A Novel Reconfigurable Architecture for Wireless Sensor Network Nodes

Johann Glaser, Jan Haase, Markus Damm, Christoph Grimm

Institute of Computer Technology, Vienna University of Technology, Austria
{glaser,haase,damm,grimm}@ict.tuwien.ac.at

Abstract

Most wireless sensor network nodes use a CPU to implement the main functionality in software. This provides flexibility compared with a pure ASIC implementation. However, due to the wakeup overhead and waiting periods the total energy consumption of periodic measurements and network listening is considerably increased.

To tackle this problem novel reconfigurable peripheral blocks are introduced into the SoC. These autonomously handle simple sub-tasks of the periodic activities while the CPU stays in an inactive low power mode. The CPU is only activated if further processing is required (e.g. transmission of a notification packet via the wireless network if the measurement value has changed). To maintain a high degree of flexibility for the implemented applications the introduced peripheral blocks are reconfigurable.1

1 Introduction

Wireless sensor networks (WSN) are applied in numerous applications like building automation, automotive systems, container tracking, and geological surveillance. Typical implementations of sensor nodes comprise a central processing unit, memory, an RF transceiver, sensors, power supply, and a software stack for both sensor control and network processing [1] as shown in Figure 2 (except for the shaded blocks).

A common challenge in virtually all application fields is the requirement for low energy consumption. This is especially difficult due to the high power consumption of the RF transceiver, sensors, and CPU. Several techniques were proposed to tackle this. One option is to limit the activity of the node to a portion of the time (duty-cycling, [2]). Another approach is to integrate most components into a single chip (ASIC), because sensor nodes built with multiple individual commercially available components waste a lot of energy in voltage level adaptations (voltage matching problem, [1]).

This work examines a different problem: A typical task of a sensor node is to periodically measure the sensor values. Therefore the CPU is activated from its inactive low-power mode. Then the power supply of an external sensor is turned on. After some settling time the sensor value is ascertained by an analog/digital conversion (ADC). The resulting value is finally compared to the previous value and only if the difference is greater than a certain threshold, further processing is conducted, like sending the value via the wireless network. In all other cases the CPU immediately switches back to an inactive low-power mode.

Another example is the MAC (media access control) layer of the network stack. For a network protocol with periodic listening intervals the receiver is turned on for a certain time waiting to receive a packet. If a packet is received, its destination address is inspected to decide upon further processing. After these relatively simple tasks the more complex tasks of the higher protocol layers follow.

These cycles consist of several simple tasks (set and read digital IOs, compare two integers) where delays occur (sensor settling time, network listening). These are performed by the CPU, which is overqualified and oversized for these tasks. It consumes power during the whole active state as well as for the wakeup procedure.

We propose to avoid this waste of energy for these periodic tasks by introducing novel reconfigurable hardware blocks into the WSN node SoC (system-on-chip) as depicted by the shaded “Sensor Interface” and “Communication Interface” blocks in Figure 2. These take over some of the simple and periodic tasks from the firmware. This allows the CPU to stay in an inactive low-power mode for extended periods and is only activated for more complex tasks.

In the next Section the power consumption of different

1This work is conducted as part of the Sensor Network Optimization through Power Simulation (SNOPS) project which is funded by the Austrian government via FIT-IT (grant number 815069/13511) within the European ITEA2 project GEODES (grant number 07013).
implementations of a sensor interface are examined. This is followed by a description of the proposed approach. This work is finished by a conclusion and outlook to future work.

2 Related Work

In [3] we investigated the power consumption of the sensor interface application mentioned above for several microcontrollers, as a hard-coded hardware block, and with the novel reconfigurable hardware architecture.

The power consumption of five different low-power microcontrollers with 8- or 16-bit RISC CPUs was investigated. The sensor control is implemented as a common C function for every chip. The execution times were calculated from counting the assembler instruction execution cycles and dividing by the operating frequency. For the waiting periods constant values were assumed. The power consumption was estimated by using typical data sheet values at 4 MHz and 25°C.

The hardware implementations were realized with an FPGA (field programmable gate array) instead of producing of a chip. Note the stack of reconfigurability, where the underlying technology (FPGA) is reconfigurable and the implemented circuit itself is reconfigurable too. Due to the usage of a commercially available FPGA instead of a full- or semi-custom chip design, no optimized multi-bit cells were implemented.

The two FPGA implementations only differ in the state machine (FSM) type. In the reconfigurable implementation a Block RAM is used while the other one uses a hard-coded state machine. The difference in current consumption is therefore mostly caused by the Block RAM which offers several different configurations, layouts, and wrappers which adds overhead in power consumption. Besides this overhead both FPGA implementations comprise overhead due to the fine granularity of the logic functions for the data-path oriented tasks as well as for the routing on the FPGA chip.

The energy consumption values of a single sensor measurement performed by the different implementations are summarized in Figure 1. While the microcontroller implementations only differ slightly (189.15 nJ to 266.22 nJ with an average of 219.08 nJ and a range of 35.2 %), the difference to the hardware implementations is tremendous (2.09 nJ and 5.61 nJ). The five bars for the microcontroller implementations are divided into the individual components (e.g., wakeup, interrupt latency, context save, ...). The FPGA implementation only consists of the function part because it doesn’t require any microcontroller related tasks.

Compared to the lowest power microcontroller MSP430F5418, the hardcoded state machine implementation in the FPGA requires over 90 times less energy. The two hardware implementations differ by a factor of ≈2.7 with the reconfigurable implementation requiring more energy. But even this requires nearly 34 times less energy than the MSP430F5418.

Due to the different semiconductor processes and the underlying architectures of the FPGA and the microcontrollers the comparison is inaccurate. A fair comparison would require to implement both, a low-power microcontroller core and the sensor interface, on a common chip. However, due to the high overhead of the FPGA implementation the total energy per sensor measurement of the hardware implementation will decrease even further. On the other hand, the firmware implementation will not experience such a large reduction because optimized low-power microcontrollers were used. So the difference between these two implementations will further increase, which strengthens the optimization potential.

![Figure 1. Energy consumption of one sensor measurement performed by different implementations. The first five bars represent the results of microcontroller implementations, broken down by contributing facility. The right two bars show the consumption of two FPGA implementations, once with a hard-coded FSM and once with a reconfigurable FSM.](image-url)
3 Reconfigurable Architecture

Starting from the above findings, we propose to introduce reconfigurable hardware blocks to a WSN SoC (see Figure 2) which independently conduct simple sub-tasks instead of the CPU. The CPU is only activated if any further (more complex) processing is required. Therefore these logic blocks act as a “filter” for these events.

3.1 Using Reconfiguration

Shifting the border in a software/hardware partitioning process towards hardware reduces the flexibility of the final application. While software can be modified by reprogramming the code memory, synthesized logic cores require a redesign of the chip. Hence, there are three important reasons for flexibility:

1. Covering multiple different applications for a higher market potential.
2. Adopting to different external components across PCB design cycles.
3. Fixing bugs without a chip redesign.

Therefore we propose to make the introduced dedicated hardware blocks reconfigurable.

In contrast to FPGA cores (e.g., [4]), which consist of fine-grained structures optimal for control dominated functions, we propose to also support multi-bit logic blocks and a multi-bit routing architecture for computational functions. The high overhead is outlined with the following example. The Xilinx Virtex FPGA architecture requires 864 bits of configuration for every configurable logic block (CLB) [5] (48 frames per column × 18 bits per row). We estimated the number of configuration bits required for only the logic function of one CLB to be 86 by counting the configurable LUTs and multiplexers (MUXes). From these numbers it is clear that the logic itself is configured by only 10% of the configuration bit stream while 90% account for the connections and routing. To avoid this high overhead, [6] proposes a multi-bit routing architecture which utilizes the inherent regularity of logic vectors.

To further reduce the power consumption and area requirement, the proposed approach requires the user to define an application class for which the reconfigurable block is inserted. This class describes the field of planned actual applications. Then the reconfigurable logic block is developed to be tailored to provide exactly those structures which are required to implement any of the desired applications. After the manufacturing of the SoC, the actual application is specified and implemented by configuring the reconfigurable block accordingly.

3.2 Components

Our approach includes concepts for the instantiated cells (multi-bit combinational and sequential cells, reconfigurable cells), the reconfigurable routing between these cells, interfaces of a block to other modules of the SoC, reconfiguration storage and interface (see Figure 3), as well as tools to assist the developer.

![Figure 3. Internals of a reconfigurable hardware block](image)


3.2.1 Cells

For the reconfigurable logic blocks a library of cells is developed. These include typical standard library cells (logic functions and registers) as well as multi-bit cells (e.g., shifters, arithmetic functions), “tactical cells” [7] (e.g., absolute difference, finite state machines, shift registers, CRC generators, timing generators), reconfigurable cells (lookup tables, sum-of-product cells) and infrastructure cells (e.g., routing switches, interface blocks, parametrization registers and especially the configuration chain).

3.2.2 Routing

FPGAs provide rich resources for signal routing between the individual logic blocks. In contrast to FPGAs the proposed approach involves an irregular topology as well as multi-bit lanes. It is crucial to include a proper set of routing resources tailored to the desired application class to provide a high degree of flexibility in the final design.

3.2.3 Interfaces

Depending on its application class, every reconfigurable logic block requires different interfaces to other peripherals like ADC, I2C bus, SPI bus, RF transceiver and memory (via an on-chip bus or direct signals). The CPU is informed of an important event via an interrupt interface. Finally, every block needs clock and reset signals.

3.2.4 Reconfigurability

Two main areas provide reconfigurability: Firstly the routing switches have to be configured to properly connect the individual cells. Secondly certain cells themselves can
adapt their functionality according to configuration, e.g., an arithmetic cell which either behaves as an adder or as a subtracter, a generic combinational cell like a LUT (look-up table), or a state-machine cell (FSM, see Section 3.3).

All these cells (including routing switches) have to be addressed and reached with low circuit and area overhead to supply the configuration information. Therefore we propose to connect their configuration storage registers as a serial shifter chain.

The interface to the configuration chain can either be accessed from the CPU on the SoC or from external pins (e.g., via an in-system programmer). In the latter case a non-volatile memory is required and the possibility of configware upgrades is limited to the case of physical access to the WSN node. A CPU configuration interface requires the firmware to download the configuration bit stream as part of the initial system setup. This enables easy configware upgrade by a firmware upgrade and even changing the functionality at runtime (e.g., adopting to different MAC protocols when entering another mobile ad-hoc network). The downside is the additionally required capacity of the firmware storage as well as CPU run-time and energy overhead for the initial setup.

A related but separate topic is parameterization. This means to set (numerical) parameters of the implemented application like the threshold of a changed sensor value or a timing parameter of the MAC protocol. These values must be changeable during the normal operation of the reconfigurable logic block without the need to reload the configuration bit stream and therefore require a separate interface.

### 3.2.5 Tools

The designer is assisted by software tools for the whole design flow. For the pre-silicon phase the logic synthesis and chip layout is performed by commercial tools. Custom tools are required in the post silicon phase for technology mapping and bit stream generation. Standardized and open interfaces (e.g., the open Liberty file format for cell libraries or the EDIF electronic design interchange format for netlists) can be utilized to connect commercial and custom tools.

### 3.3 Transition based Reconfigurable FSM

One common component of a reconfigurable logic block is its control logic, usually implemented as an FSM. In this section a novel reconfigurable architecture for FSMs is described [8].

A simple solution for a reconfigurable FSM is the use of a RAM in read-only-mode with a feedback through a clocked register. Its address inputs are fed by the current state signals and the primary inputs. Its data outputs are used as FSM outputs and as next state signals connected to the register. This architecture allows to implement any possible FSM with the given number of inputs, outputs and state signals and is reconfigurable by writing the appropriate RAM content. However, the size of the RAM grows exponentially with the number of inputs and state signals. On the other hand, the implemented FSM can have transitions from any state to any other.

FSMs in actual applications have considerably less transitions. This is especially true for FSMs with many inputs signals, where certain states are only inspecting a subset of these signals. An implementation using an (embedded) FPGA can utilize this fact by optimizing the logic functions for computing the next state and the output signals from the current state and the input signals. Unfortunately FPGA-like structures introduce a large overhead due to its flexible routing resources (compare Section 3.1).

We propose a different approach which focuses on the transitions rather than on the state transition function, and call this Transition-based Reconfigurable FSM (TR-FSM). Instead of providing a big reconfigurable block for implementing the whole state transition function, we provide several smaller reconfigurable blocks for implementing each transition.

Applying TR-FSMs for ASIC or SoC-design is again a two-stage process. In the first stage, the necessary resources for the TR-FSM are specified. This can either be an estimation based on FSM prototypes for a certain application class, or it is based on a collection of FSMs which the resulting TR-FSM must be able to be configured to.

From this specification, the semiconductor chip containing the TR-FSM is produced. In the second phase,
his TR-FSM embedded in the chip now can be configured with an actual FSM. However, since the TR-FSM is already structured like an FSM, the synthesis process is considerably less complex.

The basis of the proposed architecture is the so called transition row (see Figure 4). A state selection gate (SSG) compares the current state to its configured value and enables the transition row. A reconfigurable input switching matrix (ISM) selects a subset of the input signals which then serve as the inputs for the input pattern gate (IPG), which is a reconfigurable combinational logic block, e.g. a look-up table (LUT). If activated, the IPG outputs a “1” iff the input pattern matches a certain transition from the recognized state. The reconfigurable next state ID register (NSR) of the corresponding transition row is then selected by a multiplexer (“Select”) and fed back to the state register (SR) as the new state of the FSM.

Consider for example an FSM with a transition from state A to state B occurring for the input patterns 10--- and 01---. If the current state equals A, the SSG of the responsible transition row enables the IPG. The ISM is configured to connect the leftmost 3 signals out of the 5 input signals to the IPG. If the input pattern matches 10-- or 01-, the IPG would output a “1”, indicating that the transition to state B is active. This selects the value stored in the NSR for the next state B.

The overall architecture contains many such transition rows with varying input width (see Figure 4). The number of transition rows and their input widths are derived during the specification stage mentioned above. The output signals are computed in a separate configurable logic block ("Outputs" in Figure 4), which for example can be a LUT or an embedded FPGA IP core. The output pattern can also be included in every transition row, but this will increase the number of required transition rows if the output pattern depends on input patterns for common next states. Note that the state encoding can be chosen freely with respect to the transition logic, since both, the SSG and the NSR are fully configurable. That is, different state encodings won’t yield any benefits like a smaller number of transition rows.

As depicted in Figure 4, the current state is also fed into the next state selection logic block (“Select”). This allows for a very simple treatment of loops, i.e. transitions where the FSM remains in the same state. In the case that none of the transition rows are activated, the select logic chooses the current state as new state. Therefore, no transition rows have to be used for loops.

To summarize, the reconfigurable parts of the architecture are the state selection gate (SSG), the input switching matrix (ISM), the input pattern gate (IPG), and the next state ID register (NSR).

Our preliminary findings show that TR-FSMs need considerably less configuration bits than FPGAs (only $\approx 10\%$ of the length). While we see this as an indicator for a prospective improvement regarding area and power consumption, this is also beneficial by itself for applications with limited memory resources like wireless sensor nodes.

4 Conclusion

The optimization potential of the power consumption of a sensor interface for WSN nodes was depicted. It is leveraged by accompanying the CPU with additional reconfigurable hardware blocks that take over simple tasks from the CPU to release it from frequent wake-ups. This novel reconfigurable architecture was outlined and a details on reconfigurable FSMs were given.

Future work includes research on reconfigurable routing, development of multi-bit cells and a tool chain to ease the implementation of a reconfigurable hardware block.

References


