Feasibility Analysis of Reconfigurable Computing in Low-Power Wireless Sensor Applications

Andreas Engel[1], Björn Liebig[1], and Andreas Koch[2]

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

Abstract. With increasing complexity of sensor network applications, the trade-off between node-local processing and transmission of data to a central node for handling becomes more significant. For distributed structural health monitoring applications (SHM), we consider different realization choices of the underlying wireless sensor network and implement a key part of the application (a high-order filter) on the novel *HaLoMote* architecture, a reconfigurable wireless sensor node (rWSN) with FPGA-based processing capability. We compare different tool flows supporting development of algorithms above the RTL regarding to achievable area and energy efficiency and outline the advantage of rWSN over traditional MCU- and DSP-based sensor systems in this scenario.

Keywords: reconfigurable computing, wireless sensor network, high-level synthesis, wordlength optimization, low-power mode.

1 Introduction

Networks of wireless sensor nodes (WSN) have been used in numerous practical applications [1]. Those applications require data acquisition, some kind of processing for data aggregation, and finally the dissemination of results through the network.

In general the small compute power of the WSN allows only limited local preprocessing, restricting it to simple data logging and communication tasks. This approach is not suitable for all applications, though. If data has to be sent to the central node(s) at shorter intervals, a significant amount of energy will be required for the radio transmissions [2].

Motivated by adaptive vibration control and structural health monitoring applications in the context of the LOEWE Research Center for Adaptronics (AdRIA) [3], we have investigated the use of energy-efficient reconfigurable computing in the nodes to allow for more intelligent local processing, thus saving energy by reducing transmission data volumes. The first focus of this work will be the evaluation of real-world applications on different platforms. To ease this comparison, we concentrated on a high-order filter (384-tap FIR), a key operation in most AdRIA applications.

A. Koch et al. (Eds.): ARC 2011, LNCS 6578, pp. 261–268, 2011.

Since our work in context of AdRIA is performed in close collaboration with non-hardware designers, we also require our rWSN architecture, named *HaLoMote* (for Hardware-Accelerated Low-power Mote), to be easily "programmable". Thus, the second focus of this work is the support of automatic tool flows from descriptions in Simulink and Matlab onto the target platform. We will characterize the solutions examined in terms of computing performance, energy requirements and area required.

To this end, we will briefly survey related work in Section 2. Section 3 gives an overview of the *HaLoMote* architecture, concentrating on the processing elements. A number of tool flows for programming the *HaLoMote* from high-level descriptions will be examined in Section 4. We evaluate our benchmark kernel, the 384-tap FIR filter, implemented on the *HaLoMote* and on three traditional low-power micro-controllers and DSPs in Section 5. Finally, Section 6 concludes and looks forward to future work.

2 Related Work

2.1 Reconfigurable WSN Computing

Popular microprocessor-based WSN platforms are the Mica2 motes [4], which use an 8 bit Atmel ATmega128L [5] MCU, the TelosB motes [6] and the T-Mote sky [7], both of which are based on 16 bit MSP430 [8] MCU. A more detailed summary about conventional WSN technologies and services can be found at [1].

While many FPGA vendors claim to offer energy-efficient devices, only few devices are actually aimed at very low-power operation (e.g., in mobile devices). The key to extreme energy conservation are deep sleep modes, that can be both quickly entered and exited. While some suggestions have been made on how to extend the conventional Xilinx Spartan-3 FPGA series with this capability [9], devices which natively support such operations are preferable. Examples include the SiliconBlue iCE65 family [10] and the Actel Igloo series [11], the latter of which will be used in our *HaLoMote*.

Reconfigurable processing has been considered before for WSNs, specifically for image processing applications in a Wireless Visual Sensor Network (WVSN) [12,13]. The combination of an 8051-based MCU and a Xilinx Spartan-3 (XC3S200) FPGA was presented in [14]. However, these prior attempts have not examined power consumption in detail and do not describe active power saving measures (e.g., managing deep sleep modes) or do not consider low-power implementation alternatives.

2.2 Automatic Mapping of High-Level Descriptions to RTL

Our use-cases require mostly control engineering and signal processing applications to be executed in the WSN. Since these algorithms are often developed in Matlab and Simulink, it is worthwhile to examine the current state of tool flows supporting the automatic mapping of these high-level descriptions to reconfigurable processing units.

A number of commercial tools allow the translation of Matlab/Simulink models into RTL HDL netlists suitable for mapping onto FPGAs [15,16,17]. For Simulink, this is generally achieved by restricting models to the subset of blocks supported by the high-level synthesis tool (for which underlying hardware implementations have been developed manually). Translating Matlab, which allows greater freedom of expression than just composing blocks, is supported by fewer tools [15,17] and more severely restricted: Only very limited constructs can be automatically translated into hardware (and then often not very efficiently).

An alternative is an indirect approach by first exporting the Matlab/Simulink description as a C program using a tool such as the Real-Time Workshop Toolbox [18]. Then, the many academic (e.g., [23]) or commercial (e.g., [24,25,26]) C-to-HDL synthesis systems can be applied to hardware mapping problem. This approach is not entirely seamless, as the generated C code contains unsupported constructs (e.g., pointers, floating-point computation). We will examine later in this paper how the gap between the Matlab-exported C and the subsets accepted by the C-to-HDL tools can be closed.

Both approaches require predetermination of fixed-point word length, which should be done for each signal individually [20]. While an analytic approach exists for signal to noise ratio (SQNR) [21], other quality functions often have to be calculated by simulation.

3 HaLoMote Architecture and Prototype Implementation

Fig. 1 gives a high-level overview of the *HaLoMote* platform architecture. Some of the aspects, e.g., the sensor interfaces, memory system and power supply lie outside the scope of this work and will not be discussed further. Instead, this work will mainly deal with the actual processing elements: An 8-bit MCU is integrated into the Radio Communications SoC for low-performance tasks (e.g., running the wireless networking protocols). The FPGA can be used both to realize a software-programmable 32 bit soft-core CPU as well as dedicated accelerators directly implementing an algorithm in hardware.

For our first *HaLoMote* implementation, we selected the following devices for the platform components: 2.4 GHz IEEE 802.15.4 compliant radio communications are provided by a TI CC2530 RF-SoC [27], which includes an 8051-compatible MCU. Software running on the 8051 implements the Zigbee protocol

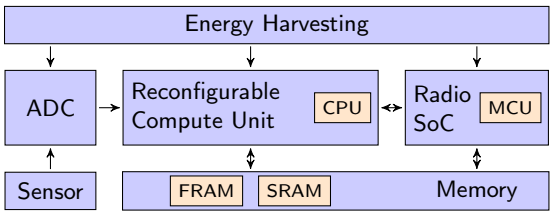

Fig. 1. Simplified overview of the *HaLoMote* platform

stack and support services. The RF-SoC consumes about 24 mW when operating at 32 MHz with radio disabled, about 92 mW when receiving data and between 86 mW and 118 mW when transmitting data. The 8051 starts operations on the RCU, implemented by an Actel M1AGL1000 Igloo FPGA [11], which was chosen for to its ultra low-power "Flash Freeze" sleep-mode, that can be entered and exited in ≈ 1 µs, saving all outputs and register content while lowering power draw to 52 µW.

4 High-Level Synthesis Tool Flows

For evaluating the high-level programmability of our architecture, specifically the RCU, we first consider the fixed-point word-length optimization, then two ways to generate hardware: One going from Simulink directly to RTL HDL, the second one exporting C and translating that to RTL HDL.

4.1 Word-Length Optimization

Due to the inefficiency (both area and energy-wise) of floating-point computations on FPGAs, we convert the computations to word-length optimized fixed-point form, allowing for separate word-lengths for each operator to achieve even smaller logic. As a quality measure, we use the mean square error over the filter response. Using a heuristic word-length approach we can generate an entire set of possible realizations (targeting different mean square error bounds) in 80 seconds on a Core 2 Duo E8500 PC. Assuming that an error bound of 0.5 dB suffices for this application, the resulting computation will have a 16 bit datapath, 14 bit coefficients and 7 bit data width on the multipliers.

4.2 Hardware Synthesis

The first approach is the direct conversion from Matlab/Simulink to VHDL. This is especially easy for our 384-tap FIR benchmark, which can directly map to a specialized block, generally also giving additional implementation options such as resource sharing (folding) or optimized constant coefficient multiplication. In the first alternative, we directly translate this model to RTL HDL for implementation on the FPGA.

The indirect approach using C based synthesis, is also applicable to models containing blocks not available in the tool vendors hardware libraries. The C code generated for the Simulink model by Real Time Workshop has to be transformed to become suitable for C-to-Hardware compilation. This includes pointer and file operations, computations split into several functions which need to be merged, and floating-point arithmetic re-introduced during the C export. For discrete single-rate Simulink models, we developed a software tool that automatically removes or rewrites the problematic constructs, ending up with C code amenable to hardware compilation.

5 Experimental Evaluation

After annotating word-lengths into the Simulink model, we translate it to hardware suitable for execution on the *HaLoMote*-RCU. We will have to refer to the design tools used for this step just by S1/S2 for direct Simulink-HDL translation, and C1/C2 for Simulink-C-HDL translation, since the tool license terms prohibit direct benchmarking. However, this approach is sufficient to evaluate the applicability of our architecture and design methodology.

5.1 High-Level Synthesis: Latency vs. RCU Area Trade-Off

Fig. 2 compares the latency required to process a single sample, and area required on the M1AGL1000-FPGA, for different implementations achievable using the different tool flows. While the Simulink-HDL tools support a very finely granular setting of resource sharing, the flows involving C-to-HDL translation are more restricted in that they support only the extremes; in the case of C2, actually only the (very small, high latency) design point. This is due to C2 not recognizing how often the loops in the FIR would be executed, and thus only allowing full sequentially executing on a single multiplier. The two RTL models generated by C1 differ in their internal storage scheme: In the small (slow) implementation, BRAMs are used to store the filter state while in the large (fast) implementation registers are used for this purpose.

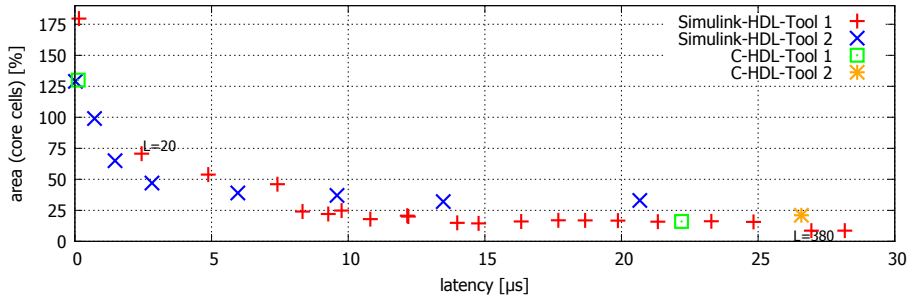

Fig. 2. Area vs. latency for final synthesis results

5.2 Latency and Area vs. RCU Power Draw Trade-Off

Fig. 2 does not consider the power drawn by the different realizations. For further analysis, two extremal solutions generated by S1 are examined: L=20 is the fastest one fitting on the FPGA and uses 19 multiplier in parallel, L=380 is the smallest one generated by all tools containing only a single multiplier. The first filter needs 20 clock cycles at a maximum frequency of 8.2 MHz for one sample, which results in a minimum execution time of 2.44 µs. The second filter has a latency of 380 clock cycles at 14.1 MHz, resulting in a minimum execution time of 26.95 µs.

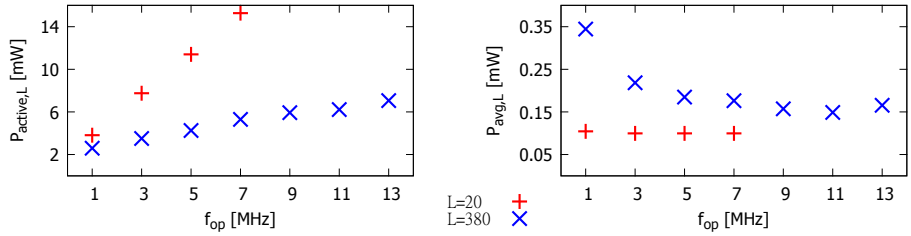

Fig. 3. Measured active and average M1AGL1000 power draw; computation of duty cycle for 384-FIR at different operating clock frequencies

To determine the power draw, each of these filters was mapped onto the M1AGL1000-FPGA and measured their core current (at 1.2 V) when operating them at a range of clock frequencies f_{op}. The left graph in Fig. 3 gives the resulting measurements as $P_{active,L}$, which show the expected result of increasing power drawn for larger logic areas toggling at higher speeds.

In a second scenario, which is typical for our vibration control application, the filter is not running continuously. Instead, the M1AGL1000-RCU is just woken up from Flash-Freeze by the RF-SoC at a rate of $f_s = 400$ Hz to process a single sample. Note that the filter hardware still executes for L cycles at the higher clock frequency f_{op}. Thus, the duty cycle of the RCU actually operating relative to the entire sample time is $\text{duty}_L(f_{op}) = \frac{f_s}{f_{op}} \cdot L$.

As the right part of Fig. 3 indicates, the average power $P_{avg,L}$ drawn in this scenario is at least an order of magnitude below $P_{active,L}$, and is actually *decreasing* with larger f_{op}. This demonstrates, that, given the availability of an ultra-low power sleep mode, it is energy-efficient to compute as quickly as possible, even accepting faster clock speeds to maximize time in deep sleep mode. It can also be seen that the power savings of L=380 due to smaller area in the first graph can not compensate the effect of faster runtime which results in a higher average power consumption of the smaller design.

5.3 RCU Performance Evaluation

Fig. 4 shows the power draw of the best design points generated by the different tool flows. S2 outperforms S1 due to its lower latency, allowing longer sleep times.

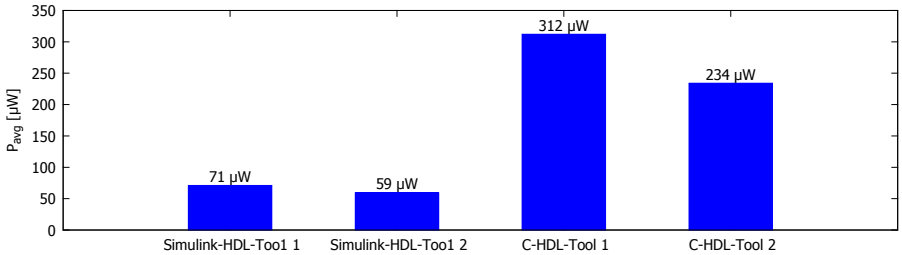

Fig. 4. Best results of different Toolflows

The two Simulink-C-HDL flows fare worse, because only their (small, slow) fully resource-shared designs actually fit on the M1AGL1000 device. These designs have long latencies and thus poor power efficiency, despite their small area.

In Fig. 5 energy efficiency of the M1AGL1000 implementation, is compared to low-power MCUs and DSPs. The 8 bit 8051 MCU could not reach the desired sampling frequency of 400 Hz, even when running at a maximum of 32 MHz. In practice, this would require radio transmission of the sample to a central node for computation. When ignoring protocol overhead, a lower bound for the duty cycle of transmission of a 16 bit value over a 250 kbps IEEE 802.15.4 channel each 2.5 ms can be calculated to draw 2.2 mW of average power.

For the 16 bit MSP430 MCU [28] an operating frequency of 9 MHz was sufficient to run the filter at $f_s = 400$ Hz. At this operating frequency, the MSP430 draws 5.6 mW, but has to run continuously (no sleeping). Stepping up the MSP430 clock frequency would allow sleeping between the samples, but also requires higher supply voltage which annihilates the power savings of sleeping.

The 16 bit ultra-low-power DSP TMS320C5515 [29] running at 3 MHz fulfilled the required sampling rate while drawing 0.75 mW.

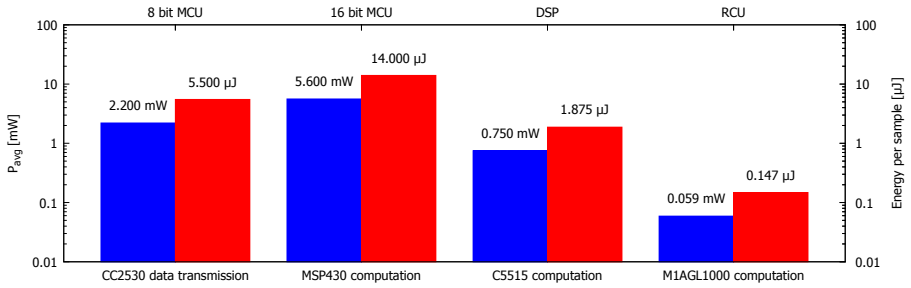

Fig. 5. Comparison of MCU- and RCU-based FIR implementations

6 Conclusion

We have shown that, with current design tools, even domain experts not proficient in hardware design can implement performance-critical parts of common algorithms on the RCU directly from their high-level Simulink descriptions.

Given a reconfigurable device capable of quickly entering and leaving an ulta-low power sleep mode, we have determined that tools should aim for the fastest possible realization, even if that requires higher clock frequencies or larger chip areas (due to limited resource sharing). The energy gains of waking up from deep sleep only briefly for computation dominate the additional power required for these faster designs.

Our "run to idle" strategy is only applicable, however, if the RCU is not used to continuously perform traditional WSN middle-ware services (e.g., communication protocols, timekeeping, synchronization). To this end, our *HaLoMote* platform, being a heterogeneous multi-processor, can leave these tasks on the low-power MCU and use the RCU just for brief bursts of computation.

References

1. Yick, J., Mukherjee, B., et al.: Wireless sensor network survey. Comput. Netw. 52, 2292–2330 (2008)
2. Sadler, C.M., Martonosi, M.: Data Compression Algorithms for Energy-Constrained Devices in Delay Tolerant Networks. In: Proc. Fourth ACM Conf. Embedded Networked Sensor Systems (2006)
3. LOEWE-Zentrum AdRIA, http://www.loewe-adria.de/
4. MICA2 WIRELESS MEASUREMENT SYSTEM, http://www.memsic.com
5. 8-bit Microcontroller with 128K Bytes In-System Programmable Flash, http://www.atmel.com
6. TELOSB MOTE PLATFORM, http://www.memsic.com
7. Ultra low power IEEE 802.15.4 compliant wireless sensor module, http://www.sentilla.com
8. MSP430x1xx User's Guide
9. Tuan, T., Rahman, A., et al.: A 90-nm Low-Power FPGA for Battery-Powered Applications. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 26(2), 296–300 (2007)
10. iCE65 Ultra Low-Power mobileFPGA Family, http://www.siliconbluetech.com/
11. IGLOO Handbook, http://www.actel.com
12. Latha, P., Bhagyaveni, M.A.: Reconfigurable FPGA based architecture for surveillance systems in WSN. In: Intl. Conf. on Wireless Communication and Sensor Computing, pp. 1–6 (2010)
13. Zhiyong, C.H., Pan, L.Y., et al.: A novel FPGA-based wireless vision sensor node. In: IEEE Intl. Conf. on Automation and Logistics, pp. 841–846 (2009)
14. Portilla, J., Riesgo, T., et al.: A Reconfigurable Fpga-Based Architecture for Modular Nodes in Wireless Sensor Networks. In: 3rd Southern Conference on Programmable Logic, pp. 203–206 (2007)
15. Synopsys Synphony Model Compiler, http://www.synopsys.com
16. Xilinx System Generator for DSP, http://www.xilinx.com
17. Simulink HDL Coder, http://www.mathworks.com/products/slhdlcoder/
18. Real-Time Workshop, http://www.mathworks.com/products/rtw/
19. Akyildiz, I.F., Su, W., et al.: A survey on sensor networks. Communications Magazine 40(8), 102–114 (2002)
20. Constantinides, G.A., Cheung, P.Y.K., et al.: Multiple Precision for Resource Minimization. In: IEEE Symp. on Field-Programmable Custom Computing Machines (2000)
21. Constantinides, G.A., Cheung, P.Y.K., et al.: Synthesis and Optimization of DSP Algorithms, pp. 27–38 (2004)
22. Sbarcea, B., Nicula, D.: Automatic Conversion of MatLab/Simulink Models to HDL. In: Intl. Conf. on Optimization of Electrical and Electronic Equipment (2004)
23. Kasprzyk, N., Koch, A.: High-Level-Language Compilation for Reconfigurable Computers. In: Intl. Conf. on Reconfigurable Communication-centric SoCs (2005)
24. Synopsys Synphony C Compiler, http://www.synopsys.com
25. Mentor Graphics CatapultC, http://www.mentor.com/esl/catapult/
26. AutoESL AutoPilot, http://www.autoesl.com/
27. CC253x SoC Solution for 2.4 GHz IEEE 802.15.4 and ZigBee® Applications, http://www.ti.com
28. CC430F613x MSP430 SoC, http://www.ti.com
29. TMS320C5515 Fixed-Point Digital Signal Processor, http://www.ti.com