Journal of Electronics and Electrical Engineering
https://ojs.wiserpub.com/index.php/JEEE/

Article

Modeling and Implementation of a Specific Microprocessor to Enhance the Performance of PLCs Employing FPGAs

Marcelo Delgado-del-Carpio[1], **A. Hilario-Tacuri**[1] **and Carlos A. Hernández-Gutiérrez**[2,*]

[1]Department of Electronic Engineering, National University of San Agustin of Arequipa, Arequipa, Peru
[2]National Technological Institute of Mexico, Tuxtla Gutierrez, México
E-mail: carlos.hg@tuxtla.tecnm.mx

Received: 14 July 2023; **Revised:** 19 September 2023; **Accepted:** 20 September 2023

Abstract: This work presents the design of a microprocessor synthesized in FPGA based on the IEC 61131-3 standard. We report the architecture, and the operation of the internal hardware which allows the execution of Instruction List (IL). One of the most important components is the operands_selector block which allows memory elements, inputs, or outputs of the microprocessor to be treated as operands. Two ALUs are available to perform a bit, integer, and floating-point operations. The implementation of this microprocessor allows concluding that our designed microprocessor implemented in xc7a100tcsg324 (Xilinx) or EP4CE10E22C8 (Intel) FPGAs is superior in their execution times compared to the microprocessor evaluated in early studies and to one of the S7-1500 family processor.

Keywords: microprocessor, FPGA, PLC, IEC 61131-3

1. Introduction

The present paper addresses the problem of execution times in microprocessors for PLCs (Programmable Logic Controllers), which have resulted in a revolution of control engineering, being used for a range of automation tasks [1–3] in areas such as industrial processes in manufacturing [4]. Thus, there are some innovations and improvements in microprocessor technology and software programming techniques that have added more features and capabilities to the PLC, enabling it to perform more complex control applications with greater speed [5]. The cumulative time of analog and digital signal scanning, execution of ladder logic program, and storage of outputs in memory is the scan time. So, with the proposed system the program execution time can be significantly reduced. The IEC 61131-3 is the PLC standard, which defines some programming languages for these devices. Moreover, there are manufacturers like Schneider Electric, Siemens, Toshiba, Mitsubishi Electric, Rockwell Automation, among others that usually do not share technology specifications about how they implement the IEC 61131-3 protocol in the PLCs. Likewise, the use of FPGAs (Field Programmable Gate Arrays) for the design of a PLC is a key field to enhance the state of the art of PLCs. Some approaches have been proposed the use of code conversions [6–10] resembling the work of a compiler, which starts from a standardized language in IEC 61131-3 to obtain a code in hardware description languages such as VHDL or Verilog. Another interesting approach is to build an IEC 61131-3 based microprocessor as was proposed by Chmiel et al. and others [11–16]. PLCs have been built based on general-purpose microprocessors or microcontrollers [17–19]. To work with programming languages like Instruction List (IL) defined in IEC61131-3 standard, a compiler is required. However, the instruction sets do not match with the standard, resulting in longer operations execution time [20].

While many modern systems employ faster-pipelined microprocessors, our design diverges from this approach, requiring four clock cycles to execute an instruction. However, for PLC technologies, the design remains appropriately suited. Throughout this study, we will demonstrate the capability of our microprocessor to address real-world industrial challenges, particularly in managing a complex floating-point PID control. So, this research centers on the development of a specific-purpose microprocessor, designed to significantly enhance response times. With an architecture primed for direct IL instruction execution, we've leveraged the dual capabilities of both Xilinx and Intel FPGAs. Distinctly departing from prior works such as Chmiel's [11], our study is neither a mere extension nor limited to previously trodden methodologies. Introducing innovative elements like the "Operands selector" and specialized memory blocks (M0_X and M1_X), we've sought to redefine benchmarks in FPGA-based microprocessor designs. Our arithmetic and simulation approaches have undergone a revamp, ensuring alignment with the latest technological standards. Currently, our research has matured to a stage where detailed simulations have been conducted, benchmarking results have been established, and our design's robustness has been tested across multiple platforms.

2. System Description

The proposed PLC microprocessor is a Harvard-modified architecture, inspired by [11], which is designed to execute the instructions of the IL language directly by the designed hardware. So, the set of instructions can perform the following operations:

- Bitwise operations for bit/double word data
- Shift and rotation for double word data
- Arithmetic operations
- Jump

The general block diagram of the system is presented in Figure 1. The key block is the instruction decoder connected directly or indirectly to all blocks of the system since it is responsible for establishing each signal at the indicated time according to the instruction cycle. In addition, the program counter with 8 input bits that come from the first 8 bits of the instruction code, this value is loaded when WR_PC is activated during a jump instruction. On the other hand, we have two important feedbacks, the first one is between the ALUs (Arithmetic Logic Units) and the dual-port RAM (Random Access Memory), this provides the second operand B (the current result), which is a bit or a 32–bit length word. The second feedback is given between the ALUs and the operands selector, this feature allows the microprocessor to set the result to the outputs or store them on dedicated memory banks M0_X and M1_X.

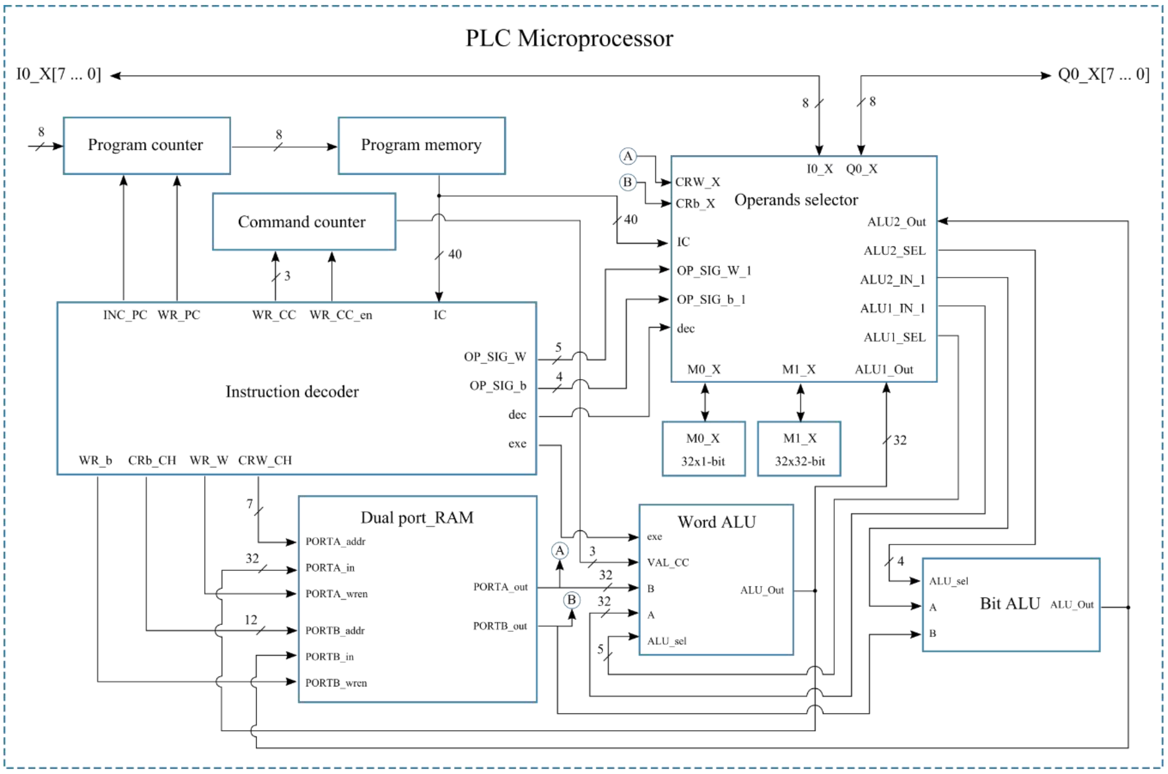


Figure 1. Microprocessor general block diagram.

2.1 *Instruction Decoder*

In the following lines, we will explain the structure of the instruction decoder (Figure 2) and its details. The command counter selector is responsible for determining how long the execution state will last. This depends on the propagation delays of each operation. The output WR_CC represents the number of clock cycles required for the execution, each clock cycle equals to 10ns and 20ns for the Xilinx and Intel FPGAs used respectively. The values of WR_CC corresponding to each instruction are listed in Table 1 under the "Clock cycles" column of ALU. The ALU_1 and ALU_2 operation selectors are responsible for determining the operation to be executed. A multiplexer is used to select the outputs of the internal blocks described above, based on the 37 to 32 bits of the instruction code IC, which is also known as the operation code. Finally, the WR_PC enable block functions as a comparator and only triggers the WR_PC when the operation code corresponds to a JMP instruction, this sets the program counter by the instruction code suffix.

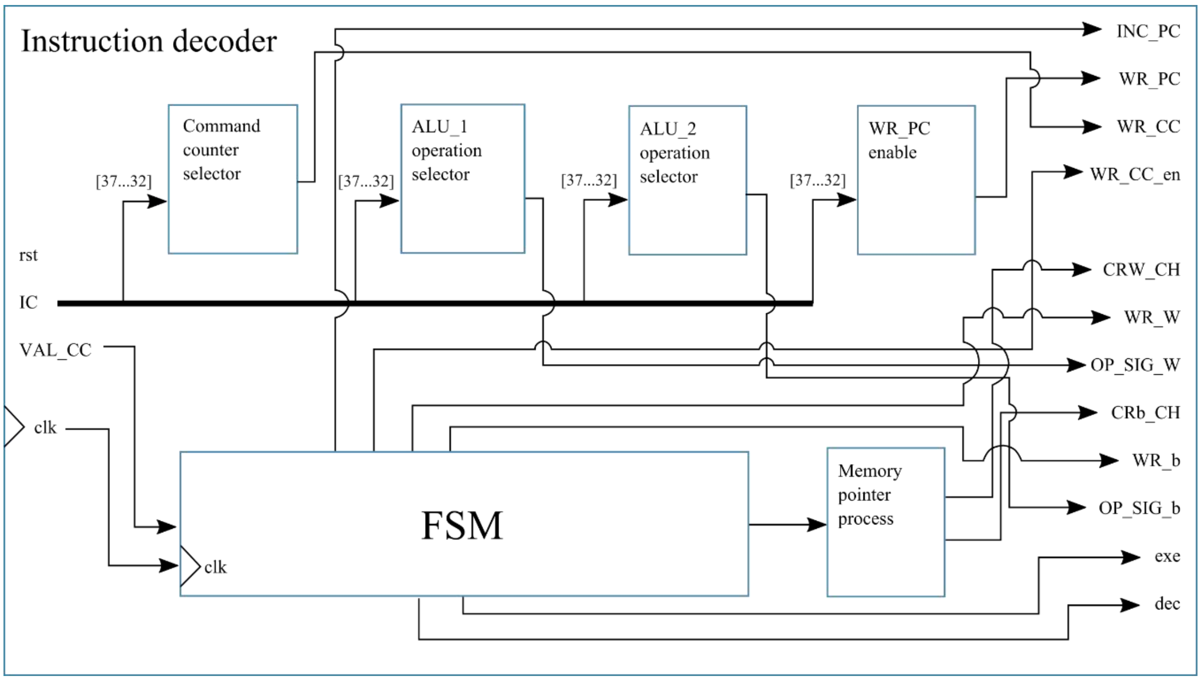


Figure 2. Instruction decoder structure.

The FSM (Finite State Machine) performs the instruction cycle (see Figure 3), all the states are explained in the following lines.

1) Initialization: The write operation enabler (WR_CC_en) is set in the command counter, which determines the duration of the execution state of each instruction. During initialization the signals INC_PC, WR_W, and WR_b are disabled. Furthermore, these signals control the increment of the program counter, the 32-bit, and 1-bit words respectively. In addition, the memory pointers from CRW and CRb stacks are incremented.
2) Decoding: The input values for the ALUs, memory banks (M0_X and M1_X), and the output values are set depending on the instruction code, this process is accomplished by the operands_selector block.
3) Execution: In this process, the instruction decoder waits until ALUs outputs are established, and the wait time is controlled by VAL_CC (while the exe output is set). Finally, when the control signal becomes to zero the execution state finish.
4) Instruction fetch: The microprocessor looks for the next instruction by increasing the program counter and enabling the write operation in CRW and CRb stacks with the WR_W and WR_b outputs.

Table 1. Comparison of execution times between Micro-XPLC, Micro-IPLC, CPU [11] and Siemens S7-1500

Instruction	Micro-XPLC				Micro-IPLC				CPU [11]	Siemens 1511(T)(F)-1 PN 1511C-1 PN	Siemens 1518(T)(F)-4 PN/DP 1518(F)-4 PN/DP MFP
	ALU		Complete Instruction		ALU		Complete Instruction				
	Clock cycles	Time (ns)	Clock cycles	Time (ns)	Clock cycles	Time (ns)	Clock cycles	Time (ns)	Time (ns)		
ADD_I	2	20	5	50	1	20	4	80	80	96	2
SUB_I	1	10	4	40	1	20	4	80	80	96	2
MUL_I	1	10	4	40	1	20	4	80	80	96	2
DIV_I	8	80	11	110	1	20	4	80	260	96	2
ADD_R	3	30	6	60	1	20	4	80	100	384	6
SUB_R	4	40	7	70	2	40	5	100	100	384	6
MUL_R	3	30	6	60	2	40	5	100	80	384	6
DIV_R	8	80	11	110	4	80	7	140	240	384	6
SL	1	10	4	40	1	20	4	80	80	72	2
SR	1	10	4	40	1	20	4	80	80	72	2
RL	1	10	4	40	1	20	4	80	80	72	2
RR	1	10	4	40	1	20	4	80	80	72	2
AND_W	1	10	4	40	1	20	4	80	80	72	2
OR_W	1	10	4	40	1	20	4	80	80	72	2
XOR_W	1	10	4	40	1	20	4	80	80	72	2
NOR_W	1	10	4	40	1	20	4	80	-	72	2
NAND_W	1	10	4	40	1	20	4	80	-	72	2
XNOR_W	1	10	4	40	1	20	4	80	-	72	2
GT	1	10	4	40	1	20	4	80	80	72	2
ET	1	10	4	40	1	20	4	80	80	72	2
AND	1	10	4	40	1	20	4	80	80	60	1
OR	1	10	4	40	1	20	4	80	80	60	1
XOR	1	10	4	40	1	20	4	80	80	60	1
ORN	1	10	4	40	1	20	4	80	-	60	1
ANDN	1	10	4	40	1	20	4	80	-	60	1
XNOR	1	10	4	40	1	20	4	80	-	60	1

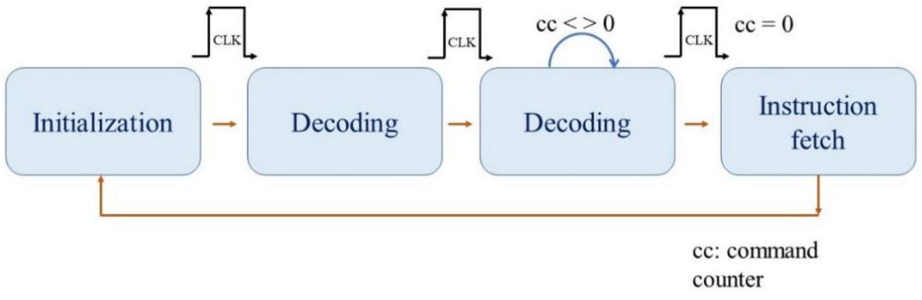


Figure 3. Instruction cycle of the proposed microprocessor.

The instruction code defines the data type, ALUs operations, and a suffix that can be either an operand or a memory pointer depending on the instruction. It is worth mentioning that Word and Bit ALU interpret the data type and operation code differently due to the difference in data that they admit, note that the stored CRW_X and CRb_X registers from the dual-port RAM (refer to subsection G) can be accessed for use in bracket operations, as demonstrated in test program 2. Moreover, the suffix has different possible values depending on the selected data type:

- Memory pointer: Memory word or bit
- 32-bit value: Literal
- bit value: Input or output

The instruction code format is shown in Figure 4a, where the fragments (1), (2) and (3) are related to the following list:

(1) Data type - Word ALU:

- Literal
- Current register CRW_X
- Memory word

(1) Data type - Bit ALU

- Input
- Memory bit
- Current register CRb_X
- Output

(2) Operation code - Word ALU

- LD_W
- ST_W
- ADD_I
- SUB_I
- MUL_I
- DIV_I
- ADD_R
- SUB_R
- MUL_R
- DIV_R
- SL
- SR
- RL
- RR
- AND_W
- OR_W
- XOR_W
- NOR_W
- NAND_W
- XNOR_W
- GT
- ET

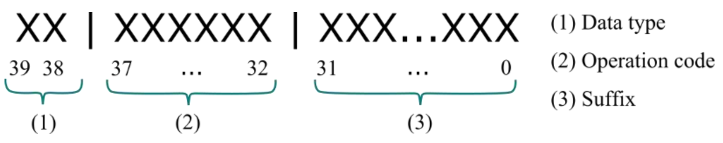
(2) Operation code - Bit ALU

- LD
- AND
- OR
- XOR
- ORN
- ANDN
- XNOR
- ST

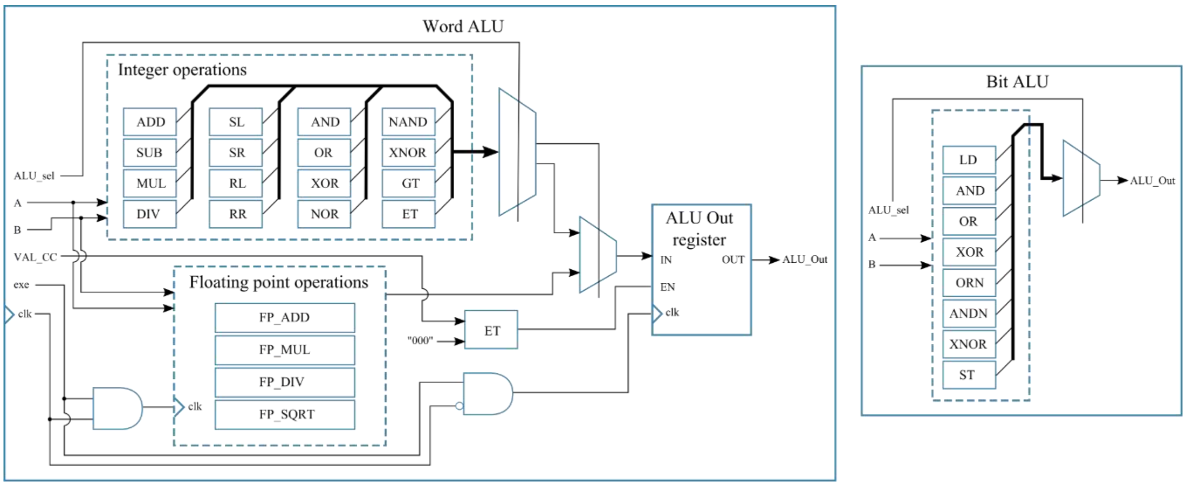
(2) Operation code - Line jump

- JMP

(3) Suffix



(a)



(b)

Figure 4. (a) Instruction code format and (b) Word ALU and Bit ALU

2.2 *Program Counter*

This is an 8-bit up-counter connected to the address of the program memory as a pointer.

2.3 *Command Counter*

It is a descending counter whose value is used to determine the duration of the execution state, different for each instruction. For example, floating-point operations take a longer time to execute.

2.4 *Word ALU*

The Word ALU executes operations between 32-bit length words, it supports 22 operations including sum, subtraction, multiplying, and dividing with integer and floating-point numbers. An "equal to" (ET) comparator is responsible for enabling the output register only when the command counter VAL_CC becomes zero. Furthermore, logical, load, and store operations are included. A detailed description of this ALU is presented in Section 3.

2.5 *Bit ALU*

This ALU is simpler, it only executes logical, load, and store operations between bits, this is quite useful for a PLC because it can implement the native ladder logic to represent bits for traditional PLCs contacts and coils. In Figure 4b the internal diagrams of the Word and Bit ALU are illustrated.

2.6 *Operands Selector*

This block is a key contribution of this work. It is responsible to select the operands that will pass to the ALU inputs after a rising edge clock. The operands can be:

- Literals
- Microprocessor inputs and outputs (I0_X and Q0_X)
- M0_X and M1_X memory elements
- CRW_X and CRb_X registers

It also has the possibility of assigning the results of the operations (ALUs outputs) to memory elements and the microprocessor outputs, useful for ST and ST_W store instructions. In Figure 5a is shown the internal diagram. Finally, the operands selector is enabled by the dec input (during decoding state).

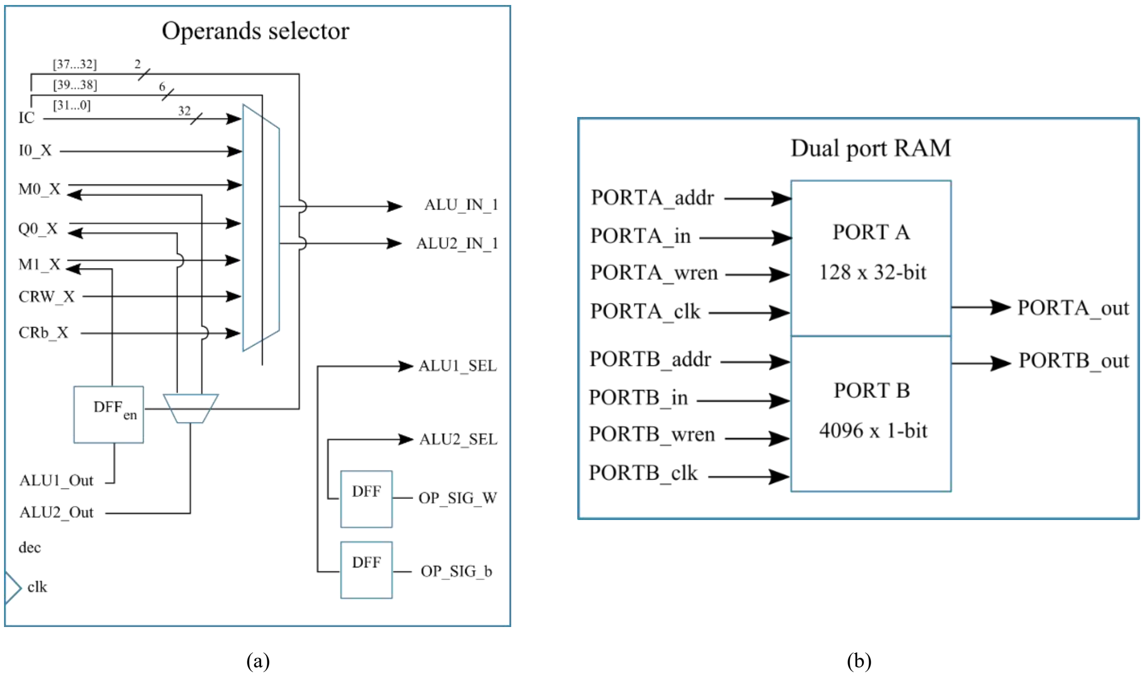


Figure 5. (a) Operands selector and (b) Dual-port RAM.

2.7 *RAM Memory*

The microprocessor integrates two RAM memories including the program memory, a dual-port RAM, and two independent banks called M0_X and M1_X. The program memory allows 256 words x 40 bits, suitable to test programs. On the other hand, the dual-port RAM stores the current results of the Word and Bit ALU – known as CRW and CRb registers–, the two ports' capacity is 128-word x 32 bits and 4096 x 1 bit respectively (see Figure 5b). Finally, the M0_X and M1_X banks store bits and 32-bit words respectively working as general-purpose registers. All the RAM memories are scalable and are limited by the number of memory blocks available within the FPGA.

3. Arithmetic Operations

The implementation of arithmetic operations is critical to achieving low execution times. To perform integer operations, the VHDL standard arithmetic packages were employed in this design [21], using the numeric_std package, which includes definitions of sum, subtraction, division, multiplication, shift, and rotation. In addition, the Vivado software 2018.3 and Quartus Prime 17.1 were capable of synthesizing all these operations without needing a low-level algorithm description, as was previously reported specifically for division [22]. Furthermore, is worth mentioning that the Artix-7 FPGA DSP48E1 slices (which have 25x18 multipliers) were taken as an advantage. Moreover, in the case of Cyclone IV FPGA the embedded 9x9-bit multipliers offer higher performance than LUTs (Lookup Tables). On the other hand, floating-point operations require special treatment because of the significant and exponent parts, the more complex implemented operation was the add/sub as is shown in Figure 6 [23]. We considered suitable designs including condition detection and adjustments, the commonly used alignment, normalization, and rounding. So, the floating-point adder/subtractor circuit was carried out in five stages as follows (Unpack, Alignment, Addition / Subtraction, Normalization and Rounding).

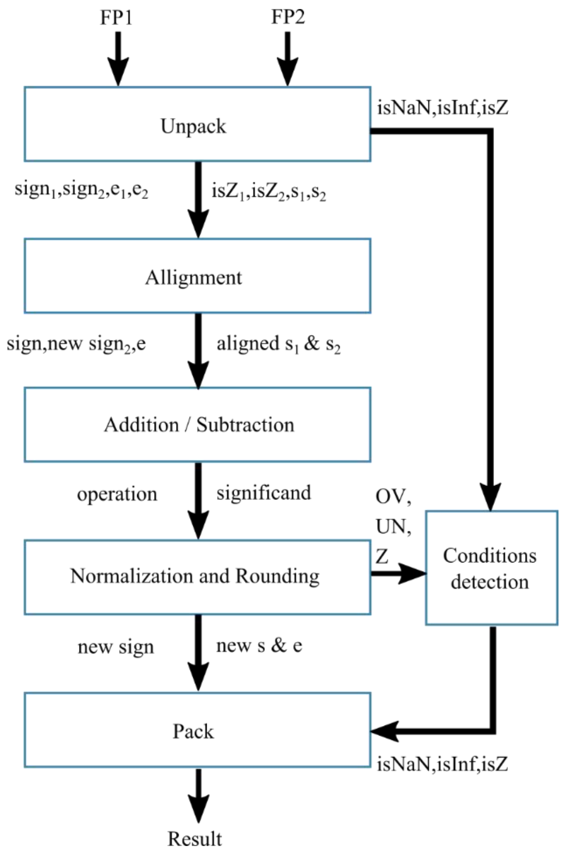


Figure 6. Simplified structure of the floating-point adder/subtractor algorithm.

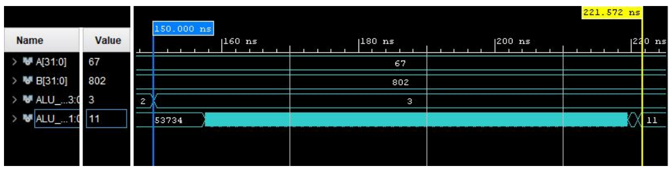
The proposed ALU was implemented as a combinational model because it serves as a reference for most optimized designs that could use e.g. resource sharing, and pipelining. The typical problem of this approach is the large number of required resources. However, our total architecture demonstrated the use of low logical resources as was presented in the results section compared with [22].

4. Results

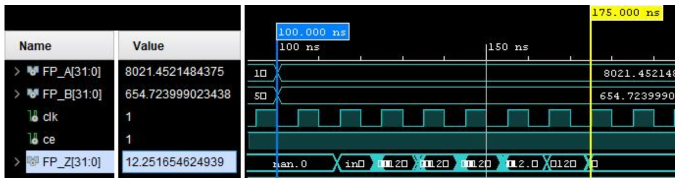
The proposed microprocessor was implemented in the xc7a100tcsg324 and EP4CE10E22C8 FPGAs. To identify each technology, we employed the labels Micro-XPLC and Micro-IPLC respectively. So, the parameters to report the performance are propagation delays, instruction execution time, and program execution time. Finally, to prove of concept two different programs were implemented for testing and analysis.

4.1 *Propagation delays*

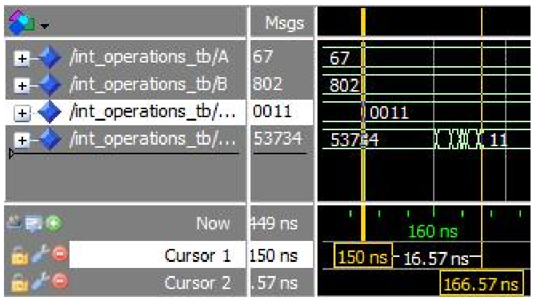
First, we calculated the instruction execution times on each FPGA, the "Post-Implementation Timing Simulation" and "Gate level Simulation" in Vivado and Quartus software were employed respectively. Where the operands are A, B, FP_A, and FP_B and the propagation delays of DIV_I and DIV_R (Division operation) are shown in Figure 7 that operations take the longest time. Comparing the performance of our system with a vendor PLC can be a difficult task as they typically only disclose timings for some bit and word operations. To overcome this, we have utilized the Chmiel et al. CPU [11] as a benchmark for comparison, which provides execution times for numerous instructions. Furthermore, we have conducted an evaluation of our system's performance by considering two of the most robust PLCs from Siemens S7-1500 family [24], as outlined in Tables 1 and 2. It is worth mentioning that in this work the fastest operations require a lower number of logical resources such as the logical and integer operations. Moreover, the propagation delays are different in each FPGA (Xilinx and Altera) due to the difference in the internal hardware and the optimization algorithm.



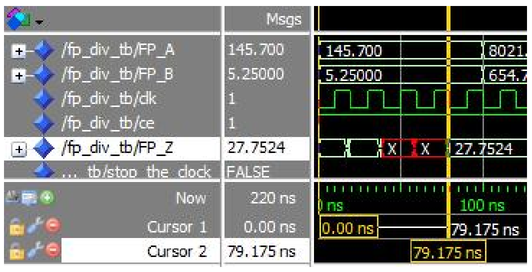
(a)



(b)



(c)



(d)

Figure 7. "Post-Implementation Timing" and "Gate level" simulation of division operations, (a) DIV_I propagation delay in Micro-XPLC, (b) DIV_R propagation delay in Micro-XPLC, (c) DIV_I propagation delay in Micro-IPLC, (d) DIV_R propagation delay in Micro-IPLC.

Table 2. Comparison of Test program 2 execution times between Micro-XPLC, Micro-IPLC, CPU [11] and Siemens S7-1500

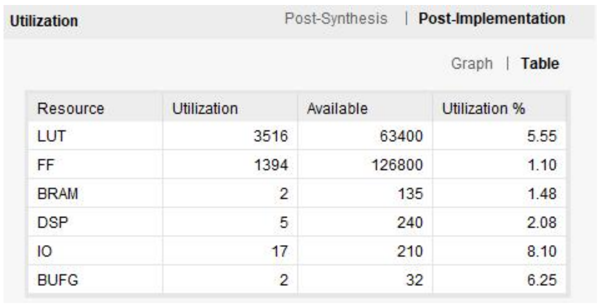
Micro - XPLC	Micro-IPLC	CPU [11]	Siemens 1511(T)(F)-1 PN 1511C-1 PN	Siemens 1518(T)(F)-4 PN/DP 1518(F)-4 PN/DP MFP
0.99	1.62	1.7	4.32	0.068

4.2 *Execution Times*

The execution times in Micro-XPLC are faster than in Micro-IPLC because of the most advanced technology of the Artix-7 FPGA and its clock frequency of 100MHz, whereas the Cyclone IV FPGA works with 50MHz. Is relevant to mention that when comparing the Micro-XPLC (refer to Table 1), it's been observed that all the relevant instructions are faster than the CPU [11], which also utilizes an FPGA. There's a clear difference between them when looking at instructions like DIV_I and ADD_R, which have a difference of 150 and 40ns respectively; compared with the lowest Siemens CPUs we have a disadvantage of 14ns and advantage of 324ns, with respect to the fastest there is a disadvantage of 108ns and 54ns on those instructions. On the other hand, Micro-IPLC is faster than CPU [11], with a difference of 180 and 20ns in the same instructions. When comparing the execution times with Siemens 1511 and 1518, the Micro-IPLC has similar performance to Micro-XPLC. However, in the DIV_ I the Micro-IPLC is 16ns faster than Siemens 1511.

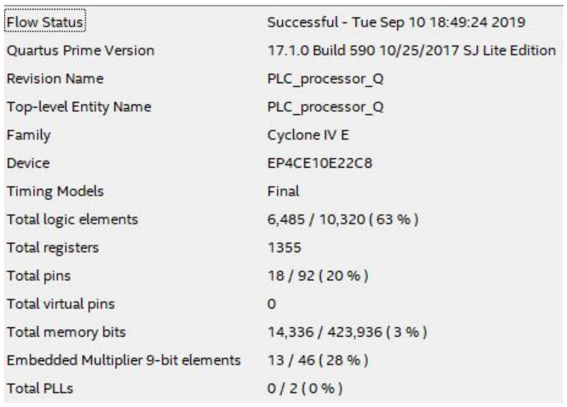
4.3 *Logical resources*

In Figure 8 the utilization of logical resources of the Micro-XPLC and Micro-IPLC are reported, for the first one 5.55% of LUTs and 1.48% of BRAMs were employed from the Artix-7 FPGA, in the second case we employed 63% of logic elements and 3% of memory bits from the Cyclone IV FPGA. Moreover, it is important to consider that the Artix-7 has 6 input LUTs and the Cyclone IV has 4 input LUTs (one per logic element). Even with the combinational ALU the employed logical resources are lower than [22], this is evidence of the efficiency of the J. Deschamps et al. arithmetic models. Also, is visible that the lower use of DSP (Digital Signal Processor) slices on the Artix-7 is a consequence of the technological evolution of this FPGA. Where DSP slices have 25x18 multipliers instead of 18x18 from the Spartan 6 family.



Resource	Utilization	Available	Utilization %
LUT	3516	63400	5.55
FF	1394	126800	1.10
BRAM	2	135	1.48
DSP	5	240	2.08
IO	17	210	8.10
BUFG	2	32	6.25

(a)



Flow Status	Successful - Tue Sep 10 18:49:24 2019
Quartus Prime Version	17.1.0 Build 590 10/25/2017 SJ Lite Edition
Revision Name	PLC_processor_Q
Top-level Entity Name	PLC_processor_Q
Family	Cyclone IV E
Device	EP4CE10E22C8
Timing Models	Final
Total logic elements	6,485 / 10,320 (63 %)
Total registers	1355
Total pins	18 / 92 (20 %)
Total virtual pins	0
Total memory bits	14,336 / 423,936 (3 %)
Embedded Multiplier 9-bit elements	13 / 46 (28 %)
Total PLLs	0 / 2 (0 %)

(b)

Figure 8. (a) Micro-XPLC logical resources utilization and (b) Micro-IPLC logical resources utilization

4.4 *Simulation of Test Program 1*

For testing purposes, we simulated a simple program (Listing 1) that loads a number (124) and multiplies it by 5, afterward there is a JMP instruction to go to an infinite loop.

Listing 1: IL code for Test program 1

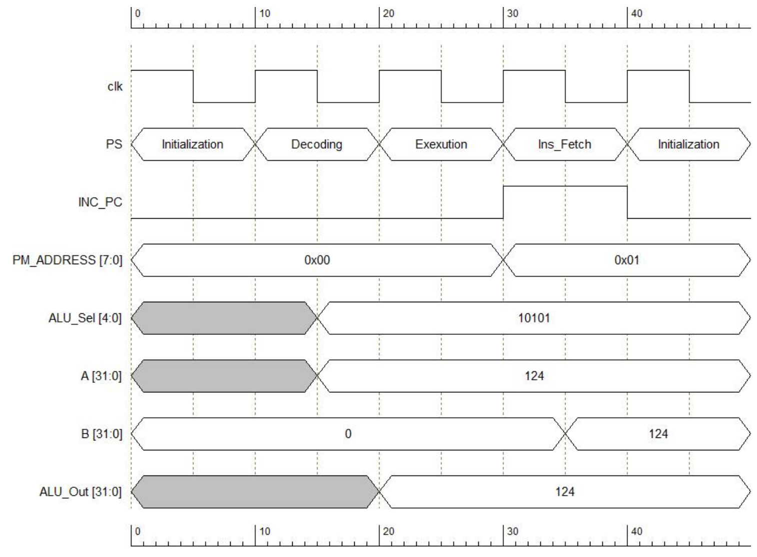
```
00: LD 124
01: MUL_I 5
02: JMP 00
```

Initially, Figure 9a shows the first instruction cycle performing the following:

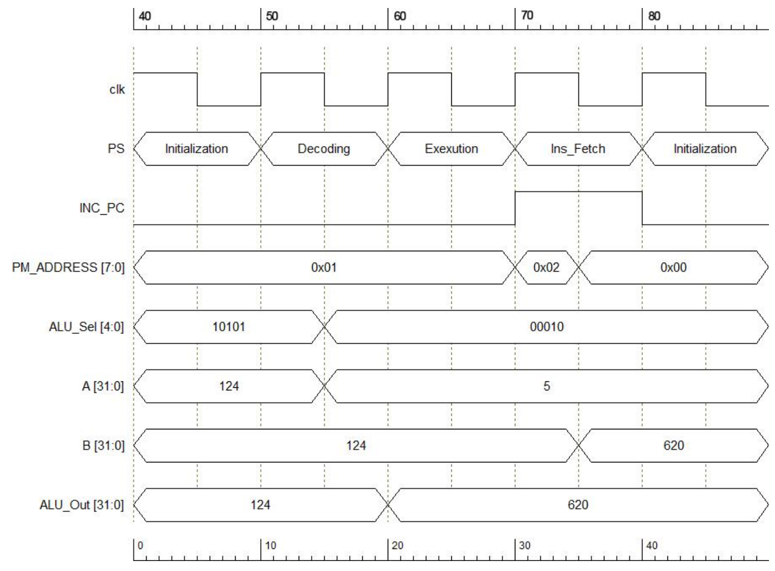
- Initialization: INC_PC becomes “0”.
- Decoding: ALU_Sel becomes “10101”, which corresponds to LD operation.
- Execution: ALU_Out becomes 124, this value is stored in CRW.
- Instruction Fetch: INC_PC becomes “1” increasing the program memory. Besides, CRW value is stored in B.

In the second part is illustrated in Figure 9b and works as follows:

- Initialization: INC_PC becomes “0”.
- Decoding: ALU_Sel becomes “00010”, which corresponds to the MUL_I operation.
- Execution: ALU_Out becomes 620, this value is stored in CRW.
- Instruction Fetch: The program memory address is increased, and in the falling edge of the clock it resets because of the JMP instruction, also CRW is stored in B.



(a)



(b)

Figure 9. (a) Simulation of Test program 1 (part 1), and (b) Simulation of Test program 1 (part 2).

4.5 *Simulation of Test Program 2*

An implementation of the well-known PID control method [25] was used to verify the operation of the microprocessor. The equations below were considered and executed in order from (1) to (3).

$$Q = K_p \cdot (SP - PV) + K_i \cdot I + K_d \cdot (PV - PE)/T_s \tag{1}$$

$$I = I + (SP - PV) \cdot T_s \tag{2}$$

$$PE = PV \tag{3}$$

Where K_p, K_i and K_d are the main control parameters of PID algorithm, *SP* is the setpoint, *PV* is the process variable, *I* is the error integral accumulator, *PE* is the previous error store variable, and T_s is the sampling time, all these values were considered as memory stored since we are focusing in the calculations. Listing 2 presents the IL code and it is important to note that CRW_2 and CRW_5 registers are used as operands in lines 05 and 0A for the ADD_R bracket instruction. These registers contain the results of lines 02 and 05, additionally, the Ki and PV values are loaded to the current register in lines 03 and 06. Then, the program execution times were calculated by adding the single times for each instruction employed, including load of values and floating-point arithmetic operations, the results are listed in Table 2. Then, in the case of Micro-XPLC, there is a reduction of the execution time of 41.76% in comparison with the CPU reported in reference [11]. However, Micro-IPLC matches almost the same time as CPU [11]. Moreover, the Micro-XPLC and the Micro-IPLC present a reduction of 77.08% and 62.5% respect to the Siemens 1511, whereas the 1518 is the fastest.

Listing 2: IL code for Test program 2

```
00:  LD SP
01:  SUB_R PV
02:  MUL_R Kp
03:  ADD_R( Ki
04:  MUL_R I
05:  )
06:  ADD_R( PV
07:  SUB_R PE
08:  MUL_R Kd
09:  DIV_R Ts
0A: )
0B: ST OUT
0C: LD SP
0D: SUB_R PV
0E:  MUL_R Ts
0F:  ADD_R I
10:  ST I
11:  LD PV
12:  ST PE
```

5. Conclusions and Future Work

The microprocessor was implemented in both FPGAs technologies (Xilinx and Altera) proving its superiority in their execution times compared to the CPU [11] and the 1511 Siemens CPU. Moreover, the xc7a100tcsg324 and EP4CE10E22C8 FPGAs have enough hardware to implement an IEC61131-3 standard-based microprocessor. Furthermore, its parallelism is useful to implement arithmetic functions, allowing to increase in the performance of the microprocessor employing its ALU. It is possible to enhance the execution time of instructions by implementing different logical structures in Word and Bit ALU and developing pipelined versions. Additionally, to overcome the LUTs performance, the model can be synthesized into an ASIC (Application Specific Integrated Circuit). Finally, since the Micro-XPLC microprocessor occupies reduced logical resources. So, it is possible to implement more than one microprocessor in the same FPGA and use co-processing techniques.

Conflict of Interest

There is no conflict of interest for this study.

References

[1] Vadi, S.; Bayindir, R.; Toplar, Y.; Colak, I. Induction motor control system with a Programmable Logic Controller (PLC) and Profibus communication for industrial plants — An experimental setup. *ISA Trans.* **2021**, *122*, 459–471, https://doi.org/10.1016/j.isatra.2021.04.019.

[2] Dwinugroho, T.B.; Hapsari, Y.T.; Kurniawanti Greenhouse automation: smart watering system for plants in greenhouse using programmable logic control (PLC). *J. Physics: Conf. Ser.* **2021**, *1823*, 012014, https://doi.org/10.1088/1742-6596/1823/1/012014.

[3] Kempegowda, V.H.; Raghukiran, N.; Reddy, K.J. Productivity enhancement by using programmable logic controller-based automated guided vehicles for material handling. *Int. J. Mechatronics Autom.* **2020**, *7*, 175, https://doi.org/10.1504/ijma.2020.110855.

[4] Bolton, W. Programmable Logic Controllers, 6th ed. Newnes: Waltham, MA, USA, 2015; pp. 1–20.

[5] Alphonsus, E.R.; Abdullah, M.O. A review on the applications of programmable logic controllers (PLCs). *Renew. Sustain. Energy Rev.* **2016**, *60*, 1185–1205, https://doi.org/10.1016/j.rser.2016.01.025.

[6] Economakos, C.; Economakos, G. Optimized FPGA implementations of demanding PLC programs based on hardware high-level synthesis. **2008**, 1002–1009, https://doi.org/10.1109/etfa.2008.4638516.

[7] Du, D.; Liu, Y.; Guo, X.; Yamazaki, K.; Fujishima, M. Study on LD-VHDL conversion for FPGA-based PLC implementation. *Int. J. Adv. Manuf. Technol.* **2008**, *40*, 1181–1190, https://doi.org/10.1007/s00170-008-1426-4.

[8] Eassa, H.; Adly, I.; Issa, H.H. RISC-V based implementation of Programmable Logic Controller on FPGA for Industry 4.0. **2019**, https://doi.org/10.1109/icm48031.2019.9021939.

[9] Jamieson, P.; Blank, D.; Ghanem, J.; McGrew, T.; Corti, G. A Methodology for an FPGA Implementation of a Programmable Logic Controller to Control an Atomic Layer Deposition System. *Int. J. Reconfigurable Comput.* **2022**, *2022*, 1–10, https://doi.org/10.1155/2022/8827417.

[10] Milik, A.; Kubica, M.; Kania, D. Reconfigurable Logic Controller—Direct FPGA Synthesis Approach. *Appl. Sci.* **2021**, *11*, 8515, https://doi.org/10.3390/app11188515.

[11] Chmiel, M.; Kulisz, J.; Czerwinski, R.; Krzyzyk, A.; Rosol, M.; Smolarek, P. An IEC 61131-3-based PLC i mplemented by means of an FPGA. *Microprocess. Microsystems* **2016**, *44*, 28–37, https://doi.org/10.1016/j. micpro.2015.11.001.

[12] Rudrawar, S.K.; Sakhare, D. High Performance Instruction List Processor on FPGA Platform. **2019**, 145–152, https://doi.org/10.1109/iccmc.2019.8819846.

[13] Mazur, P.; Chmiel, M.; Czerwinski, R. Central processing unit of IEC 61131-3-based PLC. *IFAC-PapersOnLine* **2016**, *49*, 454–459, https://doi.org/10.1016/j.ifacol.2016.12.061.

[14] Chmiel, M.; Mocha, J.; Lech, A. Implementation of a Two-Processor CPU for a Programmable Logic Controller Designed on FPGA Chip. **2018**, 13–18, https://doi.org/10.1109/icses.2018.8507283.

[15] Milik, A. Multiple-Core PLC CPU Implementation and Programming. *J. Circuits, Syst. Comput.* **2018**, *27*, https://doi.org/10.1142/s0218126618501621.

[16] 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.

[17] Yulin, D.; Chunjiao, Z. Design and research of embedded PLC development system. **2011**, *3*, 226–228, https://doi.org/10.1109/iccrd.2011.5764286.

[18] Rida, M.E.; Liu, F.; Jadi, Y. Design Mini-PLC based on ATxmega256A3U-AU microcontroller. **2014**, *2*, 1034–1037, https://doi.org/10.1109/infoseee.2014.6947826.

[19] Asif, M.; Raza, A.; Sultan, A.; Malik, F. Design of Mini PLC based on PIC18F452 Microcontroller using Concepts of Graceful Degradation. *Tech. J. UET Taxila* **2016**, *21*, 51–57.

[20] Mazur, P.; Czerwiński, R.; Chmiel, M. PLC implementation in the form of a System-on-a-Chip. *Bull. Pol. Acad. Sci.: Tech. Sci.* **2020**, *68*, 1263–1273, https://doi.org/10.24425/bpasts.2020.135386.

[21] LaMeres, B.J. Introduction to Logic Circuits & Logic Design with VHDL. 2019, https://doi.org/10.1007/97 8-3-030-12489-2.

[22] Kulisz, J.; Chmiel, M.; Krzyzyk, A.; Rosół, M. A Hardware Implementation of Arithmetic Operations for a n FPGA-based Programmable Logic Controller. *IFAC-PapersOnLine* **2015**, *48*, 460–465, https://doi.org/10. 1016/j.ifacol.2015.07.078.

[23] Deschamps, J.-P.; Sutter, G.D.; Cantó, E. Guide to FPGA Implementation of Arithmetic Functions. **2012**, https://doi.org/10.1007/978-94-007-2987-2.

[24] Siemens AG, Simatic S7-1500, S7-1500R/H, ET 200SP, ET 200pro Cycle and response times. Available online: https://support.industry.siemens.com/cs/document/59193558/simatic-s7-1500-s7-1500r-h-et-200sp-et-200pro-cycle-and-response-times?dti=0&lc=en-CA (accessed on 13 July 2023).

[25] Zhang, J.; Li, H.; Ma, K.; Xue, L.; Han, B.; Dong, Y.; Tan, Y.; Gu, C. Design of PID temperature control sy stem based on STM32. *IOP Conf. Series: Mater. Sci. Eng.* **2018**, *322*, 072020, https://doi.org/10.1088/1757 -899x/322/7/072020.