty cycles allow longer ultra-low-power sleep phases). Finally, the *COMPRESS* kernel controls the compression modules and tracks the number of generated output bytes.

Figure 5b shows the implementation of the compression module, instantiated once for each data channel. It contains a block buffer using on-chip RAM to hold the sample stream and compressed data stream. This in-place compression architecture allows to compress larger data blocks in memory-constrained systems (the on-chip RAM of the low-power FPGA used is limited to just 144 kbit). However, it requires a compression ratio *smaller* than 100 % (i.e., compression actually *reduces* the data size) for each prefix of each sample block in the sample stream. This constraint is achievable for both the primate neural activity as well

condition monitoring scenarios, where the spikiness of data is limited by the inertia of the underlying biochemical and mechanical systems.

The shaded modules of Figure 5b are used only in the adaptive prediction. For the simple static first-order *dpcm* compression, the block buffer is read once (retrieving uncompressed data), with the last sample x_{i-1} also being retained in a sample FIFO to calculate the prediction error as required by Equation 1. This prediction error is passed to the Rice coder to produce the bit sequence described by Equations 3 and 4. This bit sequence is sliced into bytes and written back to the block-buffer (now in compressed form) by a bit-buffer module.

For the adaptive *adpcm* compression scheme, an additional coefficient optimization pass precedes the compression pass. During this pass through the block-buffer, the autocorrelation sums s_k of Equation 2 are accumulated. To this end, a time-multiplexed multiply-accumulate unit (MAC) is supplied with the appropriate operands from the sample FIFO $(x_i, \ldots, x_{i-M})$ and the accumulator set $(s_0, \ldots, s_M)$. At the end of the pass, the autocorrelation sums s_k computed in the accumulators are used generate the linear equation system that has to be solved to retrieve the prediction coefficients $(a_1, \ldots, a_M)$. As a trade-off between prediction accuracy and the time and energy spent to calculate the coefficients and the prediction values, fixed point arithmetic was chosen. The resulting coefficients are stored in Q4.12 format. To conserve FPGA area and energy, we chose $M = 1$ for our experiment, which simplifies the linear equation system solver to just

$$a_1 = \frac{r_1}{r_0} = \frac{s_1 \cdot N}{s_0 \cdot N - s_0} \tag{5}$$

By restricting the block size N to a power of two, the remaining integer division can be performed sequentially in 16 clock cycles. In each step of the subsequent compression pass, the prediction coefficients have to be multiplied with the corresponding prior value(s) from the sample FIFO to accumulate the prediction of the next sample (Equation 1). The MAC unit and one of the autocorrelation sums is reused for these calculations, which are performed in parallel with the bit-buffering of the Rice code of the previous prediction error.

5 Experimental Evaluation

The hardware-accelerated data compression design described in Section 4 was synthesized for the Microsemi IGLOO AGL1000V2 FPGA using Synplify Pro H-2013.03M-1 with retiming. The block buffer size of the compression modules was fixed at 2048 samples. As shown in Table 3, the design is primarily limited by the available memory, some area remains for implementing higher order predictors (see Section 6).

To demonstrate the energy efficiency of the hardware-accelerated data compression, the measurement setup shown in Figure 6 was used. The HaLOEWEn sensor node is supplied by an external 3 V voltage source to power its internal components. The MCU drives the RCU into its low power flash freeze mode as long as no hardware-accelerated computations are required. For precise time

channels	scheme	BRAM	Core Cells	max Frequency
1	dpcm	8 (25 %)	1681 (7 %)	19.2 MHz
1	adpcm	8 (25 %)	6728 (27 %)	10.5 MHz
3	dpcm	24 (75 %)	4419 (18 %)	22.4 MHz
3	adpcm	24 (75 %)	14088 (57 %)	10.7 MHz

Table 3. Synthesis results for the Microsemi IGLOO AGL1000V2 device

measurements, an external trigger is asserted by the MCU when a compute tasks starts and is recorded by an oscilloscope. The average current drawn by the system during the task execution is measured by an Agilent 34411A multimeter, which provides a resolution of 3 µA at a sampling frequency of 50 kHz.

For comparison with the hardware-accelerated encoders described in Section 4, the *dpcm* and *adpcm* compression schemes were also implemented on the TI CC2530 MCU of the HaLOEWEn sensor node to estimate the energy consumption of a conventional microcontroller-based sensor node. The software was compiled by the Small Device C-Compiler (SDCC) v3.3 with speed optimizations enabled. The energy required for transmitting the uncompressed data stream is used as the baseline measurement. Each transmitted packet carries a maximum of 116 payload bytes.

As indicated in Figure 5a, a block of 2048 prerecorded sensor samples per channel, stored in the MCUs code memory, is used as the data source. The software implementations of the compression schemes operate directly on this data block. For the hardware-accelerated compressors, the process of moving the samples from the MCU to the RCU is included in the measurements, as we assume that in practice, the sensors would be directly attached to the RCU.

Table 4 shows the results for processing the neural data described in Section 3.2. Both *dpcm* and *adpcm* compression reduce the data size down to 38.5 %, making the simpler *dpcm* algorithm a better choice for this application. The *dpcm* execution on the MCU takes more than double the 61.2 ms required for transmitting the compressed data stream. The MCU thus has to be kept active beyond the pure transmission time of the prior packet in order to perform the compression of the current packet, leading to longer duty cycles and a loss

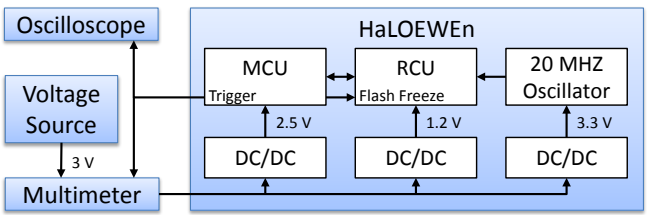


Fig. 6. Measurement setup

compression					compression + transmission					
		duration	current	energy			duration	current	energy	
scheme	on	[ms]	[mA]	[µJ]	bytes	packets	[ms]	[mA]	[mJ]	[%]
none					4096	36	158.8	41.5	19.77	**100.0**
dpcm	MCU	132.2	11.8	4680	1577	14	184.6	21.6	11.96	60.5
adpcm	MCU	249.4	11.8	8829	1577	14	310.4	17.5	16.30	82.4
dpcm	RCU	1.4	10.7	45	1577	14	62.6	41.9	7.87	39.8
adpcm	RCU	2.6	10.7	83	1577	14	63.8	41.3	7.90	40.0

Table 4. Energy to compress/transmit 2048 samples of neural data (Sec. 3.2)

compression					compression + transmission					
		duration	current	energy			duration	current	energy	
scheme	on	[ms]	[mA]	[µJ]	bytes	packets	[ms]	[mA]	[mJ]	[%]
none					12288	106	474.4	41.6	59.21	**100.0**
dpcm	MCU	531.0	11.9	18957	9836	85	912.0	24.3	66.48	112.3
adpcm	MCU	906.0	11.9	32344	9732	84	1281.0	20.6	79.17	133.7
dpcm	RCU	1.6	10.7	51	9836	85	381.6	42.5	48.62	82.1
adpcm	RCU	2.9	10.7	93	9732	84	378.9	42.4	48.15	81.3

Table 5. Energy to compress/transmit 3×2048 samples of condition data (Sec. 3.3)

of energy efficiency. In contrast, for the hardware-accelerated implementations, RCU→MCU data movement occurs in parallel to the ongoing data transmission. The very short execution time of the compression stretches the duty cycle before going back to sleep only marginally, leading to an almost complete translation of compression ratio into system-level energy savings (38.5 % ratio vs. 39.8 % energy for *dpcm*).

Processing of the condition monitoring data discussed in Section 3.3 is evaluated in Table 5. Here, the adaptive predictor improves the compression ratio over *dpcm* such that a complete radio packet is saved. On the MCU, however, compressing the three data channels takes so much time, that the energy required cannot be amortized over the transmission savings, instead leading to an efficiency deterioration. The hardware-accelerated encoders fare much better: Since all channels can be compressed in parallel, the total execution time only slightly increases over that of the single channel encoder used in Table 4. As before, the encoders using the RCU convert nearly all of the data volume savings into actual energy savings.

6 Conclusion and Future Work

In this work, lossless data compression was used to aggregate collected sensor data before transmitting it wirelessly to a data sink (central node). The trade-

off between compression quality and encoder complexity was analyzed for two different types of real-world sensor data. To this end, differential compression with a linear predictor and a downstream Rice encoder has been implemented on a Microsemi IGLOO FPGA and a TI CC2530 microcontroller. While the software compressor did reduce the overall energy consumption of compression and transmission for data compression ratios of 40 %, it did not succeed for compression ratios of only 80 %. The hardware-accelerated encoder, however, achieved almost perfect efficiency converting space savings due to compression into actual energy savings. Since for some data streams, such as the condition monitoring application, adaptive encoding yields improved efficiency, future work will evaluate the gains possible using higher-order adaptive predictors having $M > 1$.

References

1. Boonyakitmaitree, C., Nandhasri, K., Ngarmnil, J.: A low computational predictor coefficient algorithm for adpcm implementation of portable recording devices. In: Circuits and Systems, 2004. MWSCAS '04. The 2004 47th Midwest Symposium on. vol. 3, pp. iii – 187–90 vol.3 (2004)
2. Deng, X., Yang, Y.: Online adaptive compression in delay sensitive wireless sensor networks. Computers, IEEE Transactions on 61(10), 1429 –1442 (2012)
3. Donoho, D.: Compressed sensing. Information Theory, IEEE Transactions on 52(4), 1289–1306 (2006)
4. Engel, A., Liebig, B., Koch, A.: Feasibility analysis of reconfigurable computing in low-power wireless sensor applications. In: Koch, A., Krishnamurthy, R., McAllister, J., Woods, R., El-Ghazawi, T. (eds.) Reconfigurable Computing: Architectures, Tools and Applications, Lecture Notes in Computer Science, vol. 6578, pp. 261–268. Springer Berlin / Heidelberg (2011)
5. Engel, A., Liebig, L., Koch, A.: Energy-efficient heterogeneous reconfigurable sensor node for distributed structural health monitoring. In: Morawiec, D.A., Hinderscheit, J. (eds.) Conference on Design and Architectures for Signal and Image Processing (DASIP). Electronic Chips & Systems design Initiative (2012)
6. Kasirajan, P., Larsen, C., Jagannathan, S.: A new data aggregation scheme via adaptive compression for wireless sensor networks. ACM Trans. Sen. Netw. 9(1), 5:1–5:26 (2012)
7. Kimura, N., Latifi, S.: A survey on data compression in wireless sensor networks. In: Proc. Int. Conf. Information Technology: Coding and Computing ITCC 2005. vol. 2, pp. 8–13 (2005)
8. Ngau, C., Ang, L.M., Seng, K.: Low memory visual saliency architecture for data reduction in wireless sensor networks. Wireless Sensor Systems, IET 2(2) (2012)
9. Reinhardt, A., Christin, D., Hollick, M., Steinmetz, R.: On the energy efficiency of lossless data compression in wireless sensor networks. In: Proc. IEEE 34th Conf. Local Computer Networks LCN 2009. pp. 873–880 (2009)
10. Sayood, K.: Introduction to Data Compression. Morgan Kaufmann (2005)
11. Yick, J., Mukherjee, B., Ghosal, D.: Wireless sensor network survey. Comput. Netw. 52, 2292–2330 (August 2008)
12. Zhiyong, C., Pan, L., Zeng, Z., Meng, M.: A novel fpga-based wireless vision sensor node. In: Proc. IEEE Int. Conf. Automation and Logistics ICAL'09 (2009)