

Rapid Embedded Systems Prototyping – an effective approach to embedded systems development

Robert Brzoza-Woch, Łukasz Gurdek, Tomasz Szydło
AGH University of Science and Technology
Al. Mickiewicza 30, 30-059 Krakow, Poland
Email: robert.brzoza@agh.edu.pl

Abstract—In this paper we introduce the Rapid Embedded Systems Prototyping (RESP) approach aimed at accelerating the development of novel, experimental, and proof-of-concept implementations of embedded devices based on microcontrollers and Field Programmable Gate Array (FPGA) chips. It is intended to be used in the fast-paced business environment in which an early working prototype is required. The RESP approach can be useful for remote developing and temporary monitoring of various embedded devices: primarily for resource-constrained IoT platforms, microcontroller-based sensor nodes, and customized ad hoc systems. The RESP-compliant system uses a central server and one or multiple Remote Reconfiguration and Monitoring (RRM) modules. Each RRM allows the software developers to manage reprogramming and monitoring of multiple target embedded devices. It can be applied to a device that needs to be remotely reconfigured, tested, or reprogrammed in its target environment without implementing a reliable bootloader. The RRM described in this paper has been successfully implemented and its functionality and performance have been tested.

I. INTRODUCTION

IN COURSE of research projects and at an early stage of embedded systems development cycle there is often a need to quickly develop a working prototype of an embedded device which will operate in its target environment as an IoT node or as an innovative solution for control, acquisition or monitoring purposes. In the start-up business environment, a client or an investor may wish to see a demonstration of a working proof-of-concept prototype of an application-specific embedded system. In a traditional approach, an embedded device is handed to a customer after the device's software is developed to a point in which it has full functionality and, usually, its bootloader program is working to update the device's program memory or its configuration. Developing a fully functional embedded device with a reliable operating bootloader may be a tedious and time-consuming task.

When utilizing an adaptive and evolutionary approach to software development, the functionality of the final product may not be precisely specified at an early development stage. It can be beneficial to start developing the application-specific embedded hardware and software for demonstration purposes and then to continue the product development when initial results are evaluated in practice and client's expectations are more specific. In this article we discuss various aspects of

this strategy which we call the Rapid Embedded Systems Prototyping (RESP). The approach is based on fast prototyping iterations with an ability to remotely reconfigure or reprogram a hardware platform using the Remote Reconfiguration and Monitoring (RRM) module intended to support the RESP development. The idea of RESP is to deliver a working prototype of an embedded device as fast as possible to a client and demonstrate its functionality. Then, the product is evaluated in its target environment by the client or the investor. The product's operation can be monitored and its software functionality may be easily tested or changed using the remote reconfiguration and monitoring capabilities. Thus, following the RESP approach can lead to more competitive time of proof-of-concept prototype development which, in turn, may result in gaining swifter funding for a project and better time-to-market.

In the domain of extremely resource-constrained, cost-sensitive, and tentative ad hoc devices with short-term support, the RESP combined with RRM can also be useful during the development in a target environment instead of a bootloader. We hypothesize that in those classes of embedded devices the bootloader itself could even be omitted and the development can be done using the RESP approach with RRM.

The described RRM subsystem is primarily intended to be utilized for embedded software development purposes when the device under development is placed in its target environment, but a reliable remote program memory upload or reconfiguration with a bootloader is not available or is not yet developed. After the successful software development with RESP approach, the RRM module can be extracted from the target platform and the device may work as a stand-alone unit. Alternatively, depending on a specific use case, the RRM can be utilized for the embedded system long-term testing by monitoring its operation with a set of sensors and digital interfaces. Then the RRM will operate as a remote sensing node.

The concepts described in this paper can be applied to multiple classes of embedded systems including microcontroller-based IoT platforms. The RESP approach is applicable for embedded platforms (a) based on microcontroller units (MCUs) in which program memory can be reprogrammed using common in-system programmers or debugging interfaces or (b) platforms based on Field Programmable Gate Arrays (FPGAs). When using FPGA hardware platforms, the presented RRM

The research presented in this paper was partially supported by the National Centre for Research and Development (NCBiR) under Grant No. LIDER/15/0144/L-7/15/NCBR/2016.

device can be used for convenient remote development of both programmable hardware and embedded software. Also, if the RESP augmented with RRM is utilized for programming or reconfiguration of a remote device under development, the Internet connection will be required. Alternatively, the system can be utilized in a local network and in that case the Internet connection is not necessary.

The physical hardware development is not in the scope of this paper as present-day ad hoc prototypes can be based on ready-to-use computing platforms such as well-known Arduino boards family or inexpensive evaluation boards offered by many semiconductor manufacturers, for example STM32 Discovery or Nucleo series from ST Microelectronics or Freedom FRDM from NXP. Those boards can be enhanced with a wide selection of sensor or actuator modules equipped with common integrated extension circuits available at low cost. In the case of the FPGA-based platforms, we assume the use of an already developed physical hardware. The RRM can then be utilized not only to modify the embedded software but also the programmable part of the hardware project.

The rest of this paper is organized as follows. In Section II we summarize recent related research in the area of effective approaches to embedded systems development and remote reconfiguration. In Section III we present the RESP method of developing embedded device prototypes in time-critical manner and in fast-paced business competition environment. Section IV provides a general idea on how to implement a RESP-compliant system. Section V describes the construction methodology and a sample implementation of the RESP with RRM device. Finally, in Section VI we summarize our work.

II. RELATED RESEARCH

The embedded hardware platforms are characterized by their diversity when compared to the general-purpose computing platforms. Some of the differences between embedded software and computer application development are the results of the fact that the embedded software must tightly cooperate with usually non-standard, specialized hardware platform and a set of peripheral devices. An embedded software developer needs an access to either a good simulation environment or to the real hardware platform. At the very early development stages, the product requirements are not yet fully specified and they may change multiple times. Engineers can then utilize a general-purpose solution from a wide portfolio of ready-to-use sensing, data acquisition, and actuation devices controlled with e.g. National Instruments hardware and software solutions. However, a prototype can be more enticing for the investor if it could be backed up with a demonstration of a custom working hardware and software even at an initial development stage.

In commercial and industrial MCU firmware development practice, the bootloader is one of the most specialized software parts. There are multiple MCU platforms that provide a dedicated bootloader without any additional installation, but that solution usually relies on a predefined interface and protocol. Changing the default settings requires either to re-implement

the bootloader or to modify it. The Nordic Semiconductor's nRF52832 is a good example of integrated circuits (ICs) which provide a very well developed Bluetooth Low Energy (BLE) bootloader. However, not all ICs can utilize BLE to update firmware and not all manufacturers provide such a convenient firmware update functionality. Usually, a bootloader is a very application-specific part of embedded software and it needs to be either developed solely for a given platform and interface or ported from another project. Embedded systems based on an application microprocessor (with Memory Management Unit and running e.g. Linux) often use U-Boot as a first stages' bootloader. For the MCU-based embedded devices it is difficult to point out a most common bootloader solution – different platforms offer different solutions, such as the STM32 Bootloader [1]. An example of a custom bootloader is described in [2]. The bootloader program must be well designed and tested to avoid firmware corruption eventually resulting in an inability to reprogram the device with the provided bootloader. Another common problem in writing a bootloader is to make it insensitive to transmission errors and complete transmission interruption. Preventing those situations require much time, engineering effort, and some design redundancy (additional memory, correcting errors in software, etc.). All the problems mentioned here can be solved, but it usually costs additional development time.

The ability to remotely reprogram, reconfigure, and super-vise an embedded system is especially useful in the domain of programmable logic, mainly FPGA. We should also be aware that the software and hardware development flows may proceed in parallel, depending on a design (e.g. in [3]). The reconfiguration allows developers and maintenance staff not only to remotely update firmware, but also to change the functionality of the system [4]. For example the FPGA-accelerated smart camera described in [5] is able to run multiple configurations which can be substituted depending on a higher level adaptation policy. Enhancing an FPGA-based device with the remote reconfiguration feature costs design issues due to losing the programmable logic functionality during the reconfiguration process [6]. In that case the partial reconfiguration feature [7], [8] could be helpful, but it tends to complicate the hardware-software design flow hence it may be ineffective for time-critical project. Less sophisticated, but much more convenient methods include using remote programmers, such as the Intel FPGA Ethernet Cable (formerly the Altera EthernetBlaster II) as described for example in [9]. Those devices offer only limited computing capabilities at the target side. In our solution we greatly increased the computing power of the remote programmer by utilizing a single-board computer (SBC). It allowed us not only to easily implement modern communication protocols, but also to gain much more flexibility compared to other solutions.

The idea of remote programming can be extended to the remote firmware management. It is also a well-known topic, and some aspects of networked systems performing such tasks are patented e.g. in [10], [11]. Currently the remote reconfiguration and management are, however, typically per-

formed by using similar architecture and by utilizing e.g. OMA Lightweight M2M (LWM2M) protocol. The LWM2M is an increasingly popular remote management protocol for intelligent connected and IoT devices [12], [13]. It has low transmission overhead and its implementation can be relatively easily ported to many connected platforms. LWM2M also supports a framework for a remote firmware update which was especially desirable in the solution described in this paper.

Despite the fact that the embedded and general computing hardware platforms are different, embedded software development can be based on similar principles as the computer software development. For example, agile development model has been adapted to embedded systems [14], [15]. Other approaches to embedded systems development, such as the *V-Model* described in e.g. [16], can also be applied to fast development of embedded software.

As presented in this section, there are multiple systems, methods and approaches to fast development of embedded software. There are also multiple solutions for remote re-configuration, programming, and management of embedded systems. Those methods can be applied to ad-hoc embedded systems firmware and software development process. Based on the presented state of the art we propose the following solutions. First, we suggest, that the remote programmers and firmware management systems can be improved. To prove that statement we propose the practical implementation of the RRM described further in this paper. Moreover, the RRM can be utilized to implement the RESP approach. We state and prove that utilizing the proposed RESP approach can reduce time-to-market and allow developers to deliver a working prototype of the resource-constrained embedded system faster compared to traditional approaches.

III. PROPOSED DEVELOPMENT METHOD

To explain and justify the proposed RESP development method, we introduce a simplified model of the experimental embedded system software development. The model reflects practical experiences while cooperating on an innovative IoT solution with actual business representatives. We assume that the model is applicable when the MCU-based embedded hardware platform is developed and it is ready for initial firmware implementation.

The simplified development flow is following. We assume that the development time can be represented by a number of *iterations*. In this case the *iteration* can be perceived e.g. as a *time interval* or as a *programming task*. In our considerations each iteration represents an amount of work required for one developer or for a team to complete a given task. Many developers can work on the project concurrently, but in the simplified model we just count the total number of iterations as if the project was developed in a fully sequential manner. A goal is specified for each iteration. Reaching Alpha development stage requires at least two iterations: the initial development stage and the actual Alpha development-testing stage.

In this model the device is ready to be shipped to the potential customer or an investor for further review if the following two conditions are met: 1) the application has at least minimum experimental functionality with basic Alpha stage tests done, and 2) the device firmware can be reliably and remotely updated or changed to allow developers and the client to collaborate on the final firmware version and further features. The consequence of the latter prerequisite is that the bootloader should be developed up to the final release version unless the RESP approach is used. In this model the embedded device is passed to the potential customer or investor after its development reaches the Alpha stage. Then the customer initially evaluates the product. If the customer accepts the initial results, the developers shall continue to work on the product's software (*success*). Otherwise the development of the product in the current form shall be ceased (*failure*). That situation can happen e.g. when the product is unable to meet the requirements or it needs a major redesign. In the *failure* case, most of the recent developing effort is wasted because the project or the idea has been rejected by the investor or the client.

Figure 1 shows a graphical representation of a sample embedded system development time line using the presented model. The goals of each iteration are denoted inside a box representing that iteration. Two cases are analyzed.

The first case, which is shown in Figure 1 (a), represents a scenario with a classic approach applied. In order to reach the product development stage at which the prototype can be passed to the client, the developers require six iterations: four iterations for the bootloader development and two for the initial application development. In case of success, only the application needs to be further developed, but in case of

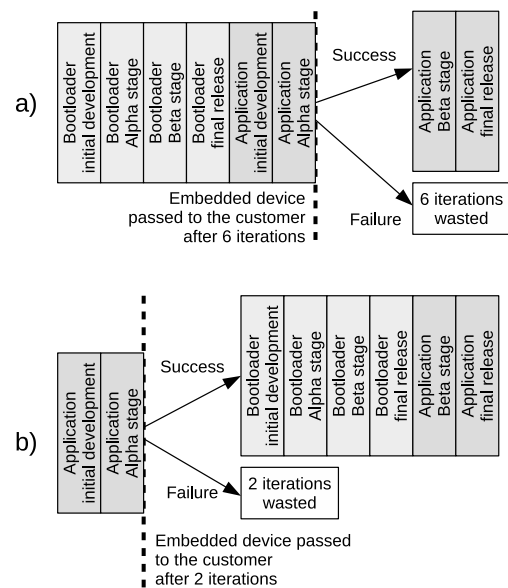


Fig. 1. Sample development time lines for the described model and: typical approach (a), utilizing RESP approach with RRM module (b).

failure, the six iterations are wasted.

In the second scenario shown in Figure 1 (b), the developers utilize the RESP approach with the dedicated RRM module. The device can be shipped to the client even at an early development stage, after just two iterations and the bootloader development can be postponed for later stages. In case of the project failure, only two iterations are wasted.

IV. IMPLEMENTATION CONCEPT AND OVERVIEW

In this section we present a general ideas which concern realization of RESP approach with the RRM module.

To implement the RESP development approach the developers should utilize the *RESP-compliant system*. The RESP-compliant system consists of a management unit and the RRM hardware with a dedicated software. One RRM can be connected to one or multiple instances of the device under development. The number of devices under development connected to one RRM depends on hardware interface capabilities of the utilized RRM embedded computer. Moreover, it is possible to manage multiple RRRMs using a single server with a remote application. Those features allow software developers to manage hundreds of devices with a single server. The sample set-up of a multi-node reconfiguration-debugging system using RRRMs is presented in Figure 2.

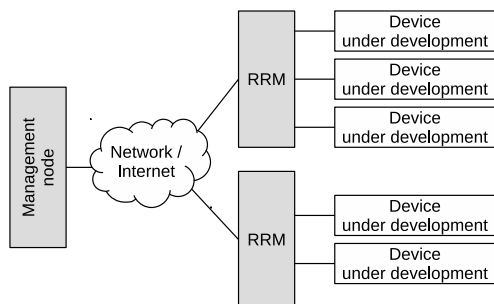


Fig. 2. Sample RESP-compliant hardware architecture with multiple RRRMs capable of managing several devices under development.

After the embedded software development is complete, the RRM can either be disconnected from the device under development, or it can be left connected to monitor the operation of the embedded system by utilizing e.g. external sensors, such as temperature, voltage or current.

To provide flexibility and advanced functionality, the implementation of RRM is based on an SBC. The use of SBC allows developers to implement much more advanced features compared to typical programmers with network interface.

The choice of generic hardware platforms combined with free and open source software solutions is one of the key concepts in the implementation of RESP-compliant system. The configuration needs to be performed with the use of scripting languages. Combining those features allows the RESP-compliant system to be very flexible and easily adapted to new target hardware platforms and specialized use cases. The SBC-based RRM can potentially log, filter, and transmit the

debug messages printed on e.g. a serial port of the device under development. The messages then can be transmitted over Internet to developers and testers in a remote location. The functionality of RRM can be extended even further. As an example, the RRM after extending its software may allow developers to obtain even a live view of the system under development. That feature obviously cannot be utilized to reliably monitor the operation of a safety-critical equipment, however it can be utilized to determine overall environment conditions for which sensors were not included in the initial design or to detect some obvious reasons of malfunction. For example the camera can facilitate determining if the light intensity in general is low or high, if the device has been covered, or if it has been moved.

V. PRACTICAL IMPLEMENTATION OF THE RESP-COMPLIANT SYSTEM

This section contains technical description of the sample RESP-compliant reconfiguration and monitoring architecture developed according to the information in previous sections of this paper.

A. RRM hardware design

Currently there are many different SBCs available at very low cost. In our sample implementation we have chosen Raspberry Pi Zero to implement the RRM. The Raspberry Pi Zero has multiple advantages as a choice for RRM. That computer is characterized by its very low cost and compact size. Another advantage of it is a General-Purpose Input-Output (GPIO) interface presence and its 3.3 V logic levels compatibility, which makes it well suited to implement versatile digital interfaces, including programming interfaces.

The network interface is provided with a generic Wi-Fi dongle with Universal Serial Bus (USB) interface. Depending on the SBC used, the network interface could also be implemented using a built-in peripheral as in e.g. full-sized Raspberry Pi computers.

Devices under development are connected to RRRMs directly using Joint Test Action Group (JTAG) or Serial Wire Debug (SWD) interface for programming and reconfiguration purposes. Those interfaces are implemented using GPIO hardware of the SBC. Devices equipped with a built-in programmer with USB interface, can connect to the SBC with that interface. In the proof-of-concept implementation no additional protection circuits were added, but they should be considered in the RRM hardware.

Another important aspect of the RRM implementation is the possibility to measure various physical quantities: the operation parameters of the device under development (e.g. input-output voltages, temperature, logic states, power supply current) and the general parameters of the environment (e.g. temperature, humidity). We have chosen Inter-Integrated Circuit (I²C) as a typical interface for RRM external sensors because it is supported in hardware and software of many SBC platforms or it can be relatively easily implemented using GPIO. There is also a wide selection of compatible sensors

with I²C interface and, what is very convenient, multiple sensors can be connected to a single bus.

As a sensor for the first implementation of the RRM we have chosen a current sensor intended to monitor a power supply current of the target device under development. The reason was to provide a sensor which is commonly needed during the process of developing embedded software. In our practice, the power management of an embedded system is one of the vital parameters that needs to be monitored and we often need embedded systems to be optimized for energy efficiency. The choice of INA219 current sensor appeared to be reasonable because it is equipped with digital I²C interface and it is able to measure current at the power supply rail (high-side).

B. RESP-compliant system logic and software architecture

The internal structure of RRM and its sample connections to multiple devices under development are shown in Figure 3.

We intended to chose a free, open source, and highly configurable solution for interfacing management software with hardware reconfiguration-programming interfaces of target devices under development. An actively developed project that seems a very good choice for that purpose is OpenOCD. It is a powerful and flexible debugging and memory programming tool which can be configured with TCL scripts and which provides multiple convenient control interfaces.

As the management interface needs to easily cooperate with standard software solutions, we decided to utilize the more and more popular OMA LWM2M protocol. The management node is implemented as a server for the OMA LWM2M protocol. Eclipse Leshan LWM2M server demo was chosen for the project implementation purposes. It provides basic unified network interface and Representational State Transfer (REST) application programming interface (API). Custom LWM2M object definitions have been added to the server in order to properly recognize custom resources which are specific to the project. The LWM2M API utilizes Firmware Update object from the specification and three custom LWM2M objects: OpenOCD LWM2M RPC, Firmware Target Selector and INA219 current sensors interface.

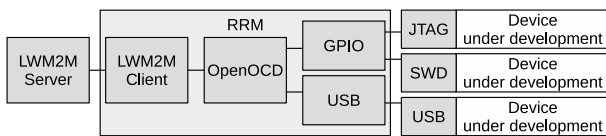


Fig. 3. General block diagram of the RRM and an example of connecting devices under development.

The internal software architecture of the RRM is shown in Figure 4. The software operates as multiple separate LWM2M client instances, one for each device connected to the RRM hardware. The LWM2M allows software developers to reprogram the target device and fetch sensors readings. It holds an OpenOCD instance for each device.

The software is written in Java with Eclipse Leshan libraries. For every device instance it creates a separate

LWM2M client instance and runs OpenOCD. In our implementation the OpenOCD uses remote procedure call (RPC) interface for clients to issue TCL commands and obtain results from TCL engine. The commands are generated according to information derived from configuration files passed as arguments. The configuration files are described in detail further in this paper (please refer to Section V-C).

When using Eclipse Leshan the binary image for the reconfiguration purposes of the device under development can be transferred from the management server. Alternatively, the reconfiguration can be initiated from the server along with providing an Unified Resource Identifier (URI) to a location in which the binary image is available. We have implemented the latter option, because in practice it proved to be more flexible and convenient for the developer who manages the process of reconfiguration or reprogramming. In current implementation, two protocols are supported for transferring firmware images: Hypertext Transfer Protocol (HTTP) and HTTP Secure (HTTPS).

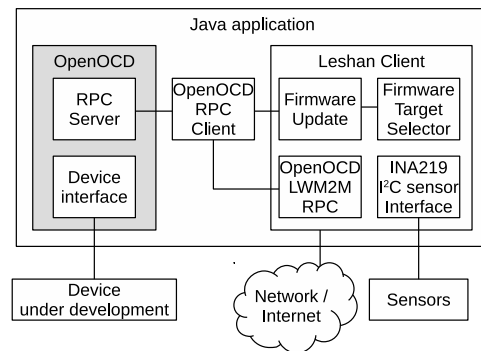


Fig. 4. Software architecture (gray block means external process managed by parent process).

C. Configuration

The configuration method was devised especially for RRM. YAML was chosen as a file format for its simplicity and readability. SnakeYAML library was chosen to load YAML files into configuration objects. It also provides runtime validation of configuration syntax.

There are two types of configuration: the first for each device class and the second for device instances connected to RRM. Device class configuration is stored in files separate for each device class (such as STMF4DISCOVERY, ZYBO). Instance configuration specifies communication interface and sensors for each of the actual devices connected to RRM (such as stm-prod, zybo1).

Listing 1. Sample instances configuration file for two identical devices connected to the RRM hardware.

```
instances:
- name: stm-by-gpio
  deviceConfigPath: stmf4discovery / config.yml
  interface:
    bcm2835gpio:
```

```

swdNum1: 25
swdNum2: 24
trstNum: 7
srstNum: 18
transport: swd
- name: stm-with-current-sensors
deviceConfigPath: stmf4discovery/config.yml
interface:
  custom:
    initCommands:
      - source [find interface/stlink-v2.cfg]
      - transport select hla_swd
sensors:
  - ina219:
      i2cAddr: 0x40
      i2cBus: 1
      shuntResistance: 0.1

```

A sample configuration is shown in Listing 1. It is a configuration file for two identical device instances connected to the RRM hardware. The first instance is connected by JTAG directly to the GPIO ports of the SBC, whilst the second instance is using board built-in USB debugger and also has current sensor attached. It was prepared for BCM2835 chip present in Raspberry Pi boards. It eliminates the necessity of writing OpenOCD commands, yet allowing to specify custom GPIO port numbers and transport.

An example of the device configuration file is shown in Listing 2. It allows system developers to define device class specific OpenOCD initialization commands as well as commands that are executed as a result of executing Update resource on Firmware Update object.

Listing 2. Device configuration file

```

initCommands:
  - source [find target/stm32f4x.cfg]
  - reset_config srst_only
firmwareTargets:
  - name: mcu
    flashCommands:
      - reset init
      - flash write_image {{ image }}
      - reset

```

D. Implemented sensor support

In the sample implementation, the RRM software supports multiple sensors connected using I²C bus supported by the physical interface of the utilized SBC. In the presented software implementation, each RRM instance supports zero, one or many INA219 current sensors as a sample implementation of that functionality. The RRM software fetches data from sensors using Pi4J library and provides on-demand access with multi-instance INA219 sensors LWM2M object (*urn:oma:lwm2m:ext:3403*). For convenience, current, voltage, and power values are provided.

E. Reconfiguration flow

Reconfiguration flow is presented as a sequence diagram in Figure 5. The reconfiguration process consists of two dependent stages. In the first stage, the developer provides a URI for a new binary image to be uploaded to the target embedded

system using Eclipse Leshan and LWM2M protocol. In the second stage, the execution of the reconfiguration itself takes place – the RRM performs the reconfiguration with an instance of OpenOCD.

F. Basic security considerations

The RRM is primarily intended to be applied only temporarily during the embedded system's development stage, but the security is still an important issue. The basic transport-level security using Datagram Transport Layer Security (DTLS) is supported by default for LWM2M. RRM software supports HTTPS to enable secure path of fetching images. User might want to add another layer of security such as an encrypted virtual private network (VPN) or Secure Shell (SSH) tunnel.

G. Achieved prototype functionality and practical verification

We have successfully developed a working prototype of RRM device according to the ideas described in previous sections. The presented example of the RESP-compliant system provides LWM2M-based remote firmware upgrade API for a wide variety of embedded systems. Thanks to the fact that we have chosen OpenOCD as the software for reconfiguration control, the RRM supports virtually any target platform that can be reconfigured or reprogrammed with OpenOCD – the main requirement is that the system administrator provides adequate configuration scripts. We have designed and implemented a specialized configuration scheme using YAML scripts. The utilization of free and open source software and common versatile off-the-shelf hardware allows developers to easily modify, extend, and tailor the functionality of the RRM for a particular use case. The implemented and presented sensor extension for measuring a target's power consumption may be extremely useful in remote debugging and development of energy constrained embedded systems.

We have measured reconfiguration times achieved with RRM and the module's average power consumption. The reconfiguration time results are summarized in Table I and visualized in Figure 6. Each reconfiguration time is an average computed from 10 sample transmissions. The results were obtained during reprogramming or reconfiguring MCU and FPGA on development boards. The reprogrammed MCU was STM32F407VGT6 and the programming interface was SWD implemented with GPIO of Raspberry Pi Zero. The reconfigured FPGA was Xilinx Zynq-7000 on Digilent Zybo ARM/FPGA SoC Trainer Board with the JTAG programming interface connected to SBC using USB interface. The SBC was connected to a local network using a generic Wi-Fi card with USB interface. We have measured power consumption of the RRM hardware. It averages to 1.3 W when idle (Wi-Fi connectivity enabled, LWM2M server is during registration) and to 1.4 W when performing firmware update. To set-up a Wide-Area Network (WAN) we used a virtual machine (VM) located in New York which was hosting LWM2M server and binary images. The LWM2M client was located in Krakow, Poland. The measured WAN Round Trip delay Time (RTT) was 120 ms at maximum transfer rate of 12 Mb/s. The

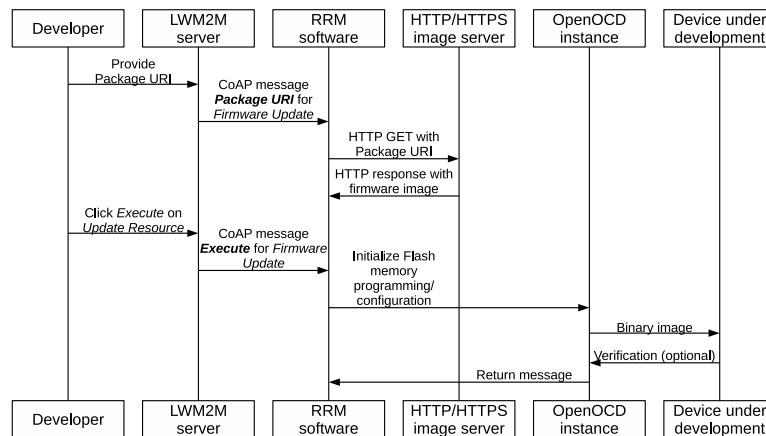


Fig. 5. Reconfiguration flow

TABLE I
RESULTS OF PRACTICAL EXPERIMENTS

Target type	Firmware/ configuration size (KiB)	Network for LWM2M	Image server protocol	DTLS for LWM2M	Total reconfiguration time (s)	Binary upload time (s)
MCU	21.8	LAN	HTTP	disabled	1.9	1.3
MCU	21.8	WAN	HTTPS	enabled	3.3	1.3
MCU	295.3	LAN	HTTP	disabled	10.5	9.1
MCU	295.3	WAN	HTTPS	enabled	12.1	9.1
MCU	978.9	LAN	HTTP	disabled	26.0	23.9
MCU	978.9	WAN	HTTPS	enabled	28.3	23.9
FPGA	4185	LAN	HTTP	disabled	12.2	9.1
FPGA	4185	WAN	HTTPS	enabled	14.7	9.1

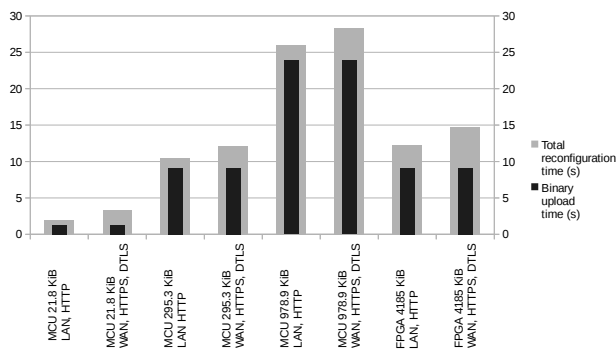


Fig. 6. Remote reconfiguration and programming times comparison.

reconfiguration time consisted of two components: an actual binary image upload time (target memory programming) and a communication overhead. According to the results the actual binary image upload was always taking an overwhelming amount of time.

VI. CONCLUSION AND FUTURE WORK

In this article we present the RESP approach for fast embedded systems prototyping. The presented RESP approach may be one of the future directions in embedded software development, especially for experimental and ad hoc systems, because it is aimed at effectiveness, low development cost, short time-to-market, and minimizing implications of a project failure. We propose a theoretical justification for the proposed solution and a sample practical implementation of the compliant system that supports the described approach. The RESP approach does not limit the use of common software development techniques and approaches, such as agile, but it may extend their possibilities and simplify their application in practice.

We also propose a new approach to remote programming and reconfiguration of microcontrollers and FPGAs by implementing much more advanced functionality in the RRM than in common off-the-shelf programmers. The use of the RRM can speed-up the embedded software development before the final version of an application-specific bootloader is developed. We present a sample working implementation of the RRM. It supports modern and promising network protocols, mainly OMA LWM2M, and flexible monitoring features. Thanks to the application of OpenOCD software along with the described implementation of scripting configuration, the RRM can be adapted for a wide variety of embedded hardware platforms, MCU-based IoT nodes, and their respective memory programming interfaces. It can be a versatile remote reconfiguration and development hardware-software tool. Thanks to the use of compact but full-fledged computer platform, the developed RRM system has far superior versatility compared to commercially available remote programmers-debuggers. The comparison between the RESP-RRM approach and solutions reviewed in Section II is summarized in Table II.

The RRM can be used not only for RESP development, but also as a multi-purpose remote reconfiguration and management extension as well as long-term operation monitor or

TABLE II
COMPARISON BETWEEN THE REVIEWED AND THE PROPOSED SOLUTIONS.

Reviewed solution	Aspects which can be improved	Aspects improved by utilizing the RESP approach and/or the RRM
Bootloaders for embedded MCU-based devices [1], [2]	Bootloaders require much development time and effort.	Bootloader is not required during initial development, and can be added at later development stages.
Remote FPGA reconfiguration, programmable logic partial reconfiguration with traditional design flow approach [5], [6], [7], [8]	Complex design which includes a reconfiguration subsystem and more elaborate design flow when considering partial reconfiguration.	RESP can improve time-to-market by allowing for remote development of both programmable hardware and software. The RRM substitutes additional fixed hardware and logic resources for remote reconfiguration. Later, the RRM can be disconnected when not needed.
Remote programmers for FPGA designs as utilized for example in [9]	Single-purpose hardware, no advanced management features, not customizable.	High flexibility of the hardware and software. Possibility to implement advanced firmware and configuration management.
General approach to remote firmware management [10], [11], [12], [13]	More versatile and general-purpose approach with standardized protocols can be considered.	The proposed solution can be easily customized to various applications.
Common approaches to embedded software development [14], [15], [16]	Usually considered for software development with hardware available locally.	RESP does not interfere with a selected embedded software development approach, but it allows developers to achieve a working prototype faster and without a direct access to a hardware platform.

logger for various embedded and connected devices, including IoT nodes. We plan to implement some of that features during further development. An example of a very useful extension is an integration of a camera module for basic visual inspection of the device under development. The RRM could also be considered as a tool which allows remote access to specialized embedded systems for educational and training purposes.

REFERENCES

- [1] "Stm32 bootloader," <https://github.com/akospasztor/stm32-bootloader>, accessed: 2017-12-29.
- [2] R. J. Landeo Márquez, "Can bus bootloader for the stm32f407vg," Master's thesis, Universitat Politècnica de Catalunya, 2017.
- [3] A. V. Parkhomenko, O. Gladkova, E. Ivanov, A. Sokolyanskii, and S. Kurson, "Development and application of remote laboratory for embedded systems design," *International Journal of Online Engineering (iJOE)*, vol. 11, no. 3, pp. 27–31, 2015. doi: 10.1109/REV.2015.7087265
- [4] M. D. V. Pena, J. J. Rodriguez-Andina, and M. Manic, "The internet of things: The role of reconfigurable platforms," *IEEE Industrial Electronics Magazine*, vol. 11, no. 3, pp. 6–19, 2017. doi: 10.1109/MIE.2017.2724579
- [5] R. Brzoza-Woch, A. Ruta, and K. Zieliński, "Remotely reconfigurable hardware–software platform with web service interface for automated video surveillance," *Journal of Systems Architecture*, vol. 59, no. 7, pp. 376–388, 2013. doi: <https://doi.org/10.1016/j.sysarc.2013.05.007>
- [6] R. Brzoza-Woch and P. Nawrocki, "Fpga-based web services—infinite potential or a road to nowhere?" *IEEE Internet Computing*, vol. 20, no. 1, pp. 44–51, 2016. doi: 10.1109/MIC.2015.23
- [7] R. Hymel, A. D. George, and H. Lam, "Evaluating partial reconfiguration for embedded fpga applications," in *Proceedings of High-Performance Embedded Computing Workshop (HPEC'07)*, 2007, pp. 1–2.
- [8] C. Conger, R. Hymel, M. Rewak, A. D. George, and H. Lam, "Fpga design framework for dynamic partial reconfiguration," in *Proceedings of Reconfigurable Architectures Workshop (RAW)*, 2008.
- [9] J. Belleman, D. Belohrad, L. Jensen, M. Krupa, and A. Topaloudis, "The lhc fast beam current change monitor," *WEPP29, IBIC*, 2013.
- [10] M. Ogura, "Remote management system, intermediary apparatus therefor, and method of updating software in the intermediary apparatus," U.S. Patent US7 516 450B2, 2003.
- [11] R. Pathak, "Remote firmware management for electronic devices," U.S. Patent US9 112 891B2, 2007.
- [12] S. Rao, D. Chendanda, C. Deshpande, and V. Lakkundi, "Implementing lwm2m in constrained iot devices," in *Wireless Sensors (ICWiSe), 2015 IEEE Conference on*. IEEE, 2015. doi: 10.1109/ICWISE.2015.7380353 pp. 52–57.
- [13] J. Prado, "Oma lightweight m2m resource model," in *IAB IoT Semantic Interoperability Workshop*, 2016.
- [14] J. Grenning, "Agile embedded software development," *ESC Boston*, 2011.
- [15] D. Dahlby, "Applying agile methods to embedded systems development," *Embedded Software Design Resources*, vol. 41, p. 1014123, 2004.
- [16] "Embedded System development Process Reference guide," Information-technology Promotion Agency, Reference Guide, 2012.