The Crux of OMNeT++ on development for a specific Wireless Sensor Node Platform, A Progress Report

Oliver Stecklina and Andreas Krumholz
IHP
Im Technologiepark 25
15236 Frankfurt (Oder), Germany
Email: {stecklina, krumholz}@ihp-microelectronics.com

Abstract—OMNeT++ is a discrete event-driven simulator and is intensively used for testing and validating research results in the area of Wireless Sensor Networks (WSN)s. In the context of the IQlevel project we developed a Wireless Sensor Node Platform (WSNP) and a Multi-Hop Routing (MHR) Protocol, which benefits by our configurable protocol stack model. It allows us to use a single protocol implementation in different environments without any adaptations. Driven by the impressive advantages of the OMNeT++ simulator we wrapped into the early decision to continue our protocol development for our WSNP with Castalia. In this progress report we will depict why these strategy was an aberration, which finished in a sloppy protocol implementation. We will describe five hard to find traps that we had to solve for porting our successfully simulated implementation to our target WSNP. As a conclusion we will mentioned an alternative approach for testing and simulating implementation for specific WSNPs.

I. INTRODUCTION

In the recent years the research in the area of Wireless Sensor Network (WSN)s becomes more and more enlarged. Publications cover a broad variety of topics, e.g. hardware, operating systems, low power systems and wireless communication protocols. Especially the design and the implementation of wireless Medium Access Control (MAC) schemes and routing protocols are deeply investigated. The addressed features as ad-hoc structures, self-organization, self-healing or mobility make the protocols and their development more and more complex. Furthermore, testing and validation become a time-consuming process and cannot be handled with real WSNPs.

Due to the complexity and size of WSN protocols and their deployments the usage of simulation frameworks becomes very attractive. Driven by a similar motivation we used Castalia, which is based on OMNeT++, for testing and validating our Fair Energy Trade (FET)-Multi-Hop Routing (MHR) protocol [1]. The protocol was developed in the context of the IQlevel project [2] and is currently in extension in context of the Aeternitas project [3]. The protocol aims to provide a long living multi-hop network with a mesh structure. We started our development on a real Wireless Sensor Node Platform (WSNP), but due to the complexity of the test and validation runs we early decided to continue our work with the Castalia framework. After finishing the simulation with promising results we were confident that we can immediately move our implementation to our MSP430-based WSNP [4], [5]. But we had to learn in a long nerve-racking process that a successful running implementation in Castalia has not to be suitable for a real WSNP. In this progress report we will describe five traps that were very hard to find and that we had to solve during our porting work. We are convinced that most of these traps may be clear for a skillful WSN engineer, but we also hope that our progress report can motivate researchers to be more carefully when using Castalia / OMNeT++ for their protocol development as well as it helps to explain why OMNeT++ and Castalia are not platforms for testing compiled code for a specific WSNP.

In the following we shortly explain few fundamentals of the OMNeT++ / Castalia framework. In section III we present our approach of a configurable protocol stack model, which allows us to use an unmodified implementation in different environments. In section IV we describe the five major traps that we had to solve for porting our protocol to a real WSNP. We conclude this short paper with a mention of an alternative approach for simulating and testing protocol implementation for a specific WSNP and a short summary.

II. OMNeT++ / CASTALIA

OMNeT++ is a discrete event simulator for modeling communication networks, multiprocessors and other distributed systems [6]. It is open-source and fully implemented in C++ and designed to support network simulation in large scale. Its development was motivated by the needs of a powerful open-source simulation tool for academic, education and research use. OMNeT++ is available since 1997 and has a large community [7]. There have been registered downloads from over forty universities worldwide and the number of OMNeT++-related projects is still growing.

The OMNeT++ model consists of modules that communicate with message passing. A message can be either objects holding arbitrary data or simple events. Modules are instances of module types, which are written in C++ and use the simulation class library. The user can add functionality to its module
via one of two alternative programming models: (I) co-routing-based programming and (II) event-processing functions. A module of the co-routing-based model runs in its own thread and contains an infinite loop with send and receive calls. The thread gets control by the simulation kernel each time the module receives a message. The event-processing function is called by the simulation kernel and returns immediately after processing the message. Modules can be hierarchically grouped into a compound module, where the number of levels is unlimited. This helps the user to transparently split its implementation in several modules. OMNeT++ supports message passing within a single level of module hierarchy or within a compound module’s hierarchical structure. To the outside a compound module acts as a 'cardboard box', transparently relaying messages from their inside. Furthermore, connections can be parametrized with delay, data rate or bit error rate. Especially the module structure and the strict separation of "nodes" and "links" provided by OMNeT++ improve the reusability of the user’s model components and simplify their deployment to a real application. An important requirement for OMNeT++ was easy debuggability and traceability of simulation modules. It offers module output windows, inspectors and automatic animation as well as tracking object ownership, doing ownership-based automatic deallocations and detecting bugs caused by aliased pointers and misuse of shared objects.

Castalia is a simulator for WSN, Body Area Network (BAN) and generally networks of low-power embedded devices. It is based on OMNeT++ and has an advanced channel model, radio model, extends sensing modeling provisions and supports node clock drift. Furthermore, Castalia inherits the excellent modularity, reliability and simulation speed from the OMNeT++ framework [8], [9].

III. A CONFIGURABLE PROTOCOL STACK MODEL

Although the implementation of our FET-MHR protocol was driven by the requirements of the IQlevel project, where Operating System (OS) and hardware were developed from scratch, we decided to use a configurable protocol stack model. This model basically divides the protocol stack in a driver, a protocol and an application layer. These layers provide a basic separation among Hardware Abstraction Layer (HAL), Hardware Independent Layer (HIL) and application.

We are convinced that a well-defined protocol as part of the HIL could be easily ported to different environments by adding small adaptation layers. Although a network protocol may make an extensive use of OS primitives like timers or tasks and needs functions for sending and receiving data via the radio interface, these functions are typically provided by the application layer or the HAL and can be wrapped by the adaptation layers. The authors of [10] show that this approach is feasible for a various set of OSs and causes a minimal runtime effort and code size impact.

We implemented adaptation layers to encapsulate our FET-MHR protocol layer and to make it portable to multiple OSs and simulators. Figure 1 shows the adaptation layers for the different environments that already exist and the integration of our FET-MHR protocol.

Castalia and the Reflex OS [11] are implemented in C++ and must be compiled by the GNU Compiler Collection (GCC). Our FET-MHR protocol and the IQlevel OS are bare C code and can be compiled by the GCC as well as by the Texas Instruments (TI) compiler. This allows us to use the TI compiler for our specific WSNP, which generates more efficient machine code. Nevertheless, our build environment supports both compilers so that our protocol can be used in all these environments without any additional implementation effort.

IV. CRITICAL TRAPS ON PORTING SOFTWARE

As just mentioned our protocol stack model and our build environment allow us to use our implementation in different environments as well as compile it with environment specific compilers. But, we learned that the compiler and architectural differences are causing a various number of pitfalls. In the following subsections we will explain the most critical traps that we found during porting our protocol from a Castalia simulation, running on an x86 architecture, to a real MSP430-based WSNP.

A. Unaligned data access

The MSP430 has a 16-bit Reduced Instruction Set Computer (RISC) architecture, where any memory access must be aligned to a 16-bit address. Nevertheless, an alignment by software is not necessary. The Micro Controller Unit (MCU) masks the lowest address bit at any memory access so that an unaligned access never traps an error. For accessing 8-bit memory addresses the compiler uses special instructions, which mask the higher or lower byte of the 16-bit data word internally.

The x86 architecture handles an 8-bit memory access in a more native way. Special instructions are unnecessary and a programmer has not to take care about memory alignment. But, when porting software from x86 to the MSP430 architecture a type cast from a 8-bit data array to a larger data type can cause an unaligned memory access and due to the MSP430’s architecture the processor will not trigger any runtime error. It reads at the wrong address instead.

Especially in network protocol implementations packets are initially stored in an unspecified 8-bit data array. Later, an array’s subset is casted to protocol header specific structures to simplify access onto header elements. While this cast is working at any address on the x86 architecture on the MSP430 the structure must be aligned. To avoid this problem aligned protocol headers can be used only, any access must be done with 8-bit operations or the data must be copied to an aligned address.

B. Stack size

Due to the limited resources of a low power MCU the stack size is often few bytes only. Furthermore, a task implementation with isolated memory sections does not exist. The...
MSP430 does not have a memory protection unit to monitor memory access. A low power sensor node OS running on an MSP430 shares a single address space and stack among all its tasks.

The stack size is growing with function calls. The processor writes at least the return address and the frame pointer to the stack at each call. While these information do not consume as much memory each local variable, which can not be stored in a register, is stored on the stack too. Especially, large local objects as structures or arrays consume a lot of stack memory.

While in the Castalia framework running on a PC the stack size is quasi unlimited and local objects are very comfortable to use on a low power MCU the programmer must take special care about this. As a result a software written for the Castalia framework, which runs in a proper way, can cause a stack overflow on a low power MCU. Due to the single address space of a sensor node OS a stack overflow can overwrite any data without causing a runtime error. This makes this kind of error very hard to find.

A way to avoid this pitfall is to take special care about the usage of local variables and to forbid the implementation of uncontrolled recursions. Furthermore, barrier variables can be used to detected a possible stack overflow during runtime.

C. Interrupt handling

As mentioned in section II the OMNeT++ framework supports objects and events for passing messages to a simulation module. While objects are a good abstraction for receiving network packets the asynchronous behavior of an interrupt can not be sufficiently emulated. The delivery of an event is controlled by the simulation kernel and its execution is based on the kernel’s scheduling scheme. In addition to this the OMNeT++ supports an 'unlimited' event queue. Events are not dropped in case that the programmer is not explicitly implementing this.

The MSP430 serves an interrupt immediately after finishing the current machine code instruction. Furthermore, interrupts are blocked by the MCU during a running interrupt service. By that reason an well-designed OS interrupt service is separated into a bottom and a top half section. The bottom half function runs in the interrupt context where interrupts are disabled. It does the most necessary operations only. Complex operations are done in the top half of an interrupt service, which runs in an application context. This scheme guarantees that interrupts can be served as fast as possible and the loss of interrupts can be avoided.

During the comfortable event handling scheme of the OMNeT++ framework the user is wrapped into a sloppy interrupt service implementation. Complex operations are not identified and not moved to a top half function running outside the interrupt context. On a real WSNP such a sloppy interrupt service implementation causes an unpredictable behavior and its solution requires a lot of re-implementation work, which affects the simulation results too.

D. Run-time constrains

Beside interrupt handling the run-time constrains of low power MCU are quite hard to emulate by the OMNeT++ framework. Although Castalia supports packet delivery delays the emulation of execution delays caused by the restricted computing power of a low power MCU must be explicitly implemented and can not reflect the real behavior in a sufficient manner.

Especially the low power modes of the MSP430 make an useful emulation very hard. The execution time of an interrupt is significantly influenced by the current low power mode. Furthermore, the MCU is often running with a lower clock speed and is not equipped with a floating point unit. Both make the usage of complex mathematic operation very time-consuming. These run-time constrains must be kept in mind during protocol implementation, otherwise they will influence the behavior of the protocol in a non-negligible manner.

E. Compiler bugs

As mentioned before we used the TI compiler to build the IQlevel OS for our WSNP. Although the IQlevel build environment also supports the GCC the TI compiler is preferred by us. But the Castalia framework running on the x86 architecture can not be compiled by the TI compiler. We build it with the GCC.

While porting the FET-MHR protocol to our WSNP we learned that the TI compiler generates incorrect machine code
for call by value function calls with large objects. The MSP430 uses 16-bit registers to pass arguments to the called function. For objects larger than 16-bits more than one register must be used to hold all data. Due to a bug the compiler generates the correct machine code for the function call preparation, but the generated machine code of the called functions does not use the additional registers. It uses the first register only. The additional registers are overwritten without reading the data before. Again, this bug does not generate a run-time error and the trap is very complex to solve.

Although this trap is basically a compiler bug. It shows that the usage of different compilers for the Castalia framework and the real WSNP opens the possibility to toddle into additional errors, which are very hard to tackle later.

V. Cycle Accurate Simulator (CAS)

We are still convinced that the OMNeT++ simulation framework is a powerful tool set for testing and validating new, complex network protocols. It can be used for interoperability testing and allows an efficient testing of large networks. But the program code is compiled for the simulation’s host, which can differ from real WSNP in a significant manner. The traps that we found and explained in the previous section were quite hard to tackle.

But we are also convinced that for developing new network protocols for specific WSNP a simulator is necessary. In the last recent years a large set of CASs are developed. A CAS emulates the sensor node at the machine code instruction set level and uses the program code compiled for this machine type. We are confident that COOJA, simulator for the Contiki sensor node operating system, should be used for interoperability tests [12]. It is based on the MSPsim, a Java-based CAS of the MSP430 MCU [13]. It provides both a realistic simulation with accurate timing and good debugging. While using a CAS the program code must be compiled for the target hardware and the simulation covers all hardware specific constrains. Although this kind of simulation is less efficient than the OMNeT++ framework, we are convinced that the development effort is even less.

In [14] we describe an extension of the MSPsim by using a SystemC interface. The presented hybrid simulator can be used for a cycle accurate simulation in combination with testing new hardware components written in SystemC. It is planned to integrated the hybrid simulator in COOJA, so that the simulator can cover simulation from hardware up to interoperability test in an efficient manner.

VI. CONCLUSION

We presented our approach of a configurable protocol model for sensor node operating systems, which aims to reduce the effort for porting from one OS to another. Furthermore, we introduced our FET-MHR that we have implemented in the configurable protocol model. We used the Castalia framework for testing and validating the protocol. Due to our configurable protocol model approach we were convinced that we can use the same implementation in the Castalia and in our real WSNP.

But we had to learn that by using the Castalia framework we were wrapped into a sloppy protocol implementation, which was very hard to port to our specific WSNP. In this short progress report we presented five major traps that we found during our porting work. We described the trap’s background and explained why these are very hard to find and solve. In the last section we mentioned an alternative approach based on a CAS, which supports testing of compiled native code. We are convinced that this approach can help to avoid these kind of errors and helps to expose problems earlier.

ACKNOWLEDGEMENTS

The research leading to these results has received funding from the Federal Ministry of Education and Research (BMBF) under grant agreement No. 16 BN1110.