REMARKS ON IMPROVING OF OPERATION SPEED OF THE PLCs

Mirosław CHMIEL, Edward HRYNKIEWICZ, Adam MILIK

Institute of Electronics, Silesian University of Technology,
Akademicka 16, 44-100 Gliwice, Poland
<Miroslaw.Chmiel, Edward.Hrynkiewicz, Adam.Milik>@ polsl.pl

Abstract: The paper presents two different approaches to optimising operation speed of Programmable Logic Controllers. First approach optimizes architecture of the CPU and the program execution. It shows the two processors bit-byte architecture which support of concurrent execution of bit and byte computation tasks. Second approach bases on hardware implemented control algorithm in reconfigurable platform based on FPGA. In second solution high performance is achieved by fully concurrent hardware execution of algorithm. Specific implementation tools enables typical PLC programming for hardware target platform.

Keywords: Programmable Logic Controller; Central Processing Unit; Bit-Byte Structure of CPU; Control Program; Scan Time; Throughput Time; Reconfigurable Logic Controller; Programmable Logic Devices; Field Programmable Gate Arrays; Parallel Processing

1. INTRODUCTION

The Programmable Logic Controllers (PLCs) introduced at the end of the sixties totally have changed the used equipment in automation control systems. Nowadays the PLCs are commonly used in civil engineering (smart buildings), industry, automotive (special vehicles) and many other areas. Due to different places of application PLC should meet a lot of different and often very difficult to fulfil requirements. One of the main requirements is speed of operation that is often expressed by so-called scan time – time of execution of one thousand of control instructions. Due to this fact it is important to have such CPU of a PLC which features with a structure enabling of fast control program execution. Many manufacturers deliver their PLCs with CPU based on microprocessor (Michel, 1990). Also very often manufactured CPUs are based on so-called bit-byte structure (Getko, 1983; Chmiel and Hrynkiewicz, 1999; 2001). In bit-byte structure of a CPU tasks operating on discrete input/output are executed by bit-processor. Such processor may be implemented in programmable structures as PLD or FPGA devices (Chmiel *et. al*., 1995; Hrynkiewicz, 1997). The byte processor – built on base of standard microprocessor – is used for control of analogue objects, numeric data processing and for execution of the instructions indirectly connected to a user program but related to the operating system tasks, as for example testing of PLC hardware, timers servicing, LAN servicing, communication to the supervision and monitoring system (SCADA) and so on. If the CPU is based on byte microprocessor only, this device executes all CPU functions. This is not convenient for carrying out instructions on discrete inputs, as it is necessary to sense the particular bits from entire word, what is inefficient and time consuming. For this reason, there are manufacturers, whose provide so-called logic solver which can be used for execution of logic instructions if CPU is based on byte processor only. One should realise that in this way a kind of bit-byte structure of CPU is created. It should be added that from the methodological point of view bit-byte

structure of CPU seems to be very natural. Bit processor executes logic control while byte processor deals with numerical control (for example it executes digital PID algorithm). Because such CPU is also cost effective – the structure is relatively simple – it is interesting to point the possibility of improving of an operating speed of a PLC with such CPU architecture.

The main drawback of traditional PLCs is the fact that the user (control) program is executed in serial-cyclic mode. It means that the inputs/outputs are processed every time interval, which is equal to a time of control program swap. If the control object is fast and user program is large it happens that some changes of inputs are not observed. An improving of the performance in this area may be done by some programming tricks and tips as segmentation of a user program, many times calling of the same segments, utilisation of released (controlled) segments, utilisation of interrupt system and so on (Modicon, 1990). The radical solution of this problem can be achieved by introducing parallel processing of a user program. However parallel processing require completely new PLC hardware platform. Tens years ago control circuits were designed as hard-wired systems. Logic controller offers new quality of design of control units by system integration and by writing of a program instead of assembling an electric or electronic circuit. Nowadays this new platform is Field Programmable Gate Array (FPGA). Hardware functionality of FPGA can be modified in programmed way. Similar to loading of a new program to the controller loading a new configuration to the FPGA allows changing its functionality.

The presented paper deals with these two problems – the improving of operation speed of a bit-byte CPU of a PLC and utilisation of FPGA for PLC design and construction.

2. Structure Of Bit-Byte Central Processing Unit

As it was mentioned above in the simplest case each programmable control system might be realised as microprocessor device. But we have to remember about applications in which we are going to use a logic controller. Those applications force special requirements and constraints. To control a real object it is necessary to process a great number binary inputs and calculates binary outputs while standard microprocessor (or microcontroller) operate mainly on bytes. Instruction list of these devices is optimised for operation on bytes or words (some of them can carry out complicated arithmetical calculation) variables that are not required in this case. Logic instructions like AND or OR on individual bits take the same amount of time as the instructions on a bytes as it is necessary to extract suitable bits from the word(s). When we take under consideration the binary inputs and outputs it is necessary to realise

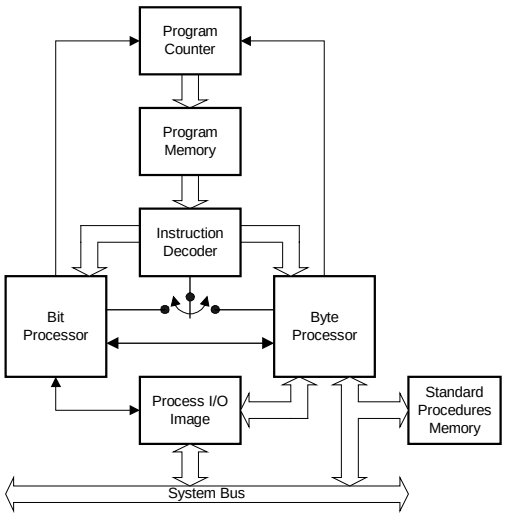

Fig.1. Block diagram of one bus CPU [Getko, 1983]

that they reach number of thousands. In such cases parallel computation of all inputs and outputs is impossible. Therefore the inputs and outputs must be scanned and updated sequentially. If we would like to achieve good control parameters the instructions on bits should be done very quickly. Creation of specialized bit-processor, which can carry out bit instruction very fast, is fully reasonable. If there is a need of computation of byte data for example from analogue to digital converters or external timers, it is required to use of additional 8,16 or even 32 bits processor or microcontroller. General structure of that device was presented in Fig.1. (Getko Z., 1983).

Presented solution consists of two processors with own instruction set. Instruction decoder recognises for which processor fetched instruction is appropriated and sends activation signal to it.

Basic parameter that was taken under consideration was program execution speed. Program execution speed is mainly limited by access latency of both processors to the internal (e.g. counters, timers) and external (e.g. inputs and outputs) process variables. The program memory and the instruction fetching circuitry also influence system performance. In order to support cooperation of both processors without conflicts and maintain their concurrent operation following assumptions were made:

- separate address buses for bit and byte processors,
- separate data buses: 1 bit wide for bit processor and 8 bit wide for microcontroller,
- separate control buses with:
  - read and write signals for byte processor,
  - read and write signals for bit processor,
  - refresh signal for latching states of all inputs and outputs at once,
  - error signal for immediate switching off of all external modules of controller.

Additionally in basic solution the following problems were taken for research and design works. These problems are structure of a memory (memories), instruction fetching by both processors, information

exchange rules between processors and access to common resources (timers, counters and flags).

The CPU structure, which meets the above requirements, is shown in Fig.2. The solution assumes main program memory for both processors (in Fig.2 this memory is marked as bit processor program memory). Each of them has unique instruction codes. Bit processor fetches instructions code and recognise it. If fetched instruction is assigned to this processor, it is immediately executed. In other cases it is send to the second processor.

The unit is equipped with 3 memory banks for control program:

- bit processor program memory,
- byte processor program memory,
- byte processor standard procedures memory.

Such CPU has three states of operation:

- both processors execute control program,
- one processor operates,
- bit processor executes control program while byte processor actualises the timers for example.

Bit processor delivers instructions to the byte processor through the instruction buffer informing about it by means of NEXT signal. On the other hand byte processor after accepting of an instruction sends to the bit processor GO signal.

The presented solution is additionally equipped with the system of fast internal condition flip-flop state exchange. This system – in simply words – causes that the processors do not wait for finishing their instructions but they execute next instructions up to the moment when instruction of waiting for result of instruction carried-out by the second processor will occur. The processors can exchange the state of their internal condition flip-flops (F_b, F_B) via buffering condition flip-flops F_{bB} and F_{Bb}. This justifies of using special instructions that are marked in Fig.2. From a side of bit processor there are two instructions: TRF_{bB} that allows writing state of F_b to F_{bB} and TF_{Bb} that allows testing state of F_{Bb} flip-flop. Similar set of two instructions is implemented for the byte processor. There are RDF_{bB} that reads contents of F_{bB} and WRF_{Bb} that transfers content of internal condition flip-flop to F_{Bb}.

This system allows parallel work of both processors with state of condition flip-flops exchange, but there is two following situations which cause that the one processor has to wait for the other:

- one processor has not yet executed instruction expected by the second processor and this one have to wait for the result – new state of buffering flip-flop is not ready ($READYF_{bB}$=0 or $READYF_{Bb}$=0),
- second processor has not yet received the previous state of buffering flip-flop and the first one can not write the next state ($EMPTYF_{bB}$=0 or $EMPTYF_{Bb}$=0).

To exclude waiting states the programs has to be written and compiled in such a way to get these two processor working possibly parallel. However in the second case one can take into account the solution basing on increased number of the accessible

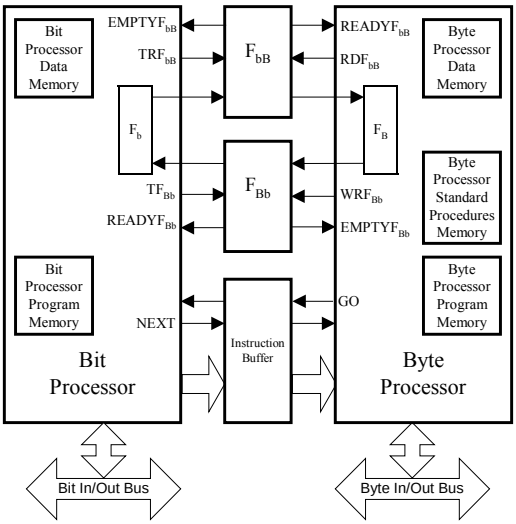

Fig.2. Block diagram of parallel CPU structure

condition flip-flops exchange or on assignment of common data memory area for the condition data exchange purpose. At that time it appears the need for assignment of the condition flip-flop to every task. One can try to solve the condition flip-flop problem in the following ways:

- the fixed marker can be assigned to every type of instruction or to each successive task that includes necessity of condition state exchange. The markers assignment process can be made by the compiler,
- the second way is to charge the programmer with the duty of marker assignment. Marker contains the instruction execution result and on the other hand marker state influences on a way of instruction execution. In similar way the Modicon PLCs are programmed [Modicon, 1990] – output state of operational blocks can be assigned to the marker by programmer himself.

First solution presented above can be implemented in hardware. Number of markers must be large enough to transfer all possible condition in the longest program. In general number of used markers is proportional to the length of program that is limited by capacity of memory.

In one possible solution condition flip-flops are grouped in two sets that passing information in both directions between two processors. The simplest implementation writes results of instruction to the queue. The opposite processor reads results from the queue as needed. Such a system may be implemented as a 1-bit wide FIFO register that allows storing all markers in order of their appearing. Processor writes condition state to the flip-flop pointed by condition counter. Opposite processor that reads condition state selects current flip-flop by its condition counter. In this way circular buffer is created that allows for reading and writing from different flip-flops.

This solution offers extremely fast operation requiring only one clock cycle, for condition state exchange, from side of each processor. Unfortunately this solution is expensive in comparison to common memory that also can be used as memory for timers, counters and flags. The memory accessed by two

processors requires special construction or arbitration system. In order to avoid arbitration process in access cycle special two gate memory must be used that allow simultaneous access to memory cells by two processors without conflicts in concurrent read and write operation.

The proposed CPU can work in one of two modes:

- the parallel–serial work of the processors with transferring instruction from bit processor to byte processor through instruction buffer,
- the fully parallel work of the processors with two programs located in separate program memory banks – bit processor program memory and byte processor program memory. Processors have to wait for each other only when buffering condition flip-flops are not ready or not empty.

3. RECONFIGURABLE LOGIC CONTROLLER

Logic controller designed with use of FPGA circuits was called reconfigurable. This term describes a method of program execution and implementation (Hrynkiewicz and Milik 2000a; 2000b; 2001). In Programmable Logic Controller program is represented as a sequence of instruction executed on given hardware platform (CPU + memory + IO) (Michel, 1992). In Reconfigurable Controller program is represented as a hardware structure that is written into programmable resources of the FPGA device (Brown, *et al.*, 1992). In this solution is obtained custom digital circuit dedicated to perform given control function.

Before the FPGA device becomes a Reconfigurable Logic Controller several problems and limitations must be solved. Logic Controller for automatic control designers consists of two basic components, which are inseparably connected. Those components are hardware platform and programming tools. Easy of control program modification and execution make logic controllers so powerful and popular. Reconfigurable Logic controller bases on the same components: hardware architecture and programming environment (Kumar, *et al.*, 1992).

3.1 Compact Reconfigurable Logic Controller Architecture

General structure of the reconfigurable compact logic controller is shown in Fig.3. This controller is the simplest solution and offer shortest throughput time. Object signals are connected and driven by controller through conditioning modules for analogue and digital signals. Modular I/O system allows for flexible selection of required modules. Each module is directly connected with reconfigurable processing system. This allows for parallel reading and writing of all signals by controller.

All calculations are executed by reconfigurable processing system. Internal structure of that

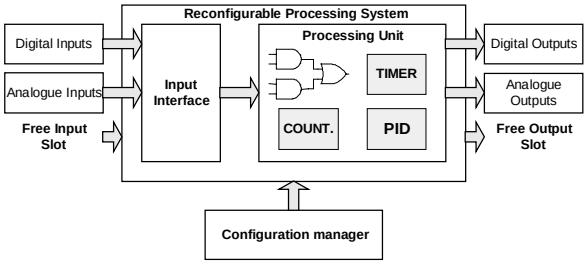

Fig.3. Compact reconfigurable logic controller block diagram

component is shown in Fig.4. The processing system consists of two units. The Input interface unit is responsible for synchronising external signals to the internal system clock. The input interface unit is passing properly timed signals to the processing unit. The processing unit consists of the user function and the output register. In general user function can be any combinatorial or sequential function (FSM). The output register allows for avoiding unpredicted states of the controller output caused by propagation process. This general description of logic controller allows to calculating throughput time. In case of reconfigurable logic controller scan time parameter is not applicable while all processing is executed in parallel in opposite to standard solutions derived from microprogrammable technique.

In general case maximal throughput time of the reconfigurable processing system is given by (1).

$$t_{THP\max} = 2T_{SYS\_CLK} + t_{pID} + t_{pQO} \qquad (1)$$

This equation takes into account propagation delay in input and output path. This propagation delay for moderate operating frequencies can be omitted. Above assumptions allow to calculating roughly throughput time given by (2).

$$t_{THP\max} \cong 2T_{SYS\_CLK} \qquad (2)$$

Other important fact in controller performance is ability of distinguishing changes and pulses applied to input of controller. As input state is stored in interface registers minimal pulse width that can be observed is given by (3).

$$t_{PULSE\min} = T_{SYS\_CLK} \qquad (3)$$

Where:

T_{SYS_CLK} system clock period

t_{THPmax} controller maximal throughput time

$t_{PULSEmin}$ minimal duration time of pulse that can be observed by the controller

t_{pID} propagation time from the input to the D input of the input flip-flop

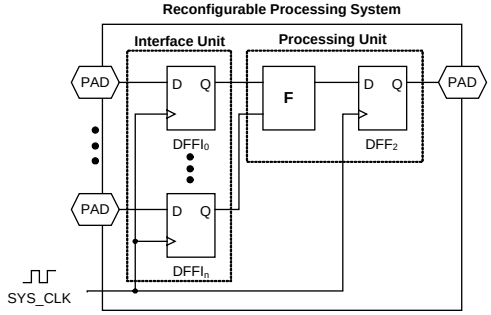

Fig.4. Reconfigurable processing system block diagram

t_{pQO} propagation time from the output of Q of the output flip-flop to the controller output

When f_{SYS_CLK} = 20MHz maximal throughput time of the controller is no longer then 100ns. The controller is able to recognise pulses that duration time is no shorter then 50ns.

3.2 Distributed Reconfigurable Controller Architecture

Compact architecture is limited to single controller and is unable to communicate with other controllers. This limit application area of controller to service a single processes and also reduces number of signals that can be connected to it. Adding to the controller network connection module allows for designing distributed architecture controller that can consists of several processing units distributed over area of controlled system or production process. The distributed architecture allows for reducing number and length of wires connecting controller with sensors and actuators. Not only length of the wire can be reduced in presented solution but also higher safety of operation can be achieved. In opposite to remote I/O in standard controllers in this architecture distributed intelligence is available.

Architecture of controller is presented in Fig.5. This architecture is derived from compact solution. Each controller is able to process delivered information. The controller treats local and remote data in the same way. Network interface operates separately from reconfigurable processing unit. In order to allow for continues access to remote signals process image register is designed in a controller. Process image register also delivers local variables to network interface. Size of process image register is variable and is fitted to design requirements during synthesis process. In each controller is allocated required set of process image registers that stores significant for controller data. This allows for reducing number of logic resources allocated in FPGA device.

Distributed controllers allow for increased safety of operation by implementing special algorithms that are executed in case of disturbances. This allows to maintain safe operation and safe process shut down.

In order to enable communication with other controllers network interface was added. Network communication employs reliable protocol that bases on token passing. Selection of this protocol allows for reliable and saves operation of the whole controller system. One of the controllers is

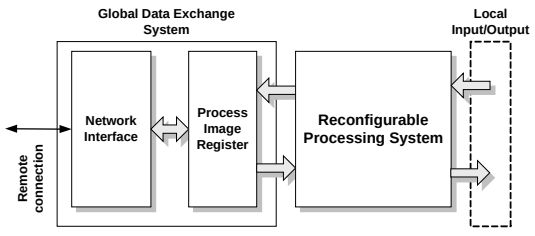

Fig.5. Distributed controller block diagram

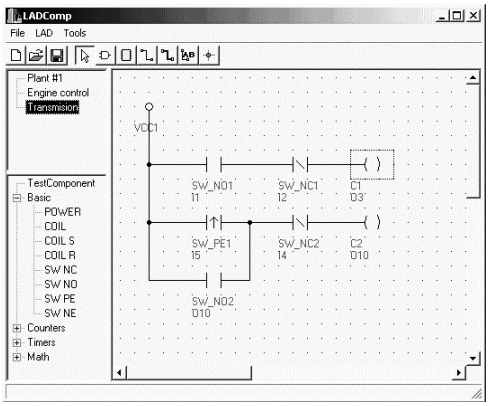

Fig.6. Ladder diagram editor and high level logic synthesis tool main window

established master. Master controller is responsible of performing start up procedure, shut down and malfunctions processing.

Number of data passed through the network is reduced due to use of distributed intelligence. It is very important to reduce number of data that is exchanged while it allows to reduce throughput time. Distributed processing units deliver to network interface partially processed data that allow for transfer only required part of information. Variable selection and network interface configuration is done automatically during synthesis of a controller (discussed in next paragraph).

4. RECONFIGURABLE CONTROLLER PROGRAMMING

Programmability and programming of the logic controller is key issue for designers and users. The reconfigurable logic controller must use the same programming methods like its ancestors and also maintain all data processing benefits. Logic controller programming is specified by IEC-1131-3 standard (IEC,1992).

Reconfigurable Logic Controller is programmed with use of ladder diagrams (LD). This graphical representation is widely used by automatic control designers. In Fig.6 is shown main window of the LD editor and high-level synthesis tool. Automatic synthesis process support user in designing its control algorithm. User can easily place optimised special function blocks like timers, counters or PID controllers. Those blocks were carefully designed and optimised in performance and resource allocation inside FPGA device.

4.1 Controller synthesis

Controller synthesis is similar for compilation process for typical program. Control program graphically represented by set of components and connections between them is translated into hardware structure. Synthesis start form design analysis. In this step graphical representation of design is converted into netlist form that can be processed. During netlist generation basic design rule check (DRC) is executed. DRC allows determining errors in diagram

drawing or improper use of components. During this check sourceless, loadless or floating nodes in circuit are determined also component parameters are checked (e.g. timer values, counter ranges etc.).
Synthesis can start after netlist generation. During synthesis process schematic components are replaced with equivalent electronics components. When direct translation is completed optimisation process is started. During optimisation process combinatorial function are merged and logical constant are removed. This operation allow for design checking against logic trimming due to constant propagation or complemented variable usage or logic optimisation. Appropriate warnings are printed out for user. When optimisation step is completed successfully interface unit is generated. Interface unit synthesis uses only inputs and outputs that remain active after optimisation process. Finally after all processes are completed result is written out in a form of gate level HDL description. In next step FPGA vendor tools can implement the controller description. Finally configuration stream is obtained that represents physical layout of control algorithm.

5. CONCLUSION

As it was shown the studies on an information exchange between the processors of the bit-byte CPU of a PLC leads to the CPU hardware solution which significantly increases a program execution speed. The most interesting result is the possibility of fully parallel work of both processors without waiting one for the other. Such mode of CPU operation become possible thanks to realising that for considered processor the other processor can be treated in the same way as a controlled object.
The authors have evaluated the operation speed of controller with all mechanisms presented in section 2 of the paper. In comparison to the standard PLCs like Siemens S7-224 execution time for the program containing about 1000 instructions was about 2.8ms for S7-224 and about 1.7ms for serial-parallel mode and 1.1ms for parallel mode.
The most profitable idea of improving PLC operation speed is based on utilisation of FPGA. For reconfigurable FPGA the PLC built on this platform is reconfigurable too and thanks to this fact it may be used in universal applications. Because the control program (for example in ladder diagram representation) is translated into hardware implementation in FPGA a design workbench should be equipped with special software as for example XILINX Foundation. As it was said for compact PLC (PLC based on one FPGA module only) the throughput time is about 100ns at clock frequency 20MHz which is not very high for nowadays FPGA. For comparison it is important to realise that Simatic S5 PLCs executes one instruction only in 1.2μs and Simatic S7 PLCs executes the same in 100-300ns. A throughput time for these controllers is created usually during execution of a lot of instruction.
It should be mentioned about other approach to improving of PLC operation parameters not described in this paper. It seems to the authors that probably good results can be obtained when PLC is considered as on event driven. It means that the system executes particular tasks in response to request on event i.e. particular element change of state. Main difference is located in events scanning and triggering technique. This approach requires design of a new controller platform that allows for task triggering. Such works are in progress.

REFERENCES

Brown S.D., R.J. Francis, J. Rose, Z.G. Vranesic, "Field-Programmable Gate Arrays", Kluwer Academic Publisher, 1992

IEC, International Electronics Commission, „International Standard IEC 1131, Programmable Controllers", Geneva, 1992

Kumar S., J.H. Aylor, B.W. Johnson, and W.A. Wulf, "The Codesign of Embedded Systems", Kluwer Academic Publisher, 1992

Michel G. "Programmable Logic Controllers – Architecture and Applications", John Willey & Sons, 1992

Modicon 984 Programmable Controller – System Manual, 1990

Chmiel M., E. Hrynkiewicz, „Parallel Bit-Byte CPU Structures of Programmable Logic Controllers", International Workshop ECMS, Liberec, Czech Republic, 1999

Chmiel M., E. Hrynkiewicz, „Remarks on Parallel Bit-Byte CPU Structures of Programmable Logic Controllers" International Workshop on Discrete Event System Design, DESDes, Przytok near Zielona Góra, Poland, 2001

Chmiel M., L. Drewniok, and E. Hrynkiewicz, „Single board PLC based on PLDs", International Workshop on Programmable Devices and Systems, PDS, Gliwice, Poland, 1995

Getko Z., „Programmable systems of binary control", Elektronizacja, WKiŁ, Warsaw, Poland, 1983 (in polish)

Hrynkiewicz E., „Based on PLDs Programmable Logic Controller with remote I/O groups", International Workshop ECMS, Toulouse, France, 1997

Hrynkiewicz E., A. Milik, "PID Module for Reconfigurable Logic Controller", IFAC Workshop on Programmable Devices and Systems, PDS, Ostrava, 2000

Hrynkiewicz E, A. Milik, „Modular Reconfigurable Logic Controller", 4th Portuguese Conference on Automatic Control, Controlo'2000, Guimaraes, Portugal, 2000

Hrynkiewicz E., A. Milik, "Reconfigurable Logic Controller Architecture, Programming, Implementation", IFAC Workshop on Programmable Devices and Systems, PDS'2001, Gliwice, 2001