BULLETIN OF THE POLISH ACADEMY OF SCIENCES
TECHNICAL SCIENCES
Vol. 56, No. 3, 2008

On reducing PLC response time

M. CHMIEL*

Institute of Electronics, Silesian University of Technology, 16 Akademicka St., 44-100 Gliwice, Poland

Abstract. The dual core bit-byte CPU must be equipped with properly designed circuits, providing interface between the two processor units, and making it possible to exploit all its advantages like versatility of the byte unit and speed of the bit unit. First of all, the interface circuits should be designed in such a way, that they don't disturb maximally parallel operation of the units, and that the CPU as a whole works in the same manner as in a standard PLC. The paper presents hardware solutions supporting effective operation of PLC CPU-s. Possibilities of solving problems concerning data exchange between a CPU and peripheral circuits were presented, with a special stress on timers and counters, and also on data exchange between the bit unit and the byte unit. The objective of the proposed solutions is to decrease the time necessary for a CPU to access its peripheries.

Key words: Programmable Logic Controller, Bit-Byte Structure of CPU, Control Program, Scan Time, Response Time, Throughput Time, Timer Function, Counter Function, Concurrent Operation.

1. Introduction

This paper presents results of research and implementation that were aimed at maximal reduction of access time to object signals. On the other hand, improved PLC CPU should also be functional and cost effective.

Entire operation cycle of a PLC consists of the following items: network communication, CPU test, object signal update and control program execution. Operation connected with object signal update and control program execution are considered in this paper. There are two typical architectures of a CPU known from literature that can be used. Those are: typical or slightly modified microcontroller (e.g. dedicated processors designed as ASICs) and dual processor bit-byte architecture with separated processing of instructions operating on binary and word variables. There also exist very expensive multiprocessor solutions [1, 2].

Dual processor architecture is optimized for faster execution of logic instructions. This feature allows obtaining not only cost effective but also functional and fast PLC CPU. Specific hardware and software solutions are required to obtain high performance dual bit-byte processor CPU. This specific mixed hardware software solution allows the features of both processors to be exploited to the maximum. First of all both processors should operate independently from each other. Independent operation allows maximizing throughput of both processors particularly taking advantage of fast operation of the bit processor. Quick response is especially required in the case of disturbances or failure in controlled process by changing the state of the appropriate executor or indicator.

Independent operation of dual core processor executing common control program requires solving a problem of common resource accessing. In a typical CPU or multi processor CPU that executes a control algorithm in a serial-cyclic fashion those problems are unknown (since there is only one processor that can access common resources at a time).

Thanks to their special features such structures are capable of achieving satisfactory performance in terms of both binary signal processing, owing to the inclusion of a dedicated bit processor, and handling of analogue signals, which is carried out by a standard, inexpensive microcontroller. Such a structure includes two separate components: a binary (bit) unit and a byte unit. Therefore, a control program has to be subdivided into two parts. What‘s more, such an approach is justified by the fact that special features of the instructions for those two processors shall be different. The bit processor executes every instruction during a single clock cycle, whereas for the byte processor every single instruction is equivalent to a procedure execution that consists of several native instructions. Such procedures are written in assembler or with use of high-level languages.

The paper deals with improving the operation speed of a bit-byte CPU in a PLC. The following improvements and modifications are proposed:

- a new architecture of CPU,
- fast inter processor information exchange,
- improved hardware-software service of timers and counters,
- improved process signal servicing.

All the improvements and modifications mentioned above lead to reduced response time and throughput time.

2. Process inputs/outputs

The way microprocessors access peripheral modules has an important influence on the operating speed of a microcontroller. One approach is to use a dedicated microprocessor

*e-mail: Miroslaw.Chmiel@polsl.pl

that is responsible for updating the state of the inputs and outputs of the microcontroller. This solution is expensive and involves complicated reading and writing of common resources. However, in the solution presented in this paper, a different approach has been applied that does not require the above-mentioned dedicated microprocessor. In addition, it should speed up the operation of the central unit of the microcontroller. Input and output units are equipped with special mechanisms that facilitate creation of process image input and output (PII, PIO) directly in the units. There are two ways the central unit could determine the rhythm of updating the process image input and output:

- after each program scan execution,
- when requested by the current refresh command.

Therefore, depending on the type of the command, one is able to read the input state that has been stored at the beginning of the current program loop or to read the current state of the input. The state of the output can be stored in the process image register and the state of the latter is then copied to the output register at the end of the current iteration of the program loop. One can also directly set a new value at the given output, which immediately updates the state of the process image output register. The units are also equipped with edge detection mechanisms. There is no need to perform a software edge detection by PLC CPU that reduces overall execution time. Block diagrams of binary I/O units are presented in Fig. 1. For the sake of legibility, the edge detection circuit is not shown in the schematic of the output unit.

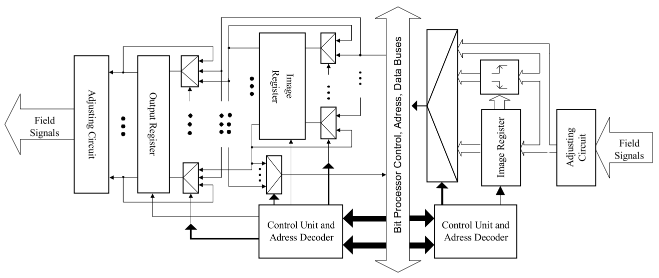
Field Signals
Adjusting Circuit
Output Register
Image Register
Bit Processor Control, Adress, Data Buses
Image Register
Adjusting Circuit
Field Signals
Control Unit and Adress Decoder
Control Unit and Adress Decoder

Fig. 1. Binary inputs and outputs module schematic

3. Timers and counters

Byte processor maintains timers and counters while bit processor mainly uses the results of their operation. They are implemented as the software modules for microprocessor but in most cases are utilized by bit processor. The operation of timers and counters may be organized using condition flip-flop exchange described in Section 4, but timer and counter operations are very frequently used in PLC, therefore special instructions for timers and counters should be created to make the system operation faster. These instructions should involve both processors in timer and counter servicing. The bit processor sets or resets the status bit of a given timer or counter. At the same time the byte processor completes its instruction (instruction subprogram). The bit processor executes its control program without waiting for the byte processor to receive its status. The status of timers and counters is stored in data memory of the byte processor. The copy of the status is sent to a special buffer register the outputs of which are located in the bit processor discrete input area. Thanks to the following hardware arrangement checking the state of each timer or counter takes one clock cycle and results in extremely fast timer and counter servicing mechanism. The structure enabling both processors to access the timers and counters independently is shown in Fig. 2.

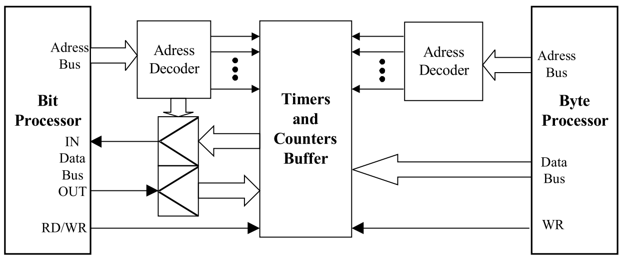

Fig. 2. Hybrid Realisation of Timer and Counter

4. Structure of bit-byte central processing unit of programmable logic controllers

Let us consider a startup procedure for a three phase AC motor as an example demonstrating the capabilities of the bit-byte central processing unit of a PLC. Motor switching relays are controlled from two outputs of the PLC. One of them switches the motor in Y configuration while the other one in Δ configuration. During the start-up phase the motor power connection configuration is changed from Y to Δ after the shaft reaches the desired minimal rotation speed. Rotation speed is measured as analogue value. From the point of view of a PLC CPU rotation speed is just a number.

The control program consists of parts that are executed by the bit processor and the byte processor. Some actions require co-operation between both processors. There are three digital inputs connected to the controller. Two of them are connected to push buttons and the other one is connected to a sensor:

- *Start* – pushing this button starts the process after a certain delay. Process begins from start-up procedure.
- *Start_up* – switches power to motor in Y configuration,
- *Stop* – pushing this button stops immediately the entire process and turn off all outputs,
- *BLK* – this signal confirms that the object meets all conditions required for process starting – start enable,

Analogue input delivers information about the actual motor speed.

Two binary outputs are also used:

- *Start_up* – is active for the time needed by the motor to reach the nominal rotation speed,
- *Out* – signal that controls output in normal working mode. Activated after start-up procedure.

A schematic diagram of signal connection is presented in Fig. 3. The control algorithm written in LD graphical language for Simatic S7-300 [3] is presented in Fig. 4. Apart from the symbols of switches and coils two block components are used. Comparator is used to compare the current speed with the reference value (nominal speed). Timer unit is used to delay the *Start-up* signal.

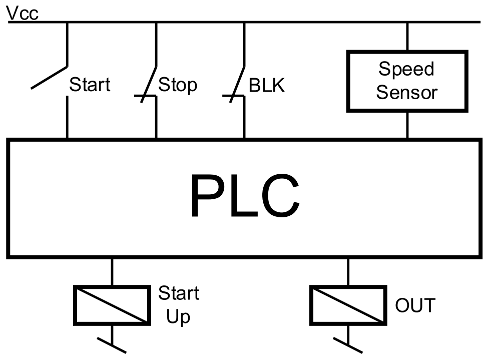

Fig. 3. Example circuit signal connection

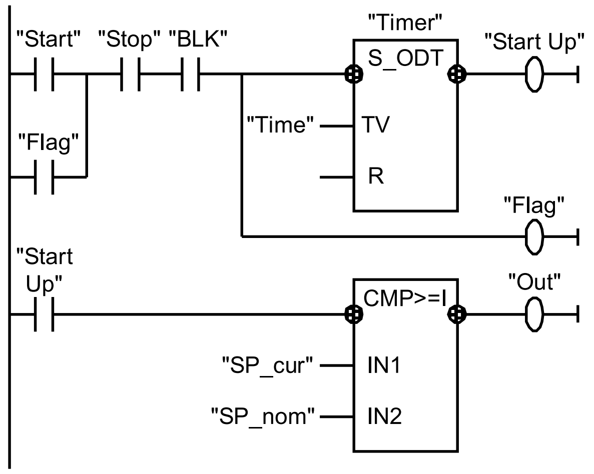

Fig. 4. The Ladder Diagram (LD) program

All places where passing of logical conditions between processors is required are marked in Fig. 4. As seen, the logical condition based on signals *Start*, *Stop*, and *BLK* activates output, uses a memory "Flag" and triggers time counting operation. Timer drives *Start_up* signal. When *Start_up* signal is activated it enables the comparison block. This block checks the actual motor speed value and activates the *Out* signal if it is at least the same as the reference value. There are four different places where information is passed between processors in CPU. Twice the information is passed from the bit processor to the byte processor. On the Ladder Diagram each block component is triggered by logic signals. On the other hand each block drives logic signals. This simple example gives overview of logic state passing in bit-byte PLC CPU between two processors. Another problem concerns control program instruction ordering, fetching and passing between processors as well operation synchronisation.

4.1. Basic bit-byte architecture of CPU. In the basic architecture of a bit-byte CPU instruction memory is common. Each processor fetches an instruction, as it is needed. In this mode processors operate in a serial way waiting for each other until instruction execution is completed. Introducing dual processor architecture even with single instruction stream is expected to improve performance. Bit operations are time consuming for a universal processor or microcontroller. Additional processor that is designed for bit operation greatly reduces time and program overhead for bit operation for bit operations [4].

In a simple dual processor architecture the program listing is similar to that presented in Fig. 5 [5]. The bit processor executes bit instructions while the byte processor remains idle. Byte instruction is executed by the byte processor while the bit processor is waiting for instruction. Complex bit-byte instructions are expanded into instructions for either or both processors in an appropriate order.

```
1.  LD   Start        ;Bit Instructions
2.  O    Flag
3.  A    Stop
4.  A    BLK
5.  =    Flag
6.  L    Time         ;Byte Instruction
7.  SD   Timer        ;Byte-Bit Instr.
8.  LD   Timer
9.  =    Start Up     ;Bit Instruction

10. LD   Start Up     ;Bit Instruction
11. L    SP_cur       ;Byte Instructions
12. L    SP_nom
13. A>=I              ;Byte-Bit Instr.
14. =    Out          ;Bit Instructions
```

Fig. 5. Program 1 written in instruction list form

The program contains 14 instructions, 9 of which are instructions for the bit processor. Thus the instructions for the bit processor constitute 64% of the whole program. Some manufacturers determine the scan time, i.e. the time of executing 1000 instructions, for such content of binary instructions in the program. The example program satisfies thus the requirements demanded by such standards.

Let us assume that instruction execution times in our example program are the same, as execution times for the well-known Siemens S7-315-2DP PLC. For this PLC execution time of a binary instruction is 0.1 μs, and execution time of a word instruction is 10 times longer – 1 μs [6]. Assuming such execution times, the PLC will execute the program presented in Fig. 5 in 5.9 μs (of course the time required for system functions, communication with I/O modules, etc. was not included).

This basic dual processor architecture can use bit-processor as a kind of co-processor for specific operation [6]. When both processor units are equally privileged then additional arbitration circuitry is required. The circuit initially decodes instruction and directs it to the proper processing unit.

The dual processor architecture [7] where instruction decoder selects processing unit for execution of current instruction is presented in Fig. 6. When instruction processing is completed the active processor increments instruction counter and the cycle starts over. Usually instruction decoder is a part of the bit processor. Instruction processing is deterministic and controlled by a common instruction decoder for both processors. Common instruction memory forces serial execution of the control program. The byte processor is also equipped with local standard-procedures memory. Common procedures for complex control instruction, such as PID or FUZZY controllers, timers, counters, advanced arithmetic functions, network and remote communication servicing etc., are stored in this memory as standard procedures. The CPU shown in Fig. 6 utilises high speed of bit processor operation however it has to execute control program serially because the system is equipped with one control program memory, one process I/O image and one system bus. In the presented solution standard-procedures memory is used. The procedures stored in this memory could be executed by the byte processor concurrently to bit processor operation. The logical conclusion would be therefore to equip each processor with its own program memory so that the relevant parts of the program could be placed in the memory of the appropriate processor.

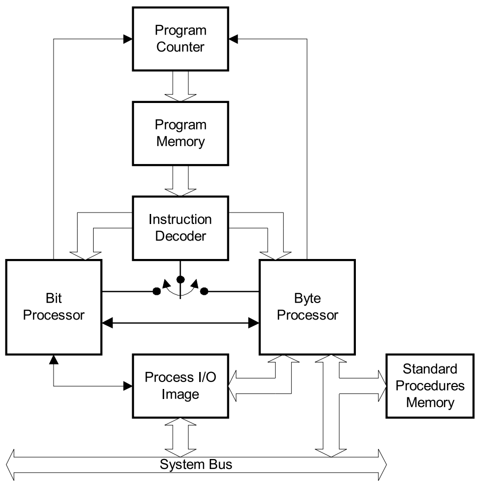

Fig. 6. Block diagram of the one bus CPU [7]

Such approach was proposed by Donandt [8]. Computation process is controlled by the byte processor that passes tasks to the bit processor. The bit processor is an autonomous calculation unit dependent on the master processor. This approach allows for concurrent operation of both processors. Once the operation of the bit processor is initialised, the byte processor (master) is able to resume its operation and continue program execution instead of waiting for the bit processor to complete its task. There are architectural limitations that reduce performance and concurrency. Only master processor can access process image memory. This limits parallel program execution.

4.2. Dual processor CPU. The presented examples of bit-byte CPUs deliver guidelines and requests for new architecture design. Dual processor CPU should be equipped with a fast bit processor. This processor may be designed as custom hardware using, for example, programmable logic components. The bit processor is responsible for instruction fetching. The byte processor is equipped with local standard-procedures memory that contains implementation of all byte instructions that this processor is expected to perform. Each instruction for byte processor is a subprogram that is executed as a request from the bit processor. This approach establishes the fast bit processor as a master processing unit. In order to avoid conflicts during data memory access each processor uses its local data memory.

The CPU architecture described above is shown in Fig. 7 [9]. a control program is stored in the main program memory. The program contains instructions for the bit processor. Byte processor instructions are represented by entry point addresses to the appropriate subprograms. After completing execution of an instruction the byte processor transfers its ready state to the bit processor. The bit processor fetches then the next instruction and passes it for execution to the appropriate processor.

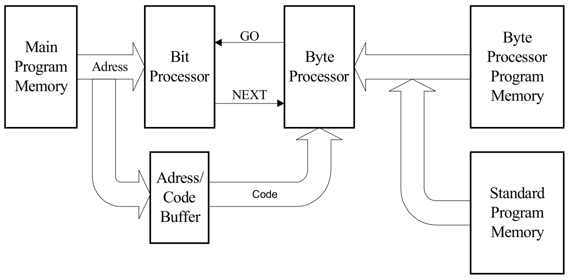

Fig. 7. Bit-byte CPU with Master Bit Processor

The flow diagram for program execution is shown in Fig. 8. In the presented circuit the processors are able to execute a single instruction or a group of instructions for a given processor concurrently. Such possibility decreases the time of control program execution.

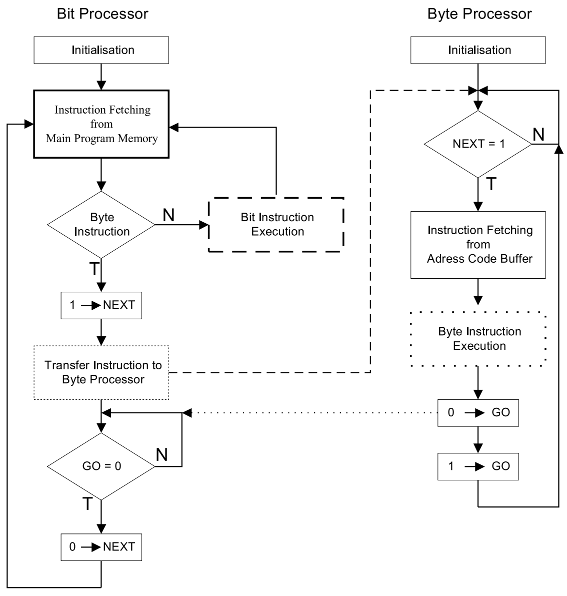
Bit Processor
Byte Processor
Initialisation
Instruction Fetching from Main Program Memory
Byte Instruction
N
Bit Instruction Execution
T
1 → NEXT
Transfer Instruction to Byte Processor
GO = 0
N
T
0 → NEXT
Initialisation
NEXT = 1
N
T
Instruction Fetching from Adress Code Buffer
Byte Instruction Execution
0 → GO
1 → GO

Fig. 8. Program execution in bit-byte CPU

Let us return to the previously considered example and consider step-by-step program execution process by the CPU from Fig. 7, which operates according to the flow diagram presented in Fig. 8. The control program shown in Fig. 9 consists of different kinds of instructions:

- Bit instructions stored in the main program memory executed usually in one clock cycle,
- Byte instructions that require byte processor program memory access,
- Complex instructions that require processor co-operation while the result of operation depends on both processor calculations performed in the appropriate order.

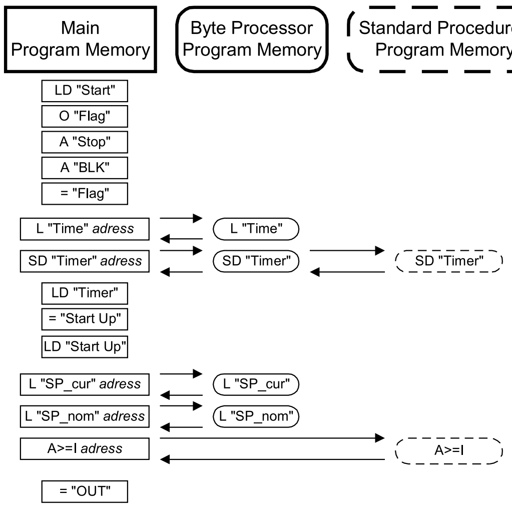

Fig. 9. Instruction allocation case 1

Byte processor instructions can have the following format:

- Simple without arguments – the instruction consists of operation code that points to program memory place where standard programs can be found ($A >= I$),
- Simple instruction with single argument – contains (L "Time"),
- Complex instructions that require co-operation of bit-processor. Instructions of this type trigger appropriate actions in both bit and byte processors (SD "Timer").

All bit instructions are executed very quickly (each instruction in a single clock cycle). Byte instructions must be passed to the byte processor. Signal NEXT is activated (Fig. 8). Bit processor enters wait state until signal GO is activated because subsequent instruction is also of the byte type. In the following situation the bit processor has to enter wait state and the byte processor must inform it about execution progress, which increases operation time. Another approach would be to make the bit processor initiate the following three byte instructions once. This allows for shortening of an instruction execution time. The same optimisation may be applied to the next two byte instructions. The proposed modification results in shortening of main program. Execution time of byte instruction group is also reduced, as they are now stored in the byte processor memory. Overall execution time of program loop is also reduced. Additionally it can be noticed that the first two byte instructions of Fig. 10 are independent of the following bit instructions. As those instructions are independent from each other they can be executed concurrently with these bit instructions. As expected, the execution of program loop was reduced by concurrent operation of bit and byte processors. When above description is considered operation speed of processors must be taken into account. Bit processor is able to perform its operations much faster than byte processor (a few dozen times). There are also operations the result of which influences the operation of the other processor. This is a very important problem of data exchange between processors.

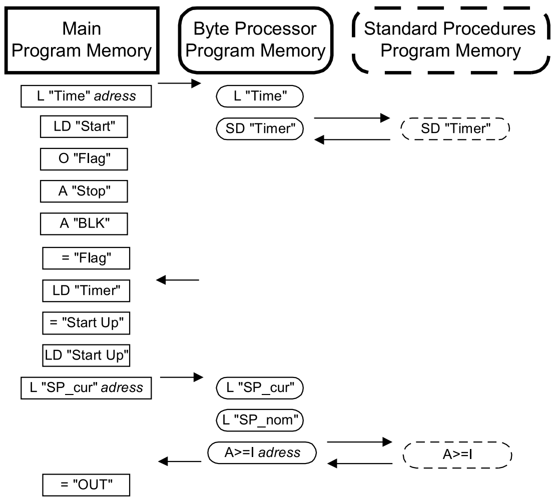

Fig. 10. Instruction allocation case 2

In a single processor unit a condition flip-flop is used to calculate logic functions that determine execution of subsequent operations. A simple two processor CPU is equipped with only one condition flip-flop too that is available for both processors. The above arrangement is correct only when serial program execution is considered. A single condition flip-flop limits parallel operation of the CPU. In the case of the considered architecture bit and byte processor must be able to process and exchange logic conditions without interrupting the operation of the other processor. This requires implementation of two logic condition flip-flops, one for each processor. This allows for execution of program parts that require logic function execution in each processor while the operation of the other processor is not influenced. As a result both processors will be able to execute program blocks that don't contain information exchange between processors.

Let condition flip-flops be called F_B and F_b for byte and bit processor respectively. Information stored in flip-flops must be transferred to the appropriate processor. It can be done by transferring value from F_B to F_b and also from F_b to F_B. This data transfer doesn't require additional hardware overhead. There is serious limitation in the described architecture whenever the processor that wants to access the condition flip-flop of the other processor has to wait until access is granted.

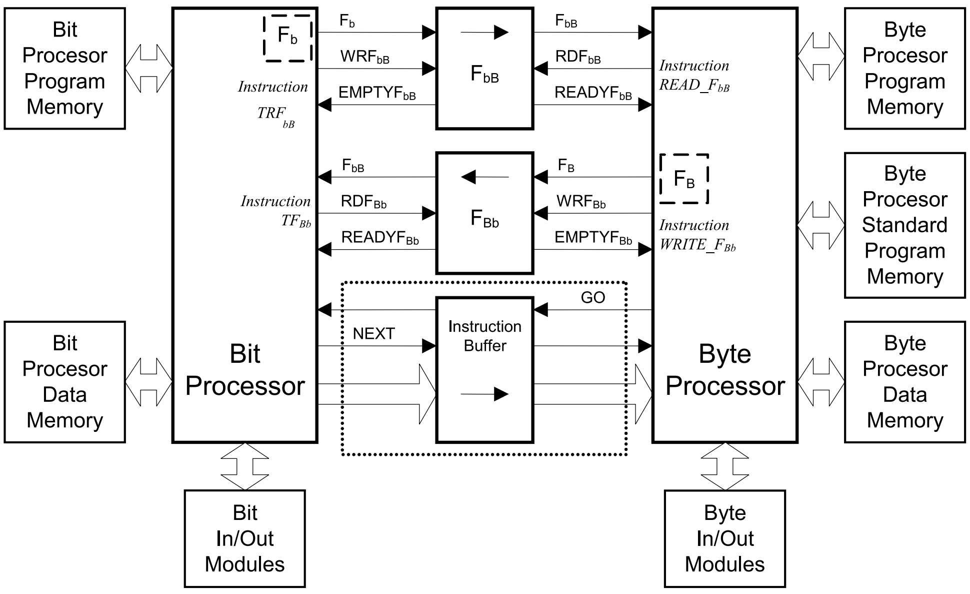

Fig. 11. Block diagram of parallel CPU structure

Whereas those problems are not very important for serial program execution, they become an object of interest in the case of concurrent program execution. In order to reduce the number of synchronisation wait states a modified architecture of condition registers may be proposed, that is introduction of specific buffered condition flip-flops that are available not only for the host but also for the other processor. The content of the internal condition flip-flop may then be transferred to an additional (buffer) condition flip-flop in order to make it available for the other processor. There are two additional condition flip-flops that store a copy of the main condition flip-flop – one for each processor. Each processor transferring condition flip-flop content to the buffer flip-flop is able to continue program execution until the next condition flip-flop update. New information can be written to buffer flip-flop only if the previous content was read out by the other processor. When buffer flip-flop is empty (doesn't contain valid condition data) condition data can be transferred and processor can immediately resume its operation. This requires appropriate compiler that is able to insert synchronisation instructions into compiled code while program designer is concentrated on problem solving.

A schematic diagram of the proposed architecture is presented in Fig. 11. Condition buffer flip-flops are called F_{Bb} (transfer from byte to bit processor) and F_{bB} (transfer from bit to byte processor).

Let us come back again to the previously considered program presented in Fig. 5. In this new situation the program may be split into two independent parts. The first part generates signal *Start_up* while the second controls signal *Out* depending on the value of *Start_up*. The first part of the program begins with 5 instructions for the bit processor that use flip-flop F_b. Instructions 6 and 7 are executed by the byte processor. A critical part of the program is instruction 13. The instruction performs a logical AND on the result of a comparison executed by the and the logic result obtained by the bit processor. Finally logic condition is obtained for *Start_up* output. Operation of both processors is required in the presented part. The obtained results influence the operation of the other processor. The bit processor executes its part of the program and finally performs logic-AND with the contents of F_{Bb} flip-flop. Byte processor performs load and comparison operations. The result of comparison is transferred to F_B flip-flop.

```
 1. LD    Sta rt
 2. O     Flag
 3. A     Stop
 4. A     BLK
 5. =     Flag
 6. =     FbB         ;Fb to FbB
 7. Read  FbB         ;FbB to FB
 8. L     Time
 9. SD    Timer
10. LD    Timer
11. =     Start_Up
12. LD    Start_Up
13. L     SP_cur
14. L     SP_nom
15. >=I               ;Both Processors
16. Write FBb         ;FB to FBb
17. A     FBb         ;FBb to Fb
18. =     Out
```

Fig. 12. Program 2 in instruction list

Program listing for dual processor CPU is presented in Fig. 12. There are two additional instructions (6, 7) that are responsible for transferring the state of *Flag* - the bit processor controls this value – finally to F_B flip-flop. Based on the state of F_{bB} flip-flop the byte processor is able to trigger timer operation. Two additional instructions are inserted in the program allowing *Flag* value to be exchanged between processors. There is one additional point of inter-processor data transfer. The status of the timer (counting the delay of the output *Start_Up*) is required by the bit processor while timer procedures are performed by the byte processor. Using the presented programming mechanism dedicated for timers, discussed in Section 3, modified timer construction changes operation of LD Timer instruction that is now executed only by the bit processor. It tests the state of appropriate timer bit.

Dual processor CPU introduces little overhead in instructions. In the presented program fragment two additional instructions appear. On the other hand this overhead allows for parallel and concurrent operation. The first six instructions for bit processor can be executed with the next load instruction for byte processor almost in parallel way (serial instruction fetching). Synchronisation point is located in the 7th instruction. At this instruction bit processor will wait until the status of F_{Bb} flip-flop will be read-out.

Further analysis of the considered program reveals two additional instructions that are responsible for transferring comparison result to F_{Bb} and logic-AND operation on the content of this flip-flop. The state of *Start_up* output must be transferred to F_{bB} flip-flop. Two additional instructions are inserted in the program for the value of *Start_up* bit value to be exchanged between processors (Fig. 12). There is one additional point of inter-processor data transfer (instructions 16 and 17). The bit processor requires the status of the compare operation while the byte processor performs the main operation. Using the programming mechanism presented above it is necessary to place two additional instructions for exchanging logic conditions between processors.

```
Bit Processor          Byte Processor
LD  "Start"
O   "Flag"
A   "Stop"
A   "BLK"
=   "Flag"             L      "Time"
=   FbB  ────────────► Read   FbB
LD  "Timer"            SD     "Timer"
=   "Start_Up"         L      "SP_cur"
                       L      "SP_nom"
LD  "Start_Up"         >=I
A   FBb  ◄──────────── Write  FBb
=   "Out"
```

Fig. 13. Program 3 in instruction list

After all modification in the hardware structure of the controller the program consists of 18 instructions while 4 of them transfer logic conditions and synchronise processor operation. It is important that all those instructions belong to the basic set of instructions and their execution is relatively short. Of course the extra instructions would extend program execution time up to 6.3 μs, if the program was processed fully serially. But in this case the program is partitioned into two parts, so as to enable assigning each part to one of the processing units: the bit-processor, and the byte-processor. A program that is split in two columns for bit and byte processor with additional transfer and synchronisation instructions is presented in Fig. 13. The discussed case requires condition transfer in both directions between processors. This simple example allows one to become acquainted with the method of simultaneous concurrent operation of processors in a hybrid bit-byte PLC CPU. As the bit processor can operate much faster than the byte processor both processors could start operation immediately. Synchronisation point should be placed for instruction – A F_{Bb}, which requires updated value of speed comparison.

It may be noticed that total execution time in this program fragment was reduced. Execution time is determined mainly by the byte processor and its program consists of seven instructions (five word and two binary instructions).

After the partitioning, the program consists of two parts: the bit-part, and the word-part. The program execution time is now not so easy to determine. We can calculate program execution times separately for each processor. Thus for the bit-processor the execution time is 1.1 μs, and for the byte-processor – 5.2 μs. Bearing in mind high discrepancy between the times we can state that the bit processor executes its instructions almost in the background of the byte processor operation. Its operation doesn't extend program loop execution time.

The conclusion is that the total program execution time, for a CPU working according to the presented principle, is equal to the execution time of the word-part of the program in the byte-processor, i.e. 5.2 μs. We save 0.7 μs, and this makes several percent of the total execution time. It should be also stated that adding to the program even 40 extra binary instructions, in practice will not extend the scan time, because the mechanism explained in Fig. 10 will work. The extra instructions will be executed while the byte-processor is busy executing its own program, and the bit-processor would have to wait for updating the state of the condition flip-flop. The observable result of the operation is hiding bit operations under byte operations. Execution time isn't the sum of all instruction execution times since byte instructions are executed concurrently with bit one.

If the same mechanism is applied to other instructions the effect of concurrent operation may be achieved. A processor waits for the other unit when synchronisation data exchange point is achieved. It may be noticed that synchronisation mechanism doesn't require passing instructions from bit processor to byte processor.

The program is also free from additional operations connected with instruction passing. Additional instructions are inserted for synchronisation and logic condition exchange. The total program execution time is equal to the maximum value of byte processor instruction execution time. Usually byte instructions are executed longer than bit instructions so the execution time is dominated mainly by the byte processor.

4.3. Concurrent operation in bit-byte CPU.

Let us consider the block diagram of the CPU from Fig. 11 in which the instruction buffer was removed while each processor owns instruction memory. Processors are synchronised by the state of condition flip-flops F_{Bb} and F_{bB} (empty/full). Instructions are delivered to the processors from their local memories so that they don't have to wait until the instruction is delivered from common program memory. Processors enter wait state only when they attempt to read empty condition register or overwrite not read condition in register. Program execution is synchronised by conditional execution of program part that depends on the result delivered by the other processor. In order to avoid long wait states the program should be written and compiled in such a way that the load of both processors is equally distributed in operation time. Further optimisation may be achieved by increasing the number of flip-flops that pass logic condition between processors or implementing a common memory area used for information exchange. In that case registers or cells have to be assigned to given tasks.

Figure 14 shows an example of application program, which is an extended version of the program presented in Fig. 5. The extension consists in additional Error output sets when Motor Start up procedure didn't work properly.

```
LD    "Start"     ;bit processor
O     "Flag"
A     "Stop"
A     "BLK"
AN    "Error"
=     "Flag"
=     FbB1        ;Fb to FbB1
Read  FbB1        ;FbB1 to FB
L     "Time1"     ;byte processor
SD    "Timer1"
LD    "Timer1"    ;bit processor
=     "Start_Up"
L     "SP_cur"    ;byte processor
L     "SP_nom"
>=I
WriteFBb          ;FB to FBb
LD    "Start_Up"  ;bit processor
A     FBb         ;FBb to Fb
=     "Out"
LD    "Start_Up"
=     FbB2        ;Fb to FbB2
Read  FbB2        ;FbB2 to FB
L     "Time2"     ;byte processor
SD    "Timer2"
TH    "Timer2"    ;bit processor
R     "Flag"
S     "Error"
A     "Stop"
R     "Error"
```

Fig. 14. Example of program in instruction list

As it can be seen, the program consists of two parts, that will be able to work completely independently, provided that the circuit is equipped with two condition flip-flops. This way we obtain an effect equivalent to two smaller programs processed in a quasi-concurrent manner. From the point of view of the condition flip-flops, such an operation of the PLC would be fully parallel. If required, however, operation of any of the processors could be suspended to wait for the other unit, but only within a specific task – the second task could be further processed. The idea is schematically explained in Fig. 15. The diagram shows a case, in which we have n tasks, mutually independent with respect to data exchange between the processor units. Thus every unit processes n tasks that require mutual data exchange. Besides, every unit can process any number of additional tasks that don't require information exchange.

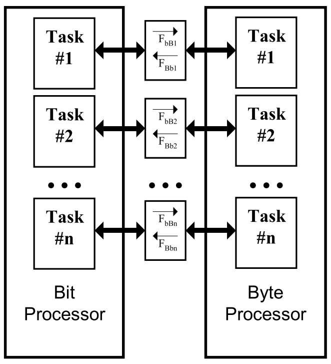

Fig. 15. CPU block diagram for multitasks control program

The mechanism incurs of course additional hardware and software costs, because instead of using one condition flip-flop, it requires applying a great, and furthermore unknown, number of them. The number of condition flip-flops can however be appointed based on the size of the program memory, and thus a possibility of assigning subsequent flip-flops to subsequent tasks requiring data exchange can be provided. The problem can also be handled in a more intelligent way, by using reconfigurable circuits, from the resources of which the condition flip flops can be freely configured, while the spare resources can be utilised to implement ordinary flags.

Now the problem should be considered, whether the handshaking mechanism applied in the presented solution of state exchange between condition flip-flops is really indispensable.

Preserving serial program execution by each processor causes that conditions are generated in the same order. That allows for designing specific hardware with extremely fast access to condition bits. The solution is based on a set of D flip-flops. Their content is written by one of the processors while the second reads its contents. The discussed approach eliminates conflicts and reduces access time. An ordinary FIFO memory cannot however be used, because of freely running processor units, for which program execution times will be different, the FIFO would lose synchronisation, and the processors would read states of wrong condition flip-flops. To avoid this a modified circuit presented in Fig. 16 ought to be built. Two flip-flop registers must be implemented – one for each direction of condition transfer. Registers are connected to one processor with write access while the other one has read access to them. Position that is currently written to is pointed to by a pointer that is incremented by processor after write operation. After reading data from register read pointer is also incremented. The presented register forms queues where conditions are written-in and read-out in order they are appearing. There is possible danger of queue overlapped in the case where one processor is executing tasks much faster than the other one and generates large number of synchronisation flags [10]. The presented architecture is extremely fast since accessing information takes no longer than one clock cycle for bit processor and port write operation for byte processor. This solution is unfortunately more expensive and complicated than using RAM. If RAM is used for conditions storing, apart from access time, which is longer for memory than for flip-flop, there is also the problem of access organising.

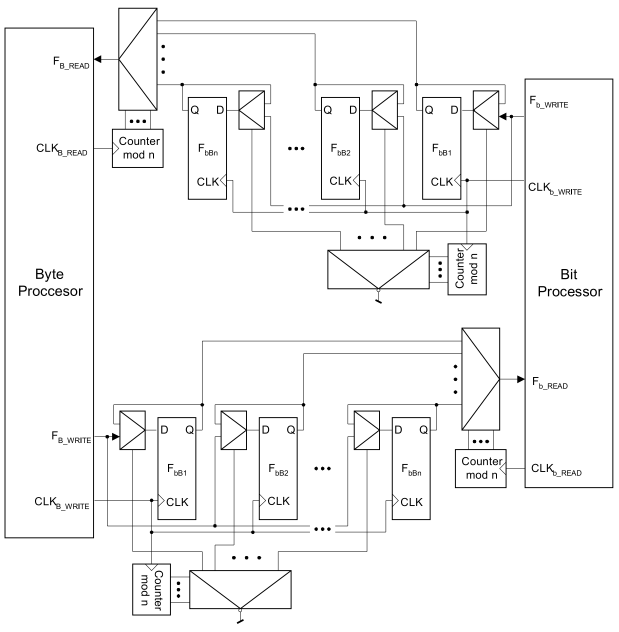

Fig. 16. Block diagram of the two-processor CPU with a flip-flop register unit

The problem may be solved by dual port RAM. This allows for simultaneous access to data without using arbitration circuitry. In order to protect the system against violation of possible write operation to the same cell from both ports, separate marker regions are assigned for bit and byte processors. The described memory access technology ensures that actual states of condition registers from one processor are available to the other like I/O signal states – in the following scans at the very latest.

It may be noticed that the presented solution allows for unconstrained operation of both processors that don't have to wait for condition passing. In the case of disproportion of program execution time between the processors some flags are updated and examined more frequently than other ones [11, 12].

At this moment the following question may be expressed. May the presented execution method lead to improper operation of the entire CPU? In general it cannot disturb the operation of the controller. In the presented structure common information is processed like other information delivered from sensors. The other processor is seen as a set of flip-flops that are being examined and based on their states proper actions are triggered. This approach has shorter response time while both processors can operate with maximal operation speed instead of waiting for calculation completing.

It seems, that the problems, if they could ever happens, would concern situations of large discrepancies between processor scan times – one of the units reads and writes back flip-flop states several times faster, than the other. It can be noticed that during control program loop executed by one processor the other may change a state of its condition flip-flop for a few times. As a result, certain parts of the control program executed by the first processor utilise different states of the condition flip-flop. The rule of serial program execution is violated. Both processors operate asynchronously with respect to each other. They are seeing the marker area as an I/O space – in order to execute suitable actions in response to changing markers.

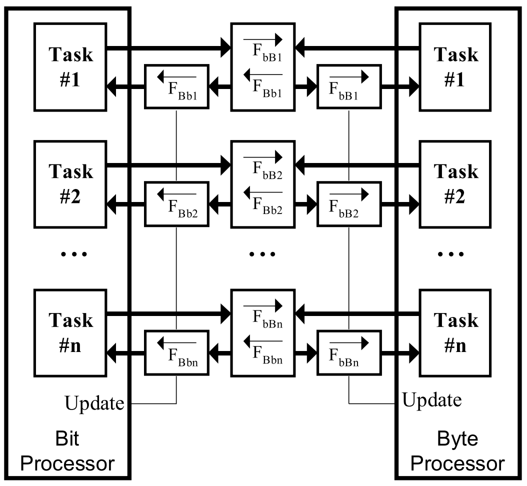

Fig. 17. CPU block diagram for multitasks control program with auxiliary buffer

From the point of view of the CPU I/O devices work slowly. It is however possible that during one program scan a state of input changes several times. In such a case the input changes will not be registered. Now similar problems arise inside our CPU, with data exchange between the processor units. To avoid the situation where one of the units processes different tasks on different states of the same condition flip-flops, additional buffering should be applied. The additional buffering consists in organising condition bit data exchange in such a way, that each processor performs writes to its own condition register at random moments, while the other decides, when the data should be copied to its own memory. The copying would be organised in a similar manner, as input scanning and output updating. Such a mechanism would cause, that the data stored in the additional register would constitute for both processors the same set of signals (with respect to the data access mechanism), as ordinary I/O-s. I/O states can also change during a program scan, but the processor is capable of acquiring this information only before the next program scan or, alternatively, the information is lost, if the changes last too short. Each processor unit would refresh from its side the data stored in the condition flip-flops, while the other unit, after completing a program scan, by generating an "update" signal, would initiate copying of the condition flip-flop states to its own image memory. The copying would be performed together with input scanning and PII updating. The idea of the condition flip-flop register with auxiliary buffer is presented in Fig. 17.

The presented CPU structures and the method of CPU operation reduce overall access time to input-output signals – during program execution (not considered as a single machine cycle). The processors are able to respond much faster to alarm and exception signals. Continued execution of program instead o waiting until an entire task of the other processor has been completed allows for much faster operation.

5. Conclusions

Solutions proposed in the paper enable to significantly speed up operation of a bit-byte central unit of the microcontroller. In consequence, time the central unit requires to access peripheral signals is also reduced. Particularly large gains are achieved when bit data is processed. Owing to the use of two separate buses and dedicated software-hardware timer and counter servicing circuit, as well as data exchange during execution of program loops, the bit processor may operate independently and in undisturbed way. The byte processor, in the case of the majority of its commands, requires relatively long time to complete the execution. Thus, during its normal operation the byte processor does not need to communicate with the bit processor as well as its operation is not disturbed by the latter. Fast access to peripheral signals and the fact that bit and byte microprocessors treat each other as peripheral devices resulted in the design of the central unit with independently operating components.

Table 1
Comparison of a few PLCs

PLC	Number of bit inst.	Number of byte inst.	Execution time [ms]
S5-100U	1030	140	9.0
S5-115U	1030	140	1.9
S7-224	780	50	2.8
S7-313	1000	115	2.7
S7-315-2DP	1000	115	1.5
Modicon A984	–	–	8.0
80C320 – serial mode	**1050**	**280**	**1.7**
80C320 – parallel mode	**850**	**140**	**1.1**

In Table 1 is presented comparison of a program execution time for a part of a practical application – the control system for a metal sheet pickling line at a Columbus Stainless ironworks (South Africa) [13]. The comparison was performed for the circuit that contained only the CPU, to avoid the influence of the time necessary for communication with I/O modules. The scan times don't contain the "empty" loop scan time, either. This enabled avoiding the influence of system-function execution time.

REFERENCES

[1] G. Michel, *Programmable Logic Controllers – Architecture and Applications*, John Willey & Sons, London, 1992.

[2] J.W. Webb and R.A. Reis, *Programmable Logic Controllers: Principles and Applications*, Prentice-Hall, Engelwood Cliffs, NJ, 1999.

[3] H. Berger, *Automating with STEP 7 in LAD and FBD – SIMATIC S7-300/400 Programmable Controllers*, Siemens AG, Monachium, 2001.

[4] M. Chmiel and E. Hrynkiewicz, "Remarks on parallel bit-byte CPU structures of programmable logic controllers", *Int. Workshop on Discrete Event System Design, DESDes* 1, 147–152 (2001).

[5] N. Aramaki, Y. Shimokawa, S. Kuno, T. Saitoh, and H. Hashimoto, "A new architecture for high-performance programmable logic controller", *Proc. IECON'97 23rd Int. Conf. on Industrial Electronics, Control and Instrumentation, IEEE* 1, 187–190 (1997).

[6] Siemens AG, *Simatic S7-200 Programmable Controller – System Manual*, Siemens AG, Monachium, 2002.

[7] Z. Getko, "Programmable systems of binary control", *Elektronizacja* 18, 5–13 (1983), (in Polish).

[8] J. Donandt, "Improving response time of programmable logic controllers by use of a Boolean coprocessor", *IEEE Comput. Soc. Press.* 4, 167–169 (1989).

[9] M. Chmiel and E. Hrynkiewicz, "Remarks on parallel bit-byte CPU structures of programmable logic controllers", in: *Design of Embedded Control Systems*, ed. M. A. Adamski, A. Karatkevich, and M. Węgrzyn, pp. 231–242, Springer Science + Business Media, Inc., Berlin, 2005.

[10] M. Chmiel and E. Hrynkiewicz, "Concurrent operation of the processors in bit-byte CPU of industrial PLC", *Int. Workshop on Programmable Devices and Systems, PDS'04* 1, 15–20 (2004).

[11] M. Chmiel, E. Hrynkiewicz, and A. Milik, "Concurrent operation of the processors in bit-byte CPU of a PLC", *Preprints of the IFAC World Congress* D, 44–49 (2005).

[12] M. Chmiel and E. Hrynkiewicz, "Improving of concurrent operation of the bit-byte PLC CPU", *Int. Conf. on Programmable Devices and Systems, PDS'06* 1, 132–137, (2006).

[13] M. Chmiel, A. Malcher, and A. Nowara, "A control system for a metal sheet pickling line", *Machine Technologies and Materials*, 20–21 (1997), (in Polish).