Kazimierz KRZYWICKI[1], Alexander BARKALOV[1], Grzegorz ANDRZEJEWSKI[1], Larysa TITARENKO[1], Małgorzata KOŁOPIEŃCZYK[2]

University of Zielona Gora, Faculty of Computer, Electrical and Control Engineering
Institute of Metrology, Electronics and Computer Science (1), Institute of Control and Computation Engineering (2)

doi:10.15199/48.2016.10.59

SoC Research and Development Platform for Distributed Embedded Systems

***Abstract**. This paper presents a novel research and development hardware platform for distributed embedded systems. The platform is based on Xilinx Zynq-7000 SoC devices and it enables a fast physical verification and behaviour analysis of the distributed systems. Furthermore, it eliminates the necessity for usage of a large number of physical devices, which results in the simpler structure and implementation, improved ergonomics in laboratory, lower costs and eliminates external, physical connection between modules. The article presents the architecture of the platform and concurrent process implementation using the EmbedCloud structure. Synthesis and optimization results for different number of end modules and an analysis of resource usage were provided.*

***Streszczenie.** W artykule zaprezentowano nową koncepcję sprzętowej platformy rozwojowo-badawczej dla rozproszonych systemów wbudowanych. Platforma oparta o układy Xilinx Zynq-7000 SoC, pozwala na szybką fizyczną weryfikację oraz analizę behawioralną systemów rozproszonych. Ponadto, eliminuje konieczność użycia dużej liczby fizycznych układów, co przekłada się na prostszą strukturę i implementację, poprawę ergonomii w laboratorium, niższe koszty oraz eliminuje zewnętrzne, fizyczne połączenia pomiędzy modułami. W artykule przedstawiono architekturę platformy oraz proces współbieżny zaimplementowany przy użyciu metody strukturalnej - EmbedCloud. Syntezy, optymalizacji i analizy użycia zasobów sprzętowych dokonano dla różnej liczby modułów końcowych. (**Modularna platforma rozwojowo-badawcza dla rozproszonych systemów wbudowanych**).*

Keywords: embedded systems, distributed systems, research platform, CloudBus protocol, EmbedCloud structure
Słowa kluczowe: systemy wbudowane, systemy rozproszone, platforma badawcza, protokół CloudBus, struktura EmbedCloud

Introduction

The dynamic development of the embedded system has also caused an increased complexity and size of distributed systems [1-4]. The design and implementation process is often complicated and requires long time [1,3]. Moreover, implementation of such systems causes greater probability to make mistakes due to large number of modules and synchronization points. While a lot of issues can be eliminated at the simulation level, hardware verification cannot cause damage of the controlled object. Furthermore, hardware verification is rather easy for simple embedded systems, but complex, for distributed embedded systems (DES). This is due to large number of end modules (EM) that need to exchange the data and synchronize between each other.

The most common verification method of the embedded system implementation is simulation [4]. Used tools depend on the used device and manufacturer, but usually they are limited to the software that is supplied with a controller. Simulation allows examining different scenarios [4,5], but also takes a lot of time and requires a powerful workstation [5,6]. Moreover, implemented system behaviour can finally differ from the represented model [7].

Another method used for system behaviour verification is its implementation with simulation of controlled object [4,5]. This method gives good results, but in the case of DESs it is not efficient. Placing a dozens of EMs in the laboratory and combine them together is expensive (cost of the end modules) and burdensome (dozens of end modules connected with each other in one place). Furthermore, changes in the implementation or debugging take additional time.

The authors lead research in the field of DESs synthesis. They have developed a protocol (called CloudBus [8]) for the data exchange between end modules in the DES. The CloudBus protocol allows for a significant reduction in the amount of data transmitted between modules, especially when compared with other protocols commonly used in the industry [8]. Moreover, it provides a process synchronization and control mechanism for a number of processing units distributed in a network. The CloudBus protocol is used by EmbedCloud method [9], which forms the basis for the automatic code generation algorithm of the DESs. These research are aimed to accelerate the implementation and verification of the DESs. So far, verification and hardware tests were performed on the platform built with microcontrollers. It is efficient and easy to use for small (up to 5-8 EMs) systems, but burdensome with a large number of modules operating in the system. Due to the above-mentioned problems, further research was more and more difficult, and took up more time. It was necessary to create a research platform which allows studying the behavior of the DESs. The main assumption was: large number of end modules "on one desk" with a single PC. Previous research [10] of the CloudBus protocol implementation on FPGA devices (Xilinx and Altera), gave encouraging results (small resource usage – maximum up to 3% for the smallest device, Xilinx Kintex-7). Therefore, as a basis for the platform described below, Xilinx Zynq-7000 [11] System On Chip (SoC) was chosen. Its architecture allow a single chip to use both: microcontroller (dual-core ARM) and reprogrammable array (FPGA).

Proposed platform architecture

The research platform is based on Xilinx Zynq-7000 SoC XC7Z020-CLG484-1 mounted on developers board called ZedBoard (Digilent Inc.). This chip provides 53,200 LUTs (Look-up Tables), 106,400 FFs (Flip-Flops) and dual-core ARM Cortex-A9. Implemented end modules operated from 1MHz to 100MHz, depending on the amount of modules that has been synthesized.

A. General architecture

The architecture of the platform is shown in Fig. 1. The platform contains a number of end modules (*End module*) and additional submodules for input/output (*I/O*) control or communication (*UART*).

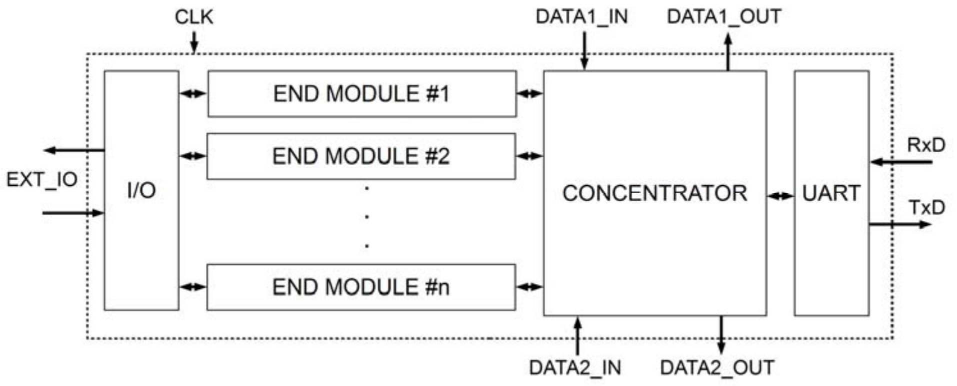

Fig.1. General architecture of the platform

The platform characteristics are the following:

- all modules forms a single distributed control system,
- each EM is self-independent and executes its own control tasks,
- modules communicate with each other for data exchange.

The *Concentrator* module is responsible for communication. It transmits the data either between EMs or outside – to other platforms/external EMs.

The *UART* module provides a communication of the entire system with a PC. This enables traffic monitoring (data transmission among EMs) and controlling the state of the system.

B. End module architecture

Each *End module* corresponds to the one hardware device of the DES. Each of modules is regarded as a separate and independent unit that cooperates with others to form a final distributed system. Furthermore, each EM can be physically implemented by a devices with a different architectures.
The architecture of the *End module* is shown in Fig. 2. The main core is a sub-module *Controller,* which:

- implements the EmbedCloud structure [9],
- parses the frames of the CloudBus protocol, collected from the *Receiver Buffer*,
- sends the CloudBus queries and answers with a state of variables/input//outputs (*Transmitter Buffer*),
- executes implemented control tasks.

Moreover, each EM includes an external input/output port (*I/O*) and sub-modules for serial communication (*Receiver* – data retrieve, *Receiver Buffer* – buffer for storing the received CloudBus protocol frames, *Transmitter* – control of the transmit line and *TransmitterBuffer* – buffer for storing the CloudBus protocol frames to be sending). Other sub-modules include the *UART* for communication with a PC (optional). *Memory* is an additional memory collection for the execution of specific task (optional).
Buffers and size of additional memory can be freely configured. In this research, the *Memory* module have been omitted and communication buffers have been limited to 36 bytes. This is possible, because the processing of each frame by the controller is done in a single clock (*CLK*) cycle. This clock also synchronizes communication modules and buffers.

C. Connection concentrator architecture

This sub-module is responsible for the data transmission between all of the EMs in the system. Basing on the signals from the EMs, it determines which of the EMs may currently broadcast the CloudBus frame. Furthermore, the *Concentrator* has two external communication ports for attaching another test platforms (forming network in the network topology); and *UART* sub-module, which allows direct monitoring both traffic and system state from the PC.

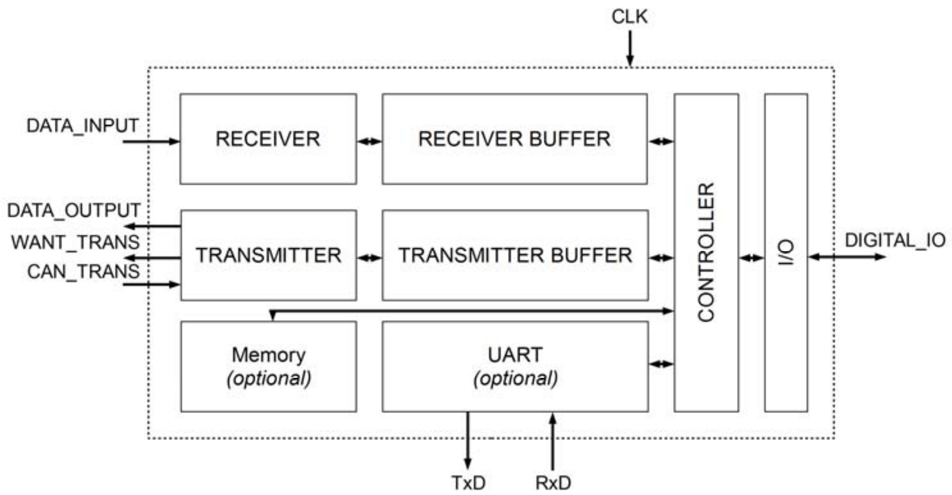

Fig.2. End module architecture

For the research purposes, the communication speed between EMs was set to 9600bps. Depending on the number of EMs it can be either increased or decreased. The selected speed stems from the need of unifying the research results and a very large number of end module implementation. Therefore, it was necessary to set constant communication speed for all examined scenarios.

D. Implementation

The platform has been implemented on the Digilent Inc. development board (called ZedBoard), which is equipped with a Xilinx Zynq-7000 AP SoC XC7Z020-CLG484-1 (dual-core ARM Cortex A9 and FPGA array). Main characteristics of the ZedBoard: 512MB DDR3, 256MB Quad-SPI Flash, SD card port, 10/100/1000 Mbps Ethernet, USB OTG 2.0, 225MHz HDMI Transmitter and Audio Codec [11,12].

The first implementation was performed manually in the Verilog HDL language. Then, the simulation (*Active HDL 9.3* [13]), synthesis (*Vivado 2015.1* [14,15]) and hardware tests were performed. After successful verification, a web application – Automatic Code Generator (ACG) was implemented using *PHP* object language. The web application generated the Verilog source code for EMs in the number indicated as an argument. The generated source code implementing the process shown in Fig. 4. All of generated structures were placed in a single module, which was ready for the synthesis process. Implementation of the automatic code generation tool was necessary due to the very large number of EMs (up to 512). Otherwise, each of them would have to be implemented manually. Without the ACG, checking the maximum possible amount of synthesizable modules would take a very long time and it would also affect both simulation and debugging times. Using the ACG, the implementation time for any of examined end-modules number was less than 1 second. The application produced output files for all end modules, sub-modules and test units, which were imported into Active-HDL and Vivado environment.

Research results

The synthesis, verification and tests were performed for different numbers of end modules (from 8 to 512). The logical structure of the platform is shown in Fig. 3. For all cases the code was generated using the web application described above. External signals were received by *End Module 1 - x1*, *End Module 2 - x2*. These modules broadcasted their status to other modules via the CloudBus protocol. This allowed the transition start and transited the specified module in the next state. The modules with the numbers starting from 3, did not have any external outputs (operating only as a local outputs without connection to *I/O* sub-module – Fig. 1). Furthermore, *End Module 1* and *End Module 2* used two external outputs: *Y0* and *Y1*.

Fig. 4 presents concurrent process described by a Petri net. All modules have implemented some tasks (*P1, P2, ..., Px, Py, Pz*) i.e. the random state change with local output state set and random time delays for simulating processing time. Time delays were implemented on 32-bit counters with a drawn residual value in the automatic code generation process.

Synthesis was made in Vivado 2015.1 using a PC (Intel Core i7 2630QM, 16GB RAM). Number of Verilog source files ranged from 22 (for 8 end modules) to 1030 (for 512 end modules). The source code files size ranged from 40kB to 14MB. This shows how the source files of the DES increases with increasing the number of modules. For this reason, manual implementation of each module would take a very long time. Moreover, with such an amount of end modules, the mistake is much easier to make. During the simulation or the system verification any mistake could be difficult to find in such a large system. This is the main reason, why the ACG was developed and used.

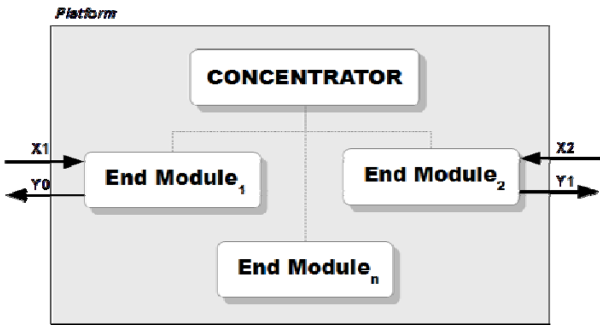

Fig.3. The logical structure of the platform

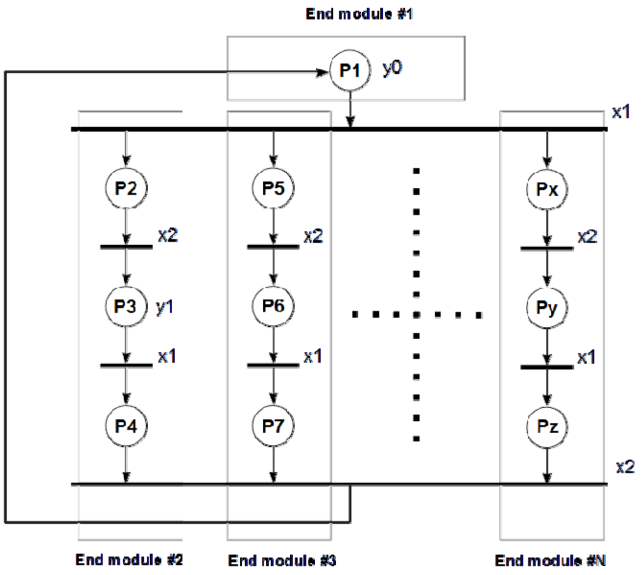

Fig.4. Implemented concurrent process described by a Petri net

Table 1. Synthesis results

End Modules Count	Resource usage				Synthesis Time elapsed [s]
	FF	*FF[%]*	*LUTs*	*LUTs[%]*	
8	1653	1,55	2149	4,04	82
16	3149	2,96	4109	7,72	138
32	3996	3,76	5663	10,65	158
48*	4838	4,55	7102	13,35	201
64*	5682	5,34	8601	16,17	250
96*	7369	6,93	11488	21,60	349
128*	9056	8,51	14789	27,80	440
256*	15812	14,86	26955	50,67	830
512*	29320	27,56	51253	96,24	1681

In Tab. 1 synthesis results are shown. Defined configuration and assumptions allowed to synthesize up to 512 EMs on a single platform. The main limitation was the number of available LUTs (total available number is: 53,200). After the implementation and optimization process, the amount of used LUTs decreased on average by 20-30%, while the number of registers (FFs) in use, remained unchanged (Tab. 2.). The *Opt_design* implementation parameter was used with the argument *-retarget*. Timing analysis showed for the number of modules that are marked (*) (above 48), TNS (Total Negative Slack) from -2.5ns to -1680ns and WNS (Worst Negative Slack) from -0.18ns to -1.4ns. To avoid changes in the architecture, the solution was gradually reducing the clock *CLK* from 100MHz down to 1MHz. With the clock set to 1MHz system worked steadily and all EMs correctly exchanged the data between each other. Network traffic was monitored by a PC via UART and *Concentrator* sub-module.

Table 2. Synthesis results after implementation

End Modules Count	Resource usage				Impl. Time elapsed [s]
	FF	***FF[%]***	***LUTs***	***LUTs[%]***	
8	1653	1,55	1728	3,25	99
16	3149	2,96	3505	6,59	117
32	3996	3,76	4723	8,88	251
48*	4838	4,55	5732	10,77	168
64*	5682	5,34	6867	12,91	185
96*	7369	6,93	8942	16,81	221
128*	9056	8,51	11301	21,24	263
256*	15812	14,86	20168	37,91	385
512*	29320	27,56	37422	70,38	753

It should be noted that the EMs implemented fairly simple functionality. For more complicated control system, the less end-modules would be able to place on the platform. It has not got great significance, because using the platform to verification and tests, allows considerable cost savings, and physical space in laboratory – even for few modules. However, the proposed architecture enables direct connection with another platforms of this type (external ports *DATA1_IN/OUT*, *DATA2_IN/OUT*) and creating network in the network system. As a result, the number of possible end modules forming a built-in distributed system can be freely and easily scaled up.

Conclusions

The paper presents a novel modular research and development platform, based on Xilinx Zynq-7000 SoC series. The proposed platform architecture enables a fast physical verification and behavior analysis of the distributed embedded systems, including a very large number of end modules. Additional advantages are: ergonomics, savings in cost of purchasing a large number of end modules and implementation/verification time (direct hardware verification takes less time than simulation which requires powerful workstation).

Presented research results for different number of end modules are satisfactory. Our approach allows for synthesize up to 512 simple EMs, on a single platform. Depending on the complexity of the control job executed by each EM, the possible number of end modules could be less than 512. However, the capacity of the platform is so large that it allows making verification and tests of very complex and complicated DESs. In the case, when the value of the resource usage exceeds a platform capability, its architecture allows connecting additional platforms. Furthermore, it is possible to combine so constructed platforms of different architecture, i.e. FPGA-FPGA, FPGA-microcontroller etc. All of this gives great flexibility in the system architecture and hardware verification. It also allows avoiding the costs associated with the purchase of individual end modules, significantly improves ergonomics and reduces implementation and debugging time. In the proposed architecture, the end modules are using for the communication, the CloudBus protocol, but it is possible to use known protocols (e.g. Modbus, Profibus or DeviceNet). In such a case, there is no need in changing the architecture and internal wires – only specific protocol implementation should be performed. This allows for network traffic analysis for the various protocols, study the system operation in different scenarios and searching the critical points of communication between end modules.

Current research focuses on traffic monitoring and analysis of the CloudBus network. Moreover, different variants of the protocol (CloudBus-BS, CloudBus-TL CloudBus-PC and CloudBus-PCTL) are examined. This is very important issue, because of the characteristics of the CloudBus protocol (broadcasting queries about the state of specific variable). When one of the system end modules fails or disappear from the system, it is necessary that the system (other modules) takes appropriate action. Additional variants of the protocol are intended to prevent such situations, detect it, and if necessary file a bug report.

Authors:
MSc. Kazimierz Krzywicki, E-mail: K.Krzywicki@wiea.uz.zgora.pl
Prof. Alexander Barkalov, E-mail: A.Barkalov@imei.uz.zgora.pl
Ph.D. Grzegorz Andrzejewski, E-mail: G.Andrzejewski@imei.uz.zgora.pl
Prof. Larysa Titarenko, E-mail: L.Titarenko@imei.uz.zgora.pl
University of Zielona Gora, Institute of Metrology, Electronics and Computer Science, ul. prof. Z. Szafrana 2, 65-246 Zielona Gora;
Ph.D. Małgorzata Kołopieńczyk, E-mail: M.Kolopienczyk@issi.uz.zgora.pl
University of Zielona Gora, Institute of Control and Computation Engineering, ul. prof. Z. Szafrana 2, 65-246 Zielona Gora.

REFERENCES

[1] H. Kopetz, “Real-time systems: design principles for distributed embedded applications”, Springer Science & Business Media, 2011.
[2] A. Sangiovanni-Vincentelli, M. Di Natale, “Embedded system design for automotive applications”. Computer, 2007, 10: p. 42-51.
[3] H. Kopetz, "The complexity challenge in embedded system design", Object Oriented Real-Time Distributed Computing (ISORC), 2008 11th IEEE International Symposium on. IEEE, 2008, p. 3-12.
[4] D. Gajski, S. Abdi, A. Gerstlauer and G. Schirner ”Embedded system design: modeling, synthesis and verification”, Springer Science & Business Media, 2009.
[5] Y. Nakamura, K. Hosokawa, I. Kuroda, K. Yoshikawa and T. Yoshimura, “A fast hardware/software co-verification method for system-on-a-chip by using a C/C++ simulator and FPGA emulator with shared register communication”, In: Proceedings of the 41st annual Design Automation Conference. ACM, 2004. p. 299-304.
[6] J. Andrews, “Co-verification of hardware and software for ARM SoC design”. Elsevier, 2004.
[7] M. Bambagini, M. Di Natale. “A code generation framework for distributed real-time embedded systems”, Emerging Technologies & Factory Automation, 2012, p. 1-10
[8] K. Krzywicki, G. Andrzejewski, “Data exchange methods in distributed embedded systems”, In: New trends in digital systems design, Fortschritt – Berichte VDI, Dusseldorf, 2014, p. 126-141
[9] K. Krzywicki, M. Adamski, and G. Andrzejewski, "EmbedCloud–Design and Implementation Method of Distributed Embedded Systems", In: Technological Innovation for Cloud-Based Engineering Systems. Springer International Publishing, 2015, p. 157-164.
[10] K. Krzywicki, G. Andrzejewski, “Hardware implementation of the CloudBus protocol using FPGA” In: Proceedings of the Prague Embedded Systems Workshop, 2014, p. 11-14
[11] Xilinx. Zynq-7000 All Programmable SoC – Technical Reference Manual, UG585 (v1.10), 2015
[12] AVNET. ZedBoard (Zynq Evaluation and Development) Hardware Users’s Guide, Version 1.3, 2012
[13] Aldec. Active-HDL Manual. (https://www.aldec.com/resources/manuals/Active-HDL/index.htm)
[14] Xlinix. Vivado Design Suite User Guide, Implementation, UG904 (v2015.1), 2015
[15] Xlinix. Vivado Design Suite User Guide, Design Anaylsis and Closure Techniques, UG906 (v2012.4), 2013