Proceedings of the Federated Conference on Computer Science and Information Systems pp. 467–470 DOI: 10.15439/2019F170 ISSN 2300-5963 ACSIS, Vol. 18

Remote Programming and Reconfiguration System for Embedded Devices

Tomasz Michalec, Maksymilian Wojczuk, Robert Brzoza-Woch, Tomasz Szydło
AGH University of Science and Technology,
Department of Computer Science, Krakow, Poland.
Email: robert.brzoza@agh.edu.pl

***Abstract*—This article presents a concept of a system which can be utilized as a remote management add-on for embedded devices. It can be applied to resource-constrained wireless sensors and IoT nodes based on a general purpose microcontroller unit or a field programmable gate array (FPGA) chip. The proposed solution facilitates remote firmware update, management, and operation monitoring. Thanks to the utilization of standard protocols and interfaces, the proposed system is very flexible and it can be easily customized for multiple modern microcontrollers or programmable logic chips. The presented system can be an efficient solution for fast prototyping and it can be an alternative to a time-consuming process of bootloader development for ad hoc devices. It can also be applied to remote laboratory access for educational purposes. A proof of concept prototype implementation has been successfully developed and evaluated. The implementation is available on a free license and utilizes a commonly available and inexpensive hardware platform.**

I. Introduction

INTERNET of Things (IoT) uses multiple nodes distributed among different physical locations. The nodes may require remote management and firmware upgrade mechanisms to be implemented. As the IoT systems are often utilized for monitoring and interaction with an environment, their operation has to be either well simulated or tested in a laboratory or in a target environment. If embedded or IoT software developers choose the approach that involves practical testing, the problem of the remote management of IoT nodes arises – it includes monitoring of a node operating condition, setting its operation parameters, and upgrading its firmware.

In production environments, a common approach of deploying a remotely manageable embedded device is to implement a bootloader. Unfortunately, the process of developing a bootloader software may be a very demanding and complex task – the software needs to be well tested because it is a crucial part of the system. In case of the bootloader malfunction, while the device's software development is at an early stage, the device becomes unusable until reprogrammed directly through a local interface. That requires physical access to the device's hardware – it can be very inconvenient in the domain of IoT and sensor nodes which may operate in remote locations.

In this paper, we present a concept and a sample implementation of a versatile add-on subsystem for remote management of embedded devices, IoT platforms, especially based on resource-constrained MCUs.

II. Related research and available solutions

A common scientific issue is designing a remote laboratory which is usually utilized to allow for remote access to laboratory infrastructure via the Internet [1], [2]. There are presented extensions of this concept, which allow designers to implement the remote laboratory on a single-board computer (SBC), but still it requires to run a full-featured operating system, e.g. Linux [3]. Such an operating system requires a large amount of hardware resources and energy.

A natural solution for firmware updates is to write a bootloader program. However, such a bootloader must be extremely reliable and it is more difficult to write a reliable bootloader which itself could be updated remotely using e.g. wireless connection [4]. Currently, one of the leading solutions in the field of remote management of the embedded devices is the utilization of the OMA Lightweight M2M (LWM2M) protocol [5], [6]. It is based on Constrained Application Protocol (CoAP) which is popular in the IoT domain [7] due to its relatively low resource requirements.

There are commercial solutions requiring additional hardware that could program flash memories of MCUs through the network, e.g. XDS220 USB/Ethernet JTAG Emulator or the Intel FPGA Ethernet Cable as used in [8]. However, those solutions are usually expensive, and their application is usually limited to a vendor-dependent subset of supported devices. Considering their application for a large number of managed nodes might not be economical.

A time-efficient approach for remote programming, software development and prototyping of embedded devices [9] may be more convenient if the remote reconfiguration is applied. The utilization of SBCs, such as Raspberry Pi, becomes more and more popular [10] even for high-end and military applications [11]. When equipped with proper software, such as the OpenOCD [12], [13], the Raspberry Pi SBCs can become remote management nodes as in e.g. [14].

III. Problem statement

After analyzing the available literature and solutions we decided to develop a concept of the remote development tool for MCU-based embedded systems with an option to expand its functionality to remote reconfiguration of FPGAs. The concept should allow designers to develop, customize, and deploy the versatile remote programmer-monitor tool for facilitating embedded software development.

The discussed problem concerns the development of multi-node systems based on embedded devices that require remote and batch firmware updates. The designed solution should meet the following crucial requirements: (1) the solution should be versatile or at least easy to adapt and extend for various MCU hardware platforms with an option for future FPGA configuration support; (2) the remote programming system should allow for easy interaction with remote embedded devices – mainly the firmware update; (3) it should detect the connected target board and adjust parameters automatically; (4) The remote reconfiguration tool should be able to operate on a commonly available and inexpensive hardware platform; (5) the interface for the user or a developer should be platform-independent.

IV. The design concepts

In this section, we present suggested choices and concepts for implementing the remote reconfiguration system based on the requirements stated in Section III.

A. General architecture and communication

We propose the following architecture of the remote programming system. The system may consist of two separate main parts: (1) the hardware part called the Remote Programming Device (RPD) further in this article and (2) the management part which is a user application for interacting with one or more RPDs.

The *RPD* is able to remotely reprogram internal memories of microcontrollers with provided binary firmware files and optionally to reconfigure FPGA integrated circuits. Primarily is intended to work as a temporary add-on to an embedded device or an IoT node during final stages of software development. It can also be utilized for diagnostic and long-term monitoring purposes in prototype and experimental IoT systems. The embedded device, which is managed, reprogrammed, or reconfigured by the RPD, is in this paper referred to as the *target device*.

The *management part* is intended to run on a user's host computer. For the purposes of communication between the two parts of the system, we have chosen and recommend the LWM2M protocol due to its popularity, basic security, and ability to communicate not only within local networks but also globally over the Internet. As the management part, we decided to use the Eclipse Leshan [1] implementation of the LWM2M server. It provides a Web-based user interface (UI) which allows for interaction with connected RPDs. That interface does not need any additional specific software to be installed on the management computer.

To provide versatility, the RPD is recommended to communicate with the target device using a standard and popular interface, primarily Joint Test Action Group IEEE 1149.1 (JTAG) and, eventually, Serial Wire Debug (SWD). An MCU for RPD may be a typical inexpensive unit for embedded systems purposes. The Ethernet was chosen to implement a convenient physical layer for Internet Protocol (IP) communication. For the prototype implementation, we have chosen the Nucleo boards equipped with STM32F429ZI MCU with ARM Cortex-M4 microprocessor core. RPD uses a USB flash drive to store programming files and configuration locally.

[1]https://www.eclipse.org/leshan/

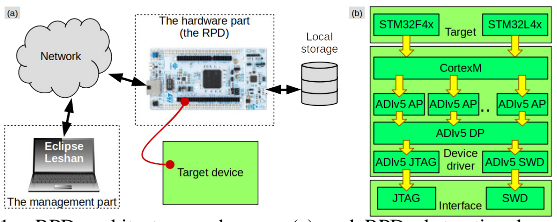

Fig. 1. RPD architecture and usage (a) and RPD abstraction layers diagram (b).

The proposed remote programming architecture has been shown in Figure 1a.

V. Selected details of sample implementation

A sample software for the remote programming system has been successfully implemented. This section contains selected details which concern practical aspects of the remote programming system operation.

A. RPD general architecture

The RPD has been designed to be easily extendable. Its architecture is based on layers. Figure 1b shows the designed organization of the layers. The first layer, denoted as *Target*, is the only layer exposed to the external interface described further in Section V-B. The *Target* layer purpose is to represent all programmable devices in a unified way. The *Device driver* is an intermediate layer which can be partitioned into multiple sub-layers. The driver sub-layers are able to communicate with devices supporting ARM Debug Interface Access and Data Ports (ADIv5 AP and ADIv5 DP). Such an architecture allows similar devices to share common parts of software implementation. This layer is independent of hardware. The *Interface* is the lowest layer and it encapsulates logic needed to use a physical medium. It is tightly coupled with hardware used to realize the RPD.

As a proof of concept implementation, we provide support for programming two different MCUs. The users and developers can extend the range of the supported programmed chips by modifying the provided source code. Further details on the RPD are elaborated in Section V-C.

B. Communication part

We used a standard LWM2M protocol stack with the CoAP over User Datagram Protocol (UDP) implemented with the LwIP stack – a commonly used TCP/IP stack designed for embedded devices. We have used Eclipse Leshan as the LWM2M server to provide the user with a generic Web-based UI for managing the programmer resources.

The communication with the management part includes two parts: the management interface using the LWM2M and the file download part with the Hypertext Transfer Protocol (HTTP). The LWM2M implementation at the RPD side uses the Wakaama code [5], [15]. The user can browse, read and update each connected device's properties through the UI. LWM2M server sends user's actions to particular devices and calls adequate procedures associated with resources. The firmware URL resource contains the URL of the current binary file. Once updated, the device downloads the newest firmware version from the HTTP server. A simplified process has been shown in Figure 2.

The whole communication between the user and the RPD is done through the LWM2M Server, excluding the binary download which is done using the HTTP. In order to ensure basic functionality for the sample implementation, we have defined an object representing the Remote Target – an embedded device to be programmed. The object contains properties necessary to monitor and control vital aspects of the remote target. The network configuration mechanism has been implemented and it can be dynamically changed without direct physical access to the device.

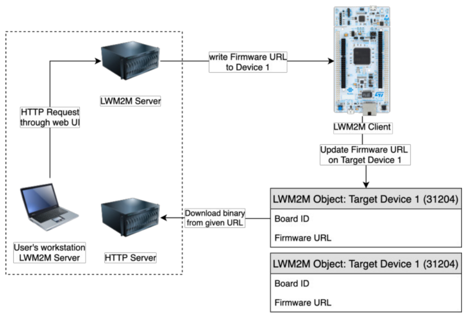
write Firmware URL to Device 1
LWM2M Server
HTTP Request through web UI
LWM2M Client
Update Firmware URL on Target Device 1
LWM2M Object: Target Device 1 (31204)
Board ID
Firmware URL
Download binary from given URL
User's workstation LWM2M Server
HTTP Server
LWM2M Object: Target Device 1 (31204)
Board ID
Firmware URL

Fig. 2. Simplified communication flow (the LWM2M Server, the HTTP Servers, and the user's workstation may or may not be running on a single machine).

C. *Programming part*

The sample implementation of the RPD is able to program flash memory of the following MCU families: STM32F4xx, STM32L4xx. Both of them are similar, but the STM32L4xx uses more energy-efficient technology and has updated hardware peripheral modules. JTAG has been selected as the hardware interface for the MCU programming in the prototype implementation. The JTAG's daisy-chaining feature is supported in the sample firmware. The RPD is also able to perform automatic discovery of connected devices by using their IDCODE registers. Discovered devices are exposed as independent targets to the management part.

VI. Prototype evaluation

Further in this section, we present a quantitative evaluation of the sample implementation of the RPD and its analysis.

A. *Network communication*

The system presented in this paper is intended to provide the possibility to transfer a new firmware binary file from the *management part* to a target embedded device programmed by an RPD. The main goal of the system is to allow for programming the target located in a distant physical location. To prove that functionality and usefulness of the created system we have tested time performance for different network environments. We conducted 5 practical tests for each of the following network conditions:

- **Network A**: The LWM2M server, the HTTP server, the user's computer, and the RPD with target boards are located in the same local network. The user's computer is connected with the router and switch through Wi-Fi interface.
- **Network B**: The LWM2M server, the user's computer, and the RPD are placed in the same local network, but the binary file is downloaded from an external HTTP server in another network but in the same city Kraków, Poland.
- **Network C**: The LWM2M server is running on the Amazon Elastic Compute Cloud (Amazon EC2) instance from Amazon Web Services in Frankfurt Datacenter and the binary files are placed also in Frankfurt, on the Amazon Simple Storage Service (the Amazon S3). User's computer, the RPD connected to target boards are located in Kraków, Poland, in the same local network.
- **Network D**: In this test set-up, the user's computer as a virtual machine, the LWM2M Server, and the HTTP Server on the EC2 Instance from Amazon Web Services were located in the United States Datacenter while the RPD was in the AGH University network in Kraków, Poland, Europe.

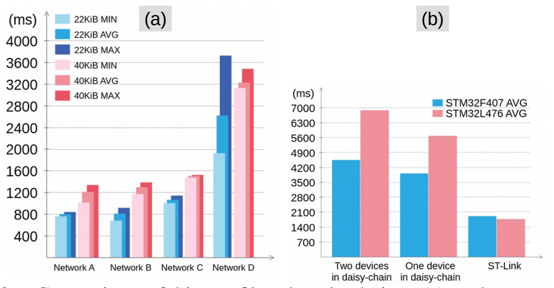

Fig. 3. Comparison of binary files download time (a) and programming time (b).

The results are shown in Figure 3a. The binary download time to the target is similar for the local network conditions and across neighbouring countries on one continent – in all of those cases the system has a similar level of its overall usefulness and the physical distance had only a limited impact on the overall system performance. The network overhead plays a greater role in large distances as in the *Network D* case. However, system can then still be considered useful because the binary download time does not exceed 4 s.

B. *The target device programming*

This section discusses its overall performance in different scenarios with reference to the underlying dependencies of this process. The information can be useful in comparison with other available commercial solutions than mentioned in this section and also to provide more complete information for scientists and engineers who wish to contribute to the RPD development.

Time required for the programming process includes overhead for communication between the RPD and a target device, erasing the target's flash memory, and the target's flash memory programming.

The JTAG clock signal (TCK) was set to 1 MHz. For erasing flash memory on STM32F4xx and STM32L4xx we utilized a mechanism to erase only those memory regions that were going to be programmed. For programming the STM32L4xx we used a binary file with size 40 KiB. STM32F4xx was programmed using a binary file with size of 22 KiB.

Tests in the following scenarios have been performed: (1) RPD with two devices in JTAG daisy-chain, (2) RPD with one device in JTAG chain, and (3) ST-Link programmer connected locally with USB. The latter scenario served as a reference for comparison with a commercial solution. All tests were repeated 5 times and the average time is presented in Figure 3b.

The process of programming a target device while another one is in JTAG daisy-chain takes longer than programming one target device only, because it was required to write the *BYPASS* instruction to all other devices in the chain. There is also additional code in the RPD that needs to be executed to handle multiple targets.

Even the STM32L4xx binary file is almost two times larger than the STM32F4xx program, time for flashing the STM32L4xx increased only 145%, because it also includes an overhead for programming operation.

The RPD firmware was compiled with the free GCC cross compiler, arm-none-eabi-gcc. With the compiler optimization level set to Og, the resulting firmware size is 149.2 KiB and the RPD program requires 135.4 KiB of static data memory.

The measured power consumption of the RPD is 1.1 W with fully operational Ethernet interface but without an external mass storage device. The total power consumption may vary depending on the utilized mass storage memory, usually a USB flash drive. In practice, the measured total power consumption with a flash drive connected is approx. 1.2 W.

VII. Conclusion and future work

In this article, we present a concept of the remote programming, configuration, and monitoring system for development and testing of embedded devices and IoT nodes.

The sample implementation of the reconfiguration system has been successful, the basic proof-of-concept functionality has been achieved, and the requirements stated in Section III are met which proves the overall correctness of the presented concept. The remote management interface is easy to use and can be run on many different operating systems thanks to the utilization of the Leshan LWM2M implementation with Web-based GUI. The created RPD software is ready for users to further develop the RPD functionality according to their own use cases, including new programmed devices, sensors, etc. The developed software is available on GitHub[2] on the free (MIT) license. The hardware price of the remote programming system presented in this paper is much lower than the commercial solutions presented in Section II. However, this implementation may require additional work needed for customizing it for specific, not yet supported use cases. The additional labor cost can be less noticeable if the multiple RPDs with the same custom firmware are deployed.

Future improvements may include implementing a support for the SWD interface, optimizing storage management and flash memory writing, as well as remote management of the RPD firmware and health checks of the connected target devices using additional sensors.

[2]https://github.com/maxiwoj/RemoteProgrammer

Acknowledgment

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.

References

[1] R. Bose, "Virtual labs project: A paradigm shift in internet-based remote experimentation," *IEEE access*, vol. 1, pp. 718–725, 2013. [Online]. Available: https://doi.org/10.1109/ACCESS.2013.2286202

[2] 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.

[3] P. Alexander and N. Radhakrishnan, "Remote lab implementation on an embedded web server," in *2015 International Conference on Circuits, Power and Computing Technologies [ICCPCT-2015]*. IEEE, 2015, pp. 1–5. [Online]. Available: https://doi.org/10.1109/ICCPCT.2015.7159525

[4] S. Schmidt, M. Tausig, M. Hudler, and G. Simhandl, "Secure firmware update over the air in the internet of things focusing on flexibility and feasibility," in *Internet of Things Software Update Workshop (IoTSU). Proceeding*, 2016.

[5] S. Rao, D. Chendanda, C. Deshpande, and V. Lakkundi, "Implementing LWM2M in constrained IoT devices," in *2015 IEEE Conference on Wireless Sensors (ICWiSe)*. IEEE, 2015, pp. 52–57. [Online]. Available: https://doi.org/10.1109/ICWISE.2015.7380353

[6] J. Prado, "OMA Lighweight M2M Resource Model," in *IAB IoT Semantic Interoperability Workshop*, 2016.

[7] B. Djamaa, M. A. Kouda, A. Yachir, and T. Kenaza, "Fetchiot: Efficient resource fetching for the internet of things," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 637–643. [Online]. Available: http://dx.doi.org/10.15439/978-83-949419-5-6

[8] J. Belleman, D. Belohrad, L. Jensen, M. Krupa, and A. Topaloudis, "The LHC Fast Beam Current Change Monitor," *WEPF29, IBIC*, 2013.

[9] A. Tutaj and J. Augustyn, "Universal serial bus as a communication medium for prototype networked data acquisition and control systems-performance optimisation and evaluation," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 665–674. [Online]. Available: http://dx.doi.org/10.15439/978-83-949419-5-6

[10] R. Baumgartl and D. Muller, "Raspberry pi as an inexpensive platform for real-time traffic jam analysis on the road," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 623–627. [Online]. Available: http://dx.doi.org/10.15439/978-83-949419-5-6

[11] F. T. Johnsen, "Using publish/subscribe for short-lived iot data," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 645–649. [Online]. Available: http://dx.doi.org/10.15439/978-83-949419-5-6

[12] H. Högl and D. Rath, "Open on-chip debugger–openocd–," *Fakultat fur Informatik, Tech. Rep*, 2006.

[13] D. Rath, "Openocd," https://github.com/ntfreak/openocd, 2005.

[14] R. Brzoza-Woch, Ł. Gurdek, and T. Szydlo, "Rapid embedded systems prototyping-an effective approach to embedded systems development," in *2018 Federated Conference on Computer Science and Information Systems (FedCSIS)*. IEEE, 2018, pp. 629–636. [Online]. Available: http://dx.doi.org/10.15439/978-83-949419-5-6

[15] O. M. Alliance, "Lwm2m specification 1.0," *Open Mobile Alliance: San Diego, CA, USA*, 2017.