Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 00 (2019) 000–000

Procedia MANUFACTURING

www.elsevier.com/locate/procedia

5th International Conference on System-Integrated Intelligence

A Hybrid Approach for Digital Representation of Sensors in Real-Time Applications

Karthik Shenoy Panambur[a,*], Shantanoo Desai[a,b], Amit Kumar Singh[a], Klaus-Dieter Thoben[a,b]

[a]*BIBA – Bremer Insitut für Produktion und Logistik GmbH, Hochschulring 20, Bremen 28359, Germany*
[b]*Faculty of Production Engineering, University of Bremen, Badgasteinerstraße. 1, Bremen 28359, Germany*

* Corresponding author. Tel.:+49 0 421 218 50111. *E-mail address:* she@biba.uni-bremen.de

Abstract

We propose a hybrid approach for peripheral device emulation of sensors on hardware platforms as well as hardware virtualization platforms. The proposed approach stems from challenges faced in IoT (Internet of Things) application development, namely, hardware selection and evaluation, software development for the hardware under consideration, and reliable reproducibility of application tests. The hybrid approach provides strong support for all development phases of IoT products and system development in the absence of physical peripheral sensor hardware. We also present an event-driven software architecture for different components and a modular workflow for implementing the emulator. Within the approach, a peripheral device emulator is conceivable by using the event-driven software architecture and modular workflow. Finally, we introduce the term *Simulatable Datasheet*, which is realizable by configuring the emulator as per technical documentation like datasheets or application notes. The *Simulatable Datasheet* provides an interactive interface to improve decision-making during the product development phase. The *Simulatable Datasheet* thus relaxes the need for hardware availability and eases testing scenarios during IoT real and non-real-time application development without modification in the firmware. Additionally, these datasheets can reduce efforts for hardware and software integration and enable faster debugging, which in turn reduces the overall time to deploy IoT applications in various fields.

Keywords: Internet of Things (IoT), emulation, simulation, peripheral device emulation, simulatable datasheet, sensors

1. Introduction

The IoT has significant potential in enhancing and influencing growth in many sectors such as manufacturing, health, consumer electronics, education, to name a few. Bringing the IoT to sector-specific components opens up the possibility for data-driven decision strategy in business using ubiquitous information obtained from a plethora of heterogeneous devices and systems [1], thus instigating many public and proprietary organizations to invest their resources in IoT research, system development, platforms, and frameworks.

A few aspects addressed by these frameworks and platforms are effective utilization of limited hardware resources, interoperability at software, hardware and system-level, scalability, flexibility, real-time and deterministic behavior at application level [1,2,3]. These aspects make IoT solutions challenging in terms of design, development, and testing. Typical IoT solutions involve multi-level, complex architectures stemming from firmware to the high-level application [1]. Thus, the typical IoT development leans towards an agile software development approach where development and testing are performed incrementally [4].

1.1. Challenges

Even with maturing IoT development platforms and frameworks, IoT system development is a complex task and deals with the following challenges:

- Hardware selection and evaluation.
- Hardware dependent development of software.
- Setting up reliable and reproducible tests.

We discuss these challenges with the impacts they impose on two sectors, namely industry and community sectors of open-source, education, and research. It is worth noting that addressing the challenge of setting up tests in a reliable and reproducible manner influence both the industry as well as community sectors, as discussed subsequently.

1.1.1. Hardware selection and evaluation

Industrial Impact: Hardware selection and evaluation are carried out at the early stages of IoT system development involving hardware and significantly impacts system and software design, time-to-market, and cost. The task of selecting and evaluating hardware devices for given application requirements is difficult and time-consuming due to the availability of multitudinous hardware components such as sensors, of varied specifications, and cost. Moreover, such a task requires an understanding of text-based information in the form of datasheets, application notes, which are usually verbose and less intuitive. Thus, requiring a designer with expert knowledge of the system and domain to carry out the task, consuming significant time efforts and resources. Further, the risk of wrong selection of hardware can have a substantial negative impact on the system along with the product development cycle, as it may require re-design or compromising features in the product. This challenge necessitates a need to have a platform where device specification can be evaluated against application requirements quickly and more intuitively.

Impact on open-source, research, and education: Open-source and academic institutions possess a limitless potential and play a vital role in the application and adoption of technology among masses and industry. By realizing such potentials and opportunities, many silicon chip manufacturers attempt to provide low-cost evaluation boards to promote their products and services. These efforts can be further enhanced by providing a low-cost simulation of hardware and emulators, where users can test and evaluate their concepts before buying physical hardware for prototype development.

1.1.2. Hardware dependent development of software

Industrial Impact: IoT system development usually involves interaction with hardware components such as sensors and actuators. Embedded software development and testing have a significant dependency on the availability of hardware. In effect, it hinders firmware development and testing, hence losing a significant amount of time waiting for hardware prototypes and fixes. This dependency of software on hardware also increases the risk of incompatibility during integration, increasing testing efforts, and time-to-market. Thus, the need for high fidelity hardware simulation with which IoT application can be developed and tested without the initial requirement for the availability of physical hardware.

Impact on open-source, research, and education: Many research and open-source communities in IoT projects face the unavailability of hardware, which gravely hinders the software development and testing efforts. Besides, prototypes built by participants may not work as expected due to issues related to hardware-specific configurations. Thus, this demands solutions comprising of high fidelity, and easily sharable simulation tools across various participants within these IoT projects.

1.1.3. Setting up reliable and reproducible tests

Industry and community impacts: IoT systems require tests before deployments. Most of the tests for such systems necessitate complex environment setups and, in many cases, expensive field trials. These environments and field trials are not easily reproducible.

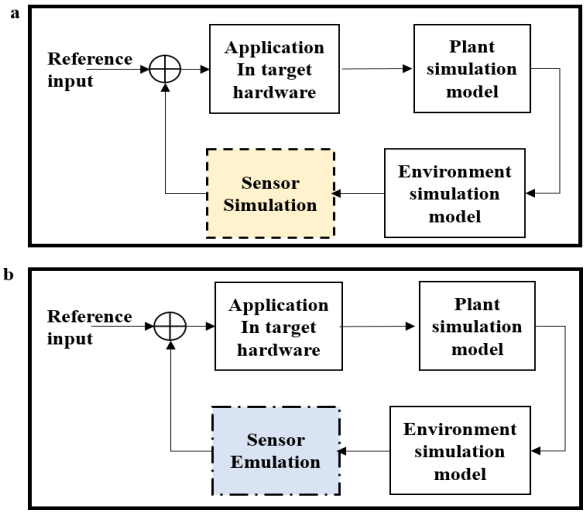

Fig. 1. (a) Processor-in-Loop (PIL) Block Diagram with Sensor Simulation; (b) Block diagram of sensor Emulation-in-Loop

The industry tackles the problem of testing via Processor in Loop (PIL). The community sector most often relies on licensed PIL solutions, which are cost-intensive. Fig.1 (a) illustrates the block diagram of PIL where peripheral devices used in IoT systems along with the plant, are replaced with simulation models, and the application under test runs on a target processor. PIL approach is very well suited for application testing but does not provide complete test coverage in terms of low-level hardware interaction.

These challenges hence demand a need for a high fidelity hardware peripheral emulator that mimics the peripheral device effectively, as shown in Fig 1 (b). The sensor emulator interacts with the virtual environment making it easier for setting up tests and lowering the cost to create reproducible complex test scenarios.

2. Related Works

This section provides an overview of current research trends that lean more in the area of device virtualization and emulation in IoT. Sendorek et al. [1] propose a testbed to provide end-to-

end testing of IoT systems from an application point of view. The principle idea is to immerse the IoT system in a virtual environment with three levels of immersion. These levels depend on whether the IoT device or the sensor are virtualized or not. Minotti et al. [5] propose the implementation of an emulation tool for microelectromechanical systems (MEMS) to be used for the characterization of MEMS. Here the authors discuss the method to realize synthesizable transfer function and several second-order effects of MEMS (such as noise, spurious modes, feedthrough capacitance) on Field Programmable Gate Array (FPGA). Szydlo et al. [6] propose the concept of a virtualized platform for testing IoT applications, which enables the creation of virtual devices (such as sensors) driven in a virtual environment illustrating real-world scenarios for real embedded device testing. The authors further discuss different layers in GNU/Linux, which are involved in sensor virtualization and model analog sensing elements with an ideal Analog-to-Digital Converter (ADC) to illustrate a temperature sensor. However, these works describe the sensor emulation term broadly without a detailed workflow to define sensor behavior consisting of analog and digital attributes like sense element model and register maps.

IoT simulators provide a deep insight into modeling the system and network behavior through virtualization. Chernyshev, Baig et al. [3] provide an extensive survey of various simulators with particular areas of interest. The iFogSim full-stack simulator supports fog computing. The simulator supports sensors, actuators, and application processing modules, which can be set into a topology that matches the real scenario. Two simulators, namely, IOTSim and SimIoT, focus on the application layer for analyzing Big Data processing capabilities of IoT application. COOJA is a Wireless Sensor Network (WSN) simulator for the ContikiOS operating system that enables cross-level simulation [7]. With cross-level simulation, COOJA is both capable of running slower instruction-level emulation of devices firmware and run the faster simulation on JAVA based nodes, thus combining simulation at sensor node hardware level as well as a simulation at high-level behavior in a single run [7]. COOJA supports hardware emulation and portrays a promising outlook as a tool for hardware virtualization platform for peripheral device emulation. However, all these simulators are yet to provide a workflow for peripheral sensor device emulation.

We propose an amalgamated approach of simulation of analog attributes and emulation of digital attributes of a sensor, along with an event-based architecture and detailed workflow, which, as a whole, will provide a digital representation. This approach is discussed subsequently.

3. Peripheral Device Emulation for *Simulatable Datasheets*

We conceptualize a hybrid approach to virtualize peripheral devices like sensors using simulation and emulation by taking into consideration the standard source of information such as datasheets, application notes. This approach introduces a new term, *Simulatable Datasheet*. *Simulatable Datasheets* are not a replacement or a direct alternative for vendor datasheets but rather used in conjunction with them. Traditionally, manufacturers provide datasheets for devices that are often verbose and non-interactive. Graphical User Interfaces (GUIs) are more interactive compared to the text within such datasheets to represent sensor behavior and characteristics in the form of graphs, lookup tables, etc.

Simulatable Datasheets are tools that functionally mimic a peripheral device like sensors as mentioned in the datasheets such that IoT application does not need any code changes for end-to-end IoT application. These datasheets provide the following benefits:

- *Interactive*: *Simulatable Datasheets* can interact with virtual environments and users via a GUI. The user can evaluate different features of the sensor and relate to the application requirements. The GUI can be used for user-specific testing and development as well.
- *Try-Before-Buy*: It facilitates the evaluation of peripheral devices without the need for physical hardware in the initial stages of development.
- *Decoupling*: Embedded software can be developed and tested on emulated peripherals without any modification in the application developed hence reducing time and effort for software development and testing while improving the test coverage.
- *Sharable*: Sharing enables open-source communities to develop and test applications without the need to have physical hardware, thus enhancing collaboration across the community. It also supports solving hardware-software incompatibility issues, which usually stems from differing hardware and software versions, updates, and setups.

3.1. Typical Digital Sensor Representation

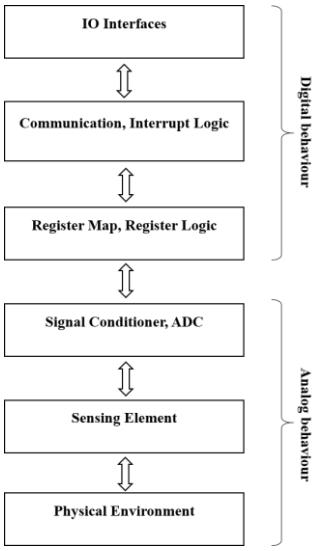

Fig. 2. Functional Block Diagram of a Digital Sensor

A typical digital sensor representation is, as shown in Fig. 2. The sensing element detects changes or measures physical properties from the physical world and converts into suitable electric signals. The signal conditioner manipulates analog electrical voltage or current to improve linearity, Signal to Noise Ratio (SNR) by filtering noise and amplifying to suitable electric signal level before sending it to ADC. The ADC converts analog voltage to a discrete digital value that is stored in registers, which are an array of memory locations to store

data. Register map stores sensor measurements, events, interrupts, other status control flags, and configurations. Registers are identified by addresses and have a fixed length of 8, 16, 32 bits with assigned read/write access control. Within each register, bits are logically grouped together as we term as "bit groups" and are interpreted as described by the vendor in the datasheet. Register logic defines sensor behavior during reading/writing a register. The configuration, status, and control registers of a typical sensor like Texas Instruments' OPT3001 light sensor, usually outline the behavior for software interrupts, communication, and information exchange with interfaced embedded devices such as a microcontroller, etc. Interrupt logic defines the sensor's digital behavior for handling interrupts, which are not controlled by register logic directly, such as setting an Input/ Output (IO) pin on interrupt, etc. Communication logic implements communication protocols and an appropriate physical layer to exchange information with an interfaced embedded device. IO interface provides access to IO pins of the sensor hardware.

3.2. Hybrid Approach for Peripheral Device Emulation (PD-EMU)

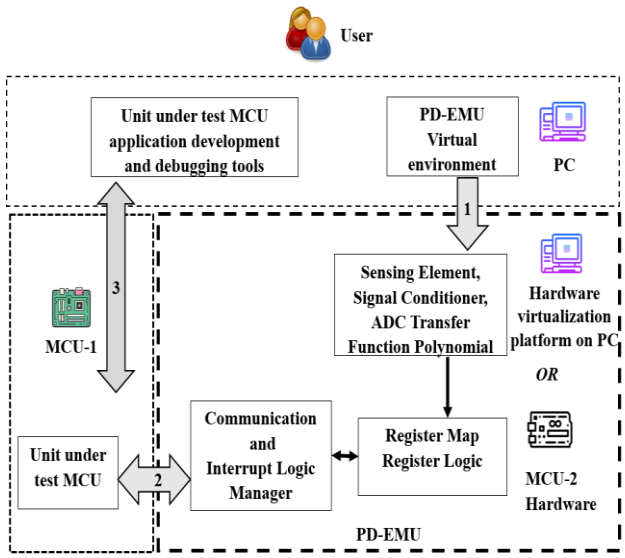

Fig. 3. Hybrid approach for PD-EMU with vital components

We broadly classify the block diagram of a sensor in Fig. 2 into two attributes, namely, analog and digital. Fig. 3 illustrates the hybrid approach for PD-EMU for a digital sensor. The PD-EMU virtual environment can mimic the behavioral characteristics of the real-world scenarios based on environmental models (e.g. physics engine, equations etc.). Typically, the user using graphical interface defines parameters, conditions or equations for these environmental models. This virtual environment interacts with the analog attributes of the sensor such as sensing element, ADC etc. via either hardware virtualization or a communication interface between and external hardware platform (MCU-2). The output of the ADC updates the register map and logic of the platform. The hardware under consideration subjected to the IoT application test obtains the data from the register map via communication and interrupt logic managers. The Interrupt Logic Manager processes the interrupts in the emulated sensor, which further interacts with the IO interface to service external interrupts. The internal software interrupts are serviced in an event-driven manner. The communication protocols and behavioral logic associated with the communication interface are implemented in the Communication Logic Manager. The user interacts with the hardware for IoT application (MCU-1) for debugging and development purposes.

The aforementioned event-driven manner is a part of the digital attribute of the peripheral device. A detailed description of these attributes is explained subsequently.

3.2.1. Sensor modelling

Analog attributes of the sensor such as sensing element, signal conditioner, ADC, as illustrated in Fig. 3, are simulated using transfer function polynomial deduced from response curves, lookup tables, equations as mentioned in the vendor datasheet. These transfer functions are obtained via curve fitting algorithm such as linear, quadratic, cubic, etc. as depicted in Fig. 4. In simulation models, these transfer function polynomials are used to create functional modules encapsulated within function blocks, which can optionally be connected with other functional blocks as illustrated in Fig.5 to form a chain of input-output relationships within the model.

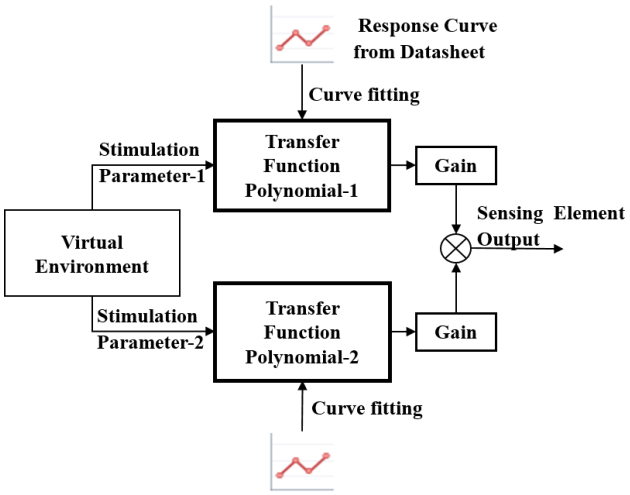

Fig. 4. Sensing Element Modelling using data from vendor datasheet

For high fidelity simulation, more detailed information like linearity, sensitivity, gain factor, measurement drift across the temperature range, etc. are added to the functional modules. The proposed workflow and software architecture subsequently allows us to add, remove, change, and update custom simulation parameters and modules.

3.2.2. Event-Driven Architecture

Digital attributes of peripheral devices such as register map interrupt logic behavior, communication, and IO interface, are emulated in an event-driven manner as illustrated in Fig. 5.

The emulation of such attributes is governed by event-driven software architecture. Within this architecture, Event

Dispatcher sends the event from the associated functional block to the connected block's Event Loop. Event Loop acts as a transporter of events and delivers it to the Event Processor of the associated block. The Event Processor consumes the relevant data and provides it to the functional module. This event-based approach introduces modularity by decoupling modules from each other. It also enhances extensibility in terms of adding custom features, scalability in terms of adding and removing more modules, and less CPU intensive due to non-polling nature. Further, this approach enables code generation due to modularity.

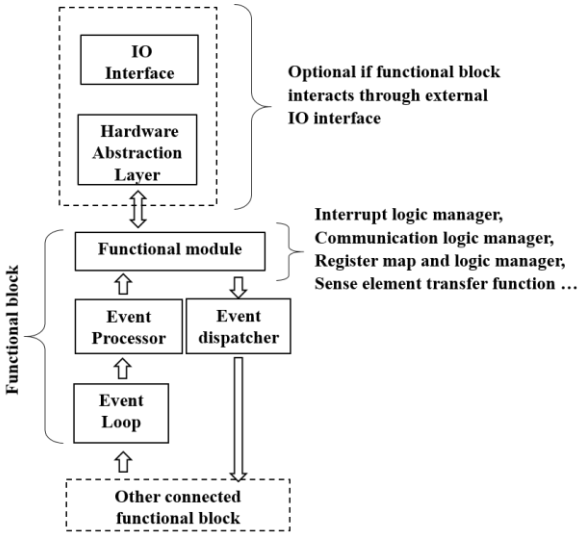

Fig. 5. Functional Block with the Event-Driven Architecture

Sensor's digital behavior is closely tied to events such as read/write activities in the sensor register and thus influencing events on the same or other functional blocks of the emulated sensor. These behaviors are modelled in Register Logic module in Fig. 3. Interrupt Logic Manager processes the interrupts in the emulated sensor, which further interacts with the IO controller to service external interrupts, and the internal software interrupts are serviced in an event-based manner. The Communication protocols and behavioral logic associated with the communication interface are implemented and handled by the Communication Logic Manager in an event-based manner.

3.2.3. Modular Workflow description

In the proposed workflow, the user represents different modules of the peripheral sensor device such as transfer function polynomial, ADC, filters, etc. as functional components in a block representation. The block representation layer provides a GUI, enabling users to add and configure different modules in a flow-based programming methodology. Here users can represent data flow interaction between functional blocks by directly inter-connecting them. The tool implementing the workflow can provide fundamental functional blocks by default, which the user can configure and use. The workflow also allows the creation of custom functional blocks according to user-specific code. Each functional block is a graphical representation of a code template, a reusable code snippet in the form of text. The connection between the blocks represents the data flow in an event-based approach, as depicted in Fig.5.

The interconnected blocks generate configuration files that can be used further by template generation engines. The code-template engine reads the configuration files and generates source code files. Subsequently, the generated source code along with platform-specific libraries are cross-compiled together to generate executable binaries for a specific processor architecture, as shown in Fig. 6. Later, the generated executable binaries are either flashed into the target hardware using flashing tools or can be executed on a Hardware Virtualization Platform (HVP).

As a practical realization, PD-EMU is capable of running as an application on dedicated embedded hardware like Espressif Semiconductor's ESP32, running Real-Time Operating System (RTOS) like FreeRTOS, and also on hardware virtualization platforms like Quick Emulator (QEMU) emulating a specific processor such as ARM Cortex-M also running on an RTOS. In addition, PD-EMU is also capable of running as an application on non-real-time operating systems but does not guarantee any timing constraints of the device being emulated.

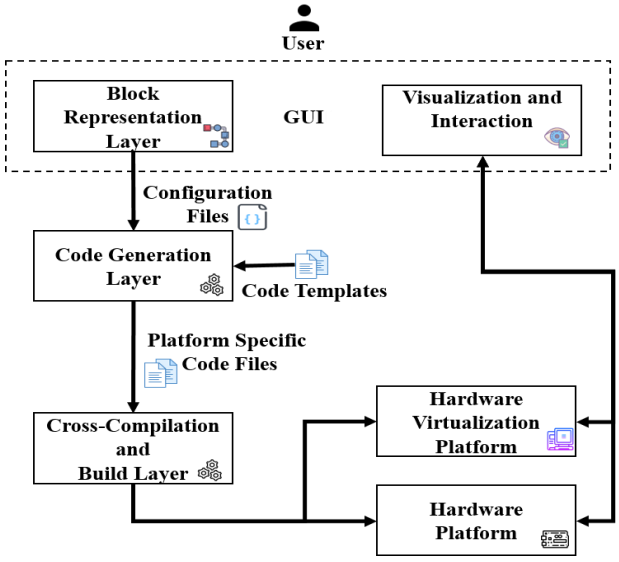

Fig. 6. Modular Workflow block diagram for the hybrid approach

4. Current Implementation and Future Works

As an initial validation for the hybrid approach of PD-EMU, we modelled the sensing element of the OPT-3001 light sensor from Texas Instruments. We created an initial version of a *Simulatable Datasheet* for the mentioned sensor by transferring the information like a normalized response for wavelength, temperature and luminance angle as well as an output luminance versus input response into their respective transfer function polynomials from the vendor datasheet. These transfer function polynomials were combined for modeling the sensing element as illustrated in Fig.4. An initial implementation of the event-driven architecture is being validated for the same sensor on an ESP32 hardware platform running FreeRTOS.

As a part of the future work, we are currently progressing towards realizing the modular workflow and generating a Simulatable Datasheet for sensors. Performance verification and validation of the modular workflow for Simulatable Datasheets will be subsequently performed alongside actual physical sensors.

5. Conclusion

The paper conceptualizes a hybrid approach using both simulation and emulation to virtualize peripheral devices like sensors. Additionally, we introduce to the new term and tool called *Simulatable Datasheet*. A Simulatable Datasheet provides an interactive tool that describes vendor datasheet more intuitively for the user and becomes beneficial for embedded software development. Furthermore, we define a workflow for implementing Simulatable Datasheets using the proposed hybrid approach of simulation and emulation. Initial validation of the proposed approach and workflow is being conducted for an implementation for a light sensor OPT3001 from Texas Instruments and testing it on a 32-bit multicore embedded hardware.

Acknowledgement

This research is partially funded by the European Commission within the H2020 Project RAINBOW (An Open, Trusted Fog Computing Platform Facilitating the Deployment, Orchestration and Management of Scalable, Heterogeneous and Secure IoT Services and Cross-Cloud Apps) Grant No. 871403, for the period between January 2020 – December 2022

References

[1] J. Sendorek, T. Szydlo, R. Brzoza-Woch, Software-Defined Virtual Testbed for IoT Systems, Wireless Communications and Mobile Computing 2018 (2018) 1–11. https://doi.org/10.1155/2018/1068261.T. Ninikrishna, S. Sarkar, R. Tengshe, M.K. Jha, L. Sharma, V.K. Daliya, S.K. Routray, Software defined IoT: Issues and challenges, in: I.I.C.o.C.M.a. Communication (Ed.), Proceedings of the International Conference on Computing Methodologies and Communication, ICCMC 2017: 18-19 July 2017, IEEE, [Piscataway, NJ], 2017, pp. 723–726.

[3] M. Chernyshev, Z. Baig, O. Bello, S. Zeadally, Internet of Things (IoT): Research, Simulators, and Testbeds, IEEE Internet Things J. 5 (2018) 1637–1647. https://doi.org/10.1109/JIOT.2017.2786639.

[4] L. Cordeiro, R. Barreto, R. Barcelos, M. Oliveira, V. Lucena, P. Maciel, Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach, in: J. Leaney, J. Rozenblit, J. Peng (Eds.), ECBS 2007: 14th Annual IEEE International Symposium and Workshop on Engineering of Computer Based Systems proceedings raising expectations of computer-based systems March 26-29, 2007, Tucson, Arizona, IEEE Computer Society; [IEEE], Los Alamitos, CA, [Piscataway, NJ], 2007, pp. 195–202.

[5] P. Minotti, L. Gaffuri Pagani, N. Aresi, G. Langfelder, MEMS Emulator: A Tool for Development and Testing of Electronics for Microelectromechanical Systems, J. Microelectromech. Syst. 27 (2018) 321–332. https://doi.org/10.1109/JMEMS.2018.2803683.

[6] T. Szydlo, J. Sendorek, Leveraging virtualization for scenario based IoT application testing, in: Communiation Papers of the 2017 Federated Conference on Computer Science and Information Systems, IEEE, 2017, pp. 229–235.

[7] F. Osterlind, A. Dunkels, J. Eriksson, N. Finne, T. Voigt, Cross-Level Sensor Network Simulation with COOJA, in: Proceedings 2006 / 31st IEEE Conference on Local Computer Networks: No. 2006, [Tampa, Florida, U.S.A. ; the 1st International Workshop on Practical Issues in Building Sensor Network Applications (SenseApp 2006), the 1st Workshop on Network Measurements (WNM 2006), the 2nd International Workshop on Performance and Management of Wireless and Mobile Networks (P2MNet), the 6th International Workshop on Wireless Local Networks (WLN 2006), the 2nd Workshop on Network Security (WNS 2006)], IEEE Operations Center, Piscataway, NJ, 2006, pp. 641–648.