On Ladder Diagrams Compilation and Synthesis to FPGA Implemented Reconfigurable Logic Controller

Adam MILIK

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

adam.milik@polsl.pl

Abstract. *The paper presents synthesis process of a hardware implemented reconfigurable logic controller from a ladder diagram according to IEC61131-3 requirements. It is focused on the originally developed a high-performance LD processing method. It is able to process a set of diagrams restricted to logic operations in a single clock cycle independently from the number of processed rungs. The paper considers the compilation of the ladder diagram into an intermediate form suitable for logic synthesis process according to developed processing method. The enhanced data flow graph (EDFG) has been developed for the intermediate representation of an LD program. The original construction of the EDFG with attributed edges has been described. It allows for efficient representation and processing of logic and arithmetic formulas. The set of compilation algorithms that allow to preserve serial analysis order and to obtain massively parallel processing unit are presented. The overview of a hardware mapping concludes the presented considerations.*

Keywords

DFG, FPGA, high-level synthesis, IEC61131-3, LD, logic synthesis, PLC, reconfigurable hardware.

1. Introduction

The Programmable Logic Controllers (PLC) have been used since 1970s and first they were applied to relay control systems. Within years of fast development of electronic technology, the requirements given to PLC become higher all the time (operating speed, handling of analog objects, the increasing reliability, etc.). Today, the areas of PLC applications include small complexity processes as well as large manufacturing lines.

The general concept of a PLC is based on the microprogrammable circuits. It consists of two inseparable parts that are a hardware platform and software. The Hardware platform is able to execute given set of logic and arithmetic instructions. A control algorithm is created in the form of instructions sequence [1], [2], [8].

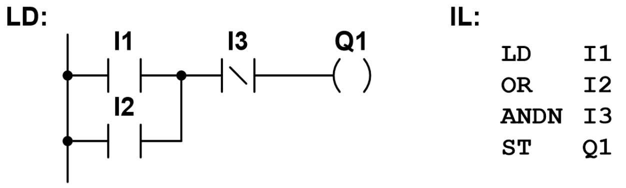
In contrast to software centric solutions, hardware offers intrinsic parallel execution of the tasks. It radically reduces the response time and offers better performance than software solutions. The implementation of the control algorithm with the use of reprogrammable and reconfigurable logic has been proposed by different research groups [3], [4], [5], [9], [11], [13], [15], [16], [19]. There have been proposed a custom FPGA architecture for direct mapping of the LD logic [17]. The significant limitation in wide use of reprogrammable digital circuits is a high design complexity of the implementation processes (in comparison to the instruction based standard approach).

A set of tools for creating a reconfigurable controller and its direct programming with well defined and commonly used ladder diagram has been developed. Presented work concentrates on transforming of control algorithm designed with the use of the ladder diagram into a form suitable for the entire process of hardware implementation incorporating: optimization, scheduling and hardware mapping. There has been considered details of the LD program execution according to IEC61131-3 requirements.

An intermediate form of the control program has been developed according to considered standard requirements. Algorithms, presented in this paper, are a part of the developed concept of reconfigurable logic controllers families and toolset for their programming according to the IEC61131-3 reference manual.

2. The LD Execution Model

The LD network is widely used method for describing control algorithms [1]. This method has been inherited from relay control systems. Contacts and coils represent logic dependencies between signals and function blocks. According to IEC61131-3 requirements a network is analyzed in a row based fashion that evaluates power flow through components rung by rung. In contrast to the electrical schematic diagram the power flow is unidirectional. The power is transmitted from the left rail to the right. There are implied limitations that prohibit of reverse power flow in ladder schematic [8]. Sequential analysis of schematic produces an ordered sequence of instructions for a PLC (Fig. 1). A switch is transformed into a logic AND operation. This operation is performed between current result coming from the predeceasing node and a signal that controls the switch. A junction merges power flow coming from several rungs. In some cases, there is a need for creating variables that enables storage of partial results for nested operations.



```
LD    I1
OR    I2
ANDN  I3
ST    Q1
```

Fig. 1: The ladder diagram and its equivalent instruction sequence.

The LD network processing speed can be increased by parallel execution of the logic operation in programmable hardware. Transforming logic dependencies into combinatorial logic allows increasing performance several orders of magnitude. It is required to develop a universal method suitable for representing not only LD programs, but other programming methods specified by IEC61131. This method should be able to synthesize logic operation, but also other operations performed by PLCs (e.g. timers, counters, arithmetic operations).

2.1. Existing Synthesis Models for LD

An LD diagram is described by two sets of Boolean variables I and Q. The set I consists of variables associated with inputs while the set Q consists of variables associated with outputs and internal markers. The logic functions are defined by rungs and create an ordered sequence of Boolean expressions:

$$q_i = f_i(I, Q), i = 1...r, \tag{1}$$

where i is the rung index. Equation (1) defines the ordered sequence of processing according to the index i. This feature has been utilized in implementations proposed by [9], [10], [16].

An exemplary LD network and its implementation have been presented in the figure (Fig. 2). In this model each rung is processed in individual cycle. The controller response time is proportional to the number of rungs in a program. In comparison to the programmatic approach, this model reduces a computation time of logic functions. It can be noticed that calculations of some variables can be processed in parallel. Distributing calculation process for each rung (q variable) introduces redundant cycles. In considered diagram (Fig. 2) variables q_1 and q_3 do not depend on other q variables. The q_1 variable and the q_3 can be evaluated in the first cycle (t_1).

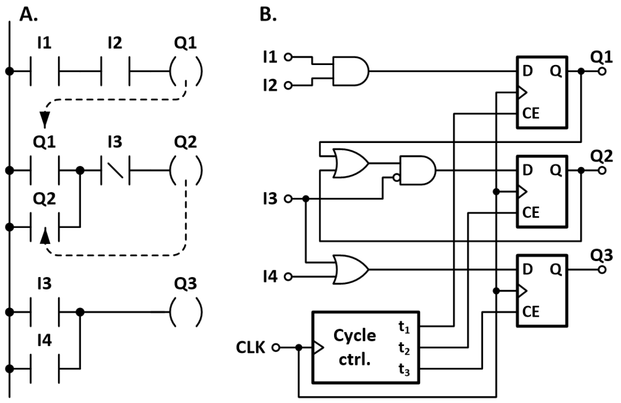


Fig. 2: The LD network (A) and its equivalent (B).

In order to reduce the number of calculation cycles dependencies between q_i variables have to be determined. In the paper [6] authors introduced an idea of using dependencies and simultaneities graphs for creating a sequential functional chart (SFC) from given LD. This idea has been employed in [3] for creating optimized hardware description. Similar idea has been employed in [15] for control algorithm partitioning. During the analysis of the LD, a dependencies graph is created. This is a directed graph that consists of nodes representing all q_i variables. The node v_i (representing variable q_i) is connected by directed edge with node v_j only if function f_i depends on variable q_j and $i > j$:

$$(v_j, v_i) \leftrightarrow f_i(q_i) \neq const. \tag{2}$$

The number of elementary cycles based on dependencies analysis is equal to:

$$T = p_{max} + 1, \tag{3}$$

where p_{max} is the longest path in the dependencies graph.

2.2. The LD High-Performance Synthesis Model

Presented dependency analysis of the ladder diagram is directly derived from sequential execution model. There are considered rungs as independent units delivering variables value. The early approaches consider each rung to be dependant of predeceasing rungs. The execution is performed in serial fashion a rung by a cycle. Applying rung dependencies analysis allows to determine the calculation dependencies in the form of the graph which is used for improved rungs scheduling.

The LD can be considered as a sequence of operations. Let assume that variables associated with inputs are updated before the start of the calculation process and remain constant during it. Let introduce the set of variables D that are assigned with a value of processed expressions. Value of the variable d_i is assigned to variable q_i at the end of calculation cycle ($q_i = d_i$). This approach allows to distinguish between two values that are calculated in the present cycle (d_i) and in the previous cycle (q_i). Equation (1) for m^{th} rung can be rewritten in following form:

$$d_m = f_m(I, d_0, ..., d_{m-1}, q_m, ..., q_n), \tag{4}$$

$$q_m = d_m. \tag{5}$$

Using proposed substitution of q variables allows to propagate calculation results through all functions bypassing registers (Fig. 3). The current value of control process is updated by single clock pulse after calculating all d_i values. In presented form the calculation process is fully parallel and completes in a single cycle that transfers values from d to respective q variables.

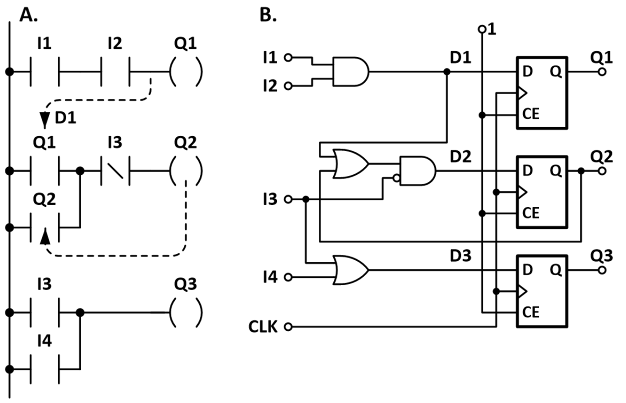


Fig. 3: The LD network (A) and its hardware equivalent obtained with proposed synthesis method (B).

3. Intermediate Representation with the Use of Data Flow Graphs

It is required to develop appropriate representation for an intermediate form of control algorithm that is suitable for high-level synthesis process. The intermediate form should be able to represent logic and arithmetic operations performed by PLCs maintaining operation sequence and reviling its dependencies. Commonly used form of intermediate representations for logic synthesis and compilers are data flow graphs [7]. A node of the graph represents elementary operation while directed edges indicate processing flow between nodes.

3.1. The EDFG Concept

For the purpose of recording PLC programs, the author has developed a form of enhanced data flow graph (EDFG). This has been inspired by concept of attributed edges used in BDD introduced by Minato [14]. In a similar way the EDFG handle unary operations like logic inversion and arithmetic complement. The other implemented extension is a multiple argument node for commutative operations. Presented modifications allow for efficient creating of data flow graph and reduces algorithmic complexity.

The Extended Data Flow Graph is given by $G = \langle V, E \rangle$ where V is a set of nodes representing elementary operations and E is a set of directed edges with attributes. The directed edge $e \in E$ is described by triple $e = \langle v_S, v_D, a \rangle$ where v_S is a predeceasing node and v_D is a successor node of the directed edge. The a is an attribute of the selected from the set A of allowed attributes.

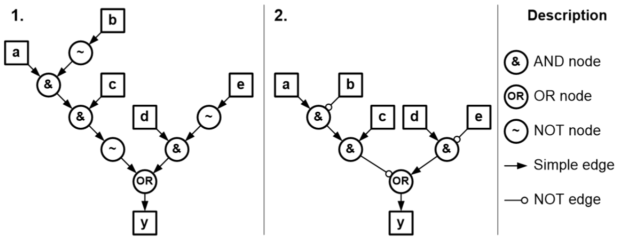


Fig. 4: Comparison of the general DFG (1) with EDFG (2).

An equivalent DFG to the Boolean formula $y = \overline{a \cdot \bar{b} \cdot c} + d \cdot \bar{e}$ is presented in the figure (Fig. 4). There have been considered two cases: a standard approach DFG (1) and with the use of the EDFG (2). The DFG (1) implements logic inversion by separate NOT nodes. Introducing attributed edges with logic inversion eliminates a NOT node EDFG (2). The attributed edge not only reduces the number of nodes in the diagram, but also allow to simplify logic operation handling.

Figure 5 shows the graph transformations for logic nodes. One of the common operation after compilation process is operation merge. Using EDFG simplifies the algorithms of node merging. Predeceasing node can be merged if it is connected with simple edge and both nodes implement the same logic operation. (shown in Fig. 5.1.). The reference node is marked with gray color. The operation is performed by modifying graph edges. There is exchanged v_D that equals v_1 to v_2. The merge rule can be extended with the use of de Morgan's laws. In the case (Fig. 5.2.) the predeceasing node is connected with inverted edge and implements opposite logic function (AND↔OR) to the reference node. The operation is performed by exchanging the v_D that equals v_2 to v_4 and the edge attribute is inverted. Finally, the v_2 node is removed The final EDFG is presented in Fig. 5.3.

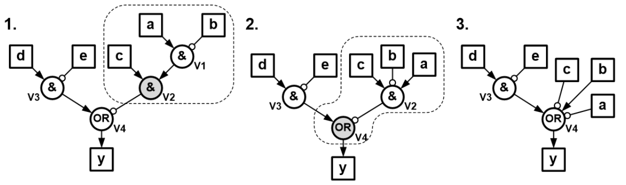


Fig. 5: Implementation of node merge (1) and de Morgan's laws (2) in DFG with attributed edges.

Thanks to attributed edges similar flexibility is achieved for arithmetic operations. In the domain of arithmetic operations, the subtraction node is replaced by an edge with complement value attribute. It reduces the set of arithmetic operations to: addition, multiplication and division. The figure (Fig. 6) shows the implementation of the expression: $y = a+b-c+d-e$ and compares the use of attributed edges for arithmetic operations. Using a standard approach with separate addition and subtraction nodes is shown in Fig. 6.A. Similar result is achieved using attributed edges (Fig. 6.B). The attributed edges simplify algorithms of operation merge and constant propagation as shown in Fig. 6.C.

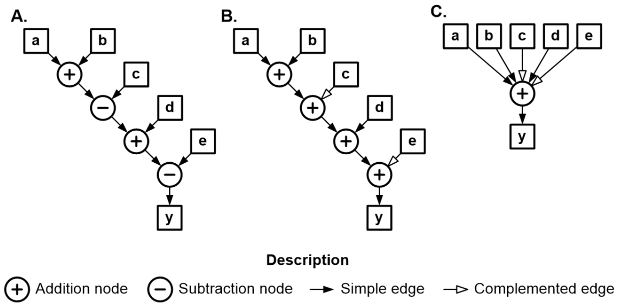


Fig. 6: Comparison of the general DFG (A) and the EDFG (B, C).

4. Converting LD to EDFG

The compilation process delivers basic items of a language [18]. Subsequent algorithms show systematic methods of translating those items into an EDFG suitable for hardware mapping.

4.1. Variables

The variables are declared at the beginning of a network according to IEC61131-3 requirements. For the purpose of the synthesis process the variables set is divided into three subsets. The variable classification is based on the signal association to input, output and internal marker areas. The variable reduction procedure takes into consideration variables membership. Variables associated with input signals are allowed to be read while value assignment is implied and made from input signals. Variables associated with outputs and markers are allowed for read and write access. The variables associated with markers can be eliminated when only write access is detected. There are two possible cases. The first one when a variable is used as a temporary storage for distributing the value and the other one when a variable is unused. The unused variable is distinguished as the only sink for the driving node. Variables associated with output signals and markers are not allowed to be read without value assignment. The moment of assignment is independent of the read access but must be completed at least once in entire calculation cycle.

The variable value access implementation assures sequential variable access according to LD description. In order to satisfy this requirement, a variable current value is accessed by following algorithm.

Algorithm 1: Let the x is a variable that the value is going to be read by node v, v_{xWR} is an assignment node to the x variable, v_{xDRV} is the node delivering value to the variable x. The variable x refers to EDFG nodes through the table pointing read and write nodes (Fig. 7). The v_{DRV} (if exists) is connected with a directed edge with v_{xWR}. The v_{xRD} is a value reading node of the x variable. Following two cases are possible depending on the value assignment sequence. If the variable x value is not assigned than the v_{xWR} node does not exist. The value of the x variable is read by creating the v_{xRD} node (this notifies that the value is coming from the previous cycle – e.g. forward coil reference). The v_{xRD} node becomes the argument of the v node (Fig. 7.1). If the variable x is assigned the v_{xWR} node exists. The directed edge connects it with driving node v_{DRV}. The v_{DRV} is used for driving the v node (Fig. 7.2). The attribute of the edge is inherited. It should be noticed that this mechanism follows recently assigned value to the x variable.

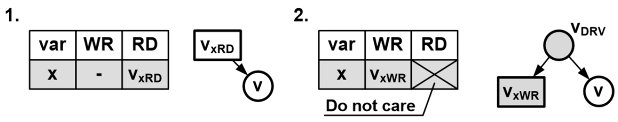


Fig. 7: The variable value access algorithm.

4.2. Nodes

The ladder diagram node merges power flow coming from multiple sources. It should deliver a logic sum of all connected signals. A general algorithm that accepts multiple drivers for a node is used. Process starts from the variable declaration. The variable name is inherited from network node name (automatically generated). It is included into a subset of internal signals.

Algorithm 2: Let the x is a variable associated with a schematic node (junction), v_{xWR} is the x variable value assignment node, v_C is the graph node that is going to drive the x variable, v_P is the node that currently drives the x variable. There are two possible cases. If the x variable is not assigned than the v_{xWR} node does not exist. The v_C node is connected with newly created v_{xWR} node (Fig. 8.1). If the x variable is already assigned then the v_{xWR} node exists. In this situation, the v_{OR} node is created. Nodes v_P and v_C are connected to v_{OR} node. The v_{OR} node becomes the only driver of v_{xWR} (Fig. 8.2). The described algorithm can be repeated iteratively for nodes with multiple driving sources.

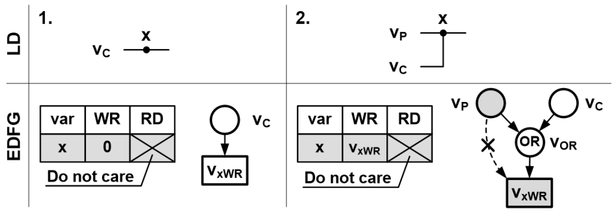


Fig. 8: The iterative conversion of the LD node into equivalent EDFG.

4.3. Switches

A switch is a basic component used for creating the logic AND operation between driving and input signals. The switch is converted into EDFG equivalent that has been shown in the Fig. 9. The EDFG procedure utilizes two previously described algorithms for variable access and node driving.

Algorithm 3: Let the x is a variable associated with an input node, a is a variable driving the switch and y is a variable associated with the output node. The v_{AND} node is created for considered switch. The value of the x and a variables are accessed with the use of the algorithm 1. The procedure returns respective driving nodes that are connected to the v_{AND} node. The attribute of the edge for variable a is set to a logic inversion for normally closed switch (Fig. 9.2). The v_{AND} node assigns value to the y variable. The assignment is performed according to the algorithm 2.

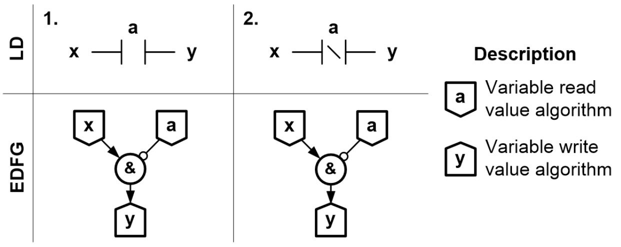


Fig. 9: The EDFG switch equivalent.

4.4. Coils

The coil assigns or reassigns value to a particular variable. Following algorithm is used for obtaining EDFG from a coil item. This algorithm is adopted to cooperate with remaining compilation algorithms, especially with variable value access.

Algorithm 4: Let the a is a driving signal, y is the variable associated with a signal driven by the coil. The variable value is read according to the algorithm 1 that returns driving node v_{aD}. Returned v_{aD} node is assigned to the variable y. If a value assignment node v_{yWR} does not exist it is created and linked with v_{aD} (Fig. 10.1). If the variable y is already assigned than the directed edge is reconnected to the recent driving node (Fig. 10.2). The edge attribute is set according to the coil type (e.g. inverted coil - Fig. 10.3).

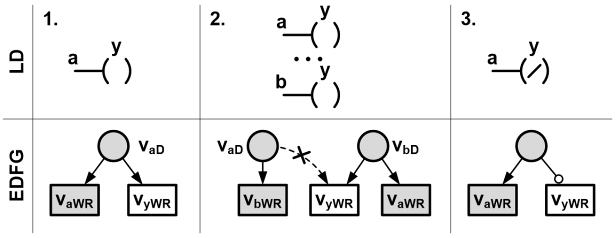


Fig. 10: The coil compilation scheme.

4.5. Functional Modules

The complex arithmetic or mixed arithmetic-logic functionality of the controller is implemented with the use of functional blocks. In the form of blocks are implemented timers, counters and arithmetic functions [1], [8]. Those blocks are controlled by logic and arithmetic signals. All logic signals are connected into ladder network while numeric variables are referenced by identifiers (names).

All these blocks perform conditional execution controlled by input logic signals. This implies the conditional execution of arithmetic operations. The EDFG requires introduction of the conditional selection node that implement selection and assures parallel operation execution). The conditional selection node shown in Fig. 11.1 reflects the high-level synthesis concept based on EDFG. It enables selection between arguments of same data type and can be used for flow control in logic and arithmetic paths. The optimization process can be separately applied to the data paths and control (selection input) path. The additional optimization rules are defined for selection nodes. This is important for optimization performed in arithmetic operations paths. When considered for logic paths the conditional node is transformed into multiplexer equation formed from logic nodes. This operation creates consistent logic EDFG that further can be optimized.

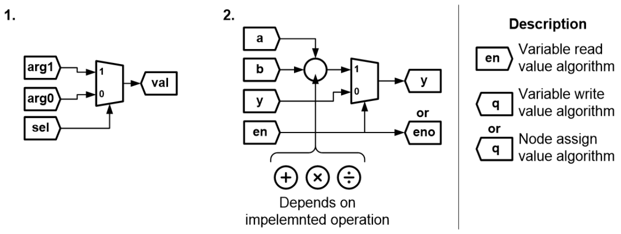

1.
arg1
arg0
sel
1
0
val
2.
a
b
y
en
1
0
y
or
eno
Depends on
impelemnted operation
Description
en
Variable read value algorithm
q
Variable write value algorithm
or
q
Node assign value algorithm

Fig. 11: The EDFG conditional selectin node (1) and general implementation EDFGimplementation of arithmetic blocks (2).

The exemplary general arithmetic module is shown in Fig. 11.2. This block performs calculations conditionally depending on the enable (en) signal. An arithmetic block is combined into EDFG between source and sink nodes. Conditional execution of the block forces an automatic variable implementation. It is achieved by executing algorithm 1 for the output variable before calling assignment algorithm 4 This variable is responsible for driving output when block is disabled ($en = 0$). The optimization procedures of selection nodes allow for eliminating covered by logic condition intermediate nodes. At the block level there are used two different value assignment procedures. For logic variables, the algorithm 2 is used while for numeric variables the algorithm 4 described for coils is used.

The figure (Fig. 12) shows an EDFG implementation of TON timer. This block is delivered in the form of sub EDFG that is incorporated into final EDFG during the compilation process. The timer is a specific implementation of a counter. It declares a hidden clocking signal t that is triggered with time base period. This signal enables counting of the timer unit. It is driven from the controller framework created during the implementation phase. Similarly to the arithmetic modules, there are declared internal variables responsible for storing elapsed time (et) and timer activity (q).

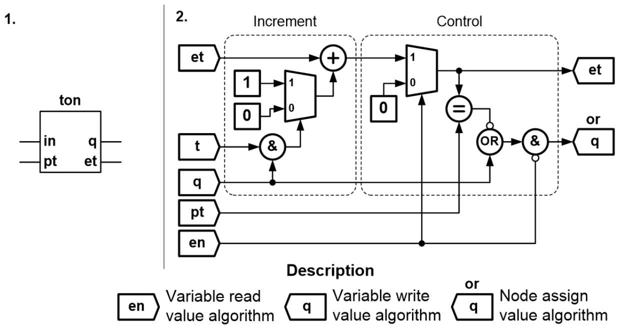

1.
ton
in
pt
q
et
2.
Increment
Control
et
1
0
1
0
0
1
0
t
&
=
OR
&
et
or
q
q
pt
en
Description
en
Variable read value algorithm
q
Variable write value algorithm
or
q
Node assign value algorithm

Fig. 12: The timer ton equivalent sub EDFG.

5. The EDFG Hardware Mapping

A synthesizable HDL model optimized for an FPGA target is obtained from described EDFG structure. The mapping procedure for an EDFG starts from the optimization process. The EDFG allows for limited optimization of the logic operations as presented in chapter 3. After initial operation merge and logic absorption, the Espresso minimization is used. This allows for further reduction of the logic operations and optimization of unused paths. In the domain of the arithmetic operation, a constant merge and common subexpression extraction are performed. Finally, multiple argument arithmetic nodes are expanded into a two argument nodes that can be directly mapped into arithmetic modules (adders and multipliers). The expansion process balances the propagation delay of operations in EDFG paths [12].

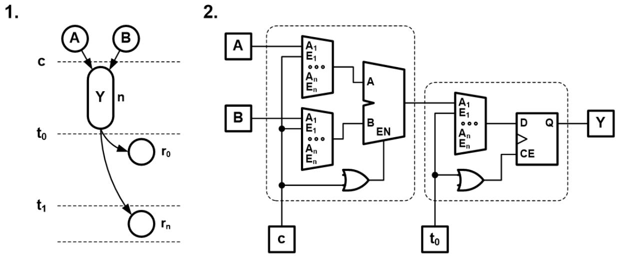


Fig. 13: The EDFG scheduling and mapping process.

The optimized EDFG is a subject of scheduling. It can be directly implemented with the use of greedy approaches with ALAP or ASAP scheduling methods [7]. In contrast to logic operations, arithmetic operations resource requirements are much higher. The greedy mapping approach will lead to quick resources run out. To overcome this limitation, a scheduling method based on list approach with original operation sorting is applied. The operation schedule takes into consideration the operation mobility and local dependencies. Scheduled nodes are mapped into a set of arithmetic resources. After operation schedule, the register allocation is made with the use of modified left edge algorithm. Schematically EDFG hardware mapping process is shown in (Fig. 13). There is shown a schedule result in the form of an EDFG (Fig. 13.1). On the graph are marked: operation start time (c) operation end time (t_0) and last access time to the result (t_1). Obtained mapping reflects the hardware structure shown in Fig. 13.2.

The mapping procedure optimizes the resource distribution by minimizing the cost of argument multiplexing. Structure of the controller and principles of its operation are shown schematically in Fig. 14. The specific EDFG depicts the allocated registers (small circles) and operations (large circles). The thick line connecting small circles denotes the variable lifetime. There are three computation stages. The calculation process starts from image registers update and internal variables exchange. After this operation, a calculation process takes place. The cycle is ended with a result write back.

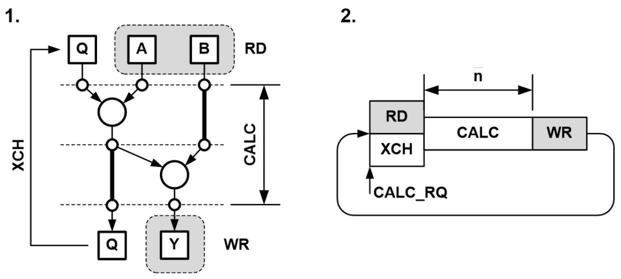


Fig. 14: The calculation cycle and its EDFG representation with marked variable lifetime.

The proposed method of implementation has been compared with direct EDFG mapping approach. The result are presented in Tab. 1. There has been selected 3 representative FPGA families that are Spartan II, Spartan 3 and Spartan 6. The Spartan II and the Spartan 3 are equipped with 4 input LUTs while the Spartan 3 is additionally equipped with combinatorial 18x18 multipliers. The Spartan 6 family is equipped with 6 input LUTs and DSP48A1 units. For illustrating implementation, two representative projects for automatic control have been chosen. The C2 project implements double PID controller with low pass filtering and hysteresis trigger. The T8 project implements the cascade of 8 timers controlling time dependant process. The optimization process can influence the controller response time by extending calculation time due to resource sharing. In the case of the C2 project there was used method of resource sharing that prohibits performance loss. This approach presents a non-redundant greedy assignment accommodated to FPGA architecture sharing. In general, the controller area has been reduced between 50 % - 60 % of the initial area. There can be observed a strong reduction of multipliers usage (50 %).

Tab. 1: The FPGA resource usage comparison.

FPGA	Proj.	Direct	Opt	Gain
		[LUT/MUL]		[%]
Spartan II (LUT4)	C2	1021/-	612/-	59.9 %
	T8	341/-	93/-	27.2 %
Spartan 3 (LUT4 + MUL)	C2	1252/4	717/2	57.2 %
	T8	341/-	93/-	27.2 %
Spartan 6 (LUT6 + DSP)	C2	1274/4	598/2	46.9 %
	T8	205/-	95/-	46.3 %

The T8 projects demonstrate the idea of resource sharing with an acceptable increase of response time. In the case of timers, the flat structure response time is 2 cycles. For this structure, the algorithm takes benefits form use of distributed RAMs that allows to reduce resource requirements about 3.67 times on expense of response time increase. This method is applicable when controller performs other calculation or performance reduction is acceptable.

6. Conclusion

The paper presents entire synthesis process of hardware implemented reconfigurable logic controller from a ladder diagram to hardware mapping. The paper is focused on the compilation of the ladder diagram into an intermediate form suitable for logic synthesis process. As it was presented, chosen intermediate form and methods of creating it has extremely high impact on the final result of the synthesis. The author has developed a method of intermediate representation based on the enhanced data flow graph that utilize attributed edges. It significantly simplifies the graph construction and processing. The intermediate form is created from the ladder diagram with the use of presented algorithms. Due to limited space only general overview of the LD compilation has been described. There are also developed method of representing arithmetic operations and complex functional blocks like timers and counters.

The graph representation is well suited for further processing oriented to FPGA implementation. Finally, a brief overview of mapping and implementation of synthesizable HDL description was given. Developed optimization methods allow to reduce controller size between 1.66 – 2.13 times without performance loss. Significant reduction of the controller size (about 3.6 times) is observed for implementation where specific features of FPGAs are used and little performance loss is acceptable.

Presented algorithms belong to originally developed a hardware PLC synthesis tool capable of synthesizing custom hardware implementation from LD, IL and SFC [11], [12]. The compilation and synthesis tool is subject of ongoing research and development. It is planned to extend arithmetic support to floating point numbers and improving scheduling and mapping processes.

References

[1] BOLTON, W. *Programmable Logic Controllers.* Burlington: Elsevier Newnes, 2009. ISBN 978-0-7506-8112-4.

[2] CHMIEL, M. and E. HRYNKIEWICZ. Concurrent operation of processors in the bit-byte CPU of a PLC. *Control and Cybernetics.* 2010, vol. 39, no. 2, pp. 559–579. ISSN 0324-8569.

[3] DAOSHAN, D., X. XIAODONG and K. YAMAZAKI. A study on the generation of silicon-based hardware PLC by means of the direct conversion of the ladder diagram to circuit design language. *International Journal of Advanced Manufacturing Technology.* 2010, vol. 49, no. 5, pp. 615–626. ISSN 1433-3015.

[4] ECONOMAKOS, C. and G. ECONOMAKOS. FPGA implementation of PLC programs using automated high-level synthesis tools. In: *IEEE International Symposium on Industrial Electronics.* Cambridge: IEEE, 2008, pp. 1908–1913. ISBN 978-1-4244-1665-3.

[5] ECONOMAKOS, C. and G. ECONOMAKOS. C-based PLC to FPGA translation and implementation: The effects of coding styles. In: *16th International Conference on System Theory, Control and Computing (ICSTCC).* Sinaia: IEEE, 2012, pp. 1–6. ISBN 978-1-4673-4534-7.

[6] FALCIONE, A. and B. H. KROGH. Design Recovery for Relay Ladder Logic. *IEEE Control Systems.* 1993, vol. 13, iss. 2, pp. 90–98. ISSN 1066-033X.

[7] GAJSKI, D., N. DUTT, A. WU and S. LIN. *High-Level Synthesis - Introduction to Chip and System Design.* New York: Kluwer Academic Publishers, 1992. ISBN 978-1-4613-6617-1.

[8] JOHN, K. H. and M. TIEGELKAMP. *IEC 61131–3: Programming Industrial Automation Systems: Concepts And Programming Languages, Requirements for Programming Systems, AIDS to Decision-making Tools.* Berlin: Springer Science & Business Media, 2001. ISBN 978-3540677529.

[9] ICHIKAWA, S., M. AKINAKA, R. KIEDA and H. YAMAMOTO. Converting PLC instruction sequence into logic circuit: A preliminary study. In: *IEEE International Symposium on Industrial Electronics.* Montreal: IEEE, 2006, pp. 2930–2935. ISBN 1-4244-0497-5. DOI: 10.1109/ISIE.2006.296082.

[10] LIU, Y., K. YAMAZAKI, M. FUJISIMA and M. MORI. Model-driven programmable logic controller design and FPGA-based hardware implementation. In: *ASME 2005 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference.* Long Beach: ASME, 2005, pp. 81–88. ISBN 0-7918-3766-1. DOI: 10.1115/DETC2005-85119.

[11] MILIK, A. and E. HRYNKIEWICZ. Synthesis and implementation of reconfigurable PLC. *International Journal of Electronics and Telecommunications.* 2012, vol. 58, no. 1, pp. 85–94. ISSN 2300-1933.

[12] MILIK, A. On Mapping of DSP48 Units for Arithmetic Operation in Reconfigurable Logic Controllers. *Programmable Devices and Embedded Systems.* 2012, vol. 11, no. 1, pp. 249–254. ISSN 1474-6670. DOI: 10.3182/20120523-3-CZ-3015.00048.

[13] MILIK, A. On Mapping of DSP48 Units for Arithmetic Operation in Reconfigurable Logic Controllers. *Programmable Devices and Embedded Systems.* 2006, vol. 6, no. 1, pp. 14–16. ISSN 1474-6670.

[14] MINATO, S. *Binary Decision Diagrams and Applications for VLSI CAD.* Berlin: Springer, 1996. ISBN 978-1-4613-1303-8.

[15] MOCHA, J. and D. KANIA. Hardware Implementation of a control program in FPGA structures. *Electrical Review.* 2012, vol. 88, iss. 12, pp. 95–100. ISSN 0013-4384.

[16] WELCH, J. T. Translating Relay Ladder Logic for CCM Solving. *IEEE Transactions on Robotics and Automation.* 1997, vol. 13, iss. 1, pp. 148–153. ISSN 1042-296X. DOI: 10.1109/70.554356.

[17] WELCH, J. T. and J. CARLETTA. A direct mapping FPGA architecture for industrial process control applications. In: *International Conference on Computer Design.* Austin: IEEE, 2000, pp. 595–598. ISBN 0-7695-0801-4. DOI: 10.1109/ICCD.2000.878352.

[18] WIRTH, N. *Algorithms + Data Structures = Programs.* New Jersey: Prentice Hall, 1976. ISBN 978-0130224187.

[19] ZIEBINSKI, A., R. CUPEK and W. SROKA. Application in Java language realizing the function parser of pseudocode describing structure of a specialized coprocessor of PLC in VHDL. *Measurement Automation and Monitoring.* 2011, vol. 57, no. 8, pp. 148–153. ISSN 0032-4140.

About Authors

Adam MILIK received M.Sc. and Ph.D. degrees from Silesian University of Technology of Gliwice in 1997 and 2003 respectively. Since 2003 he is a professor assistant at Silesian University of Technology of Gliwice. His main interests and research areas are: high-level logic synthesis and implementation, algorithm implementation, technology mapping in FPGA devices, the hardware high-level modeling systems based on HDLs and its integration with other tools like MATLAB, Simulink or SystemVue.