Modeling, hardware in the loop simulation and code generation for wireless sensor networks based on ZigBee and TinyOS platforms

Mohammad Mostafizur Rahman Mozumdar*, Anwar Al-Khateeb*, Luciano Lavagno*, Laura Vanzago†
{mohammad.mozumdar, anwar.al-khateeb, lavagno}@polito.it, laura.vanzago@st.com

*Department of Electronics, Politecnico di Torino Torino, Italy
†STMicroelectronics Milano, Italy

Abstract—Hardware and software platforms for Wireless Sensor Networks (WSNs) are almost as diverse as the application areas, with very limited standardization. This makes application development and debugging for these platforms extremely challenging. Often times same application needs to be rewritten for each platform and there is hardly any support for application debugging, except for a few blinking LEDs. Similar problems are solved in application domains that have several similar constraints, such as automotive, by the use of model-based design. We address the lack of model-based design tools for the WSN domain by providing a framework (based on Simulink, Stateflow and Stateflow Coder) in which an application developer can model a WSN application by using Stateflow constructs and then use a single model to perform multi-platform Hardware-In-the-Loop (HIL) simulation and platform-specific application code generation.

I. INTRODUCTION

Although WSNs have experienced great advances in the last decade, still application development in this domain is quite challenging. There is a lack of tools in this domain that can provide modeling, Hardware-In-the-Loop (HIL) simulation and automatic code generation for multiple platforms from a single high level abstraction. Just like other embedded systems, WSN applications need to be verified functionally before being implemented on the actual platform. But most of the available sensor nodes on the market (such as Mica, Telos, etc.) only provide a few on-board blinking LEDs for debug aids. This makes code development on the actual platform virtually impossible. The available functional analysis packages, such as TOSSIM for debugging of TinyOS application, OmNet and NS, fall into two main categories. One is very platform- and OS-specific (such as TOSSIM), and others are generic network simulators (such as OmNet, NS, etc.). Both have significant drawbacks when it comes to complex application development.

To ease WSN application development and improve platform independence (e.g. ZigBee, TinyOS, etc.), we propose a model based framework implemented primarily on top of the MathWorks tool chain. Application developers are able to model applications using Simulink/Stateflow, simulate network behavior with detailed or abstract channel models (including PHY and MAC layer protocols, in order to tune channel parameters for optimization of required metrics), and perform HIL simulation by interacting with the sensor nodes. To remove platform dependencies from Stateflow modeling, we developed a set of generic interfaces (for sending/receiving packets and also for acquiring data from different sensors) and an event mechanism, which ultimately allows developers to specify applications in Stateflow without requiring detailed knowledge of ZigBee or TinyOS coding. After HIL simulation, our framework can generate the complete application code for ZigBee and TinyOS from a single Stateflow application model.

Our proposed framework is an extension of the work described in [1], which did not have HIL simulation capability. In [2] and [3], the authors address problems like simulation, debugging or code generation separately, but do not simultaneously support multi-platforms HIL simulation and code generation from a single high level abstraction.

II. PLATFORM INDEPENDENT MODELING FRAMEWORK

The complete framework is depicted in figure 2. The WSN algorithm (application or middleware) is initially modeled
by a Stateflow block. While modeling, the application developer uses Stateflow constructs (such as state, transition diagram, event, etc.) which are independent of the specific APIs and programming languages for TinyOS or ZigBee, and uses a set of library functions to interact with sensors. The sensors physically reside on the node, and are transparently accessed either by using a serial cable, or by using pre-recorded streams saved in Matlab. This algorithm block is connected to other Stateflow / Simulink blocks that generate events from other software or hardware components of the platform (such as CLK for periodic scheduling and PKT for radio data reception). In order to transparently support HIL simulation, these events can also come from Simulink models or from the actual node over a serial connection. The Stateflow model can be refined repeatedly, in order to tune the functionalities and performance of the algorithm. At the end of development, the same model also serves as an input for the code generation phase. The first step is to generate C from the Stateflow algorithm model, to be automatically customized for the different platforms, using Stateflow Coder. The next step is to adapt the generated ANSI C code to the target platform. Target Language Compiler (TLC) scripts are used for this purpose. TLC provides mechanisms by which one can generate platform specific code by creating sections (such as includes, defines, functions, etc) from ANSI C code and also by adding custom code for the target platform.

### III. Hardware In the Loop Simulation

HIL simulation has two main components on the Simulink/Stateflow side: the node stub and the network model (which can be simulated in Simulink/Stateflow or implemented over the air). The communication between them is done by a serial or USB cable. Simulink blocks like pulse_generator and serial_port_packet_reader provide events from the actual hardware to the algorithm block, and vice-versa. We developed several Matlab functions (such as getAccelerometerAxisValue, getPacketPayload, sendPacket, etc) by which Stateflow can interact with the stub or other Simulink components. The stub contains minimal code to join or interact with a ZigBee or TinyOS network, without any application-specific part. When it receives a packet from the network, it stores the packet locally and in the next request from the Stateflow model, it transfers the packet payload. In the same way, when the stub receives a packet from the Stateflow model, it constructs the actual packet and transmits it to the network.

### IV. Multi-Platform Code Generation

Stateflow coder can generate ANSI C code from a Stateflow diagram. The computational bodies of ZigBee tasks and event handlers of TinyOS are essentially written in C. So, the code generated from Stateflow code can be directly used in ZigBee and TinyOS with very small, easily automatable modifications. For code generation, we disconnect the WSN algorithm block (Stateflow model) from stubbed input and output signals, and change these signals into internal events. After that, we need to delete all library functions that are used in Stateflow modeling and create a C file that contains empty definitions of these functions, just in order to allow Stateflow Coder to complete its job. Real implementations of these functions will be added later in the platform specific code generation phase. After completing these steps, we can generate ANSI C code for the WSN application block. The next step is to generate platform specific code from the C code by using TLC scripts. Application code for TinyOS and ZigBee looks very different, so the TLC script performs the following tasks to generate platform specific code:

- Generate platform specific base code.
- Generate platform specific application files by taking different sections (such as includes, defines, functions, etc) from C code of the Stateflow.
- Generate platform specific implementation of the library functions and
- Generate make or configuration files for each platform.

### V. Conclusion

We described an extensible framework for modeling, HIL simulation and multi-platform code generation of sensor network algorithms based on MathWorks tools. The reason for choosing the MathWorks tools over, for example, TOSSIM, NS, OmNet, is that they are well known and already provide rich libraries for digital signal processing and control algorithm behavior simulation. They also provide extensible mechanisms for efficient code generation and platform-specific re-targeting.

### REFERENCES

