Architecture of FPGA Embedded Multiprocessor Programmable Controller

Zbigniew Hajduk, *Member, IEEE*, Bartosz Trybus, *Member, IEEE*, and Jan Sadolewski

***Abstract*—This paper presents the design and implementation of a multiprocessor programmable controller in field-programmable gate array (FPGA). The novelty of the proposed solution is that it combines two approaches used so far in the domain of FPGA implementations of control algorithms, i.e., program based and hardware coded, and applies multiple processors in a single FPGA chip. The controller is programmed according to the IEC 61131-3 standard and runs control tasks in parallel. Performance tests of the prototype show that it is able to execute control programs significantly faster than industrial programmable logic controllers.**

***Index Terms*—Field-programmable gate array (FPGA), industrial control, programmable logic controllers (PLCs).**

I. Introduction

PROGRAMMABLE logic controllers (PLCs) are a base for industrial automation due to a wide area of applications, robustness, and easy programming using IEC 61131-3 languages [1]. Since some time, field-programmable gate array (FPGA) implementations of PLC algorithms have become an important research area because of structural flexibility and speed of FPGA chips [2], [3]. Dedicated controllers [4] or specific filters for image processing [5] are examples of such approach. Because performance of standard PLCs does not always satisfy requirements of highly responsive systems [6], [7], parallel execution of programs provided by FPGAs creates remarkable potential.

Papers involving various techniques for implementations of control algorithms in FPGA may be divided into two groups. The first one applies a conversion from a formal description of a program to a code in hardware description language (HDL). For example, conversion of PLC instructions into very high-speed integrated circuit HDL is presented in [7], and conversion of ladder diagrams (LDs) in [8] and [9]. Implementation of reprogrammable logic controllers based on matrix model of Petri nets is described in [10]. Much faster execution of hardware-coded programs is the basic advantage of the first approach. However, any change of control concept requires the use of three software tools, i.e., PLC program editor, conversion to HDL, and FPGA synthesis.

The second approach assumes that complete programmable controller, or at least its essential components, is implemented in FPGA. Such FPGA-based PLC prototype can be programmed by a single tool without conversion to HDL. This is demonstrated in [11], where a general-purpose FPGA micro-PLC executes LD. Some of the proposed solutions involve the main processor and a coprocessor, as, for instance, in [12], where a reduced instruction set computer (RISC) processor is dedicated for PLC, and a Boolean coprocessor accelerates bitwise operations. The hardware–software platform for testing two FPGA central processing units (CPUs), i.e., one for bit and another for byte (word) operations, is described in [13].

FPGA technology is also becoming interesting for manufacturers of control equipment. The FPGA-based FM 352-5 Boolean processor module from Siemens AG [14] may be an example. The module improves speed of logic calculations and may operate as a component of a larger system or as a standalone controller. It is programmed using Siemens STEP-7 LD or function block diagram (FBD) languages and executes program instructions in parallel. More powerful programmable automation controllers involving FPGAs are offered by National Instruments (NI). The CompactRIO platform [15] includes a user-programmable FPGA chassis, an embedded controller with floating-point (FP) processor, and input/output (I/O) modules. CompactRIO is programmed by (NI) LabView graphical programming tool. The Siemens and NI solutions are close to the first approach of PLC program implementation in FPGA. Program is developed by using an integrated software, which implicitly invokes FPGA HDL synthesis tools.

There is also a growing interest in FPGA implementations of multiprocessor systems [16]. For instance, the design of a symmetric multiprocessor system for parallel execution of multiple threads is presented in [17]. Another system implements wideband code division multiple access algorithm for software defined radio [18]. Multiple very long instruction word processors accelerate calculations of the oxygen saturation map [19].

In this paper, we present a multiprocessor controller composed of identical CPUs implemented in a single FPGA chip, programmed in all IEC 61131-3 languages. Thus, the idea corresponds to the second approach of PLC FPGA implementations aforementioned, but by using multiple CPUs, the control program can be split into tasks executed in parallel, hence much faster. Moreover, the controller integrates the so-called hardware function blocks (HFBs) developed according to the first approach, i.e., configured directly in HDL, what increases the speed even more. An HFB may represent continuous controller, filter, etc. Thus, the solution combines the two existing approaches, i.e., program based and hardware coded. The preliminary realization of this idea was presented in [20], where an FPGA controller with a single processor

Manuscript received March 25, 2014; revised June 23, 2014 and August 12, 2014; accepted August 31, 2014. Date of publication October 14, 2014; date of current version April 8, 2015.

The authors are with the Department of Computer and Control Engineering, Rzeszow University of Technology, 35-959 Rzeszow, Poland (e-mail: zhajduk@kia.prz.edu.pl; btrybus@kia.prz.edu.pl; js@kia.prz.edu.pl).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIE.2014.2362888

was designed. The concept of using multiple processors for execution of more than one control task was proposed in [21]. However, that paper, after presenting a preliminary architecture, dealt with software aspects of the solution.

Here, we present results of further development with the following novel aspects: enhanced multiprocessor architecture for parallel execution and synchronization of IEC 61131-3 tasks, extended architecture of HFBs, and custom FP unit (FPU). The new hardware involves improved access to global memory, wider 32-bit global memory bus, direct access registers (DARs) for fast task synchronization, and configuration registers (CRs) for instant triggering of a processor either by other processors or by an external input. Performance improvements are justified by a set of tests, with comparison to a few commercial PLCs. A complete prototype of the FGPA-based PLC multiprocessor controller is also described.

The FPGA controller presented here is programmed by the Control Program Developer (CPDev) engineering tool, developed some time ago [22]. The tool has been extended for FPGA to handle multiprocessing and HFBs.

This paper is organized as follows. Section II explains the concept how general-purpose microprocessors programmed so far by the CPDev can be replaced by FPGA. Architecture of FPU is presented in Section III. Section IV presents an integration of the HFBs and CPU. General architecture of the multiprocessor controller with configurable number of CPUs is described in Section V. Extension of CPDev tool for multitasking, use of global memory, and synchronization mechanisms are described in Section VI. Prototype of the multiprocessor controller is presented in Section VII. Results of performance tests involving comparisons with industrial PLCs and reductions of time in terms of CPUs, HFBs, and hardware improvements are given in Section VIII.

II. Concept and Architecture of CPU

The concept of the FPGA-based programmable controller was originally affected by the operation structure of the CPDev programming environment [22]. The primary idea was to prepare FPGA-based platform for CPDev programs written in IEC 61131-3 languages.

CPDev is a target-independent tool originally developed for programming controllers with general-purpose AVR, ARM, and x86 processors. Similar environments supporting IEC 61131-3 programming include well-known CoDeSys, ISaGRAF, or MULTIPROG. Much like the three, CPDev supports various platforms; however, it is mostly oriented toward small- and medium-scale controllers with limited resources (computing power, memory). Although it lacks some advanced features of the commercial counterparts, some unique solutions are available, such as SysML modeling and support for unit tests [23].

CPDev has some industrial applications, including ship control and monitoring [24] or small distributed control systems [23]. Overall structure of the tool is shown in Fig. 1. Programs written in structured text (ST) or instruction list (IL) textual languages or translated to ST from FBD, LD, or sequential function chart (SFC) diagrams are compiled into the virtual

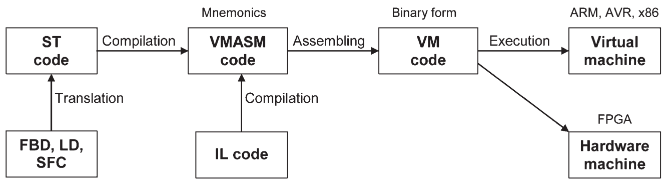

Fig. 1. Translation stages of CPDev programs for the hardware machine.

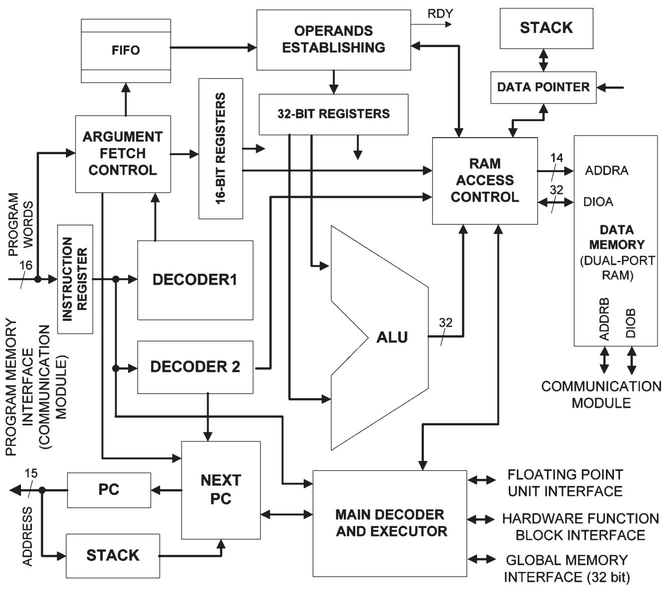

Fig. 2. CPU architecture of the programmable controller.

machine assembler (VMASM) universal executable code, and executed at the target processor by the virtual machine (VM), unlike, e.g., CoDeSys, which directly generates native machine code for several CPU types. VMASM is an assembler-type language, not related to any particular processor, but oriented much toward IEC 61131-3 [25]. The software VM machine is written in C; thus, it may run on different hardware platforms, from 8-bit microcontrollers up to 32/64-bit general-purpose processors. It consists of universal and platform-dependent modules (the latter prepared by controller manufacturers). The VM model and architecture has been presented in [25].

Having such a tool, it is probably not surprising that the basic concept behind development of FPGA controller was to replace the VM by the hardware machine implemented in FPGA. Thus, the controller is in fact a dedicated FPGA processor targeted for execution of VMASM code. In addition, the FPGA implementation provides new capabilities, such as multiprocessing and HFBs, not available in general-purpose processors.

A dedicated 32-bit CPU, which implements VM instructions is an essential part of the controller. Two versions of the CPU have been investigated in [20]. The one with better outcome of performance tests has been chosen for further extensions here.

The architecture of one CPU in the multiprocessor controller is shown in Fig. 2. The ability to perform some internal operations in parallel is an important feature having direct influence on CPU performance. For example, when a consecutive argument is read from program memory, data memory is accessed and an operand is read and stored in the 32-bit register at the same cycle. Analogously, writing an arithmetic logic unit (ALU) result is performed at the same clock cycle as reading the next instruction code from program memory (similarly to pipeline processing).

The extensions introduced into the CPU for the new multiprocessor controller in comparison with the earlier design [20] include communications with other CPUs via global memory, interfaces to FPU and HFBs, additional instructions for semaphore operations on global variables, and in-circuit debugging capabilities. In comparison with [21], WAIT instruction for tasks synchronization has been added and instructions related to global memory communication have been altered and optimized. The NEXT PC block (see Fig. 2) responsible for asserting the next value of program counter (PC) has also been upgraded, i.e., instructions that modify PC (jumps) are now performed in a single instruction cycle. The mechanism applied here is similar to the one presented in [26], being rather unique feature as far as typical microprocessors are concerned.

III. FPU

FPGA structures are optimized for fixed-point computations. Implementation of FP arithmetic requires large amount of resources; thus, maximum clock frequency becomes relatively low [27]. To overcome this in part, efficient implementations of FP arithmetic have been designed, for example, sequential and pipelined divider and square root block [27], logarithm and exponential function [28], and a set of specific libraries [29].

The ALU of the CPU mentioned in the previous section (see Fig. 2) handles integer numbers only. REALs in the CPDev tool are single precision FP numbers of the IEEE 754 standard. Apart from basic operations like addition, subtraction, multiplication and division, the controller FPU also performs comparisons (equal, more than, etc.), integer to FP two-way conversion with rounding and truncating, determination of absolute value, sign inversion and some functions of IEC 61131-3 (MIN, MAX, LIMIT).

Acceptable tradeoffs between resource requirement and speed of computing have been an important goal while designing components of the FPU for the multiprocessor controller. As an illustration, let us briefly discuss design issues of the FP multiplier.

Multiplication of two FP numbers is executed by multiplication of mantissas and addition of exponents. Simplified architecture of the FP multiplier is shown in Fig. 3. Normalization block on the right side is a simple shift register, which conditionally right shifts the result of mantissas multiplication. The result is rounded using round to nearest even mode. Rounding requires three additional bits determined in A3 block, i.e., guard, round, and sticky bits.

Four versions of the multiplier have been investigated. They differ in the design of the fixed-point multiplier block A3, whose structure has essential influence on final implementation parameters, such as FPGA resource requirement and calculation speed. The standard serial paper-and-pencil multiplication algorithm, as well as parallel, pipelined, and fast array multipliers with or without FPGA embedded hardware multipliers, has been examined. Performance tests of the multiplier versions

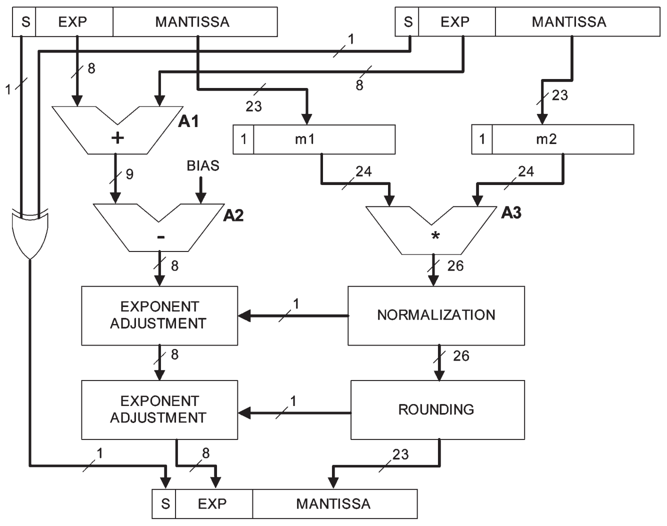

Fig. 3. Architecture of FP multiplier.

and implementation parameter evaluations have shown that the paper-and-pencil algorithm is a good compromise between speed and resource requirements. Therefore, this version has been chosen for the FPU FP multiplier. Similar evaluations have been carried out for other components of the FPU before final implementation.

IV. HFBs

HFBs are another feature of the FPGA controller. They may drive custom-designed peripherals and provide access to low-level hardware, e.g., to display. Moreover, HFBs can also be used as hardware accelerators to increase computing speed. They may implement specific control algorithms, such as continuous control, neural networks, etc., becoming a bridge to direct implementation of PLC programs in FPGA (see Section I).

HFBs can be particularly useful for time-demanding applications, as, for instance, drivers of fast sensors and actuators built in microelectromechanical system technology, whose response time is, in some cases, less than 50 ns.

Here, HFBs are designed as intellectual property (IP) cores; thus, they can be integrated with the controller by independent designers. The way of HFBs connection with CPU or HFB access arbiter is presented in Fig. 4. A PC application, which is developed to facilitate HFBs integration, automatically generates Verilog HDL code for connection between the blocks and CPU. Dedicated handshaking protocol for data transfer between hardware blocks and CPU has also been implemented.

V. Multiprocessor Programmable Controller

As stated in the introduction, FPGA implementations of multiprocessor systems become increasingly important [16], [17], [19]. In this section, architecture of multiprocessor system on-chip (MPSoC) implemented in FPGA for the controller is presented. It consists of a number of processors with FPUs, HFBs integration section and communication module (CM). The MPSoC operates as programmable controller, which is able to run multiple control tasks simultaneously. Each task is

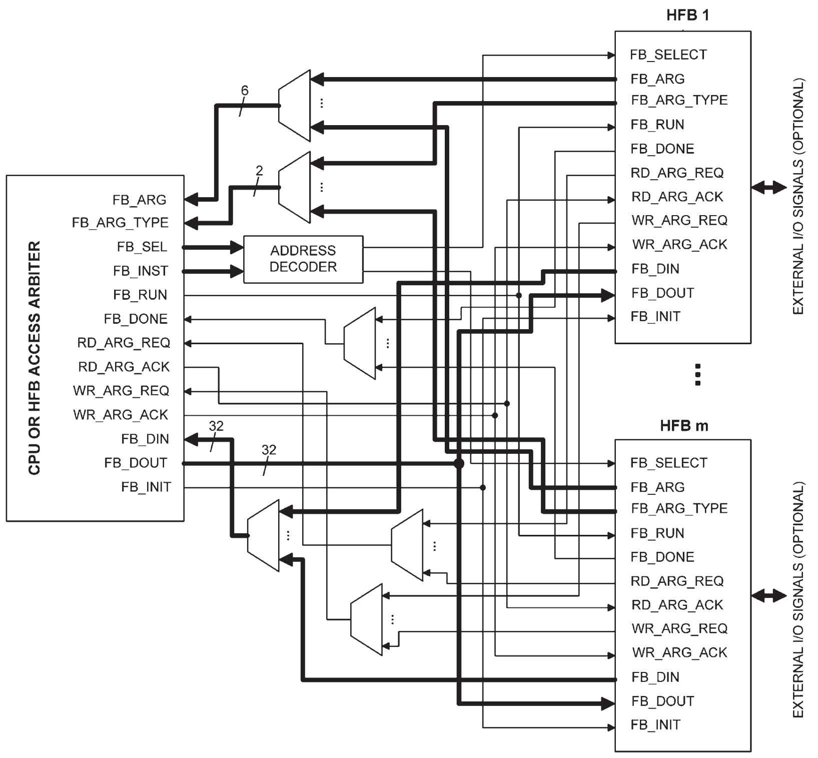

Fig. 4. Connection between CPU and HFBs.

executed by a separate processor. The design goal has not been to make a new contribution to the development of multiprocessor systems but rather to apply such systems into the domain of programmable controllers. Here, control, technical computations, communications, etc., can be assigned to different CPUs. In addition, time-critical and nontime-critical processing functions can be separated.

General concept of our multiprocessor system architecture is primarily determined by IEC 61131-3 software model (see Section VI). In addition, we have introduced some features to improve interprocessor synchronization and communication, such as WAIT instruction and semaphores.

Simplified architecture of the controller is shown in Fig. 5. A number of CPUs (or another words, CPU cores), equipped with FPUs described in Sections II and III, respectively, are essential components of the controller. It also includes an initiator core, being a simple microprocessor responsible for initialization of global variables. The initiator is triggered on power up or after loading new program into the controller. Only after initialization, all CPU cores begin to execute their tasks.

Following original architecture of the CPDev VM [25], the CPU cores communicate via global memory. As shown before in Fig. 2, each CPU is interfaced to global memory by 32-bit bus. Since the CPU has also its own integrated data memory, therefore, global memory is used only when common data need to be exchanged. This significantly reduces possible bottlenecks.

Global memory address space is divided into two ranges accessed in the same way. The lower range, i.e., from address 0 up to a configured value, is reserved for I/O peripherals. The upper range maps physical synchronous RAM, implemented as dual-port block RAM, and special function registers (SFRs). The memory stores typical global variables, whereas SFRs, which occupy the highest address space, include DARs and CRs. Collision-free access is provided by the global memory and I/O access arbiter block (see Fig. 5). DARs are an exception, since read operation can be done at any time by any CPU without involving the arbiter. This remarkably reduces the time of synchronization and communication between CPUs. CRs control triggering of selected CPUs (if in single mode) by other CPUs or in response to active edges of selected input signals.

The arbiter uses protocol similar to token-passing to grant access. Handshaking is applied for data transfer between CPU cores and the arbiter. Two arbitration algorithms have been examined. First one, relatively simple, was applied in the preliminary version of the controller prototype [21], but with poor throughput. In the current version, the complex priority encoder and the optimized clock scheme (granting access reduced to one clock cycle) have improved global data throughput substantially.

HFBs can be implemented in the controller in two ways. In the first one, each CPU core has its own set of HFBs. The second way assumes that HFB instances can be shared among CPUs. In Fig. 5, the HFB instances $1, \ldots, m$ (top left) are permanently assigned to CPU 1. These blocks are not accessible to CPUs $2, \ldots, n$. On the other side, the HFB instances $m + 1, \ldots, q$ (top right) are shared among all CPUs except CPU 1.

Although in the IEC software model [1], function blocks are not shared among resources (CPUs), sharing may be useful in case of their hardware counterparts. For example, each CPU core may access common peripherals, such as display, touch panel, etc. The sharing requires an arbiter block to provide collision-free access to HFBs. The arbiter uses a similar handshaking protocol as the one for the global memory and

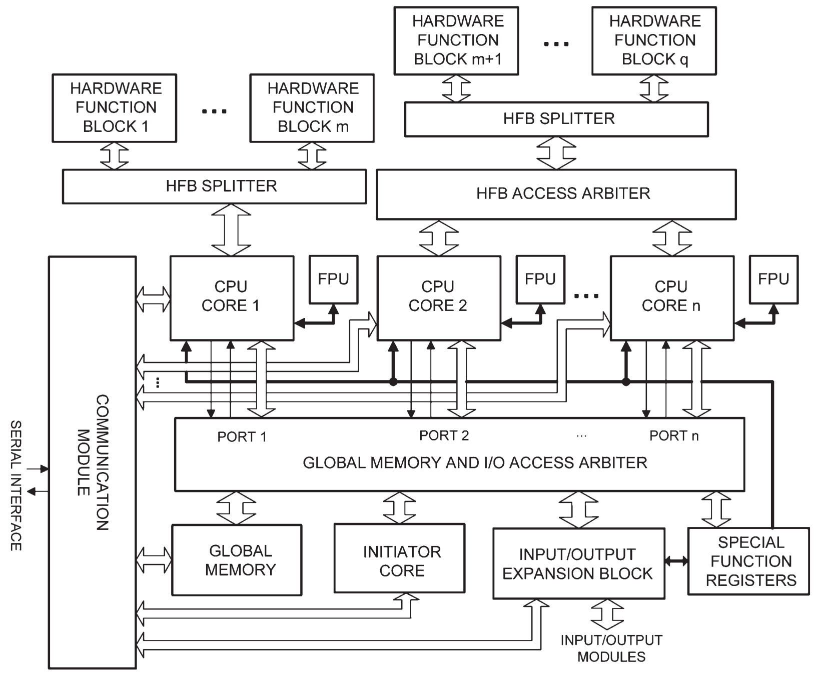
HARDWARE FUNCTION BLOCK 1
HARDWARE FUNCTION BLOCK m
HARDWARE FUNCTION BLOCK m+1
HARDWARE FUNCTION BLOCK q
HFB SPLITTER
HFB SPLITTER
HFB ACCESS ARBITER
CPU CORE 1
CPU CORE 2
CPU CORE n
FPU
COMMUNICATION MODULE
SERIAL INTERFACE
PORT 1
PORT 2
PORT n
GLOBAL MEMORY AND I/O ACCESS ARBITER
GLOBAL MEMORY
INITIATOR CORE
INPUT/OUTPUT EXPANSION BLOCK
SPECIAL FUNCTION REGISTERS
INPUT/OUTPUT MODULES

Fig. 5. Architecture of the multiprocessor programmable controller.

I/O access arbiter. The HFB splitter in Fig. 5 consists of multiplexers, address decoders, and specific connections shown in Fig. 4.

As a comment we explain, why two interfaces, i.e., one for global memory and another for HFBs have been applied, instead of adopting one of standard interfaces, such as processor local bus (PLB) used in MPSoC [30]. The answer is the following: implementation of the PLB requires an IP block with a PLB core, which is usually vendor dependent. In addition, functionality of the PLB remarkably exceeds needs of the controller. On the contrary, our two interfaces are simple, require small amount of logic resources and can be optimized for specific operations.

CM shown in Fig. 5 provides data transfer from/to the CPDev programming tool. Basic function of the CM is to load program code from an external flash memory or from a PC into program memory of each CPU core, implemented as dual-port block RAM. CM also reads and writes global variables during online monitoring and testing, and provides access to program and data memories of each CPU. Some special functions as in-circuit debugging are also available.

CM implements RS-232 serial interface to PC host. Communication protocol applies Intel HEX format. Each data record, equipped with a check sum, must be acknowledged by the receiver (PC or CM module) to ensure reliable data transfer.

VI. Software Architecture

The FPGA multiprocessor controller is programmed in CPDev engineering environment [22], [25]. Here, we describe extensions to CPDev to handle HFBs, multitasking, and synchronization. Some comments on compilation and resulting binary file are also given.

A. HFBs

A software project is created in hierarchical manner, beginning from the selection of the number of CPU cores. IEC 61131-3 program organization units (POUs), i.e., programs, function blocks, and functions, are then created or acquired from libraries. Two types of function blocks are available, i.e., program blocks and hardware blocks. Program blocks are implemented in software, as typically in PLCs. Hardware blocks are written in HDL and configured in FPGA chip (see Section IV). However, HFBs must still be declared, similarly to program function blocks, but with a special compiler directive indicating a hardware block, e.g.,

```
(*$HARDWARE_BODY_CALL ID : 0002*)
```

The directive denotes a HFB-type identifier (ID), which corresponds to the value of the CPU interface bus FB_SEL shown in Fig. 4. To handle multiple instances of the HFB defined in a program, each one is automatically given a successive number (related to FB_INST in Fig. 4). Based on FB_SEL and FB_INST, the proper block is selected and activated by the address decoder from Fig. 4.

The HFBs instances can be invoked from the program in the same way as program blocks. Unlike their software counterparts, HFB instances run concurrently with the controller CPUs, i.e., HFBs can do their job between calls.

B. Control Tasks

Since the controller consists of a number of CPUs, it may be viewed as a group of single-task controllers (or task execution units). It can be configured to run a number of control tasks in parallel, each one by separate CPU. For instance, fast logic control (LC) may run in parallel with slower continuous control, communications and human–machine interface (HMI).

The original single-task CPDev tool [22] has been extended to incorporate a number of tasks into a single project. Thus, the IEC 61131-3 software model [1] is implemented in the following way. Configuration is created for the controller itself. The user creates a task and assigns to it appropriate POUs. Each task is executed by single CPU core, which corresponds to a resource in terms of IEC. The user specifies which CPU will execute particular task.

The tasks can be executed cyclically with selected periods (real-time periodic tasks, common in control applications), as endless loops or as aperiodic real-time tasks (on demand). Aperiodic tasks are triggered by dedicated Boolean variables mapped to CRs (Section V). If such variable becomes true, the task is executed immediately. Aperiodic tasks can be also triggered by an external input signals and used for alarm routines, start-ups, shut-downs, handling HMI, etc.

Global variables shared by all tasks are the way of data exchange. To provide consistency of global variables during calculations, the tasks do not operate directly on these variables but on so-called process images or, in other words, on local shadows. At the beginning of the execution the CPU copies global variables into internal memory, creating the shadows. Only the shadows are used for execution of the task; thus, changes of global variables caused in the meantime by other CPUs do not affect calculations. At the end the updated shadows are stored back in the global memory. Collision-free access to the global variables (what means that two or more tasks cannot read/write particular variable at the same time) is provided by the hardware arbiter. The programmer still has to make sure that global variables are used in correct way, e.g., writing the same output variable by two tasks can be treated as a programmer's error.

C. Task Synchronization

Depending on the system being designed, synchronization of tasks can be either the problem of cooperation or competition (for a shared resource). The multiprocessor controller provides mechanisms to handle both cases.

Frequent approach in parallel programming is to implement a server task, which performs calculations upon request of other tasks (clients). To handle such scenario, the controller involves a special WAIT instruction used both by the server and the clients. While executing WAIT, the server task goes into wait state expecting requests from clients. A client can then trigger the server and continue execution up to complementary WAIT at the point where the server result is needed. The following code uses the variable SERVER to synchronize client (left) and server tasks at particular times.

```
PROGRAM CLIENT_TASK        PROGRAM SERVER_TASK
(*start server*)           (*wait for request*)
SERVER := TRUE;             WAIT(TRUE, SERVER);
(*do other things*)        (*calculate result*)
WAIT(FALSE, SERVER);        (*indicate job done*)
(*process the result*)     SERVER := FALSE :
```

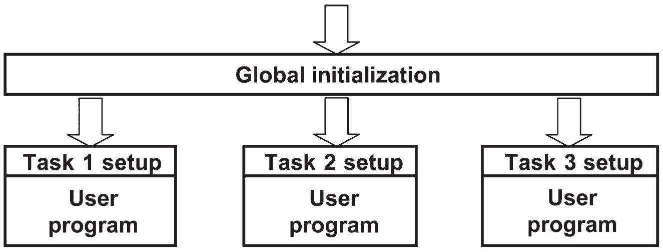

Fig. 6. Structure of executable code for the multiprocessor controller.

Some of the HFBs control peripherals what may create conflicts in parallel environment. The conflicts are avoided either by assigning such blocks to particular CPUs or by HFB access arbiter (see Fig. 5). In the second solution, the controller implements counting semaphores for program-based task synchronization and mutual exclusion. Actually, the solution applies an extension of two-state semaphore block SEMA defined in the first version of IEC 61131-3. Here, a semaphore is a global integer variable accessed from task by LOCK and UNLOCK functions, implemented as dedicated CPU instructions. UNLOCK increments semaphore value, whereas LOCK decrements it. When the value is zero, the locking task is suspended until one of the other tasks unlocks the semaphore. To check (without waiting) whether a semaphore is unlocked, the instruction TRYLOCK is used.

D. Program Compilation and Execution

Stages of translation of IEC programs into executable form have already been shown in Fig. 1. During compilation, POU code is first converted into VMASM mnemonics and then assembled into binary machine code [22]. VMASM instructions handle IEC 61131-3 data types: BOOL, BYTE, INT, WORD, REAL, TIME_OF_DAY, DATE_AND_TIME, and others. Machine code involves arithmetic, Boolean and bitwise operations, comparisons and jumps [25]. In addition, operations to handle arrays and structures are available.

Execution times of the machine instructions are determined by a number of clock cycles and actual clock frequency. Since they are known before the program is run, one can estimate to some extent how long the execution will take. Following this idea, CPDev environment has been extended with estimation mechanism, which simulates program execution; collects time information for each instruction; and presents maximum, minimum, and average. Thus, the programmer can check whether the code fulfills time cycle requirement.

In case of the FPGA controller, the CPDev compiler creates a separate binary file for each task, i.e., for each CPU core. In addition, a code for initialization of global variables is created; thus, general structure of the executable code looks as in Fig. 6.

The CPDev tool involves a component called CPSim that loads, runs, and tests the compiled software using the controller CM module (see Section V). During online testing (commissioning), global and local task variables are monitored. Actual execution time of each task is measured and given with 1-μs resolution. The current state of the selected CPU can be displayed, i.e., system clock, cycle counter, flags of cycle

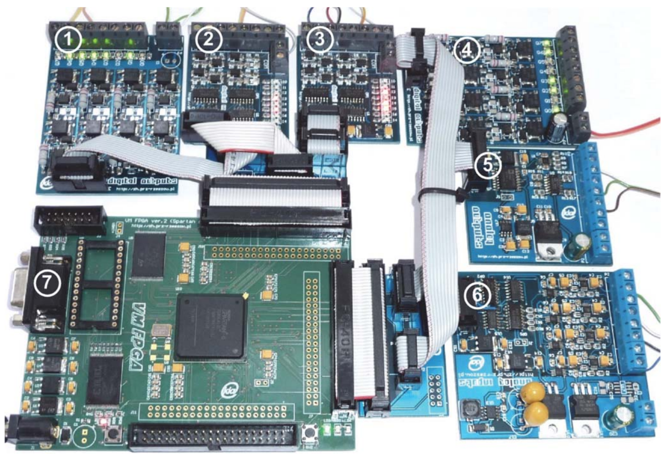

Fig. 7. Prototype of the multiprocessor programmable controller.

overrun, and violation of array sizes. CPSim also provides basic debugging facility in the form of breakpoints that stop hardware machine at particular lines of code to check values of variables.

VII. Prototype of the Controller

The multiprocessor programmable controller has been designed as a set of IP cores described in Verilog HDL. The description does not depend on specific FPGA family or vendor. Hence, the controller may be implemented in any FPGA chip with an enough amount of logic resources. A prototype hardware platform developed for verification of functional properties and performance tests is described in the following.

Although the controller HDL description is not oriented toward any particular FPGA family, we have chosen a popular and relatively inexpensive, Spartan-6 family from Xilinx as an essential component. The prototype is shown in Fig. 7. It consists of seven modules, i.e., digital outputs (1), two modules with digital inputs (2 and 3), digital outputs (4), analog outputs (5), analog inputs, (6) and the main module with FPGA chip (7).

The main module involves Xilinx Spartan 6 FPGA XC6SLX100-3FGG676. Analog I/O modules use specialized Analog Devices integrated circuits. All I/O modules are galvanically separated from the main module (by means of optical and digital isolators).

The FPGA chip has enough logic resources for implementation up to 16 CPU cores. The maximum achievable clock frequency slightly exceeds 40 MHz.

It is shown from Fig. 7 that the assembly of the FPGA controller prototype is relatively simple. Overall cost is also rather low, below 450 USD.

VIII. Results of Performance Tests

Performance of the FPGA controller prototype (40 MHz) has been evaluated in comparison with two conventional PLCs, i.e., GE Fanuc VersaMax and Siemens S7-1200, and with soft-PLC CP6607 from Beckhoff with IXP430 CPU (533 MHz). The tests have involved three algorithms, i.e., logic control (LC), proportional–integral–derivative (PID), and crisp output

TABLE I
Comparison of FPGA Controller With Selected Industrial PLCs Test Execution Times (T) and Speed Factors (SF)

Controller	Test algorithm					
	LC		PID		P1-TS	
	T[μs]	SF	T[μs]	SF	T[μs]	SF
GE Fanuc Versa-Max	446.8	38.8	2294	417.1	36700	312.9
Siemens S7-1200	339.8	29.5	122.9	22.3	2383	20.3
Beckhoff CP6607	1.0	0.08	9.4	1.7	139.2	1.2
FPGA 1 CPU	11.5	1	5.5	1	117.3	1

calculation of a specific class of Takagi–Sugeno–Kang fuzzy rule-based system, called P1-TS [31]. The first algorithm executes simple sequential control of an industrial process and can be described by a state machine diagram with timers. One-hot encoding scheme determines Boolean equations for the machine diagram implementation. The second algorithm is a discrete transfer function realization of PID function block. The time-domain difference equations were used. The third P1-TS algorithm involves four input variables, requires remarkable computational power, and may be applied for control of mobile robots. As shown in [31], P1-TS can be viewed as a special case of the Kolmogorov–Gabor polynomial; thus, the test involves calculations of polynomial values (more details can be found in [32]). Both PID and P1-TS algorithms use FP arithmetic.

Results of the tests are shown in Table I. For each of the three algorithms, calculation times and speed factors are given. The speed factors indicate how many times the FPGA controller with single CPU (speed factor 1) is faster than the others. As seen, the FPGA controller is much faster than GE Fanuc (38, 417, and 312 times for LC, PID, and P1-TS, respectively) and Siemens S7-1200 (29, 22, and 20 times). In the case of the LC algorithm, the FPGA controller is significantly slower than Beckhoff soft-PLC (11.5 times–0.08 speed factor), however it is slightly faster for the other two algorithms. Taking into account that the soft-PLC is clocked 13 times faster than the FPGA controller (533 MHz versus 40 MHz), the latter still shows reasonable performance, particularly in the case of FP computations. It is also worth noting that the FPGA controller and Siemens PLC execute the PID program faster than the logic control algorithm. It is the opposite for the other two PLCs; thus, FP computations seem to be weak side of the Beckhoff and Fanuc PLCs.

Computing time can still be reduced by using more CPU cores. Two scenarios are possible: 1) parallelizing computations within specific algorithm; or 2) running multiple control tasks concurrently. The first case has been implemented in the P1-TS algorithm by using two, four, and eight CPUs to calculate specific parts of the Kolmogorov–Gabor polynomial in parallel, as well as in the PID algorithm for two CPUs only. The results are presented in Table II. As seen, reasonable time reduction is obtained for the P1-TS fuzzy algorithm. However, the reduction is not a linear function of the number of CPUs, since sequential part of P1-TS algorithm, cannot be parallelized. In general, time reduction can only be achieved for

TABLE II
COMPARISON OF MULTIPLE CPUS IN FPGA CONTROLLER TEST EXECUTION TIMES (T) AND TIME REDUCTION FACTORS (TRFS)

FPGA Controller	Test algorithm			
	PID		P1-TS	
	T[µs]	TRF[%]	T[µs]	TRF[%]
1 CPU	5.5	0	117.3	0
2 CPUs	4.8	12.7	63.0	46.3
4 CPUs	-	-	38.7	67.0
8 CPUs	-	-	28.7	75.5

TABLE III
FPGA CONTROLLER WITHOUT OR WITH HFBS TEST EXECUTION TIMES (T) AND SPEED FACTORS (SFS)

FPGA Controller	Test algorithm					
	LC		PID		P1-TS	
	T[µs]	SF	T[µs]	SF	T[µs]	SF
1 CPU	11.5	1	5.5	1	117.3	1
1 CPU + HFB	0.0042	2738	0.44	12.5	10.3	11.4

more demanding computations, when calculation time of single CPU is reasonably longer than time needed for synchronization and data transfer between CPUs. This is not the case for very simple LC algorithm. In addition, for not complicated PID algorithm, two CPUs provide only modest reduction.

The use of multiple CPU cores may be particularly useful when two or more tasks are executed concurrently. Consider, for example, the case when control program implements two algorithms from Table I, i.e., logic LC and fuzzy P1-TS. Assume in addition that response time for the LC must be less than 50 μs. If the program is run as a single task, the total computation time is the sum of the times of corresponding algorithms, i.e., $11.5 + 117.3 = 128.8$ μs here; thus, it does not satisfy the 50 μs requirement. However, the use of two CPUs, each one running with its own cycle, gives 11.5 μs cycle for the LC (and 117.3 μs for P1-TS); thus, the requirement is satisfied.

A similar concept can shorten the execution of a program involving a few PID function blocks for control loops. In case of four PIDs, the single CPU controller would calculate the outputs in $4 \times 5.5 = 22$ μs (see Table I). By using four CPUs, each PID can be executed every 5.5 μs.

If cycle time provided by multiple CPUs is still not short enough, HFBs become a solution. Table III shows test times and speed factors when HFBs are used. The speed factor indicates how many times the controller with HFB is faster than without it. The HFB for the LC algorithm calculates the result in a single clock cycle and can be clocked at 240 MHz (for Spartan 6; recall that HFBs can be clocked with higher frequencies than CPU). Now the calculation time (0.0042 μs) is 2738 times shorter than for the controller without HFB. It is also 238 times shorter than for the Beckhoff PLC (1.0 μs—see Table I). The PID HFB consists of four FP multipliers and three adders. This requires 36 clock cycles, since only a few arithmetic can be parallelized. Thus, the HFB implementation of the PID algorithm (clocked at 82.6 MHz) is only 12 times faster than its software counterpart. Structure of the P1-TS HFB is very similar to the one presented in [32] (the recursive algorithm) and requires 932 clock cycles (10.3 μs for 90 MHz). As may be expected spectacular acceleration is provided by direct hardware implementation of the logic control algorithm, where FPGA parallel logic structures can be fully exploited. The acceleration is not as high for arithmetic computations (PID and P1-TS algorithms). It can also be observed that calculation times of P1-TS algorithm by eight CPUs and by the HFB are relatively close (28.7 μs versus 10.3 μs). This results from recursive, hence serial, hardware implementation of the algorithm.

The FPGA controller prototype presented here can be also compared with FPGA-based FM 352-5 Boolean processor from Siemens (see Section I). According to the specification [14], execution time of FM is 1 μs and does not depend on the number of LD/FBD components. The FPGA controller with software implementation of the LC algorithm is 11.5 times slower (fifth column in Table I) than FM. However, when LC is implemented as HFB (second column in Table III), it becomes $1/0.0042 = 238$ times faster than FM. Nevertheless, programming in STEP-7 remains important advantage of FM 352-5, whereas the LC HFB implementation requires explicit use of FPGA synthesis tools.

Performance of the FPGA controller is proportional to clock frequency. If it were implemented in a faster FPGA chip (Spartan-6 does not represent the latest FPGA technology), the speed factors given before would increase accordingly. Hence, the presented comparisons are rather conservative. It is also worth noting that the FPGA controller does not have cycle jitter, whereas in the conventional PLCs the cycle fluctuates within some range near prescribed value. The execution cycle of FPGA controller really remains constant, what can be important in real-time applications.

Since the increase of global memory throughput has been one of the design goals; thus, the architecture previously presented is the result of gradual improvements of the preliminary version. Assessment of these improvements is briefly described in the following to provide some insight into development process. The improvements have been the following: A—CPU global memory communication instructions optimized and adopted for DARs and 32-bit data bus, B—arbitration algorithm altered, C—32-bit global memory data bus introduced, D—DARs applied, E—WAIT instruction added. For the assessment, a kind of P1-TS algorithm, distributed over eight CPUs and restructured for intensive data transfer between CPUs has been used. Table IV gives execution times of 4000 iterations of test

TABLE IV
TEST EXECUTION TIMES (T) AND TRFS FOR FIVE HARDWARE IMPROVEMENTS

Improvement	Influence type			
	Separate		Cumulative	
	T[ms]	TRF[%]	T[ms]	TRF[%]
REF	332.778	-	332.778	0
A	258.561	0	258.561	22.3
B	208.649	19.1	208.649	37.3
C	211.530	18.1	160.136	51.9
D	182.543	29.4	134.922	59.4
E	195.446	24.4	90.670	72.7

program for the five improvements A to E, implemented either separately or combined cumulatively with the others. TRFs are referenced either to A for separate influence type (without A other improvements would not be possible), or to REF for cumulative influence type. REF indicates the preliminary version.

As seen, the highest reduction of the time is provided by D (29.4%, DARs), followed by E (24.4%, WAIT), B (19.1%, arbitration), and C (18.1%, 32-bit global memory bus). The overall cumulative reduction factor of all improvements is 72.7%, what means that final version of the controller is 3.7 times faster than the preliminary.

IX. Conclusion and Perspectives

It has been shown that FPGAs can be a platform to build a low-cost, single-chip, and multiprocessor PLC that provides high execution speed, multiprocessing and IEC 61131-3 programming, i.e., similar features as larger industrial PLCs. The speed can be increased either by distribution of control program over multiple CPUs or by implementing it as a HFB. Unlike existing FPGA-based control platforms, e.g., Siemens FM 352-5 and NI CompactRIO, the presented controller prototype can be programmed in IEC languages; thus, for standard applications it does not require FPGA synthesis tools. However, to obtain the highest execution speed, the HFBs are needed. One can describe them directly in HDL or even generate from a C/C++ specification using Xilinx HLS (high-level synthesis) technology. Nonetheless, development of a library with ready-to-use HFBs involving typical control and signal processing algorithms would be of much help.

The proposed architecture is fairy flexible, applicable in small FPGA-based PLCs (e.g., LOGO-size), as well as in larger controllers. Due to FPGA reconfiguration capability, it is also relatively easy to customize (number of CPUs and HFBs) and to upgrade (e.g., remotely or by SD card). The HDL description may be optimized for a target FPGA by exploiting features of specific embedded hardware. Another approach may involve IP core domain. The IP core of the multiprocessor controller can be used for implementation and integration in custom-made automation systems. Performance of the controller and precision timing seem adequate for such application areas as real-time industrial control, transportation, and robotics.

The tests have also revealed some weak points of the prototype. Current CPU instructions list, originally developed for software VM, has not been optimized for high performance of FPGA implementation. Hence, a new RISC CPU is being developed, with data flow type changed from three to one address and different instruction set supporting fast calculation of Boolean sum of products. Preliminary design considerations and simulation results point out that the new CPU should be 6–7 times faster for logic control algorithms than the current version. It is hoped that the new instructions set will also reduce requirements for logic resources (currently relatively high) and increase maximum clock frequency. Although communication between CPUs has been already significantly improved, another topology of CPU connections, e.g., mesh (a grid), may be also considered instead of the global memory communication scheme. Application of soft-hard architecture as in Xilinx Zynq [33] seems another interesting research area. Additional acceleration may be provided by NEON coprocessor and acceleration coherency port for data exchange between the processor and FPGA part.

Acknowledgment

The authors would like to thank the Editor-in-Chief, Prof. C. Cecati, and the anonymous Reviewers whose valuable comments have enhanced the quality of this paper.

References

[1] International Electrotechnical Commission, 2013-02, IEC 61131-3, Int. Standard Edition 3.0.
[2] E. Monmasson and M. Cirstea, “FPGA design methodology for industrial control systems. A review,” *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.
[3] E. Monmasson *et al.*, “FPGAs in industrial control applications,” *IEEE Trans. Ind. Informat.*, vol. 7, no. 2, pp. 224–243, May 2011.
[4] M. Curkovic, K. Jezernik, and R. Horvat, “FPGA-based predictive sliding mode controller of a three-phase inverter,” *IEEE Trans. Ind. Electron.*, vol. 60, no. 2, pp. 637–644, Feb. 2013.
[5] A. Gabiger-Rose, M. Kube, R. Weigel, and R. Rose, “An FPGA-based fully synchronized design of a bilateral filter for real-time image denoising,” *IEEE Trans. Ind. Electron.*, vol. 6, no. 8, pp. 4093–4104, Aug. 2014.
[6] C. Economakos and G. Economakos, “Optimized FPGA implementations of demanding PLC programs based on hardware high-level synthesis,” in *Proc. IEEE Int. Conf. Emerging Technol. Factory Autom.*, 2008, pp. 1002–1009.
[7] S. Ichikawa, M. Akinaka, H. Hata, R. Ikeda, and H. Yamamoto, “An FPGA implementation of hard-wired sequence control system based on PLC software,” *IEEE Trans. Elect. Electron. Eng.*, vol. 6, no. 4, pp. 367–375, 2011.
[8] D. Du, Y. Liu, X. Guo, K. Yamazaki, and M. Fujishima, “Study on LD-VHDL conversion for FPGA-based PLC implementation,” *Int. J. Adv. Manuf. Technol.*, vol. 40, no. 11/12, pp. 1181–1190, Feb. 2009.
[9] S. Subbaraman, M. M. Patil, and P. S. Nilkund, “Novel integrated development environment for implementing PLC on FPGA by converting ladder diagram to synthesizable VHDL code,” in *Proc. IEEE Int. Conf. Control Autom. Robot. Vis.*, 2010, pp. 1791–1795.
[10] C. F. Silva, C. Quintáns, A. Colmenar, M. A. Castro, and E. Mandado, “A method based on Petri nets and a matrix model to implement reconfigurable logic controllers,” *IEEE Trans. Ind. Electron.*, vol. 57, no. 10, pp. 3544–3556, Oct. 2010.
[11] D. Gawali and V. K. Sharma, “FPGA based micro-PLC design approach,” in *Proc. IEEE Int. Conf. ACT*, 2009, pp. 660–663.
[12] X. Mei-hua, R. Feng, C. Zhang-jin, K. Shu-feng, and L. Run-guang, “IP core design of PLC microprocessor with Boolean module,” in *Proc. IEEE Conf. High Density Microsyst. Des. Packag. Compon. Failure Anal.*, 2005, pp. 1–5.
[13] M. Chmiel, J. Mocha, and E. Hrynkiewicz, “A FPGA-based bit-word PLC CPUs development platform,” in *Proc. 10th IFAC Workshop Programm. Devices Embedded Syst.*, 2010, vol. 10, pp. 132–137.
[14] Siemens AG Operating Manual, SIMATIC S7-300 FM 352-5 high-speed Boolean processor, Jul. 2011.
[15] R. Glorioso, “Combining PLC and FPGA architectures,” in *Proc. EngineerIT*, Aug. 2006, pp. 64–67.
[16] T. Dorta, J. Jimenez, J. L. Martin, U. Bidarte, and A. Astarloa, “Overview of FPGA-based multiprocessor systems,” in *Proc. IEEE Int. Conf. Reconfigurable Comput. FPGAs*, 2009, pp. 273–278.
[17] S. Santner, W. Peck, J. Agron, and D. Andrews, “Symmetric multiprocessor design for hybrid CPU/FPGA SoCs,” in *Proc. Reconfigurable Comput., Architectures, Tools Appl.*, vol. 4943, *Notes in Computer Science*, 2008, pp. 99–110.
[18] R. Airoldi, T. Ahonen, F. Garzia, D. Milojevic, and J. Nurmi, “Implementation of W-CDMA cell search on a highly parallel and scalable MPSoC,” *J. Signal Process. Syst.*, vol. 64, no. 1, pp. 137–148, Jul. 2011.
[19] D. Stevens *et al.*, “BioThreads: A novel VLIW-based chip multiprocessor for accelerating biomedical image processing applications,” *IEEE Trans. Biomed. Circuits Syst.*, vol. 6, no. 3, pp. 257–268, Jun. 2012.

[20] Z. Hajduk, J. Sadolewski, and B. Trybus (2011, Sep.). FPGA-based execution platform for IEC 61131-3 control software. *Elect. Rev.* [Online]. *2011(8)*, pp. 187–191. Available: http://pe.org.pl/abstract_pl.php?nid=5057

[21] Z. Hajduk, J. Sadolewski, and B. Trybus, "Multiple tasks in FPGA-based programmable controller," *e-Informatica*, vol. 5, no. 1, pp. 77–85, 2011.

[22] D. Rzońca, J. Sadolewski, and B. Trybus, "Prototype environment for controller programming in the IEC 61131-3 ST language," *Comput. Sci. Inf. Syst.*, vol. 4, no. 2, pp. 133–148, Dec. 2007.

[23] M. Jamro, D. Rzońca, and B. Trybus, "Communication performance tests in distributed control systems," in *Computer Networks 2013 Communications in Computer and Information Science 370*, A. Kwiecien, P. Gaj, and P. Stera, Eds. Berlin, Germany: Springer-Verlag, 2013.

[24] Praxis Automation Technology B.V., Mini-Guard Ship Control & Positioning System, 2010. [Online]. Available: http://www.praxisautomation.com

[25] B. Trybus, "Development and implementation of IEC 61131-3 virtual machine," *Theor. Appl. Informat.*, vol. 23, no. 1, pp. 21–35, 2011.

[26] Z. Hajduk, "An FPGA embedded microcontroller," *Microprocess. Microsyst.*, vol. 38, no. 1, pp. 1–8, Feb. 2014.

[27] A. J. Thakkar and A. Ejnioui, "Design and implementation of double precision floating point division and square root on FPGAs," in *Proc. IEEE Aerosp. Conf.*, Mar. 2006, pp. 1–7.

[28] J. Detrey and F. de Dinechin, "Parameterized floating-point logarithm and exponential functions for FPGAs," *Microprocess. Microsyst.*, vol. 31, no. 8, pp. 537–545, Dec. 2007.

[29] P. Echeverría and M. López-Vallejo, "Customizing floating-point units for FPGAs: Area-performance-standard trade-offs," *Microprocess. Microsyst.*, vol. 35, no. 6, pp. 535–546, Aug. 2011.

[30] IBM, 128-Bit Processor Local Bus Architecture Specifications Version 4.7, Hopewell Junction, NY, USA, May 2007.

[31] J. Kluska, "Analytical method in fuzzy modeling and control," in *Studies in Fuzziness and Soft Computing*, vol. 241. New York, NY, USA: Springer-Verlag, 2009.

[32] J. Kluska and Z. Hajduk, "Hardware implementation of P1-TS fuzzy rule-based systems on FPGA," in *Proc. Artif. Intell. Soft Comput.*, vol. 7894, *Notes in Computer Science*, 2013, pp. 282–293.

[33] Xilinx, Zynq-7000 All Programmable SoC Overview, San Jose, CA, USA, Dec. 2013.

Zbigniew Hajduk (M'13) received Ph.D. degree (with honors) in computer engineering from the University of Zielona Góra, Zielona Góra, Poland, in 2006.

He is an Assistant Professor with the Department of Computer and Control Engineering, Rzeszów University of Technology, Rzeszów, Poland. His main areas of interest include digital system design with FPGAs.

Bartosz Trybus (M'13) received the Ph.D. degree in computer engineering from AGH University of Science and Technology, Cracow, Poland, in 2004.

He is an Assistant Professor with the Department of Computer and Control Engineering, Rzeszów University of Technology, Rzeszów, Poland. His research involves real-time systems and runtime environments of control applications.

Jan Sadolewski received the Ph.D. degree in computer engineering from Silesian University of Technology, Gliwice, Poland, in 2012.

He is an Assistant Professor with the Department of Informatics and Control, Rzeszów University of Technology, Rzeszów, Poland. His interests are modern programming languages and compiler implementations.