applied sciences

MDPI

Article

FPGA Implementation of IEC-61131-3-Based Hardware Aided Counters for PLC

Miroslaw Chmiel [1], Robert Czerwinski [1,*] and Andrzej Malcher [2]

[1] Department of Digital Systems, Silesian University of Technology, Adademicka Str. 16, 44-100 Gliwice, Poland; mchmiel@polsl.pl
[2] Department of Electronics, Electrical Engineering and Microelectronics, Silesian University of Technology, Adademicka Str. 16, 44-100 Gliwice, Poland; amalcher@polsl.pl
* Correspondence: rczerwinski@polsl.pl; Tel.: +48-32-237-1720

Abstract: The article discusses counters defined in the IEC 61131-3 standard. The possible implementations of standard counters function blocks in FPGAs are presented. First, counters are implemented as classical hardware-based modules. Second, counters are designed as the FPGA built-in memory blocks with a single common executing unit. These solutions are compared to each other and compared with counters realized in commercially available PLCs like Siemens SIMATIC S7 controllers. The structure of integrated hardware–software CPU with counters is presented. The paper presents how the designer can take advantage of the specific features of the FPGA devices to optimize both the utilization of resources and speed of realization of the particular blocks. Experimental results prove the high efficiency of the proposed solutions.

Keywords: programmable logic controllers (PLC); counters; IEC 61131-3; field programmable gate arrays (FPGA); function blocks; central processing units (CPU)

Citation: Chmiel, M.; Czerwinski, R.; Malcher, A. FPGA Implementation of IEC-61131-3-Based Hardware Aided Counters for PLC. *Appl. Sci.* **2021**, *11*, 10183. https://doi.org/10.3390/app112110183

Academic Editor: Paris Kitsos

Received: 22 September 2021
Accepted: 26 October 2021
Published: 30 October 2021

1. Introduction

IEC 61131 is the international standard for programmable logic controllers (PLCs). The third part of this standard is focused on the basic software architecture with programming languages [1]. Basically, it defines textual and graphical languages with necessary elements like data types, variables, program organization units (including functions and function blocks) and others. Among the programming languages, the Instruction List (IL) language occupies a special position. It is an assembler-like language and all other program representations—graphical and textual—can be converted into IL. The CPU should be optimized for efficient realization of the IL code as the low-level representation of the control algorithm. Unfortunately, it must be emphasized that the standard is not precise and requires many comments as well as improvements [2–4]. However, provisions of the standard are implemented in industrial PLCs and in experimental or academic structures. Great opportunities in this area offer field programmable gate arrays (FPGAs). FPGAs are integrated circuits configured by the customer. The main structure contains programmable logic and programmable connections, so it enables us to implement designed functions in a flexible way. The logic structure enables us to implement real concurrency that is the main advantage over microprocessors. Moreover, the microprocessors can be implemented in FPGAs as hard or soft cores, so the system-on-a-chip could be built.

Designing a dedicated microprocessor takes quite a bit of effort. It is much simpler to design a PLC using a standard microprocessor or microcontroller. Often, the problem is the operations execution time. The ways to minimize the execution time of operations have been used in CPUs built using standard microcontrollers [5]. A useful tool in using standard programming languages for PLCs is a program translator to ANSI C language. Such a translator was proposed in the paper [6]. In addition, this translator takes into account industry safety standards. In fact, the resources of a standard microprocessor/microcon-

troller are not aligned with the standard—the instruction list of a standard microprocessor is severely mismatched with what the standard defines. By developing a specialized microprocessor compliant with IEC 61131-3, these drawbacks can be eliminated [7–11].

The use of FPGAs provides completely different possibilities. The control program can be directly implemented in logic, which is presented, among others, in the works [12–15]. In this case, the program is converted into a hardware description language, and then this description is synthesized. However, this is a completely new approach and is still developed. However, the advantage of the FPGA that is concurrency is exploited.

The concurrency can be also exploited in order to design hardware accelerated modules of the PLC, like function blocks. According to the IEC-61131-3 standard, a function block (FB) is an organizational unit of software that, upon execution, can provide one or more values at its output, as opposed to a function that has only one output. FBs may be used many times, but for each call there must be created an appropriate unique data structure that stores information about the state of the called FB. Such a structure is called an instance. The standard FBs are:

- bistable elements,
- edge detection elements,
- counters,
- timers.

This paper concerns the counter implementation. The standard defines three types of counters:

- CTU—the counter that counts up pulses supplied to the count up (CU) input. The counter has the reset input (R) that enables clearing of the counter content to zero as well as the register that stores preset value (PV). If the current content of the counter equals to the preset value, the binary output (QU) is active,
- CTD—similar to CTU, but the counter counts down pulses supplied to the count down (CD) input,
- CTUD—the counter that is able to count in two directions and has functionalities as well as inputs and outputs of both types of counters: CTU and CTD.

In modern PLC applications, the availability of a large number of fast counters is increasingly required. Examples of applications include counting the number of manufactured or packaged small components (e.g., electronics) or products of the pharmaceutical industry. Reducing the execution time of a single counter operation entails reducing the entire program cycle and therefore increasing the performance of the PLC. This is especially important in the era of Industry 4.0, where the smart technology integration requires significantly increased system performance. The main contribution of the paper is to propose the structure of the counter function block for PLC. The fast counters may be implemented in FPGA logic devices as separate modules or can be integrated with CPU in a form of integrated hardware-software PLC.

2. Theoretical Background

2.1. The Counter

Figure 1 shows the appearance of the counter in graphical form. The interface of such FB is defined by:

- CTUD (C_NO)—counter number or instance name,
- CU—the Count Up input; sensitive to rising edge,
- CD—the Count Down input; sensitive to rising edge,
- R—the input of resetting the counter value,
- LD—the input of presetting the counter value,
- PV—value for initializing the counter and evaluation the QU value,
- Q (QU; QD)—status of the counter,
- CV—count value.

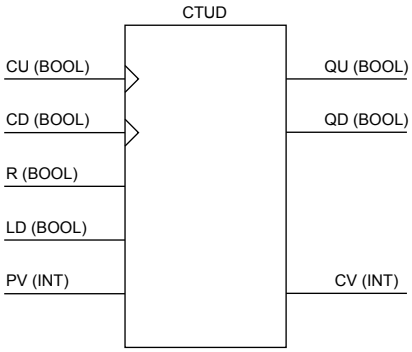


Figure 1. The symbol of CTUD counter.

To show relations between inputs and outputs (also priorities), the specification of the CTUD function module with INT format number in the ST language described in standard [1] is shown in Listing 1.

Listing 1. The specification of the CTUD counter in ST language [1].

```
FUNCTION_BLOCK CTUD
VAR_INPUT
        CU : BOOL R_EDGE;
        CD : BOOL R_EDGE;
        R : BOOL;
        LD : BOOL;
        PV : INT;
END_VAR

VAR_OUTPUT
        QU : BOOL;
        QD : BOOL;
        CV : INT;
END_VAR

IF R THEN CV:=0;
ELSIF LD THEN CV:=PV;
ELSE
        IF not (CU and CD) THEN
                IF CU and (CV<PV_MAX)
                THEN    CV:=CV+1;
                ELSIF CD and (CV>PV_MIN)
                THEN    CV:=CV-1;
                END_IF;
        ENDIF;
ENDIF;
QU := (CV>=PV);
QD := (CV<=0);
END_FUNCTION_BLOCK
```

Implementation of a counter needs mapping of a suitable data structure in the controller memory to enable storage of the current content of the counter as well as the status of the PV input and the status of QU and QD outputs. However, QU and QD can be calculated during their reading. In such a case, the data structure may not contain them. Two more memory cells may be necessary as well to store previous statuses of the CU and CD inputs (CU' and CD'). The example data structure for a single counter unit is shown in Figure 2.

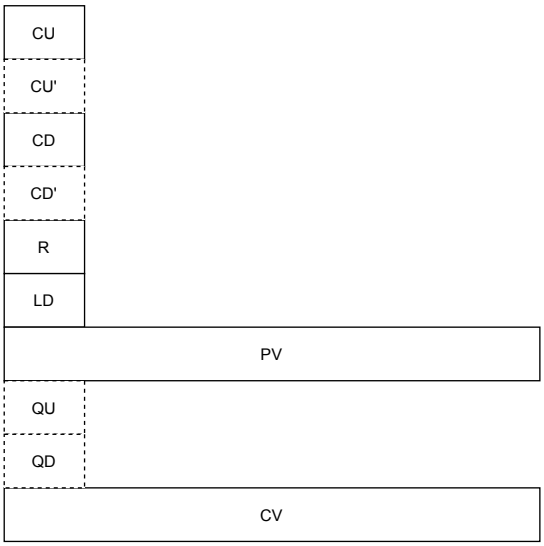


Figure 2. Minimum representation of a counter data structure.

2.2. The Counter CALL

The standard allows three ways of calling counter operations in IL. The first is to call a function block with the current parameters passed in parentheses [1]—e.g.,:

```
CAL      CNT (CU:=IN_CU, CD:=IN_CD,R:=IN_RES,
                                    PV:=IN_PV, LD:=IN_SET, CV=>CNT_VAL)
```

This type of call may have a complete list of assigned parameters or may have only a portion of the parameters assigned. In this case, the unassigned parameters take the values stored previously ([1]: Table 42, page 105). This way of calling function blocks is available in CoDeSys [16], Concept [17] software, and also in the IEC counters of SIMATIC S7 PLCs [18].

The next two ways are more ambiguous and require more commentary. The second one consists in preparing suitable data in particular fields of the counter structure, and then calling the function updating the counter without any parameters using the CAL command. Not all fields of the counter structure have to be filled in immediately before the function is called. If some values are not assigned to the structure fields, the function uses the values of the structure fields stored earlier. This way of calling the counter function block in Instruction Language (IL) is shown in Listing 2. As can be seen, there is a set of instructions (LD) for loading data into the current result (CR) register and storing data to the individual fields that make up the counter structure (ST). All these instructions merely fill in the relevant fields. The counter functionality is realized only when the CAL instruction is executed. This method of calling counter operations is available in the CoDeSys [16], Concept [17], and ISaGRAF [19] environments.

However, the provisions of the standard allow also using so-called short IL operators. When, in fact, the operation updating the state of the counter is performed, since the standard allows the use of operators (CU, CD, R, LD and PV). In this case, each short operand updates the counter state, which may improve the readability and conciseness of the program code. However, the standard says that each use of the short command is interpreted as sending a value from CR to the appropriate field of the counter structure, followed by a call (CAL) to the function that updates the counter. For the system designer it is a big difference that significantly affects the implementation of the counter block and also has a big impact on the execution time of this block. In the Siemens Simatic S7-300/400 controllers the basic counters for this system (Simatic Counters) do not form a function block, so there is no problem to use only counter elements that are necessary at the moment. It may seem that in this case the processing time of the counter is shorter than the longest one. Among the implementations of the standard known to the authors,

only Concept allows using operator based programming. Other implementations, such as CoDeSys and ISaFRAF, do not allow this method of calling counter functions.

Listing 2. Counter block calling using CAL instruction.

```
LD      IN_CU
ST      CNT.CU  ;CR -> CNT.CU
LD      IN_CD
ST      CNT.CD  ;CR -> CNT.CD
LD      IN_RES
ST      CNT.R   ;CR -> CNT.R
LD      IN_PV
ST      CNT.PV  ;CR -> CNT.PV
LD      IN_SET
ST      CNT.LD  ;CR -> CNT.LD
CAL     CNT     ;Counter Block Execution
                ;IF CU Positive Edge THEN CNT.CV+1 -> CNT.CV
                ;IF CD Positive Edge THEN CNT.CV-1 -> CNT.CV
                ;IF LD THEN CNT.PV -> CNT.CV
                ;IF R THEN 0 -> CNT.CV
LD      CNT.CV
ST      CNT_VAL ;CNT.CV -> CR
```

In the field of the short operators, the standard is somehow weak. The example code is presented in Listing 3. It is easy to imagine now that the counter does not require the existence of the CAL instruction in the program, which actually implements the counter's functions based on the information gathered earlier, but simply executes individual instructions that exist in the program. In fact, a program written using the short commands (operators) needs to be translated to the second way, and thus will take longer to execute than a program explicitly using the CAL command, because every short operator requires store operation and implicit CAL instruction.

Listing 3. Counter block using operators.

```
LD      IN_CU
CU      CNT     ;IF CR Positive Edge THEN CNT.CV+1 -> CNT.CV
                ;Equivalent to: ST CNT.CU + CAL CNT
LD      IN_CD
CD      CNT     ;IF CR Positive Edge THEN CV-1 -> CNT.CV
                ;Equivalent to: ST CNT.CD + CAL CNT
LD      IN_RES
R       CNT     ;IF CR THEN 0 -> CNT.CV
                ;Equivalent to: ST CNT.R + CAL CNT
LD      IN_SET
LD      CNT     ;(CR) IN_SET -> CNT.LD
                ;Equivalent to: ST CNT.LD + CAL CNT
LD      IN_PV
PV      CNT     ;CR -> CNT.PV
                ;IF CNT.LD THEN CNT.PV -> CNT.CV,
                ;update QU
                ;Equivalent to: ST CNT.PV + CAL~CNT

                ;Operators take into account the current state
                ;of all bits in the CNT structure
                ;Explicit CAL not required
```

There is another problem. The CU and CD inputs have built-in edge detectors. It is conceivable that the edge detectors will become an integral part of, either special load operations or increment and decrement operations. However, in a classical processor version, this will not change much, as these tasks will simply disappear from the IL notation, but the commands will still require their execution. It is worth noting that there exist implementations of the standard like ISaGRAF [19] where the CU and CD inputs do not have built-in edge detectors. In such a case, the user should implement edge detectors

as external blocks or functions preceding the counter call. Anyway, edge detectors must be somehow executed. Realizing a CPU based on FPGAs, the execution of these operations will be entrusted to the hardware part. This applies to the instructions already described, like CU and CD with simultaneous edge detectors, as well as to the R and LD instructions, which are also not really simple operators.

3. FPGA Counters Implementations

Counters' function block is proposed in this section together with the CPU unit integration. Hardware and software-like counters are presented.

3.1. CPU Unit

Field Programmable Gate Arrays offer great opportunities for hardware support of some tasks related to the implementation of the function blocks. In the classical solutions the microprocessor-based CPU is equipped with procedures/functions/macros that are responsible for function blocks functionality. This solution is based on the program processing of data stored in the data memory (Figure 3). Software routines for standard function blocks are part of the PLC operating system. The implementation of tasks consists in creating an appropriate data structure in the Data Memory and performing a specific function. It is necessary to have memory to store data structures and to spend CPU time on calling a specific function. Such a unit turns out to be ineffective from the point of view of resource consumption and time of individual tasks.

The basis of the integrated hardware–software controller is a dedicated CPU with a developed microprocessor, the machine language of which complies with the IEC 61131-3 standard. Problems of the CPU construction are presented in [11]. As shown in the paper [11], the FPGA-based central processing unit has the ability to perform operations in two clock-blocks. Consequently, it should be noted that the execution times of all basic operations will be 20 ns at a CPU clock frequency of 100 MHz. As shown in the work [20] and [21], this also applies to instructions that have been proposed to be equivalent to the function blocks of flip-flops and edge detectors. Tests have been carried out for standard FPGAs, which have shown that all basic operations and operations associated with binary function blocks are performed in one clock, so complex operations take the same amount of time as the basic instructions.

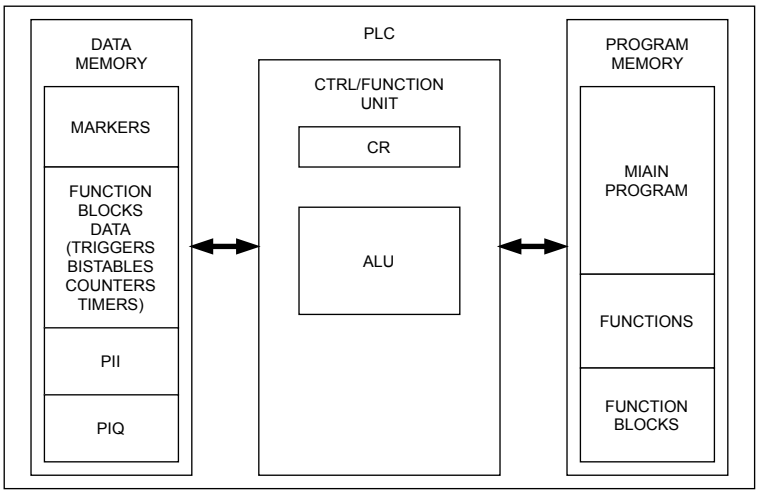


Figure 3. The architecture of the classical CPU of the PLC.

The solution proposed in the paper is to construct a controller with integrated blocks of hardware supporting the performed operations—IHSPLC (Integrated Hardware-Software PLC). Among others, it can be implemented in the FPGA system. Figure 4 presents the idea of the controller implemented in this way.

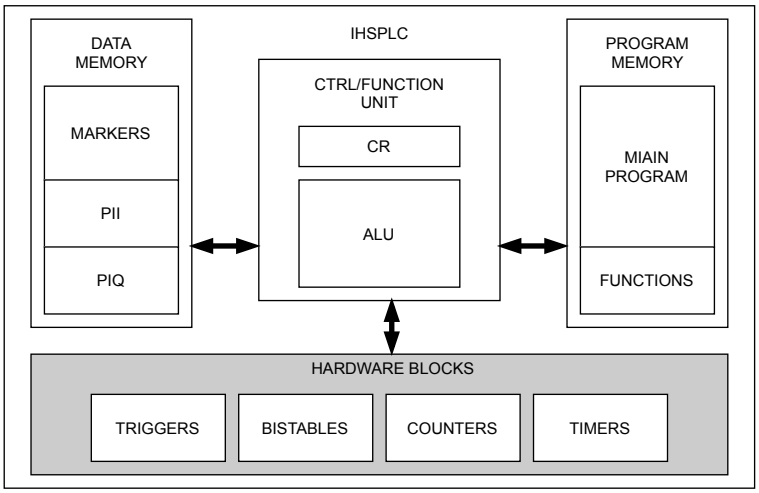


Figure 4. The idea of the Integrated Hardware-Software PLC.

The standard [1] defines no data width for Current Result. It is easy to implement a software-based PLC with dynamically resized width of CR, but it is hard to implement this idea in hardware, especially if it must work in an atomic way. The proposition is presented among others in [11]—this concept may be combined with bit-byte idea presented in [22–24]. As the effect, the proposed in this paper bit.WORD IHSPLC works on CR_bit (CR_b) and CR_word (CR_W) accumulators (Figure 5). It is not inconsistent with the provisions of the Standard, because the user can write common instructions (e.g., LD), and it is up to the compiler to decide (considering the data type) which CR the processed data is sent to.

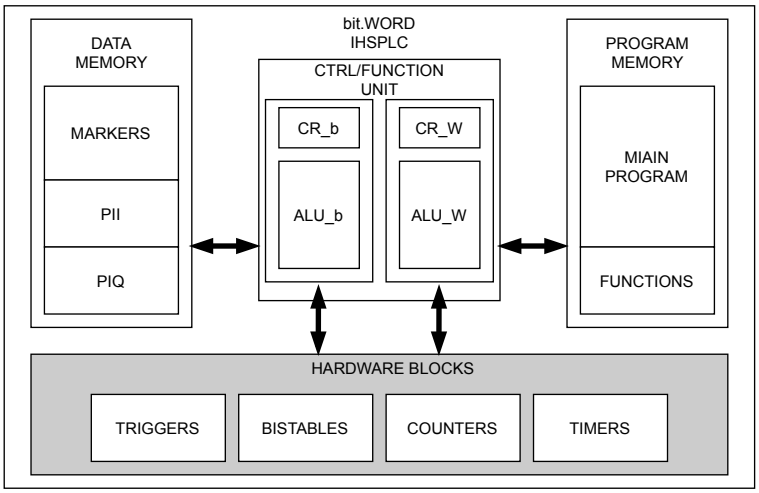


Figure 5. The structure of the bit.WORD IHSPLC.

In order to accelerate the operation of standard function blocks they are implemented in the hardware. Appropriate logical resources are used in FPGA to implement hardware acceleration. In other words, the functions that were responsible for processing data organized into special structures are replaced with hardware logical structures that determine the results. These logical structures connect data structures with operations. However, the main problem is on the junction of the hardware units and CPU.

For newly designed microprocessors, it is possible to integrate function blocks into the CPU structure. It can be easily integrated with the rest of the structure. However, when designing hardware accelerated blocks, it must be designed with minimum latency for communication. The structure of counters function block proposed in the paper is just driven by means of the CR, address, clock signal and set of enable signals. It ensures

minimum delay for data exchange between function blocks and CPU (Figure 6). It does not matter whether the counter function blocks are hardware counters, software-like or even fully software implemented blocks.

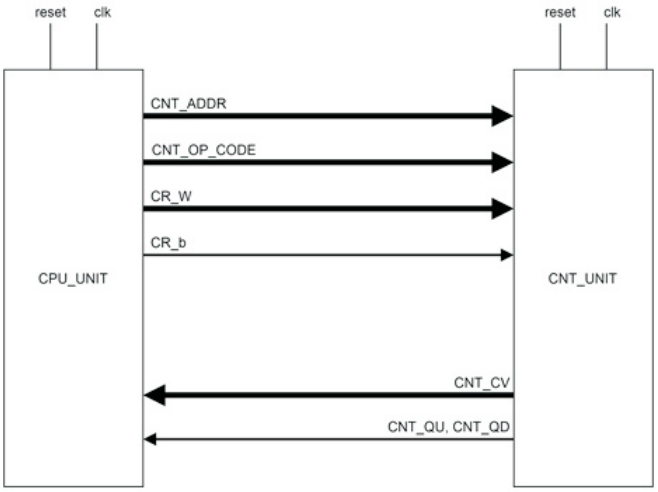


Figure 6. The interface between function unit and hardware-supported blocks.

3.2. Hardware Counters

In the hardware counter function block design the main problem is not to design the counter, but the whole environment. The counter itself is not a problem, because this task is well controlled in the HDL synthesis. The description of the counter presented in the standard is presented in Listing 4a, while the Verilog description is presented in the Listing 4b.

However, the main problem presented in detail in Section 2 is the execution moment of the counter. It is not the problem in the hardware counter as described in Listing 4b. The active clock edge executes the counter. However, such a counter is executed based on the state of inputs like 'CU', 'CD', etc. Those inputs must be stored somehow, because the counter can be driven only by means of CR_b. Moreover, one of the most important parts of the counter is PV that must also be remembered. It can be constructed as presented in Figure 7.

The counter presented in Figure 7 receives an active clock edge together with other parts of the environment. It is unambiguous with the counter execution, so it is compliant with conclusions drawn in Section 2 for Listing 3 and short operators. When the design should be compliant with the action presented in Listing 2, every counter's input must be remembered, but the counter may be executed after CAL instruction. The proper structure with counter's Verilog description is presented in Figure 8 and Listing 5, respectively. The call (CAL) is implemented by means of clock enable (CE) input that is built-in in the FPGA D flip-flop. The RTL (Register-Transfer Level) schematic of the counters function block is presented in Figure 9.

One more thing needs some comment. The structure presented in Figures 7 and 8 returns 'cv', 'qu' and 'qd'. However, the CPU expects only CR_b and CR_W inputs, so 'qu' and 'qd' must be multiplexed and every single counter must be connected to the CR_b and CR_W by MUXes too. The structure presented in Figure 10 is a complete hardware counters function block for four counters.

Listing 4. (**a**) The description of the counter presented in the standard, (**b**) the Verilog description of the counter.

```
(a)
IF R
  THEN CV := 0;
ELSEIF LD
  THEN CV := PV;
ELSE
  IF NOT (CU AND CD)
    THEN
      IF CU AND (CV < PV_MAX)
        THEN CV := CV + 1;
      IF CD AND (CV > PV_MIN)
        THEN CV := CV - 1;
      END_IF
   END_IF
END_IF

QU := (CV >= PV);
QD := (CV <= 0);

//-------------
(b)
always @(posedge clk)
  if (reset)
    cv <= 32'b0;
  else
    if (ld)
      cv <= pv;
    else
    if (!(cu && cd))
      if (cu && (cv < PV_MAX))
        cv <= cv + 1'b1;
      else
        if (cd && (cv > PV_MIN))
          cv <= cv - 1'b1;


assign qu = cv >= pv;
assign qd = cv <= 0;
```

3.3. Software-Like Counters

It is quite easy to imagine a software counter. It is just a variable that is incremented/decremented in case of particular events. Hardware-based counters that are designed by means of D (or others) flip-flops are also well known. Hardware-based counters work very fast in relation to software counters. Software counters must be executed somehow by means of the program, so it lasts (Listing 1, Section 2). However, there is no problem to design a very big block of independent counters. It is clear that those counters cannot work concurrently, but it is not a problem in the PLC that by definition works serially. The drawback of hardware-based big counter structure is the flip-flop and connections utilization. So, the idea is to compound those two implementations—in a nutshell, to implement software idea in hardware.

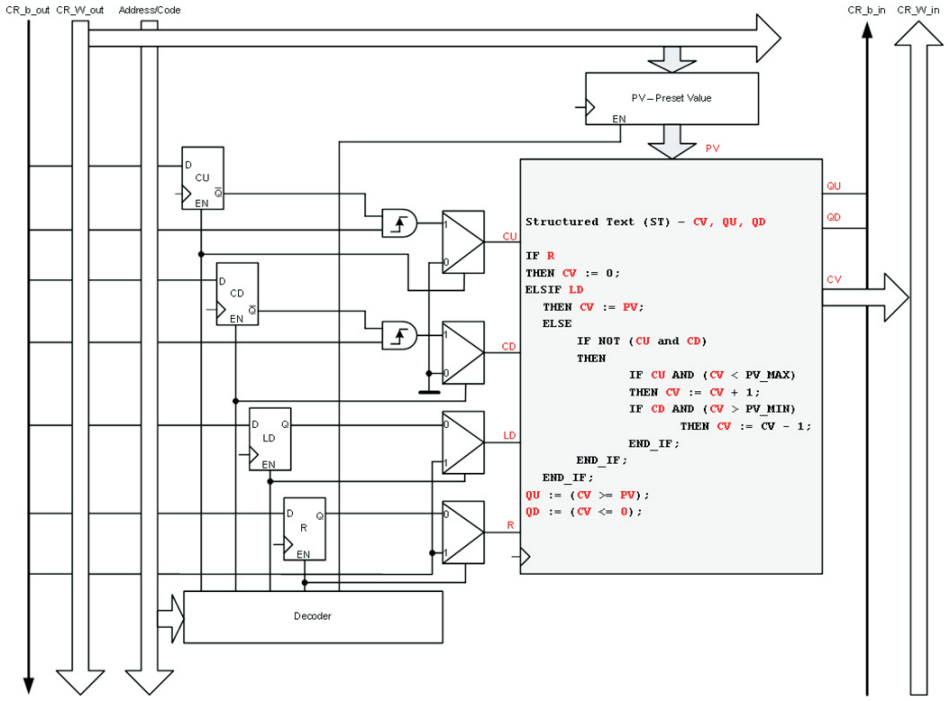


Figure 7. The structure of the hardware-based counter block.

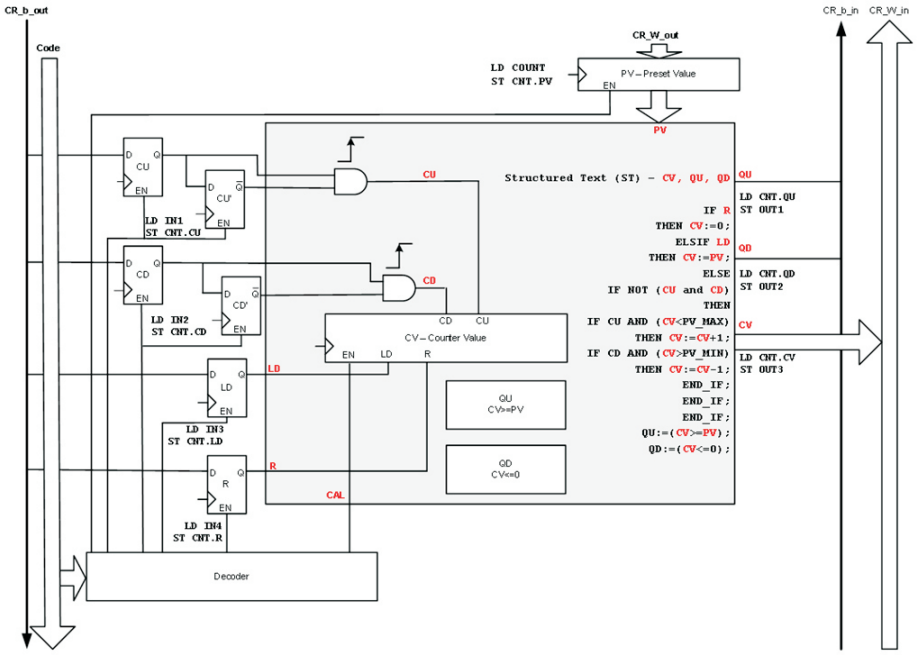


Figure 8. The structure of the hardware-based counter block with CAL.

Listing 5. Verilog description of the counter block with CAL.

```
always @(posedge clk)
  if (reset && ce)
    cv <= 32'b0;
  else
  if(ce)
    if (ld)
      cv <= pv;
    else
    if (!(cu && cd))
      if (cu && (cv < PV_MAX))
        cv <= cv + 1'b1;
      else
        if (cd && (cv > PV_MIN))
          cv <= cv - 1'b1;

assign qu = cv >= pv;
assign qd = cv <= 0;
```

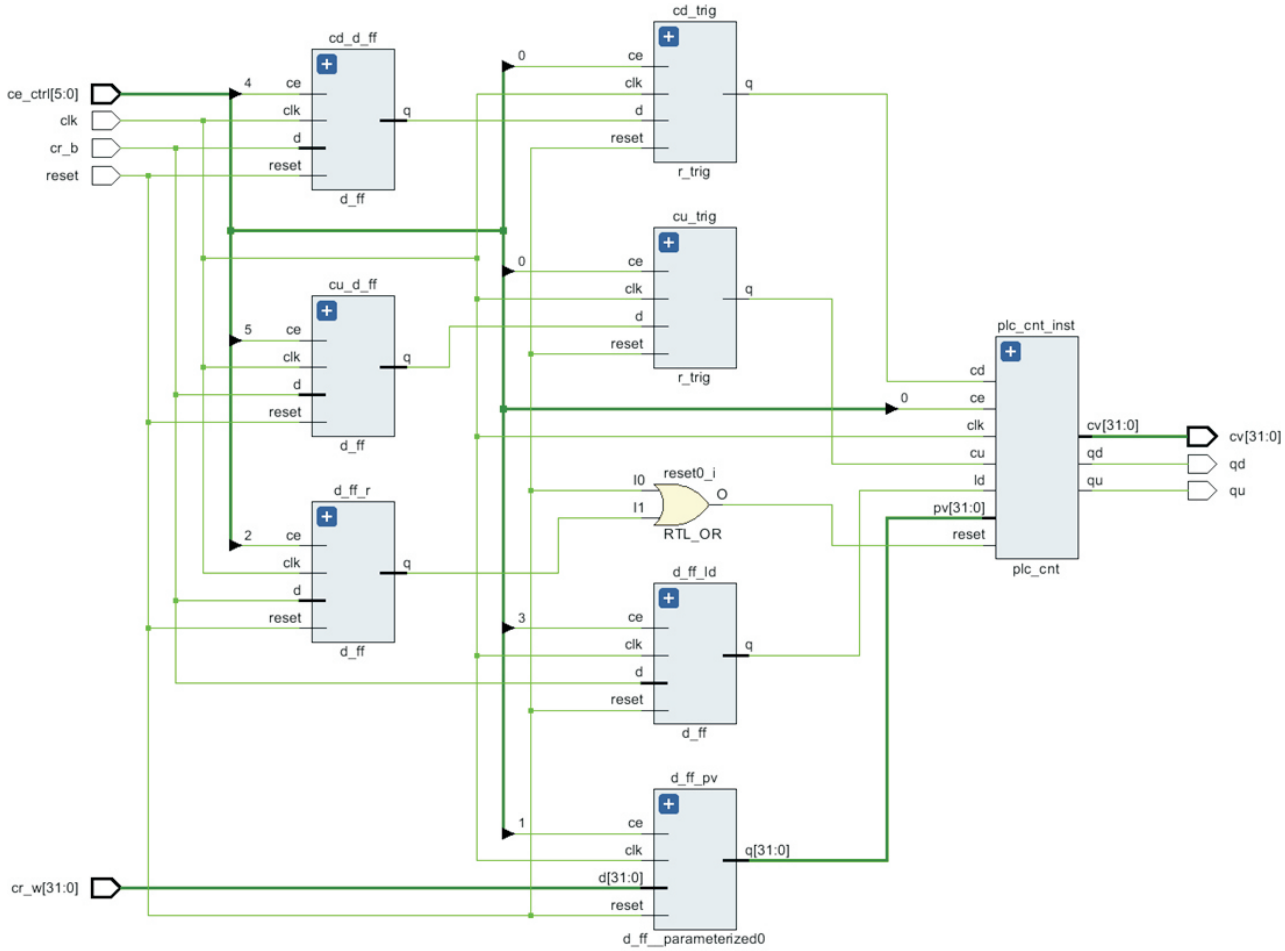


Figure 9. RTL schematic of the proposed counter structure.

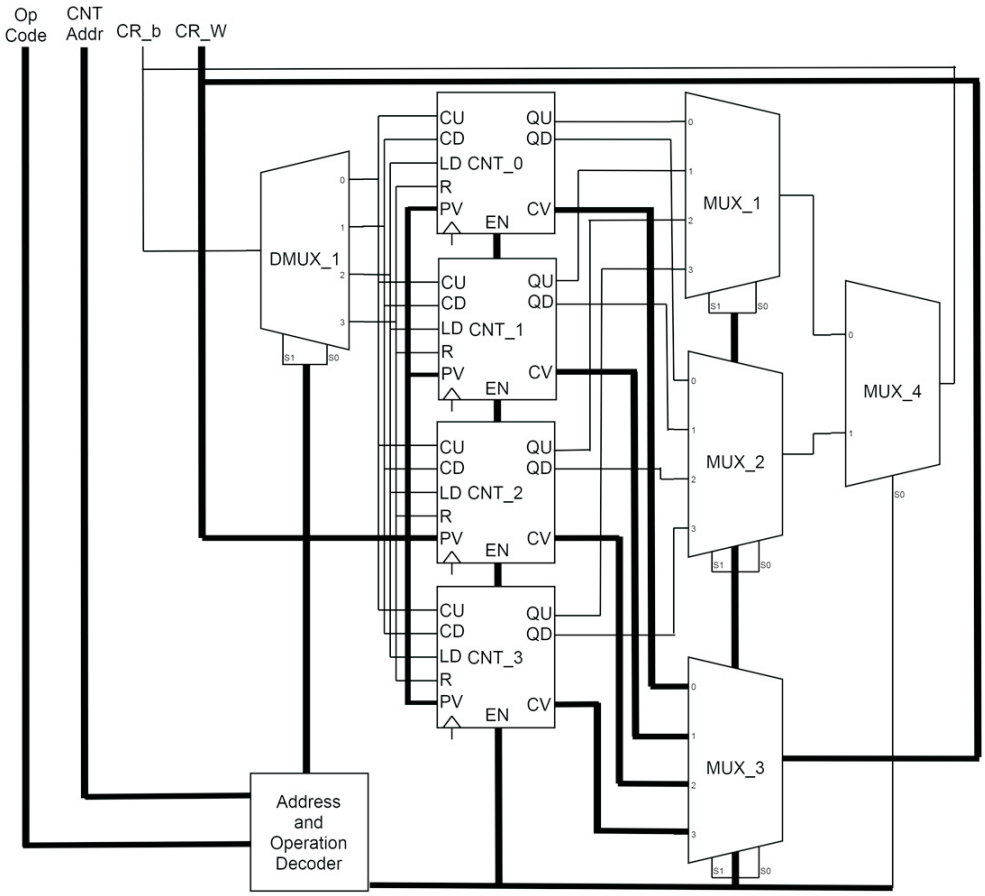
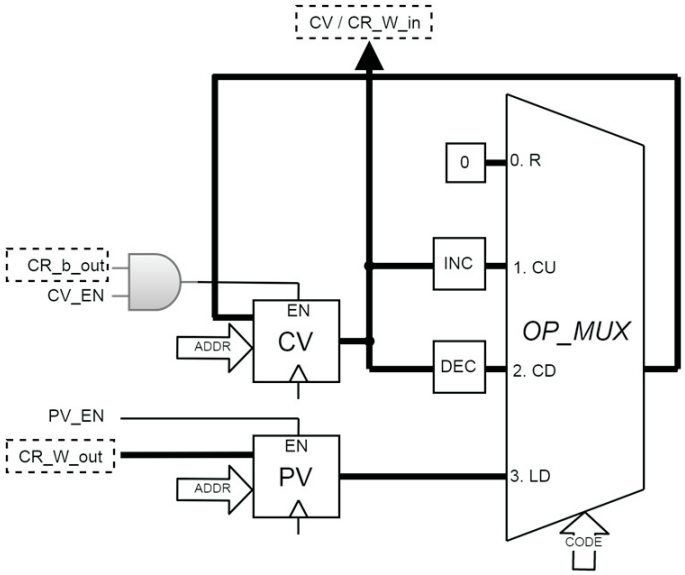


Figure 10. The structure of the counter function block for four counters.

To design software-like counters, memories that implement elements of the structure presented in Figure 2 are necessary. Of course, memories should be driven by means of CRs (CR_b and CR_W), enabling inputs that realize consent to store data to particular memories and common address inputs. However, it would not be good to read data to CR, perform operations and write back data to the memory. It would be time ineffective. The best way is to design MUXes that are driven by decoded instruction. The structure with only Counter Value as a circuit result is presented in Figure 11.

The idea is based on a MUX, which decides what happens to the counter at any given event—increment, decrement, set, reset—depending on which counter instruction is currently being executed. The ENable bits for each instruction are involved in controlling the multiplexer, as well as the signal coming from the CR_b that conditions every change of the CV state. This signal additionally controls the ENable signal of the counter cell. In the first case, such writing is blocked using an additional AND gate.

Similarly, the question can be asked whether it is necessary to continuously calculate the state of the binary outputs of the counter—QU and QD. Note that this may mean the determination of their state during each cycle of the loop, so it takes time. An alternative is to determine the state during the execution of the command testing the binary state of the counter. The third solution requires additional memory, but the QU and QD calculation is executed once during CAL. Calculating the state of the binary outputs requires, of course, performing the comparison operation, which, in case of the existence of the CAL operation, is done during the execution of this operation. A fragment of a program for calculating the binary state of the counter, using the basic operations of the controller, could look as it is presented in Listing 6.



Function	CODE	CV_EN	PV_EN
0 -> CV	00	1	0
CV + 1 -> CV	01	1	0
CV - 1 -> CV	10	1	0
PV -> CV	11	1	0
CR_W_out -> PV	xx	0	1

Figure 11. Software-like counters with conditioning.

Listing 6. Proposed method of Counter outputs generating using standard PLC operations.

```
// QU
LD              CNT.CV
GE              CNT.PV  (GE - Greater or Equal)
ST              CNT.QU
// QD
LD              CNT.CV
LE              0       (LE - Less or Equal)
ST              CNT.QD
```

The implementation of such a program would only require that the computing unit have access to the current state of the counter (CV) and the Preset Value (PV) cell.

The structure of soft-like counters block dedicated for short operators is shown in Figure 12. The most important element of the structure is the CV memory, which stores the current counter state. The multiplexer on the data input of this memory allows us to write the value from the PV memory, to keep the current value of the counter or to write a value modified by count up, count down or reset operation. The second important element of the structure is the PV memory. This memory can be modified only by the state of the CR_W. All these functionalities are implemented by means of OP_MUX. AND gates work as the edge detectors for the CU and CD inputs. In order to detect the rising edge of a variable coming from CR_b, two memories for remembering its previous state must be used. Additionally, two comparators are applied that are used to control the reaching of limit values by the counter—MIN and MAX. These values are constant for each data type of the counter value.

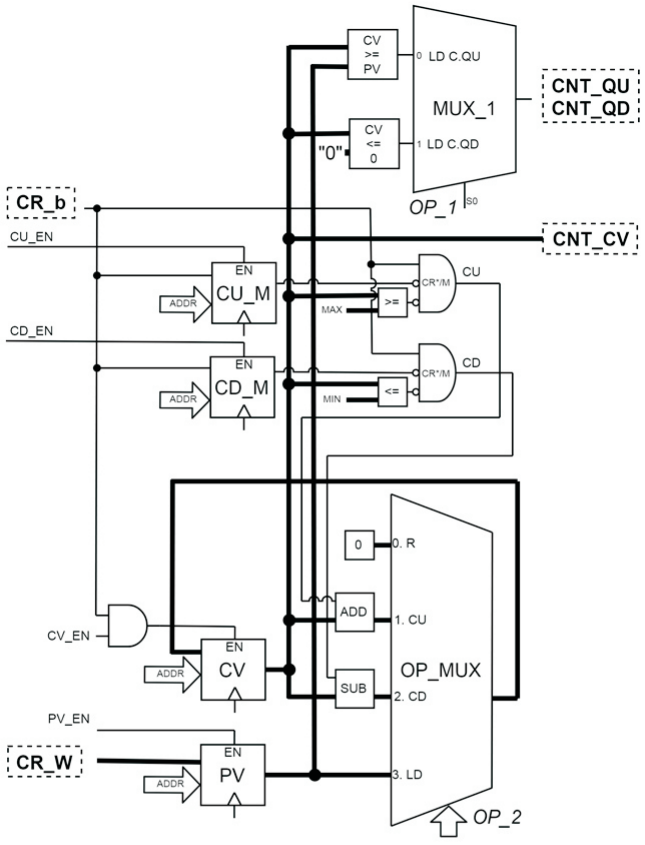
The MUX_1 multiplexer is used to produce the state of the counter's binary outputs—QU and QD. It is implemented in a combinational way, so it is calculated after every input change. The QU and QD outputs are calculated during reading their state. There is no need to implement the memory, but in opposite to microprocessor-based solutions, it takes no clock cycles to elaborate the QU and QD values. The content of CV and PV

memories, for particular address, are compared for QU and CV memory is compared with zero for QD.

However, the idea presented in Figure 12 has the same problem as hardware counter presented in Figure 7—it realizes every instruction like it was an operand executed immediately. This nuance was presented in detail in Section 2. It is necessary to introduce a block that could realize CAL. In such a case, memories form an environment to a call block. The call block (CALL_UNIT) presented in Figure 13 is just a simple combinational process consistent with Listing 5, so it works as a decoder for MUX_1. This CALL_UNIT block takes into account the MIN and MAX values.

As can be seen in Figure 13, edge detection this time requires the use of two memory cells CU and CU' (CD and CD'), because the rising edge detection itself occurs during the execution of the CU (CD) operation, but this result is consumed only at the CALL operation. This requires storing not only the state of the previous CR bit, but also information about current CR_b state.

Presented in Figure 13 construction requires extra elements—CU', CD', R and LD bit memory and CALL_UNIT. The undeniable advantage of this design is that it remains a single-clock-edge design.



Function	CV_EN	PV_EN	CU_EN	CD_EN	OP_1	OP_2
R - 0->CV	1	0	0	0	x	0
LD - PV->CV	1	0	0	0	x	1
CU - CV+1->CV	1	0	1	0	x	2
CD - CV-1->CV	1	0	0	1	x	3
PV - CR_W->PV	0	1	0	0	x	x
QU - QU->CR_b	0	0	0	0	0	x
QD - QD->CR_b	0	0	0	0	1	x

Figure 12. The structure of soft-like counters block dedicated for short operators (CAL not required).

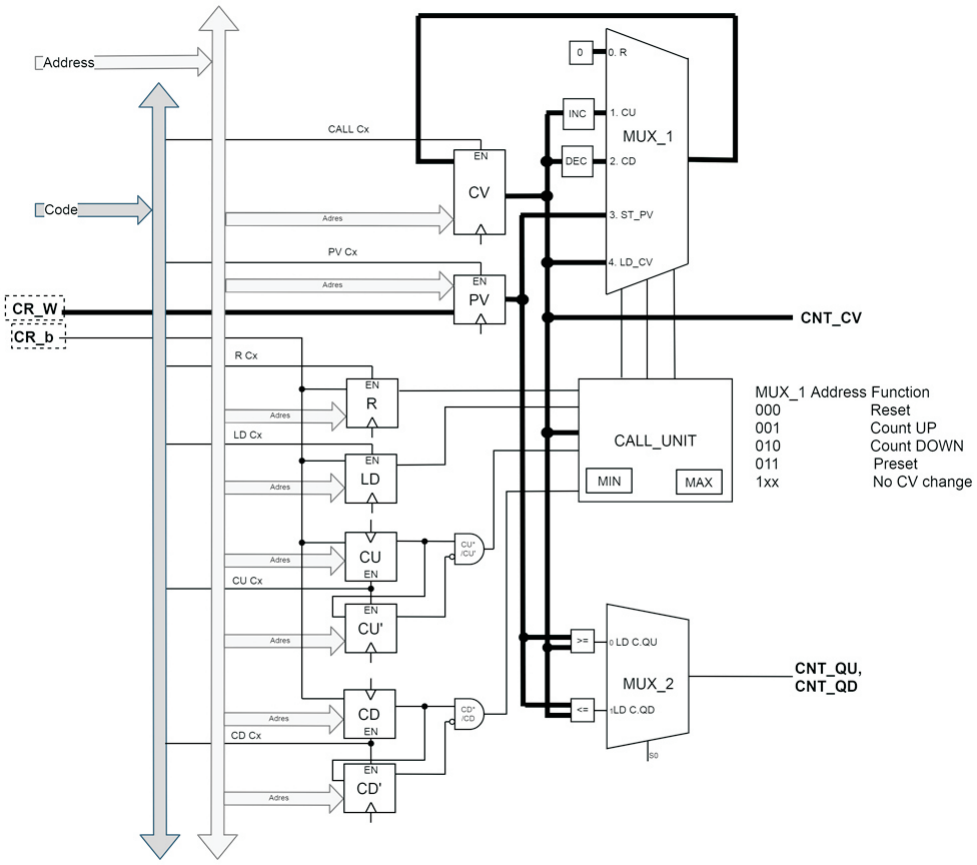


Figure 13. The structure of the soft-like counter block with a CALL_UNIT.

3.4. Summary

Presented in Section 3, the solutions are single clock edge units. This means that the machine cycle is executed within one clock cycle. However, it must be stressed that two kinds of units were presented:

- An operator that executes whole counter (Figures 7 and 12 for hardware and software-like counters, respectively),
- Operators that prepare data in memories, but the counter execution shall take place after the CAL instruction (Figures 8 and 13).

Furthermore, it must be stressed that the first idea prevents the use of a CAL operand as a separate execution moment, so this unit implements only a subset of the features defined in the standard. The second idea enables us to execute a counter after every single counter operator, e.g., CU CNTx, but the compiler must split the operator into two instructions (store + CAL). The idea with separate CAL (Figure 13) is more flexible, but its drawback is that simple operators are realized in two clock cycles.

4. Experimental Results

Units presented in Section 3, that is, hardware and software-like counters, have been implemented using Verilog HDL. Simulations have been run to check for the bugs and to improve the units. After implementation of each solution, a comparison was made between FPGA resources utilization, as well as the determined maximum clock frequencies. Second part of the experiments concerned Siemens PLCs. Different units were tested in order to compare counters in ready-made PLCs with presented in the paper solutions.

The comparison of the logic utilization for hardware and software-like counters is presented in Table 1 and Figure 14. One of the simplest devices of the Xilinx 7th family has been used (xc7a100tcsg324). Synthesis has been done for different numbers of implemented

counters: 2-1024 (address: 1-bit to 10-bit). The versions with CAL are implemented (hardware—Figure 8; software-like—Figure 13).

Table 1. The comparison of the logic utilization for hardware and software-like counters.

No. of Counters	Hardware Counters			Software-Like Counters		
	LUT	LUTRAM	FF	LUT	LUTRAM	FF
2	169	0	140	184	70	0
16	1321	0	1120	184	70	0
64	5407	0	4480	184	70	0
128	10,919	0	8960	254	140	0
256	21,600	0	17,920	394	280	0
512	43,298	0	35,840	760	560	0
1024	86,687	0	71,680	1330	1120	0

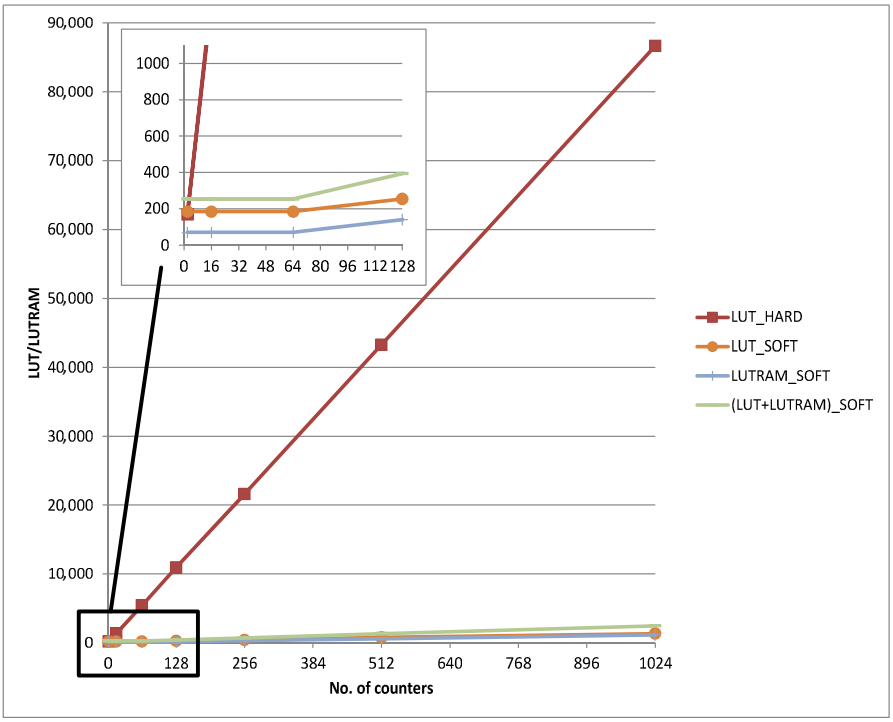


Figure 14. The comparison of the logic utilization for hardware and software-like counters.

As can be seen in Figure 14 the number of LUTs utilized for hardware counters increases linearly (80 LUTs/CNT) with the number of counters. Furthermore, the number of FFs also increases linearly (Table 1; 70 FFs/CNT)—this is a serious disadvantage of this solution. This solution quickly utilizes huge amounts of resources. However, hardware counters do not use LUTRAMs. Software-like counters are completely different after their implementation. The execution unit of the counter requires about 184 LUTs and 70 LUTRAMs, but each counter instance increases the number of LUTs by approximately 1.1, and the number of LUTRAMs increases by 1 LUTRAM per counter. Their functionality is based on memory blocks, so LUTRAMs are widely utilized. Moreover, LUTs and LUTRAMs used in FPGA are 6-input tables, so the utilized resources are constant for the number of counters not exceeding 64. The occupancy of elements of the structure is in this case smaller and increases much slower than for the hard structure.

To fully compare hardware counters with software-like ones, the timing analysis is necessary. The maximum clock frequency (in MHz) of those solutions is presented in Figure 15.

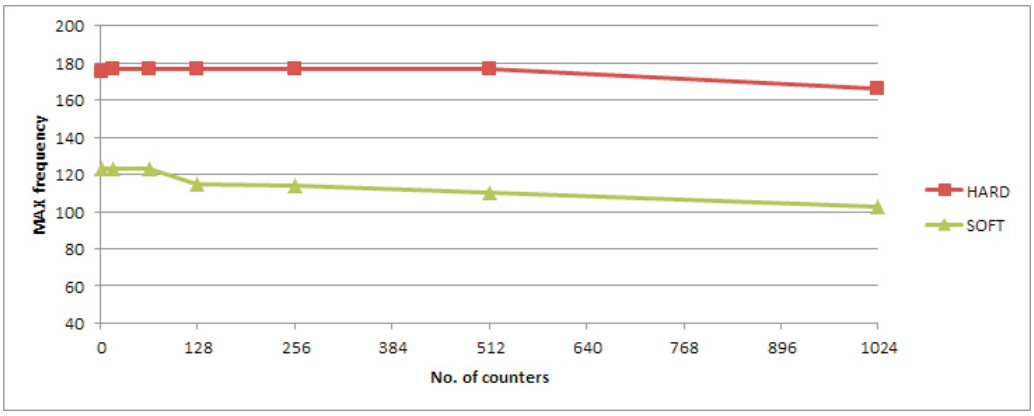


Figure 15. The time analysis for hardware and software-like counters ([MHz]).

As can be seen for software-like counters, the frequency is always lower and decreases faster with increasing the number of instances of the counters. Hardware counters are much faster, even for the big structures.

Summarizing the results for the utilization and time analysis, it can be concluded that for a small number of counters required in a given application, they can be implemented in a hardware way, while for a large number of counters, a software-like solution should be used.

In order to prove full functionality of the idea presented in the paper, simple CPU units with basic instructions were implemented together with counters as presented in Figure 6. Synthesis was carried out for xcvu5p-flvc2104-1-i (Virtex UtraScale+). Results are in Table 2—it considers the conclusion that hardware counters are suggested for small numbers of instances while software-like ones are dedicated for a large number of counters instances. The xcvu5p FPGA is much more powerful than xc7a100t, but in this application, the difference manifests itself mainly in maximum frequency.

Table 2. Result for CPU with counters.

No of Counters		CPU + Hard CNTs	CPU + Soft CNTs
2	LUT/FF/LUTRAM/BRAM	300/179/33/0.5	-
	fmax [MHz]	302	-
16	LUT/FF/LUTRAM/BRAM	1481/1159/33/0.5	-
	fmax [MHz]	300	-
64	LUT/FF/LUTRAM/BRAM	5384/4523/33/0.5	338/39/103/0.5
	fmax [MHz]	298	230
128	LUT/FF/LUTRAM/BRAM	-	408/39/173/0.5
	fmax [MHz]	-	224
256	LUT/FF/LUTRAM/BRAM	-	548/39/313/0.5
	fmax [MHz]	-	218
512	LUT/FF/LUTRAM/BRAM	-	828/39/593/0.5
	fmax [MHz]	-	216
1024	LUT/FF/LUTRAM/BRAM	-	1467/39/1153/0.5
	fmax [MHz]	-	216

Units with simple operators (Figures 7 and 12) were also implemented in xc7a100t FPGA. The direct comparison for 16, 256 and 1024 counter instances is presented in Table 3. In terms of both speed and utilized logic, there are no drastic differences between units for simple operators and with CAL instruction. For hardware units, it is even more advantageous to implement a unit with CAL in terms of speed. This is probably due to the 'cutting' of combinational circuits with registers, which is good for critical paths. In a soft-like unit, more memory is implemented for the CAL unit, which in turn adversely affects the operating frequency.

Table 3. Direct comparison of designed counter block.

Counters	16 LUT/FF/LUTRAM	16 fmax [MHz]	256 LUT/FF/LUTRAM	256 fmax [MHz]	1024 LUT/FF/LUTRAM	1024 fmax [MHz]
Hard CNTs (operators)	1401/1056/0	164	22,097/16,896/0	145	89,364/67,584/0	150
Hard CNTs (CAL)	1321/1120/0	177	21,600/17,920/0	177	86,687/71,680/0	166
Soft-like CNTs (operators)	198/0/66	130	396/0/264	119	1271/0/1056	108
Soft-like CNTs (CAL)	184/0/70	123	394/0/280	114	1330/0/1120	103

A comparison of the time required for the CPU to process all the operations associated with the counter has also been made. There are 16 such instructions. For solutions with the CAL operator, the execution of this instruction must be added in addition, while for solutions with short operators, the CAL instruction must be added to each instruction that affects the counter, that is: CU, CD, R, and PV/LD R. Depending on how the state of the QU and QD outputs are working out, these executions may also be required for operations of reading these states. For the presented CPU, the execution of each instruction requires one clock cycle, so determining the time to operate the entire counter is to multiply this number by the execution time of a single instruction. The situation is different for the Siemens solutions presented for comparison. Such a comparison for a few selected Siemens solutions is presented in Table 4. The table shows the measured processing time for the program using all capabilities of the counter, as well as using its individual fragments.

There is huge disproportion for the older controller S7-315 between the implementation of the counter operations as a whole for the classical solution and the standard compliant (Table 4). This disproportion has been removed in newer solutions. It can also be seen that the structure proposed by the authors significantly exceeds the time performance of all units taken for comparison.

However, due to the different possible solutions and the fact that the vendor PLCs perform even relatively more tasks than the presented solution, it was decided to make another, seemingly more interesting comparison. For example, in the S7-315 controller, the edge detection operations take three times more than basic instruction and these operations for the S7-319 take almost four times more time to execute. The operation of loading the counter status into CR for this controller takes more than thirty times longer than the basic instruction! In the authors' proposed solution, these operations take the same amount of time as the basic instructions. The execution times of individual counter tasks were compared with respect to the duration of the instruction that seems to be basic for every CPU controller—the instruction for reading the status of the contents of the PLC's memory cell M (Marker, Memory) was taken into account. Now it is much clearer that the implementation of counter operations proposed by the authors in terms of time is much more efficient than the one occurring in Siemens PLCs, regardless of whether counters are 16-bit or 32-bit, and whether in the classic version (not compliant with the provisions of the Standard), or in the version compliant with the Standard (most closely to short operators' ones). In the best case, for the S7-315 controller and the counter not conforming to the Standard the difference reaches more than four times, while for the counter conforming to the Standard and the S7-1516 controller, the difference is more than five times. These differences are due to the fact that the counter operations, for the proposed units, thanks to the hardware support, are implemented at the same time, or rather should be said, the same number of clock periods, as the basic instructions (such as reading or writing a memory cell).

Table 4. Comparison times of counters operators' execution times for Simatic Controllers.

	S7-319 Classical Counter	S7-319 IEC Counter 16 bit	S7-315 Classical Counter	S7-315 IEC Counter 16 bit	S7-1214C IEC Counter 32 bit	S7-1516 Classical Counter	S7-1516 IEC Counter 32 bit
Full Counter	957/106	1228/136	7700/59	26,730/206	7050/176	720/65	932/85
Only CU Input	137/15	1018/12	1050/8	19,630/151	1430/36	100/9	162/15
Only CD Input	133/15	40 */4.5 *	1140/9	1000 */7.5 *	970 */24 *	90/8	120 */11 *
Preset Counter	197/22	80 */9 *	1350/10	2200 */17 *	1800 */45 *	100/9	147 */13 *
Reset Counter	114/13	80 */9 *	790/6	1500 */11.5 *	1240 */31 *	100/9	143 */13 *
Read QU	N.a.	40 */4.5 *	N.a.	900 */7 *	410 */10 *	N.a.	120 */11 *
Read QD	41/4.5	30 */3 *	510/4	500 */4 *	200 */5 *	70/6	130 */12 *
Read CV	114/13	20 */2 *	1050/8	1000 */7.5 *	1000 */25 *	70/6	110 */10 *
Basic instr.: load memory bit	9	9	130	130	40	11	11

A/B—A number is a time of execution in ns, and the B is a quotient of this time and the time taken from the last row. * Calculated as increase of the block execution time compared to values in row "Only CU Input".

The comparison of execution times for proposed in the paper units is presented in Table 5.

Table 5. Comparison of execution times for proposed in the paper units.

	Hard Figure 7 (OPs)	Hard Figure 8 (one CAL)	Hard Figure 8 (OP + CAL)	Soft-Like Figure 12 (OP)	Soft-Like Figure 13 (one CAL)	Soft-Like Figure 13 (OP + CAL)
Frequency [MHz]	300	300	300	200	200	200
Full Counter	53.28/16	56.61/17	113.4/32	80/16	85/17	160/32
Only CU Input	6.67/2	10/3	10/3	10/2	10/2	10/2
Only CD Input	6.67/2	10/3	10/3	10/2	10/2	10/2
Preset Counter	6.67/2	10/3	10/3	10/2	10/2	10/2
Reset Counter	6.67/2	10/3	10/3	10/2	10/2	10/2
Read QU	6.67/2	6.67/2	6.67/2	10/2	10/2	10/2
Read QD	6.67/2	6.67/2	6.67/2	10/2	10/2	10/2
Read CV	6.67/2	6.67/2	6.67/2	10/2	10/2	10/2
Basic instr.: load memory bit	3.33	3.33	3.33	5	5	5

A/B—A number is a time of execution in ns, and the B is a quotient of this time and the time taken from the last row.

Most often, control programs do not use all the functionality of counters. An example of a program that only realizes counting up events (CU), allows clearing the counter state (R), and also reads its binary output (QU)—indicating that the setpoint (PV) has been reached—is presented in Table 6.

Table 6. Example program execution times.

	Hard Figure 7 (OPs)	Soft-Like Figure 8 (OPs)	Hard Figure 12 (one CAL)	Soft-Like Figure 13 (one CAL)	Hard Figure 12 (OP + CAL)	SOft-Like Figure 13 (OP + CAL)
Frequency [MHz]	300	200	300	200	300	200
Basic instruction	3.33	5	3.33	5	3.33	5
Execution time [ns]	26.67	40	30	45	36.65/30	55/45
Program	Listing 7a		Listing 7b		Listing 7c/Listing 7d	

As can be seen from Table 6, the program executed with the use of operators for the circuits of Figures 7 and 8 has the best time performance. It is also worth noting that in the case of solutions from Figures 12 and 13, only one operator command containing the default CAL instruction can be used, while the remaining commands can be as in the solution from the middle columns. The execution time of the program so written will then be equal to the execution time of the program from the middle column.

Listing 7. Example programs.

```
(a)
LD   INP_UP
CU   CNT
LD   INP_RES
R    CNT
LD   17
PV   CNT
LD   CNT.QU
ST~OUT_UP

(b)
LD   INP_UP
ST   CNT.CU
LD   INP_RES
ST   CNT.R
LD   17
PV   CNT.PV
CAL  CNT
LD   CNT.QU
ST~OUT_UP

(c)
LD   INP_UP
CU   CNT (ST+CAL)
LD   INP_RES
R    CNT (ST+CAL)
LD   17
PV   CNT (ST+CAL)
LD   CNT.QU
ST~OUT_UP

(d)
LD   INP_UP
ST   CNT.CU
LD   INP_RES
ST    CNT.R
LD   17
PV   CNT (+CAL)
LD   CNT.QU
ST   OUT_UP
```

The above example uses only some of the functionalities of the counter, as it is often used in practice. The time needed to execute the above functionality for the S7-319 controller is, respectively: 1328 ns for counters compliant with the standard and 415 ns for counters realized in the Simatic standard. The S7-1516 unit—from the family of the newest and the most efficient units—need to execute the program, respectively, in: 405 ns for counters compliant with the standard and 250 ns for Simatic standard counters. So the fastest PLC, using a 16-bit classic counter, runs the presented program about 5 times longer than the slowest unit presented in this paper. In the case of the fastest unit, presented in the most left-hand side column of Table 6, this disproportion is twice as large.

5. Conclusions

This paper presents the idea of hardware support for IEC61131-3 compliant counter operations. The developed concept can be used in practice, whereby the counter block can work with a dedicated microprocessor/microcontroller or a general-purpose unit. The minimalistic interface between the CPU and the counter block allows building PLCs with very low latency for counter operations. The concept of bit.WORD integrated hardware/-software controller is presented. One part of the microprocessor performs operations on BOOL-type variables only, while the other part enables operations on WORD-type (32-bit) variables. Such a concept enables some parallelism of the performed operations that are executed concurrently, e.g., bit and word data could be written to counter block concurrently.

The presented concepts have been described in Verilog HDL language and implemented in an FPGA device with a dedicated software core CPU performing basic operations.

Two types of counters are presented. The first one is a classical hardware counter built on the basis of flip-flops. The second type, software-like, is also hardware-based. However, it draws on ideas of software implemented counters in its design. It is based on memory cells on which operations are performed. Both structures are very fast and make it possible to execute the counter update operation in a single clock cycle. Nevertheless, hardware counters provide faster structures. However, they utilize flip-flops, making these structures effective for small counter blocks. Hardware counters with some small modifications can work as high-speed counters which count events regardless of the program [25]. Software-like counters are a slightly slower design (in the range of max frequency), but they are based on memory-dedicated FPGA structures. This makes even a counter block for 1024 counters not a big problem. Experimental results show the validity of the adopted solutions.

Changes accompanying the fourth industrial revolution, Industry 4.0, such as large-scale communication, increased automation, improved communication, self-monitoring, and production of smart machines, have led to the need for increased performance of programmable logic controllers. The Integrated Hardware-Software PLC idea is especially suitable for that purpose. First, the control loop is relatively short in relation to classical solutions. Second, bit.WORD units can work concurrently and in the future, support for "out of order" technology is planned. Third, different communication modules can be easily integrated with the PLC core, so the integration in digital communication networks is possible.

Author Contributions: Conceptualization, M.C. and R.C.; methodology, M.C., R.C. and A.M.; software, M.C., R.C. and A.M.; hardware, R.C.; validation, M.C., R.C. and A.M.; formal analysis, M.C.; investigation, M.C., R.C. and A.M.; writing—original draft preparation, M.C. and R.C.; writing—review and editing, M.C., R.C. and A.M.; supervision, R.C. and M.C. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by the Polish Ministry of Science and Higher Education funding for statutory activities.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

ALU	Arithmetic-Logic Unit
CD	Count Down
CE	Clock Enable
CPU	Central Processing Unit
CR	Current Result (accumulator)
CR_b	Current Result: bit
CR_W	Current Result: Word
CU	Count Up
CV	counter value
FF	Flip-Flop
FPGA	Field Programmable Gate Array

IHSPLC	Integrated Hardware-Software PLC
IL	Instruction List
LD	Load preset value
LUT	Look-Up Table
LUTRAM	Look-Up Table Random Access Memory
PII	Process Image Input
PIQ	Process Image Output
PLC	Programmable Logic Controller
PV	Preset Value
R	Reset counter
RTL	Register Transfer Level
ST	Structured text
QU	Output Up
QD	Output Down

References

1. International Electrotechnical Commission. *EN 61131-3:2013, Programmable Controller—Part 3: Programming Languages*; Technical report; International Electrotechnical Commission: Geneva, Switzerland, 2013.
2. John, K.; Tiegelkamp, M. *IEC 61131-3: Programming Industrial Automation Systems*; Springer: Berlin/Heidelberg, Germany, 2010.
3. de Sousa, M. Proposed corrections to the IEC 61131-3 standard. *Comput. Stand. Interfaces* **2010**, *32*, 312–320. [CrossRef]
4. Plaza, I.; Medrano, C.; Blesa, A. Analysis and implementation of the IEC 61131-3 software model under POSIX real-time operating systems. *Microprocess. Microsyst.* **2006**, *30*, 497–508. [CrossRef]
5. Chmiel, M.; Mocha, J.; Hrynkiewicz, E.; Polok, D. About Implementation of IEC 61131-3 IL Function Blocks in Standard Microcontrollers. *Int. J. Electron. Telecommun.* **2014**, *60*, 41–46. [CrossRef]
6. de Sousa, M. Data-type checking of IEC61131-3 ST and IL applications. In Proceedings of the 2012 IEEE 17th Conference on Emerging Technologies & Factory Automation (ETFA), Krakow, Poland, 17–21 September 2012; pp. 1–8.
7. Carrillo, S.; Polo, A.; Esmeral, M. Design and Implementation of an Embedded Microprocessor Compatible With IL Language in Accordance to the Norm IEC 61131-3. In Proceedings of the IEEE International Conference on Reconfigurable Computing and FPGAs. ReConFig 2005, Cancun, Mexico, 9–11 December 2011; pp. 28–30.
8. Okabe, M. Development of processor directly executing IEC 61131-3 language. In *SICE Annual Conference*; The University of Electro-Communications: Tokyo, Japan, 2008; pp. 2215–2218.
9. Rudrawar, S.; Patil, M. Design Furthermore, Implementation of FPGA Based High Performance Instruction List (IL) Processor. *IOSR J. Electron. Commun. Eng. (IOSRJECE)* **2012**, *1*, 38–45. [CrossRef]
10. Hajduk, Z.; Trybus, B.; Sadolewski, J. Architecture of FPGA embedded multiprocessor programmable controller. *IEEE Trans. Ind. Electron.* **2015**, *62*, 2952–2961. [CrossRef]
11. Mazur, P.; Czerwinski, R.; Chmiel, M. PLC implementation in the form of a System-on-a-Chip. *Bull. Pol. Acad. Sci. Tech. Sci.* **2020**, *68*, 1263–1273.
12. Monmasson, E.; Idkhajine, L.; Cirstea, M.; Bahri, I.; Tisan, A.; Naouar, M. FPGAs in industrial control applications. *IEEE Trans. Ind. Inform.* **2011**, *7*, 224–243. [CrossRef]
13. Ichikawa, S.; Akinaka, M.; Hata, H.; Ikeda, R.; Yamamoto, H. An FPGA implementation of hard-wired sequence control system based on PLC software. *IEEJ Trans. Electr. Electron. Eng.* **2011**, *6*, 367–375. [CrossRef]
14. Milik, A.; Hrynkiewicz, E. Synthesis and implementation of reconfigurable PLC on FPGA platform. *Int. J. Electron. Telecommun.* **2012**, *58*, 85–94. [CrossRef]
15. Mocha, J.; Kania, D. Hardware Implementation of a Control Program in FPGA Structures. *Electr. Rev.* **2012**, *88*, 95–100.
16. 3S-Smart Software Solutions GmbH. In *User Manual for PLC Programming with CoDeSys 2.3*; 3S-Smart Software Solutions GmbH: Kempten, Germany, 2010.
17. Schneider Electric. In *Concept 2.6 User Manual*; Schneider Electric: Rueil-Malmaison, France, 2010.
18. Siemens AG. *SIMATIC S7 S7-1200 Programmable Controller System Manual*; Siemens AG: Nurnberg, Germany, 2019.
19. ISaGRAF. ISaGRAF, Software Release 5.2, User Guide, 2009. Available online: https://docplayer.net/21858762-Isagraf-getting-started-software-release-5-2.html (accessed on 20 September 2021).
20. Chmiel, M. FPGA-based implementation of bistable function blocks defined in the IEC 61131. *Microprocess. Microsyst.* **2019**, *65*, 37–46. [CrossRef]
21. Czerwinski, R.; Chmiel, M. Hardware-Based Single-Clock-Cycle Edge Detector for a PLC Central Processing Unit. *Electronics* **2019**, *8*, 1529. [CrossRef]
22. Chmiel, M.; Hrynkiewicz, E. Concurrent Operation of the Processors in Bit-Byte CPU of a PLC. *Control Cybern.* **2010**, *39*, 559–579. [CrossRef]
23. Chmiel, M.; Kloska, W.; Mocha, J.; Polok, D. FPGA-based two-processor CPU for PLC. In Proceedings of the International Conference on Signals and Electronic Systems (ICSES16), Kraków, Poland, 5–7 September 2016; pp. 247–252.

24. Chmiel, M.; Mocha, J.; Lech, A. Implementation of a Two-Processor CPU for a Programmable Logic Controller Designed on FPGA Chip. In Proceedings of the International Conference on Signals and Electronic Systems (ICSES18), Cracow, Poland, 10–12 September 2018; pp. 13–18.
25. Siemens. *Application Examples for High-Speed Counters (HSC), Application Example*; Siemens AG: Nurnberg, Germany, 2016.