EmbedCloud – Design and Implementation Method of Distributed Embedded Systems

Kazimierz Krzywicki(✉), Marian Adamski, and Grzegorz Andrzejewski

Department of Electrical Engineering, Computer Science and Telecommunications,
University of Zielona Gora,
ul. Licealna 9, 65-417 Zielona Gora, Poland
K.Krzywicki@weit.uz.zgora.pl,
{M.Adamski,G.Andrzejewski}@iie.uz.zgora.pl

Abstract. This paper presents a novel design and implementation methodology of the distributed embedded systems, called EmbedCloud. It defines structured implementation model for each module in the system. EmbedCloud forms the basis for the automatic code generation algorithm of the distributed embedded systems which accelerates and simplifies synthesis process of such systems. The EmbedCloud utilizes CloudBus protocol demonstrated in previous publications, which provides a process synchronization and control mechanism for a number of processing units distributed in a network. The CloudBus protocol allows to significant savings in the amount of transmitted data between end modules in the distributed embedded system, especially when compared with the other protocols used in the industry. To verify and evaluate the performance of the EmbedCloud, a concurrent process was described using Petri nets. Hardware tests and synthesis verification of the distributed embedded system was performed on the testing platform built with AVR, ATmega series microcontrollers. The tests confirmed the correctness of the developed source code and EmbedCloud method. Furthermore, resource requirements and reaction time analysis were performed.

Keywords: Distributed embedded systems · Embedded system design · Hardware synthesis · Process control · EmbedCloud method · CloudBus protocol

1 Introduction

The dynamic development of the automatic industrial process control has caused an increased complexity and size of embedded systems. Typically, they are composed of execution units that are connected together in a logical way and form a distributed embedded system. The design and implementation of such systems is often complicated and requires long time for a large number of end modules operating in a system [1, 4, 7]. Moreover, the operating costs of systems based on PLCs (Programmable Logic Controllers) networks is high.

Most of the currently available models and tools for automatic code generation provide synthesis to a single hardware execution unit without any internally implemented process synchronization and communication with external hardware units [2, 3, 5, 6].

L.M. Camarinha-Matos et al. (Eds.): DoCEIS 2015, IFIP AICT 450, pp. 157–164, 2015.
DOI: 10.1007/978-3-319-16766-4_17

For a large number of end modules it is necessary to design them separately, e.g. using GALS (Globally Asynchronous Locally Synchronous) architecture [7, 8, 10]. These problems have led us to develop a new method which significantly reduces the design and implementation time of distributed embedded systems.

This paper proposes a novel design and implementation method of the distributed embedded systems, called EmbedCloud. It allows a structured implementation for each module in a distributed embedded system. Furthermore, the EmbedCloud is the base for the automatic code generation algorithm which accelerates and simplifies synthesis process of such systems. The end module communication and process synchronization is achieved through the CloudBus protocol presented in [9].

In Section II relationship to Cloud-based Engineering System is described, Section III describes the CloudBus protocol. Section IV presents synthesis model and EmbedCloud method. Section V discuss verification and research results in the term of resource usage and reaction time. Section VI, concludes the paper.

2 Contribution to Cloud-Based Engineering Systems

Large and complex distributed embedded systems form a cloud-based systems, due to large number of modules that perform various tasks and exchange data. However, such system is presented to an end-user as a single, complex, global system.

The proposed EmbedCloud accelerates and simplifies synthesis process of such systems and through the use of the CloudBus protocol provides a process synchronization and control mechanism for a number of processing units distributed in a network. The CloudBus protocol allows fast reconfiguration of the implemented system and allows to form the distributed embedded system with devices based on a different architectures.

The distributed embedded systems are able to overcome the limitations of single-unit environments and scatter the system into a various number of processing units.

3 CloudBus Protocol

The CloudBus protocol is one of the methods of the data exchange and concurrent process synchronization in the distributed embedded systems. It realizes distributed control method with a various number of end modules, where all of them are equal to each other and each of them implements a functional part of designed concurrent process. The CloudBus communication model allows the significant savings of the transmitted data between end modules [9]. Schematic diagram of the CloudBus protocol network topology is shown in Fig. 1.

The data transfer between end modules, necessary for process synchronization is executed only when one of the end modules requires information (input or variable state) from outside their own native resource variables. The end module broadcasts question to the system (other end modules) about the state of the specified variable, e.g. *if p1 = 0?*.

The end module that natively controls this variable (e.g. *p1*) sends the answer to the system (other end modules) when it gets previously quested state.

Basic data frame of the CloudBus protocol is shown in Fig. 2.

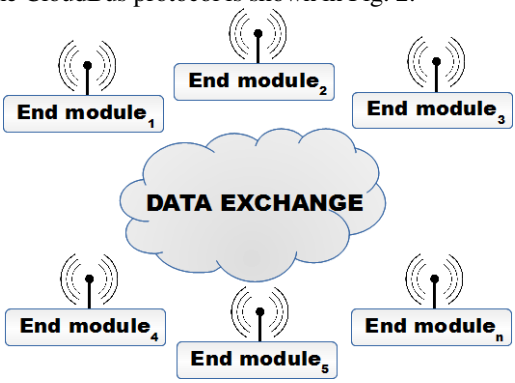

Fig. 1. Schematic diagram of the CloudBus protocol network topology

CNT	FUNC	VARS	DATA	CRC

Fig. 2. Data frame structure of the CloudBus protocol

The fields of the protocol corresponds to: *CNT* – entire frame length; *FUNC* – command code (e.g. question/answer); *VARS* and *DATA* represents binary arrays of the variables and their states (values); *CRC* – the CRC error checksum.

In order to ensure safety and proper operation of the system, each module has a defined maximum time for receiving a response with quested variable. Alternatively, the CloudBus protocol allows to check presence of other end modules in the system.

The comparison of the CloudBus protocol with other protocols used in the industry (e.g. Modbus RTU, Profibus DP or DeviceNet protocol) showed significant savings in the amount of transmitted data between end modules in the distributed embedded system [9].

4 EmbedCloud – Design and Implementation Method

4.1 Hardware-Software Synthesis of Embedded Systems

The general requirements [1, 7] of the distributed embedded system are listed below:

- concurrency – concurrent process execution,
- openness, flexibility, scalability – fast reconfiguration and development,
- resource sharing – ability to share data between end modules,
- platform independent model architecture – end modules may have different hardware architecture.

Fig. 3 presents a comparison of two synthesis models of the embedded systems: *(a)* where one module performs the entire concurrent process, and *(b)* where the process has been decomposed into individual sub-processes and is implemented by three independent end modules.

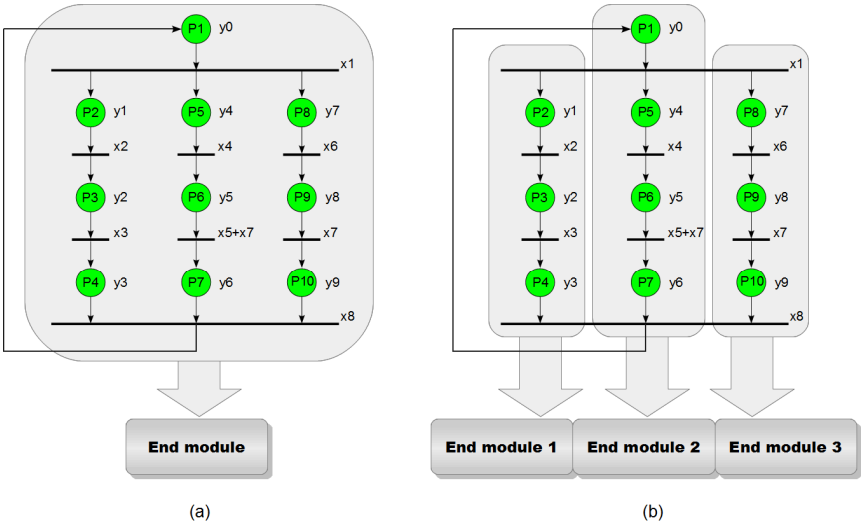

Fig. 3. Single module (a) embedded system vs. distributed (b) embedded system synthesis model

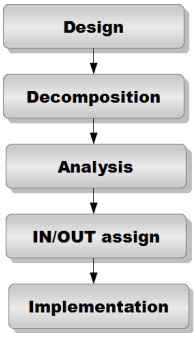

Fig. 4. General diagram of the concurrent process synthesis of the distributed embedded system

A general diagram of the concurrent process synthesis of distributed embedded system is shown in Fig. 4. The first step is to *Design* (describe) concurrent process using one of the models e.g. Petri nets. The next step is *Decomposition* into individual sub-processes. The decomposition can be realized by different criteria imposed by the designer such as: system reaction time, hardware resource usage or designer's manual decomposition. After structural and behavioural *Analysis* of the decomposed process, *IN/OUT assign* is made – the designer assigns inputs/outputs for each end module. Finally, each module in the distributed embedded system is implemented.

4.2 EmbedCloud Structure

The proposed new design and implementation method (called EmbedCloud) of the distributed embedded system, defines an implementation model of ordered structure

for each end module, Fig. 5. Implementation structure is the same for all end modules of the distributed embedded system.

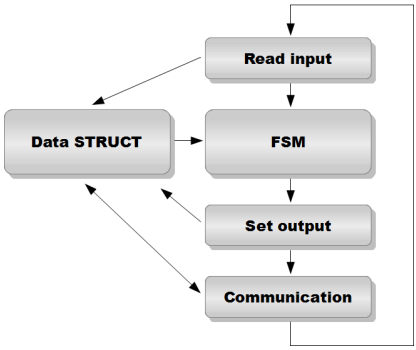

Fig. 5. General diagram of the EmbedCloud implementation method

The main step in the EmbedCloud implementation flow is *DATA STRUCT*. It is responsible for storing the information about the state of the native I/O and the state of shared variables in the system. In the *Read input* block native I/O states are read and stored in *DATA STRUCT*. Afterwards *FSM* (Finite State Machine) block using the data from *DATA STRUCT*, executes a selected control process or some of its parts. In the next step (*Set output*) the native I/O states are set. After performing all process operations, the information is exchanged (*Communication*) between end modules using the CloudBus protocol. The CloudBus and EmbedCloud method are closely related, because the CloudBus protocol is responsible for obtaining and making available the necessary data from other end modules. Moreover, it provides a process synchronization mechanism. In the *Communication* step, the end module broadcasts to the system (other modules) messages about the state of specific variables that were asked. The received responses are stored in the *DATA STRUCT* and algorithm returns to the *Read input* and starts again.

Such structure of the EmbedCloud allows to implement end modules, which differ only in the following elements: the native I/O and implemented control model (*FSM* block). This allows to use different architecture of each end module, such as micro-controller, FPGA or DSP devices. Furthermore, it forms the basis for the automatic code generation algorithm.

4.3 The Conception of the Automatic Code Generation Algorithm

The EmbedCloud structure allows to propose a simplified version of the algorithm for the automatic code generation for the end modules, Fig. 6.

After designing the concurrent model (*Model design*), the algorithm starts performing the decomposition. The *Decomposition* output are files in PNML or HPN format that contains divided sub-processes (represented by structured text) for each end module in the system. In the loop, files are parsed and processed for the structural

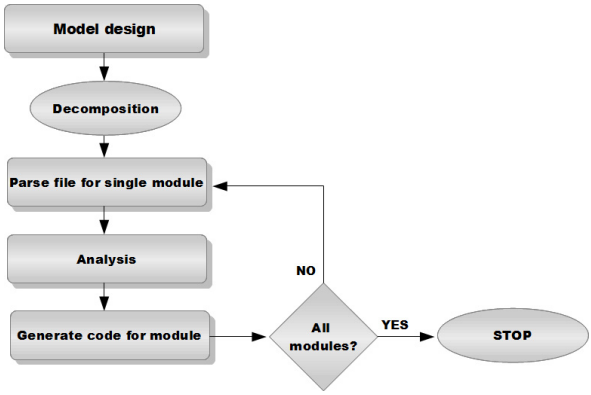

Fig. 6. Proposed algorithm for automatic code generation

and behavioural *Analysis*. After processing the code is generated using the Embed-Cloud method. The process stops, when all files have been processed. The output of the algorithm is source code for each end module.

5 Experimental Results

To verify and evaluate the performance of the EmbedCloud method a concurrent process was designed with interpreted 1-bounded Petri net model, Fig. 3 (a). Fig. 3 (b) presents Petri net model after manual decomposition to three individual sub-processes for each end module. For the *Module 1* and *Module 3* it was necessary to add resting places (*RP1*) to preserve the proper functioning and synchronization of the end modules. The *Module 2* executes the *P1* place with *y0* output. The transitions and allowing arc are described by logic equations e.g. *x3*P7M2*, which means: input *x3* AND place *P7* from *Module 2*.

Hardware tests and synthesis verification of the distributed embedded system was performed on the testing platform built with AVR family, ATmega8 microcontrollers.

Table 1 presents a comparison of the synthesis results in the terms of memory usage and reaction time. The results were obtained with EmbedCloud method to generate a source code for ATmega8 microcontrollers. All of the end modules were transmitting data with the RFM12 transceiver (manufactured by Hope Microelectronics) using the CloudBus protocol. The results demonstrated following memory usage for the ATmega8 end module: 25% of total (8k bytes) program memory, 6% of total (1k byte) data memory. The maximum reaction time (CPU clock set to 16MHz) is 0,07ms for CPU processing and less then 2ms when external communication (at 115,2kbit/s) is also performed. Concluding, the hardware resource usage and reaction time is negligibly small.

It should be noted, that memory usage and reaction time depends on the concurrent model and its decomposition. Incorrectly performed decomposition may negatively affect the obtained results, hence it is necessary to make appropriate optimization and decomposition process.

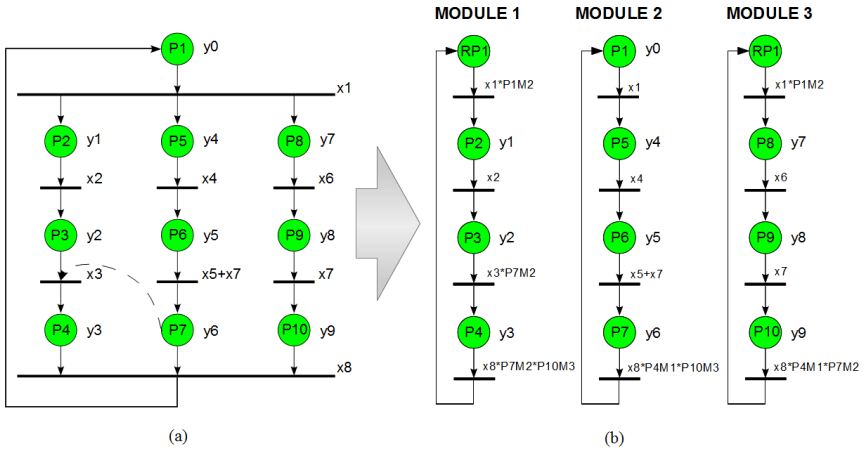

Fig. 7. Petri net model before (a) and after (b) decomposition

Table 1. Memory usage and reaction time for implemented distributed embedded system

End Module	Program Memory Usage [Bytes]	Data Memory Usage [Bytes]	Max. CloudBus Data Transmit [Bytes]	Max. CloudBus Reaction Time [ms]	Max. CPU Reaction Time [ms]	Max. Total Reaction Time [ms]
Module 1	1928	60	25	1,74	0,06	1,80
Module 2	2134	60	25	1,74	0,07	1,81
Module 3	1888	60	20	1,39	0,06	1,45

6 Conclusion

The paper presented the new EmbedCloud design and implementation methodology of the distributed embedded systems and formed the basis for the automatic code generation algorithm to accelerate and simplify the synthesis process of such systems.

Experimental results verified the proposed methodology and demonstrated a negligible memory usage (less than 2k bytes of program memory and 60 bytes of data memory usage) and minimal reaction time to the data requests (reaction time for CPU processing was 0,07ms; after including the data transmission to other modules – the result was less than 2ms for entire sub-process). However, it is necessary to implement and investigate various number of real distributed embedded systems with different wired and wireless network connections.

Current and further research focuses on developing the automatic code generation algorithm and graphic software tool where the embedded system designer, draws concurrent process model, assigns I/O and gets generated source code as output. Depending on the requirements, the software tool gives code in ANSI C language

or one of the HDL (Hardware Description Language), such as VHDL or Verilog. Using the automatic code generator and design tool, allows the significant time savings in the design and implementation of the distributed embedded systems.

References

1. Marwedel, P.: Embedded system design. Springer, New York (2011)
2. Nicolescu, G., Mosterman, P.J.: Model-Based Design for Embedded Systems. CRC Press, Boca Raton (2009)
3. Moreira, T.G., Wehrmeister, M.A., Pereira, C.E., Petin, J., Levrat, E.: Automatic code generation for embedded systems: From UML specifications to VHDL code. In: Industrial Informatics (INDIN), pp. 1085–1090. IEEE (2010)
4. Hsieh, W.H., Kao, S.P., Tan, K.H., Chen, J.L.: Energy-saving cloud computing platform based on micro-embedded system. In: Advanced Communication Technology (ICACT), pp. 739–743. IEEE (2014)
5. Raghav, G., Gopalswamy, S., Radhakrishnan, K., Hugues, J., Delange, J.: Model based code generation for distributed embedded systems (2010)
6. Babic, J., Marijan, S., Petrovic, I.: Introducing Model-Based Techniques into Development of Real-Time Embedded Applications. Automatika **52**(4), 329–338 (2011)
7. Adamski, M., Karatkevich, A., Wegrzyn, M.: Design of embedded control systems, vol. 267. Springer, New York (2005)
8. Bukowiec, A., Tkacz, J., Gratkowski, T., Gidlewicz, T.: Implementation of Algorithm of Petri Nets Distributed Synthesis into FPGA. International Journal of Electronics and Telecommunications **59**(4), 317–324 (2013)
9. Krzywicki, K., Andrzejewski, G.: Data exchange methods in distributed embedded systems. In: New Trends in Digital Systems Design, Fortschritt - Berichte VDI, Dusseldorf, vol. 836, pp. 126–141 (2014)
10. Bukowiec, A., Mroz, P.: An FPGA synthesis of the distributed control systems designed with Petri nets. In: Networked Embedded Systems for Every Application (NESEA). IEEE, London (2012)