jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2012, VOL. 58, NO. 1, PP. 85–94
Manuscript received January 15, 2012; revised March 2012. DOI: 10.2478/v10177-012-0012-8

Synthesis and Implementation of Reconfigurable PLC on FPGA Platform

Adam Milik and Edward Hrynkiewicz

***Abstract*—The paper presents a set of algorithms dedicated for synthesis of reconfigurable logic controllers implemented on FPGA platform and programmed according to IEC1131 and EN61131. The program is compiled to hardware structure with a massive parallel processing. The developed method automatically allocates resources and operations. It controls resource usage and operation timing. Using mixed concept of operation allocation that considers operation timing and forms combinatorial chains of operations number of execution cycles can be reduced. An example of logic functions, PID controller and mixed arithmetic and logic programming examples are considered. Introducing the automatic implementation method allows flexible implementing the control algorithms. The maximal possible parallelism (limited only by the algorithm dependencies and available resources) is introduced.**

***Keywords*—PLC, LD, IL, FPGA, high level synthesis, logic synthesis, arithmetic circuits, reconfigurable hardware.**

I. Introduction

THE Programmable Logic Controllers (PLC) have been introduced in 1970s. In the beginning they were used as relay control systems. However, together with fast development of electronic technology, observed during last few decades, the demands (high operating frequencies, handling of analogue objects, increased reliability, etc.) of PLCs were constantly increasing. Today, the areas of PLC applications include small complexity processes as well as large manufacturing lines in industrial plants.

General concept of a PLC is based on the microprogrammable circuits. It consists of two inseparable parts that constitutes its operation. Those parts are hardware and software. Hardware part is able to execute given set of logic and arithmetic instructions. Software is an ordered sequence of instructions that allows solving problem with use of given hardware platform. This approach is very simple and effective in case of programming and modifying program that can be also called control algorithm [1], [2].

Modern PLCs belong to high level electronic and computing devices, but a question arises: is it possible to improve performance of those systems? This paper proposes solution for direct hardware algorithm implementation.

A. Program vs. Hardware

All PLCs executes their tasks indirectly by executing ordered sequence of instruction. Operation of a PLC is based on continues execution of control program in closed loop. For external observer it seems that PLC responds simultaneously to different signal changes applied to its input. In fact basic concept of microprogramming assumes that complex tasks are decomposed to serial chain of instructions. Control program is processed in serial fashion instruction by instruction. Program execution time is proportional to the number of instructions.

Different techniques of program acceleration are used. There can be observed a continuous development of CPU hardware architecture and optimizations of programming technique. Researches that are carried in domain of CPU architectures show possible solution in concurrent operation of different bit-byte execution units or independent processors [3], [4]. Other approach is based on event driven triggering of program blocks that allow to save time by skipping program blocks that haven't changed since last calculation [5].

The most efficient approach is to construct hardware that is able to perform control algorithm. It allows increasing operation speed several times. The response time is independent of number of instructions. Each task is performed with constant calculation time that does not dependent on number of tasks computed by a controller.

This paper presents high-level synthesis approach that automates hardware platform design for process automatic control designers with commonly used programming languages. This approach is similar to high level programming languages where generation of assembler is automatic and performed by a compiler [6], [7], [8].

This paper presents the developed synthesis algorithm of logic controller capable to process Boolean as well as fixed point variables. Presented algorithm simplifies determining the Boolean dependencies. The other novelty is incorporation of arithmetic operation synthesis aimed for efficient resource allocation.

II. Reconfigurable Controller

A reconfigurable controller is a term that describes logic controller with hardwired control program [9], [10], [11], [12]. A hardware platform must be able to perform at least static reconfiguration in order to maintain programmability. Static reconfiguration describes a reconfiguration process when device enters a special configuration mode [13]. Operation of controller is suspended during static configuration. Normal operation of controller is restored immediately after configuration process is completed.

The research work reported in the paper was supported by Ministry of Science and Higher Education of the Republic of Poland, Grant No. 5391/B/T02/2010/38.

A. Milik and E. Hrynkiewicz are with the Institute of Electronics, Silesian University of Technology, Gliwice, Poland (e-mails: {amilik,ehrynkiewicz}@polsl.pl).

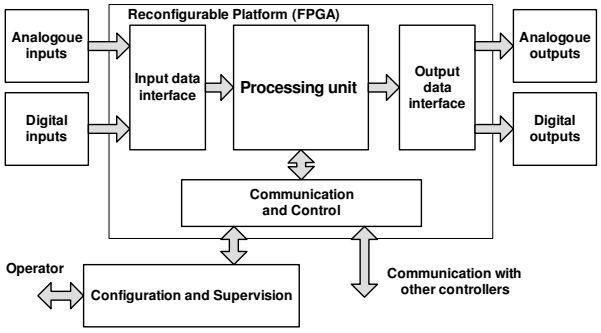

Fig. 1. The reconfigurable logic controller block diagram.

A. Controller Architectures

A general architecture of controller is presented on schematic diagram Fig. 1. Central part of the reconfigurable controller is called processing unit. The processing unit is not able to operate by itself. To distribute object signals and maintain operation of processing unit additional blocks are needed. Not all components can be reconfigured inside controller. Some blocks are determined by controller architecture and selected method of data processing.

The logic controller process input signals from different sensors and delivers control signals to actuators. Input and output signals can be discrete or analogue. Those input and output signals must be conditioned by interface modules and converted into appropriate form. Next signals reach input interface block that is a general structure of different interfaces that are responsible for proper data exchange with modules. The interface module architecture depends on user program requirements. This approach allows taking benefits from reconfigurability that allows choosing the most appropriate hardware structure for input interface.

Controller operation depends on supervising and configuration unit. This unit maintains configuration process but also enables to run, suspend and stop operation of configured controller. Configuration data in this case can be compared to The control program that is loaded into PLC CPU. The configuration not only determines functionality but also the internal hardware structure. The supervising unit is partially constructed outside of the reconfigurable platform while it must be able to operate when configuration is not loaded.

III. Programming

The PLC programming methods have been inherited from automation design like ladder diagram representing relay electric control systems. Different methodologies of description have been standardized. The PLC programming is covered by standards given by IEC1131 and EN61131 [14], [1] reference manuals. Those manuals specify the methods of programming. As the base for PLC programming can be assumed the instruction list language (IL) and the ladder diagram (LD). The high level languages like structured text (ST) and the sequential functional chart (SFC) can be compiled or translated to the instruction list that is the base of the controller operation. The data processing model established by PLC hardware and programming languages determines the interpretation of instructions.

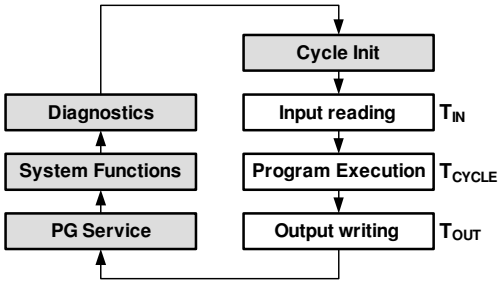

Fig. 2. A PLC operation cycle.

A. A PLC Data Processing Model

The calculation process of a PLC consists of reading inputs, calculating the results and finally transferring the results to the outputs Fig. 2. The calculations are carried out on the memory array called process image memory. Entire memory is divided into fragments that are assigned to the input signals, the output signals and the internal markers. The process image memory speeds up the calculation process and eliminates transient results delivery to the controlled devices [2]. Moreover it delivers quasi-concurrent control of concurrent processes.

The main difference in data processing can be observed in LD representation and its processing. The ladder diagram is inherited from electric diagrams of automatic control devices. The diagram presents graphically energy flow through switches and relays and other complex components. In general it is assumed that energy flows through the components from the left power rail to the right power rail (not always is shown on the LAD diagrams). The program is organized into set of networks. Entire program is processed in serial fashion network by network. Depending on analysis method used by a PLC itself or programming environment slightly differences in operation can be observed. There are two basic approaches based on column or row scan directions.

The column based processing method has been developed by Modicon. The column based analysis is applied to single network under processing. This method closely resembles behaviour of the real hardware. The LD schematic is processed column by column. Outputs are updated after processing all preceding switches. When a switch representing the output is used the value from the previous cycle is used. The

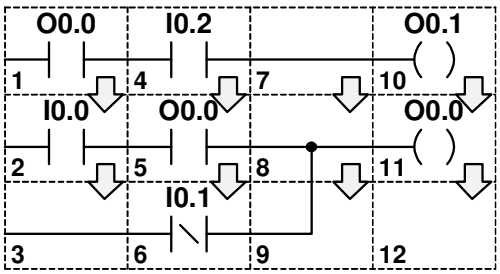

Fig. 3. Column based LD network analysis.

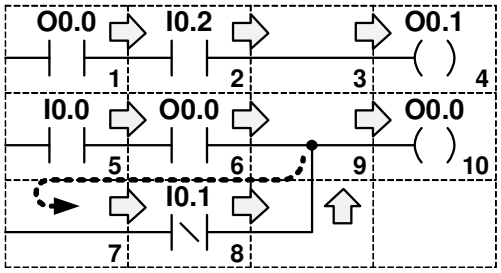

Fig. 4. Row based LD network analysis.

column based processing method requires creating an array of temporary variables. The array size is determined by the maximal number of rungs that can be recorded in one network. It can be noticed that this method is dependant on the location of items on the network.

The row based processing method is schematically shown in Fig. 4 (used by Simatic PLCs.) The rung items (usually switches) are analysed until reaching a coil or a junction. The coil causes the immediate assignment of logic value. The junction merges energy flowing from joined rungs (two or more) and requires analysis of other rungs that contribute energy (logic value) to the node. The analysis is stopped until all contributing rungs are evaluated.

It can be noticed that results of logic calculation are dependant on the components' appearance order in the network. In opposite to electric diagrams not all circuits with identical netlists are evaluated in the same way. Designers get used to the sequential manner of the analysis and the calculation results propagation. It is expected that new implementation will follow the well known processing rules of LD.

IV. A PLC to Hardware Compilation Model

The LD schematic due to specific serial processing produces different control flow than electrical schematic netlist implementation. The early implementation of the hardware PLCs were oriented to the direct schematic implementation [15], [10], [16]. A development of correct data processing model is required for the proper generation of hardware structure that behaves like the microprogrammable PLCs. The only difference concerns its performance.

The microprogrammable processing is able to handle a single variable at a time. This limitation is a source of differences in the comparison to the massively parallel hardware implementation. The compiler is also responsible for optimizing generated structure according to the image memory functionality by eliminating those statements, which results are overwritten provided that they are not a part of other expressions.

A. Previous Models of LD to Hardware Conversions

The proposed in [17], [18] model of LD translation model consists of three graphs: the simultaneity graph, the dependencies graph and the condensed simultaneity graph. The simultaneity graph consists of nodes representing all coils in the system. Unidirectional edges connect all nodes (coils) that can be executed concurrently. The directed dependencies graph records the coil dependencies. It presents the execution order of rungs. Finally, after creating the dependencies graph and all simultaneities are recorded the structure of controller is created.

The approach presented in the algorithm is oriented to HDL description rather than hardware synthesis. It produces circuit that requires several steps to complete the operation. Is it possible to introduce a synthesis method of lower complexity and offering comparable or better results?

B. Proposed Model of IL and LD Synthesis

The LD and IL synthesis process combined with finite state machines synthesis approach simplifies the entire synthesis approach. The LD and IL due to the sequential processing imply variable dependencies that should be properly recorded. Building appropriate processing model allows reducing the complexity of the compiler.

The creation of the compilation model requires building the model of data processing in hardware. The main limitation is the maintenance of sequential features of PLC defined by programming methods and its data processing.

The processing model consists of the combinatorial part responsible for implementing switches functionality. The results of combinatorial logic calculations are stored in registers. Registers hold values between calculation cycles and assure constant driving of object signals.

At the beginning a pure combinatorial function without any feedback is considered Fig. 5A. This is the simplest processing example. It does not depend on the outputs and can be processed during a single calculation cycle. The switches are translated into a combinatorial network that drives the data input of the register. Figure 5 depicts all transformation stages applied to LD that are required for obtaining the processing

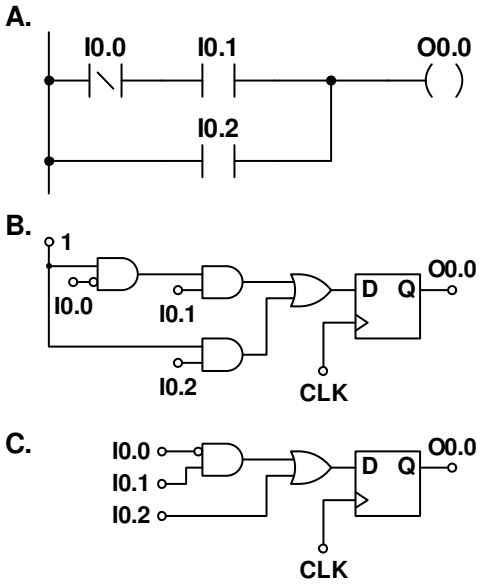

Fig. 5. Combinatorial function compilation: A. LD representation, B. Direct translation, C. After optimization.

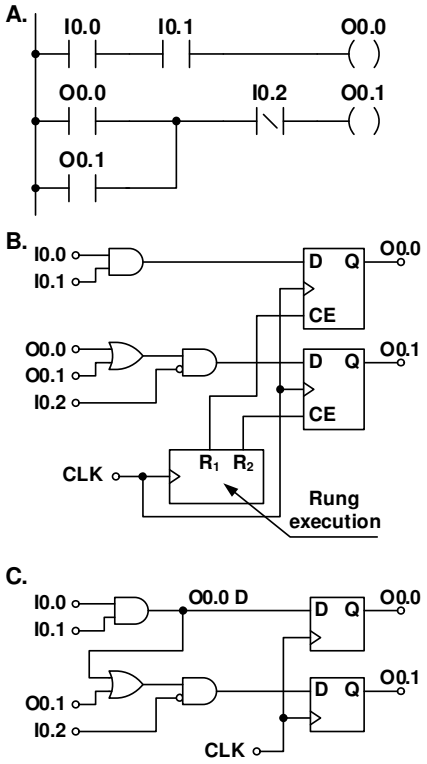

Fig. 6. Sequential LD dependencies implementation.

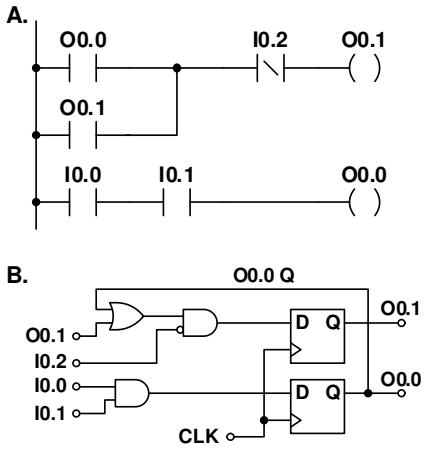

Fig. 7. The LD with memory loop.

system. Each switch is substituted by AND gate. One of the inputs is driven by the current logic value while the other one is driven by switch signal. The junction inserts OR gate into the translated structure that brings together all merged rungs. In the optimization phase constants are propagated and gates of the same type are merged. Further optimizations are possible by applying combinatorial function minimizations algorithms like Espresso or Exact of Quinee-McCluskey. Combinatorial synthesis algorithms allow monitoring correctness of formulated control statements informing about removed switches (signals) in the optimization process.

Implementing a design where rungs depend on results of other rungs requires implementing a sequential processing that assures the proper result calculation and partial result distribution inside a system. Very important factor is the complexity of the controller that manages data flow. Reducing complexity of the calculation flow will also reduce the hardware overhead connected with data flow management. The developed processing model should reduce the number of processing cycles. There are considered two cases of feedback signal use where feedback signal update precedes the expression or follows it.

Fig. 6A presents the LD network where signal O0.0 is fed to the next rung. This is a typical example of complex combinatorial function implementation where the partial arguments are shared and passed through intermediate nodes. Programmer expects that the signal changes are propagated thorough the cascade of switches during a single calculation cycle. The calculation cycle is sequential with calculation of O0.0 followed with calculation of O0.1 output. The schematic diagram of LD hardware implementation is shown in Fig. 6B. The proper data flow is maintained by a simple ring counter. The output of the counter activates register that represents coil on the LD schematic. For the external observer outputs are updated in the same cycle during the result write back from the process image memory to the output modules. This problem can be considered as a finite state machine synthesis. The immediate update of outputs means that there are not sequential dependencies that require sequential processing of signals. The FSM synthesis approach distinguishes the current state value and next state values. The current state value comes from the memory block and here it is a register system that holds the output or the marker. The next state value is worked out by a combinatorial function. Registers for signal O0.0 and O0.1 assure continues driving of the signal nodes. We can introduce two types of signals one with D suffix representing the next state value and the other with Q suffix representing the current value.

Using presented approach the circuit can be implemented in a form shown in Fig. 6C. The O0.0 signal that is processed in the second rung is fed directly from the combinatorial unit (net labelled O0.0 D).

Figure 7A presents the LD where the signal O0.0 is used before it is updated in the current cycle. This is an example of implementing a memory that delays a value for one computation cycle. The same model of the finite state machine implementation concept is used as in the previous example. It is expected that a partially computed variable will arrive in the next computation cycle. To address the mentioned problem and assure the proper computation order the variable I0.0 Q coming from the register is used. The proposed implementation allows for simultaneous processing of all variables independently of their dependencies without splitting the operation into cycles.

The above considerations for the LD sequential processing can be directly used in the IL synthesis. All described situations in the LD representation reflect the possible organization of the processing flow of the IL.

The proposed computation model allows processing a combinatorial expression in a single computation cycle. In opposite to models based on simultaneity and dependencies graphs the

proposed computation model offers the quickest calculation and reduces the complexity of creating the custom processing unit. It should be noticed that computation unit does not require the additional flow control unit. The overhead of the computation sequence controller has been eliminated.

C. Compilation Algorithm

The proposed compilation model can be put down in a form of the algorithm for processing the combinatorial expression in LD and IL forms. For the processing purposes the creation of the logic expression evaluator is required. The evaluator collects the partial logic expression during the LD analysis or parsing of the IL. For evaluation purposes we can use symbolic logic evaluator [19] or BDD package [20], [21], [22]. Using BDD for partial evaluation potentially can lead to inefficient variables ordering. Logic evaluator produces logic expressions from processed rungs (LD) or instructions (IL). Each expression has a reference counter. When the expression is assigned the reference counter is increased. When the expression is deassigned the reference count is decremented. When the reference count reaches zero the user is given a message informing that a given statement is not used. The statement is disposed. It should be noticed that the expression disposal is an iterative process of reducing the reference count in all referenced expressions.

For the simplicity and consistency of the representation, a system is considered as a set of variables regardless of their origin (input, output or marker). The value of each variable depends on the assigned expression from the expression collection.

The processing starts with empty set of variables. Each time the variable is referenced in a form of switch the read variable operation is performed. The read variable operation looks up for a variable in the variable collection. If the variable is not found it is created. A newly created variable is assigned an assignment expression of itself. It can be explained as the reading of Q output of the variable register.

The current variable expression is added to the temporary expression. The reference count for the expression is incremented. When the variable is referenced as a coil it is equal to the write variable operation. Similarly the variable is looked up in the variable collection if variable can not be found the new variable is created with the self assignment. The expression of the returned variable is deassigned. If reference count of deassigned expression reaches zero the notification message is generated. The notification informs about unused statement. This situation helps to detect potential errors in design. The temporary expression is assigned to the variable.

The IL statements are processed in the same way as the LD diagram. Figure 8 depicts the exemplary LD diagram and its data structure obtained in the compilation process. The data structure is constructed from two types of structures variables and operations. The operation structure allows representing all logic operation that can be handled. Reading (RD) and writing (WR) operations holds the pointer to the variable (the variable is not represented as a separate structure on the drawing). The variable points to the expression that is currently assigned to it.

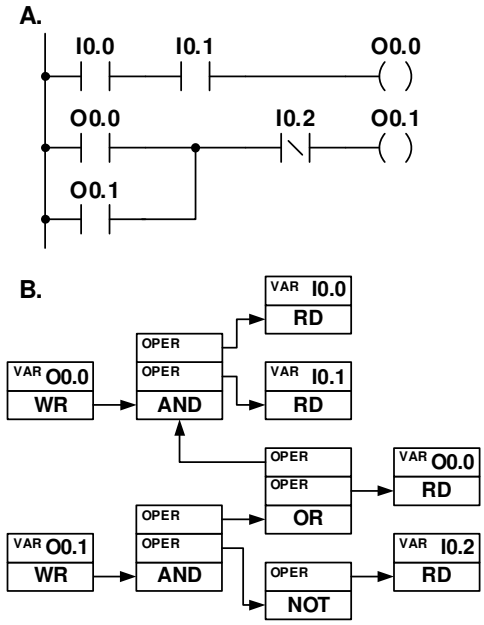

Fig. 8. The LD and its compiled data structure.

The operation capable of handling multiple arguments (AND, OR) stores them in the argument expression list.

After completing the compilation process further low level synthesis can be run. Alternatively, any of commercially available synthesis tools can be used to generate HDL description.

V. Arithmetic Operation in Reconfigurable PLCs

Until now the implementation of Boolean expression has been considered. Special attention has to be paid for implementing processing dependencies of programs created with use of the LD and the IL.

The control functions of PLC require the implementation of arithmetic operations and operations linking numerical values with Boolean variables like comparison. The arithmetic operations are foundation of other complex blocks like counters, timers or signal processing blocks like PID.

The implementation of arithmetic operations requires much more logic resources than Boolean variable processing. The arithmetic expression processing requires different approach than in case of Boolean expressions.

A. Arithmetic Operations

The arithmetic expressions can be represented graphically by a network of functional blocks (LD) or in a form of statements (IL). Sometimes a mixed form of the representation is used. It combines graphical and textual components in a form of sequential functional charts (e.g. MATLAB Stateflow).

The mentioned standard forms of mathematical expressions' representations are commonly used for building and designing of the control algorithm. A lot of tools limit synthesis process to the direct mapping of expressions into hardware form [10]. One of the simplest approaches can be called the program representation. This method of mapping executes/assigns one arithmetic operation into the calculation cycle. This approach

requires building a specific arithmetic processing unit (ALU) that processes data in a form of an embedded program (it can be compared to Single Instruction Single Data unit – SISD unit). The utilization of logic resources is limited, while the calculation time is proportional to the number of operations and all benefits of the massive parallel processing of Boolean variables are wasted.

The other approach, which assigns the unique hardware resource to each operation, can be called the direct hardware approach. This approach promotes the fastest execution of the calculation. The greedy hardware resource allocation strongly limits the number of operations (arithmetic instructions) in the entire program that can be processed (here allocated) by the controller. The mapping algorithms apart from the control algorithm representation must solve problems concerning the reconfigurable controller framework allocation. This problem concerns the proper data flow between input and output modules and the control of data processing cycle [6].

The design of the arithmetic operation synthesis package for a reconfigurable controller allows addressing mentioned problems of a control program mapping. The package must take into account the efficient resources allocation that are limited by the utilized programmable circuit and must assure the possible shortest calculation time. Further considerations are limited to the basic four arithmetic operations.

B. Arithmetic Operation Support in FPGA

The FPGA devices are continuously developed to meet different design requirements. Since the early implementation they are equipped with different arithmetic support. Further considerations are limited to Xilinx FPGA families. First implemented arithmetic expressions were addition and subtraction. These blocks have been introduced for the first time it in XC4000 families (the early third generation of FPGA devices). Those LUT generators were supplemented with additional hardware that implemented the direct carry chain among vertically neighbouring cells. This relatively simple implementation allows improving implementation of adders, subtractors and all addition based components e.g. counters, sequential multipliers and dividers etc. The signal processing requires fast multiplication. The combinatorial multiplier has been built from adders (based on general purpose logic resources). A typical combinatorial multiplier 16x16 bits allocates 266 LUT generators. To improve the performance and reduce general logic resource consumption multiplier cores have been implemented starting from the Virtex II structure. There are implemented 18 bit signed multipliers with selectable registered or combinatorial inputs and outputs. The algebraic expressions implementation considers the multipliers as atomic operations. The Virtex 4 family brings first highly specialized component for DSP implementation [23]. The DSP48 has taken the name after accumulation adder width. The slightly modified and better accommodated for DSP calculations version of the DSP48 core has been implemented in derivative family Spartan 6 under name DSP48A [24]. The simplified block diagram is presented on Fig. 9. The basic structure of the unit suggest it use as multiply and accumulate unit (MAC). Data registers

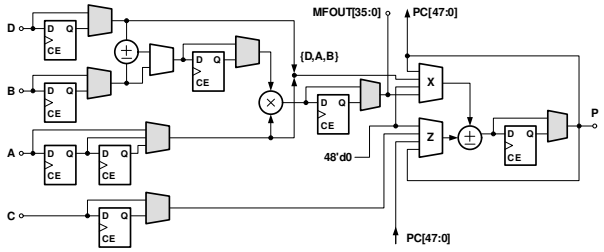

Fig. 9. The Spartan 6 DSP48A arithmetic core.

allow configuring modules to operate in pipelined fashion. The signal paths (direct or registered) are statically configured. It is possible to dynamically reconfigure the device. The dynamic reconfiguration is not suitable for computation components with the clock to clock cycle modification requirements. The dynamic reconfiguration procedure requires exchanging entire configuration frame. Some of the functions can be controlled dynamically at the run time. The multiplexers X, Y and Z are dynamically controlled. This allows for direct control of the operations. It can be noticed that one of the multiplier's arguments can be calculated as the sum or difference of A and D. The efficient use of DSP cores for implementing arithmetic operations given by high level language requires creating automatic tools that will aid designer in the implementation process.

C. Data Flow Graph Model of DSP48A Unit

The DSP48A implements chained multiplication based on addition. Blocks incorporate a signed 18x18 multiplier and final stage 48 bit adder or subtractor. The final stage adder when used as an accumulator has a 12 bit margin. The overflow can be expected after accumulation of 4096 items. The hardwired connections between arithmetic components does not allow for direct mapping of basic operations separately. Internal multiplexers allow controlling final stage adder arguments. The operations implemented in DSP48 and DSP48A slightly differ. These differences reflect used mapping procedures. The DSP48 block implements the configurable three argument adder (1)

$$\begin{gathered} P = Z \pm (X + Y) \\ Z = \{0, C, P, PC\}; X = \{0, A \times B, C\}; \\ Y = \{0, A \times B, P, A : B\} \end{gathered} \tag{1}$$

In this configuration both X and Y arguments are used by the partial multiplication results $(A \times B)$. There is implemented an ability of signed shifting of P or PC input right by 17 bits. It is useful for the implementation of long multipliers (multiplicands longer than 18 bits).

$$P = C \pm (A \times B) \tag{2}$$

When used with multiplier the structure allows to add or subtract results of the multiplication from given argument (2). The presented structure perfectly fits for implementing FIR or IIR filters where the "multiply and accumulate" (MAC)

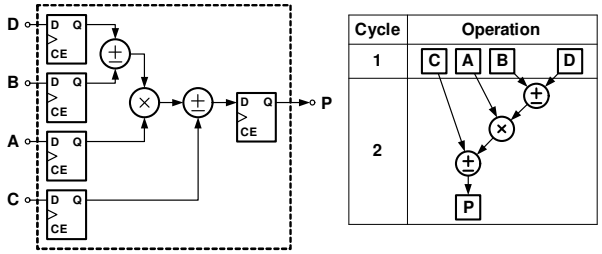

Fig. 10. The DSP48A block diagram with respective operation data flow graph.

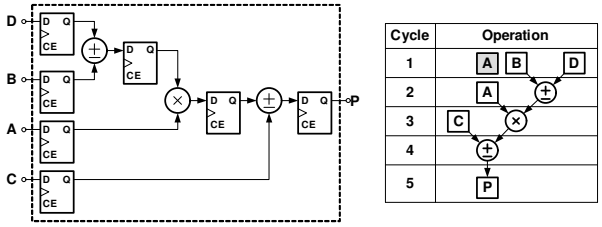

Fig. 11. The DSP48A pipelined mode block diagram with respective data flow graph.

operation is commonly used. The DSP48A brings some modification in comparison to DSP48. The final adder adds only two arguments:

$$\begin{gathered} P = Z \pm X \\ Z = \{0, C, P, PC\}; X = \{0, A \times B, D : A : B\}; \end{gathered} \tag{3}$$

The additional 18-bit adder before multiplier can be bypassed. Than the calculated result (4) is identical with (2).

$$P = C \pm (A \times B) \tag{4}$$

Selecting additional adder allows calculating the sum or difference of multiplier (5).

$$P = C \pm (A \times (D \pm B)) \tag{5}$$

This additional adder perfectly fits the requirements of the symmetrical FIR implementation or other calculation with symmetrically distributed coefficients. This ability is also worth to embedding into the mapping procedure.

The arithmetic functionality is the most important factor of the DSP blocks. There is additional architecture property that strongly impacts scheduling and mapping procedures. Each arithmetic stage inside the block can be connected directly (as a combinatorial unit) or separated by registers (the pipelined structure). The data path can be formed statically during the configuration. In the run time the register path structure can not be changed (only using the dynamic reconfiguration methodology). That introduces two architectures here called the combinatorial (Fig. 4) and the pipelined (Fig. 5). The pipelined architecture of DSP48 offers maximal performance of 400 – 500 MHz for DSP48 depending on the chip speed grade [25], [26]. Combinatorial architecture exhibits reduced performance between 250 – 317 MHz respectively. The combinatorial configuration offers better and simpler operation allocation. The pipeline architecture requires insertion of an empty cycle between the multiply and add or accumulate operations and the pure addition. It was decided to select the register buffered combinatorial structure of the unit.

VI. The Operation Mapping

The data flow graph – DFG set is used for the operation mapping [19], [7]. The DFG can be directly mapped into hardware, but usually it leads to the inefficient implementation. In the direct mapping each operation node is assigned to a separate hardware resource that offers the shortest calculation time. It should be noticed that the assigned unit is used only once per a calculation cycle that strongly reduces hardware resources utilization. The limited number of resources available in the FPGA requires developing methods that distribute the calculation in time (serially) and in the space (the parallel calculation).

The proposed mapping algorithm is built of limited number of arithmetic and logic resources. It is oriented to use DSP48 and DSP48A arithmetic blocks for implementing the multiplication and addition or subtraction. The division is accomplished with use of separate core implemented in general purpose logic resources.

The FPGA device is used as a main processing element of the reconfigurable controller. The circuit is equipped with the limited number of general purpose logic resources and hardware DSP cores. The arithmetic operation processing unit is a part of the whole controller that operates together with two state (Boolean) variables processing unit. The constraint of the limited number of resources is fully justified. The available logic resources are partially reserved for Boolean variables processing and the controller framework.

The mapping algorithm has been developed basing on several well known mapping algorithms [7], [27], [28], [8], [29]. The as soon as possible – ASAP, as last as possible – ALAP and the list of scheduling algorithms constitute the foundation for the formulation of the mapping operations algorithm in the limited number of resources with dynamic resource allocations. The mapping algorithm incorporates timing control that determines the operation execution sequential timing.

The ASAP algorithm in each step allocates all operations that arguments are known. The ALAP method determines the last possible moment of time when the operation has to be executed to maintain the proper calculation flow. Both methods represent a greedy approach where no resource limits are implied. The other simplification used in both methods is the atomic operation time. Assigning the same duration for different operations would require extending the calculation cycle to match the longest operation in the set. Comparing moment of execution of given node in ALAP and ASAP method the possible mobility range of variable can be determined. The list method assigns calculation resources from the finite set of computation resources to DFG nodes that arguments are known. The set of operations is defined by available computation resources before the scheduling procedure starts.

The proposed method is supposed to allocate the arithmetic operations with use of DSP48 components for the addition/subtraction and multiplication while the division is

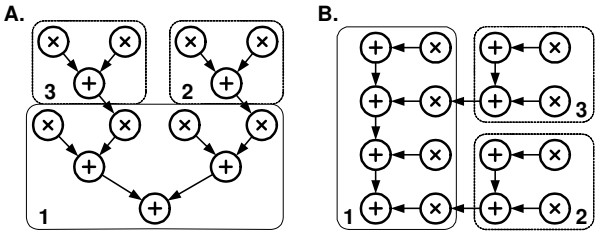

Fig. 12. The DSP48 data flow graph mapping transformation.

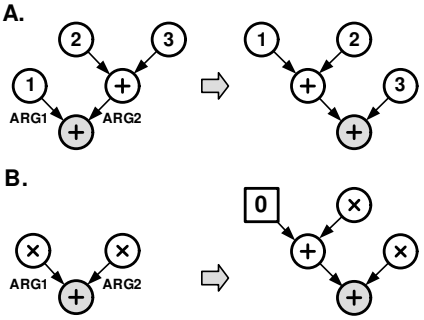

Fig. 13. The DFG rules of nodes exchange and creation.

implemented with use of general purpose logic components. The data flow and control unit developed as the result of scheduling are also mapped in general purpose logic resources. The algorithm begins from rearranging the DFG to DSP48 cores requirements. The most desired form of the graph that can be mapped to the DSP units is form of branch implementing accumulative addition. The previously reduced and optimized graphs (Fig. 12A) are transformed into a form of a vine branch (Fig. 12B). This transformation prepares the DAG for DSP48 mapping. The transformation moves the pointers among addition nodes. The commutative operation (e.g. addition) allows exchanging the arguments. The graph is transformed according to node exchange rules (Fig. 13). There are considered two cases. The node under transformation is marked with gray colour. The case A (Fig. 13A) considers the ARG2 arc. If the ARG2 is an addition node than it is placed in ARG1 and appropriate arguments are exchanged. Described operation is applied to the node until the ARG2 does not satisfy described condition. The case marked with B considers situation when both ARG1 and ARG2 nodes are multiplication. To satisfy mapping requirements only one multiplication is available. The mapping procedure creates new addition node. The ARG1 from considered node is attached to the ARG2 in newly created node while to the ARG1 is assigned output of newly created node.

Presented mapping transformation allows determining a lower bound of calculation cycles (CY_{MIN}). The minimal number of calculation cycles can be obtained from calculating number of elementary operations (OP) and number of available DSP blocks. It is assumed that all n DSP units operate with the full utilization.

$$CY_{MIN} = ceil\left(\frac{OP}{n}\right) \tag{6}$$

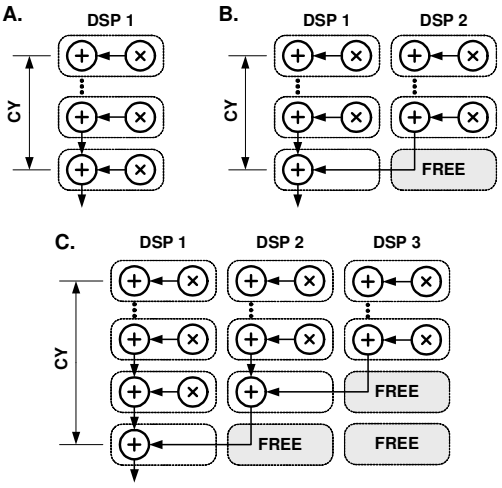

Fig. 14. The long sequence of operations distributions.

The elementary operations are a pair of the addition type (addition or subtraction) and multiplication nodes or an addition node type only. The previously made assumption still holds in general case, where mainly the addition and the multiplication are dominant and the division is a rarely used operation.

The applied transformation creates the DAG in a form of linear fragments that perform accumulative addition of arguments. This transformation contradicts the requirements of time reduction of operations and the parallel processing while expansion of the tree height and the serialization of processing are observed. The mapping process into DSP cores requires distributing of mapped operations in space to assure approaching to the CY_{MIN}. During distribution we face the problem of implementing long linear chain of operations (e.g. high order FIR filter). In general the length of the chained operation can exceed significantly the minimal number of calculation cycles in worst case. For the purpose of minimizing the length of linear fragments they can be split into smaller pieces. The idea is presented in Fig. 14. The distribution operation takes into consideration the device architecture. The DSP48 cores are placed in columns. Each DSP48 uses direct PC lines to connect with the unit placed above (Fig. 9). Dedicated connections allow reducing use of general purpose programmable logic resources and controlling complexity. There are presented three cases of operation distributions among DSP48 units. The case A (Fig. 14A) presents the implementation without the distribution. The case B distributes operations between two DSP48 units while case C considers distributions between 3 DSP48 units.

$$\begin{aligned} N_A &= CY \\ N_B &= 2 \cdot CY - 2 \\ N_C &= 3 \cdot CY - 3 \end{aligned} \tag{7}$$

The performance in terms of basic operations is described by (7). It can be noticed that during distributed execution of the operation some units are released before completing the calculation. Those freed units can be allocated for other calculations. The other problem that has to be addressed is timing control of operation sequences that depends on result from

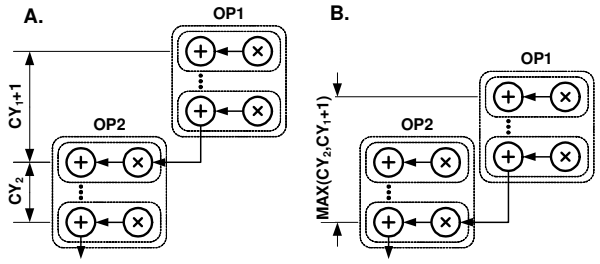

Fig. 15. Reducing calculation time of dependant processes.

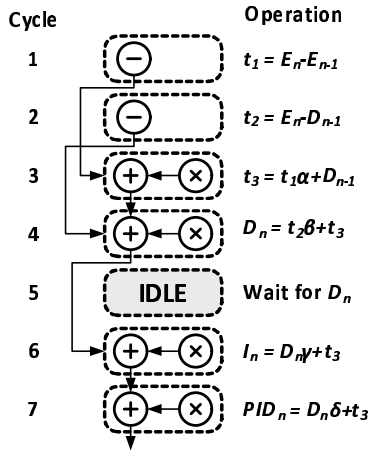

Fig. 16. The PID controller scheduled DFG.

other calculations. To reduce the calculation time and assure the most possible parallel execution the operation mobility range can be exploited. The operation mobility range allows selecting the most suitable location for a given operation in a sequence. The problem and the possible solution is shown in Fig. 15. It presents the calculation of two chained operations. The operation OP2 depends on the result from the operation OP1. Moving a dependant operation node down to OP2 allows to earlier starting calculation of the OP2 chain. Using the operation mobility range approach the pipeline transport delay must be taken into consideration for variable arrival scheduler.

The scheduling operation attempts to distribute operations in time to assure the proper calculation flow and to achieve the possible highest hardware utilization. To optimize data flow and the variable multiplexing system the variable exchange is performed. The optimization can be applied to the commutative operation and to the set of the same operations (Fig. 12). The proposed process of the automatic implementation of arithmetic expressions is used for PID controller implementation. The controller is given by the following set of discrete differential equations (8) [30], [9].

$$\begin{array}{c} D_n = D_{n-1} + \alpha \cdot (E_n - E_{n-1}) + \beta \cdot (E_n - D_{n-1}) \\ I_n = I_{n-1} + \gamma \cdot D_n \quad P_n = \delta \cdot D_n \quad V_n = I_n + P_n \end{array} \tag{8}$$

The DFG has been generated from (8) and mapped according to the developed methods. The automatically obtained implementation assures the hardware utilization of 86% (Fig. 16).

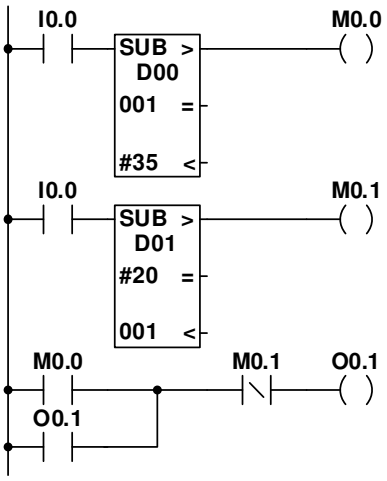

Fig. 17. The LD with mixed arithmetic and logic operations.

A. Bringing together logic and arithmetic operations

Presented algorithms for Boolean operation synthesis and the arithmetic operation synthesis require merging into one the procedure that allows obtaining fully functional hardware implemented logic controller. The concept introduced for intermediate logic operation representation is extended to incorporate the arithmetic expressions. The node set has been enriched with arithmetic operations. The general node represents elementary operations that can belong to logic or arithmetic sets of operations.

An exemplary LD that combines logic and arithmetic operations is presented in the Fig. 17. The LD presents implementation of the controller with the hysteresis. The switch points are determined by constant values in SUB (subtract) modules. This module not only calculates the difference but also delivers the relation of subtracted values. The subtract values are written to the marker space D00 and D01. Markers are not used in calculation process and can be optimized. The compiled diagram is presented in Fig. 18. During the compilation process the DAG of the operation is created. The nodes implement logic as well as arithmetic operations. The scheduling of arithmetic operations is based on described methods. Due to sequential nature of arithmetic operations execution there appear sequential dependencies on Boolean operations. Boolean operations are scheduled one cycle after completing the arithmetic operations they are depend on. The logic operations are always completed in a single cycle.

The arithmetic operations are conditionally triggered. Implementation of this property requires the introduction of the conditional write of results. The nodes that are writing back arithmetic results are conditionally triggered. In proposed implementation conditional enable expression enables write of a result or its propagation to other blocks.

The tree of arithmetic operation can be merged provided that the write operations are controlled by the same expression. In case of tree-like structures and the proposed method of the building data structure comparison of the pointers assures condition equality.

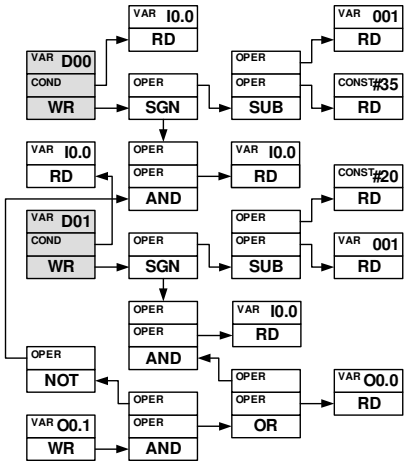

Fig. 18. Compiled data structure of the mixed operations LD network.

VII. SUMMARY

The paper presents the complete set of algorithms for synthesizing the reconfigurable logic controller. The synthesis processes are carried out on standard representation of programs for PLCs. They are based on the ladder diagram (LD) or the instruction list (IL) specifications. Implementation of Boolean operations can be efficiently done in programmable devices. The implementation of Boolean functions has been presented in details at the beginning. The PLCs operation also depends on arithmetic operations. In opposite to logic operations implementing arithmetic operations requires much more resources. The direct approach of the implementation leads to inefficient results. The direct implementation is not able to utilize specific properties of complex functional blocks incorporated into modern FPGAs. There has been proposed algorithm of arithmetic operation mapping into DSP48 blocks. The proposed scheduling operation distributes operation in time and space. It attempts to reduce entire computation time. Finally, the complete synthesis algorithm has been presented. It is able to process logic and arithmetic blocks.

The obtained results of automatic synthesis are very promising. The work over implementation of the package is continued. The based skeleton of the synthesis concept will be tested with different optimization schemes.

REFERENCES

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

[2] G. Michel, *Programmable Logic Controllers - Architecture and Applications*. John Willey & Sons, 1992.

[3] M. Chmiel, E. Hrynkiewicz, and A. Milik, "Concurrent operation of the processors in bit-byte CPU of a PLC," in *Proceedings of IFAC World Congress*, July 2005.

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

[5] M. Chmiel, E. Hrynkiewicz, and A. Milik, "Compact PLC with event-driven program tasks execution," in *Preecedings of 3rd IFAC Workshop on Discrete Event System Design, DESDes'06*, September 2006, pp. 99–104.

[6] S. Devadas, A. Ghosh, and K. Keutzer, *Logic Synthesis*. McGraw-Hill Inc., 1994.

[7] D. Gajski, N. Dutt, A. Wu, and S. Lin, *High-Level Synthesis. Introduction to Chip and System Design*. Kluwer Academic Publishers, 1994.

[8] A. Mishchenko. (2010) Abc: A system for sequential synthesis and verification. [Online]. Available: http://www.eecs.berkeley.edu/alanmi/abc/.

[9] A. Milik and E. Hrynkiewicz, "PID module for reconfigurable logic controller," in *Proceedings of Programmable Devices and Systems 2000 Conf., Ostrava, Czech Rep.*, 2000.

[10] A. Milik, "High level synthesis - reconfigurable hardware implementation of programmable logic controller," in *Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems*, February 2006.

[11] S. Shanta and S. Dipali, "A new generation of PLC-an FPGA based PLC," in *Proceedings of the SICE Annual Conference, SICE 2005 Annual Conference in Okayama*, 2005, pp. 2367–2370.

[12] L. Yadong, Y. Kazuo, F. Makoto, and M. Masahiko, "Model-driven programmable logic controller design and FPGA-based hardware implementation," in *Proceedings of the ASME International Design Engineering Technical Conferences and Computers and Information in Engineering Conference-DETC2005*, 2005, pp. 81–88.

[13] Xilinx, *DS-099, Spartan-3 FPGA Family, ver.2.1*. Xilinx, 2006.

[14] *International Standard IEC 1131, Programmable Controllers*, International Electronics Commission Std. IEC, Geneva, 1992.

[15] J. Welch, "Translating unrestricted relay ladder logic into boolean form." *Computers in Industry*, vol. 20, pp. 45–61, 1992.

[16] S. Ichikawa, M. Akinaka, R. Kieda, and H. Yamamoto, "Converting PLC instruction sequence into logic circuit: A preliminary study," in *Proceedings of IEEE International Symposium on Industrial Electronics*, vol. vol.4, July 2006, pp. 2930–2935.

[17] D. Du, X. Xu, 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*. Springer London, 2010, vol. 49.

[18] D. Du, Y. Liu, X. Guo, K. Yamazaki, and M. Fujishima, "Study on LD-VHDL conversion for FPGA-based PLC implementation," *The International Journal of Advanced Manufacturing Technology*, vol. 40, pp. 1181–1190, 2009.

[19] A. V. Aho, R. Sethi, and J. Ullman, *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[20] S. Akers, "Binary decision diagrams," *IEEE Transactions on Computers*, vol. C-27, pp. 509–516, June 1978.

[21] R. E. Bryant, "Graph based algorithms for boolean function manipulation," *IEEE Transactions on Computers*, vol. C-35, pp. 677–691, August 1986.

[22] S.-I. Minato, *Binary Decision Diagrams and Applications For VLSI CAD*. Kluwer Academic Publisher, 1995.

[23] Xilinx, *UG073, XtremeDSP for Virtex-4 FPGAs User Guide*. Xilinx, 2007.

[24] ——, *UG389, Spartan-6 FPGA DSP48A1 Slice*. Xilinx, 2009.

[25] ——, *DS302, Virtex-4 FPGA Data Sheet: DC and Switching Characteristics*. Xilinx, 2007.

[26] ——, *DS162 Spartan-6 FPGA Data Sheet: DC and Switching Characteristics*. Xilinx, 2011.

[27] G. Hachtel and F. Somenzi, *Logic synthesis and verification algorithms*. Springer, 1996.

[28] S. Hassoun and T. Sasao, *Logic synthesis and verification*. Kluwer Academic Publisher, 2002.

[29] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, "Combinational and sequential mapping with priority cuts," in *Precedeenigs of Computer Aided Design Conference*. IEEE, 2007, pp. 354–361.

[30] R. J. Bibero, *Microprocessors in Instruments and Control*. John Willey & Sons, 1990.