PERGAMON

Annual Reviews in Control 27 (2003) 55–61

Annual Reviews in Control

www.elsevier.com/locate/arcontrol

Review

# Implementation of safety critical logic controller by means of FPGA

Marek Wegrzyn*

*University of Zielona Góra, Institute of Computer Engineering and Electronics, ul. Podgórna 50, 65-246 Zielona Góra, Poland*

**Abstract**

In the paper, a synthesis method of logic controllers for safety critical systems into field programmable logic is presented. The specification of the system in the form of symbolic, *if-then* or *if-then-else* conditional decision rules is used for directly mapping to netlist format, or transformed into a HDL format that is accepted by standard FPGA simulators and synthesis tools. In addition, models of Logic Controllers are verified using the well-developed Petri net theory, and then they are translated through automated processes into selected FPGA specification format.


*Keywords:* Process control; Programmable logic controllers; Petri nets; Rule-based systems; Digital systems; Logic arrays; FPGA; HDL

## 1. Introduction

The main aim of this paper is a presentation of two methods of synthesis of Logic Controller (LC) programs for Programmable Logic (PL) for safety critical systems.

The behavioral specification of Logic Controller programs (Halang & Jung, 1994), related with subset of IEC 1131-3 standard, is modeled as Petri net. The textual, rule-based description of Petri net, called Petri Net Specification Format version 2—PNSF2 (Wegrzyn & Adamski, 1999), is used as an entry format for dedicated CAD tool. In another approach, controllers are described by state tables, and then they are modeled in HDL. Finally, in both methods, LCs are implemented into PL as a dedicated Reprogrammable Logic Controllers.

In the first of presented methodologies, a rule-based behavioral specification of Petri net model is directly transformed into hardware library (on the netlist level) by means of Xilinx implementation system (Wegrzyn & Adamski, 1999; Wegrzyn, Adamski, & Monteiro, 1998). The *if-then* decision rules (non-procedural conditional statements) are mapped into Xilinx XNF format after some simple additional symbolic transformations. From the conceptual point of view, a set of event-oriented (Petri net transition-oriented) *if-then* conditional statements (Adamski, 1999) is easily replaced by the equivalent local state-oriented (Petri net place-oriented) *if-then-else* decision rules. In such a way, the Petri net image in FPGA structure is straightforward and simple. Standard design tools perform the final multi-level, combinational optimization, placement and routing.

A control unit, represented by state table (or decision table), can be also behaviorally specified by means of using hardware description languages (HDLs). Its specification, modeling, validation and FPL synthesis can be directed towards modern, well-accepted VHDL or Verilog-based synthesis tools. The second proposed method of structured implementation of state table is based on self-evident mapping of decision rules into HDL. The *if-then-else* conditionals may serve as both flexible and formal intermediate forms.

## 2. Background and motivation

### 2.1. Field programmable gate arrays

Field Programmable Logic is an electronic circuit, which can be configured by a user to perform particular combinational or registered logic functions. The design process is greatly simplified by FPL compilers by some new CAD tools, which are recently available, like Xilinx Foundation Express (supporting VHDL and Verilog). The effective simulation allows the Logic Controller to be debugged before the device is programmed. If a design change is needed, it is a simple matter to reedit the original specification and then re-program or exchange the old device. The most flexible and high density FPL devices are Field Programmable Gate Arrays (FPGAs). Fig. 1 depicts a conceptual diagram of a typical FPGA.

* Tel.: +48-68-328-2484; fax: +48-68-324-4733.
*E-mail address:* M.Wegrzyn@iie.uz.zgora.pl (M. Wegrzyn).

1367-5788/$ – see front matter © 2003 Elsevier Ltd. All rights reserved.
doi:10.1016/S1367-5788(03)00007-5

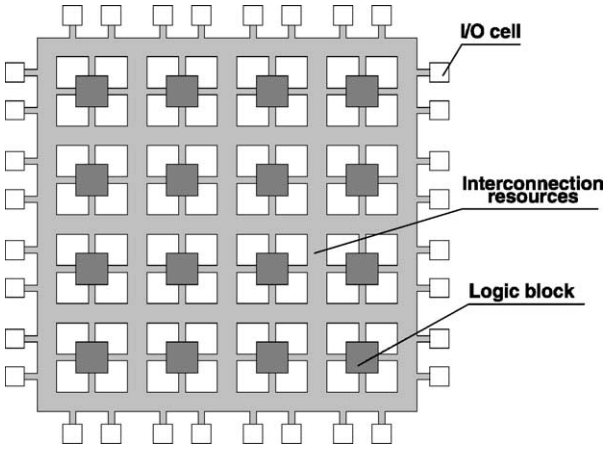


Fig. 1. A conceptual FPGA.

FPGAs were introduced in 1985 by the Xilinx company, and since then, many different FPGAs have been developed by a number of companies: Actel, Altera, Plessey, AMD, QuickLogic, Algotronix, Concurrent Logic, Crosspoint Solutions, and others. As shown in Fig. 1, FPGA consists of a two-dimensional array of logic blocks that can be connected by general interconnection resources. The interconnection comprises segments of wire. The segments may be of various lengths. In the interconnection there are programmable switches that enable one to connect the logic blocks to the wire segments, or one wire segment to another. Logic circuits are implemented in the FPGA by partitioning the logic into individual logic blocks, and then interconnecting the blocks as required via the switches (Xilinx, 2003).

To facilitate the implementation of a wide variety of circuits, it is important that an FPGA is as versatile as possible. This means that design of the logic blocks, coupled with that of the interconnection resources, should facilitate the implementation of a large number of digital logic circuits. There are many ways of designing an FPGA, involving trade-offs in the complexity and flexibility of both logic blocks and interconnection resources. Logic-block architectures can be designed in many different ways: as simple gates (e.g. 2-input NAND gates), or as more complex structures, such as multiplexors or look-up tables. In some FPGAs, a logic block corresponds to an entire PAL-like structure. Fig. 2 shows a Xilinx Virtex FPGA logic block with more details. The variety and complexity of currently produced FPGAs make them suitable for use in a wide set of applications. Two significant advantages of FPGAs are as follows: lower prototype costs and shorter production time.

As user-programmable application-specific integrated circuits, they provide a valuable compromise which combines the benefits of standard microcontrollers with many of the benefits of other semi-custom logics. The design process is greatly simplified, due to the powerful FPGA compilers, as well as some new modern and effective CAD tools, which

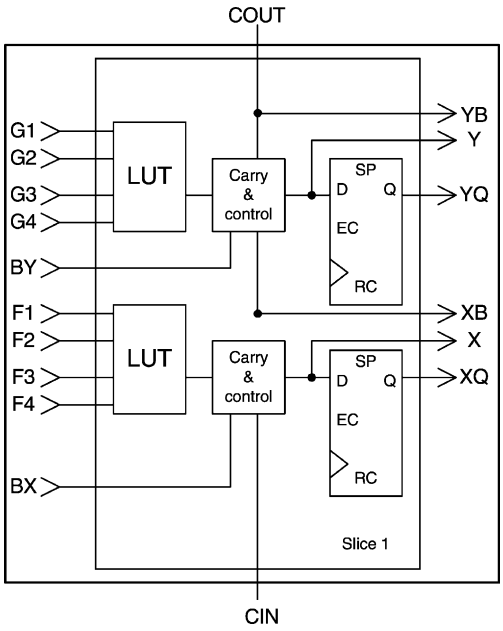


Fig. 2. Example of a logic block structure.

have recently become available. The effective simulation allows a logic controller to be simulated and debugged before the device is programmed. If design changes are needed, it is a simple matter to re-edit the original specification and then re-program the former device. It could be expected that after some time industrially oriented complex PLD and FPGA would be introduced. Advanced PLDs or FPGAs, together with modern CAD tools, may easily replace some of the old fashioned programmable controllers (Mandado, Marcos, & Perez, 1996). The Xilinx design software is called Xilinx Foundation Series, and it supports design entry, design simulation, and design implementation. Design entry occurs by schematic capture or HDL specification.

However, devices address the issues that are essential to the safety critical systems market (e.g. aerospace and defense):

- Commercial manufacturing strengths result in more efficient process flows.
- Reliability of supply. Controlled mask sets and processes insure the same quality devices, every time, without variation, which remain in production for an extended time.

The Xilinx QPRO family of ceramic and plastic QML products (Qualified Manufacturers Listing), certified to MIL-PRF-38585 and complemented by ISO-9000 certification, provides system designers with advanced programmable logic solutions for next generation designs. The QPRO family also includes select products that are radiation hardened for use in satellite and other space applications. Designers can confidently design with Xilinx for High-Reliability systems with the knowledge they are getting unsurpassed quality and reliability, and long-term commitment to the safety critical applications market.

The QPRO family provides a wide variety of devices. The broad range of devices is available in various speed and package options. Both military temperature and full QML/SMD versions are available as standard off-the-shelf products. The Virtex members of the QPRO family offer FPGAs with high-performance up to 200 MHz, and densities greater than 3,000,000 system gates, and even larger devices planned for the future. Dramatic increases in silicon efficiency result from optimizing the new architecture for place-and-route efficiency and exploiting an aggressive 8-layer-metal 0.15 μm CMOS process. These advances make QPRO Virtex FPGAs powerful and flexible alternatives to mask-programmed gate arrays.

### 2.2. Safety critical control applications

Programmable electronic systems for safety critical control and regulation applications are rather a new scientific and engineering research field that stands at the beginning of its treatment.

Control unit of digital circuits can be described in several forms. However, for safety critical controls only very-well verified methods can be applied (Cullyer & Pygott, 1987), e.g. interpreted Petri nets, cause/effect tables, decision tables, and HDL models.

Control oriented, 1-bounded Petri net is considered as a formal model for the set of decision rules. It is used also as a distributed, local state-based model for the generated HDL descriptions. In the considered interpreted Petri net model, a particular combination of Boolean external inputs, and eventually some internal conditions have to be true, for the transition to be fired.

Conditional logics is treated as an intermediate behavioral level language for the description of concurrent digital systems. Implicitly, it gives the *procedural* specification of distributed, concurrent digital system, implemented in FPL as a single State Machine with Data Path. The global states (Petri net markings) of Concurrent State Machine (direct hardware implementation of Petri net) are formed from local states (Petri net places). The places are encoded in such a way, that the global state code is implicitly obtained by superposition of the local state codes. A rule-based specification, considered in the paper, is composed from discrete local state symbols (places $P$), input signal symbols ($X$) and output signal symbols ($Y$) of the controller. The name of transition ($t$) is treated only as a rule label.

The textual description of developed implementation in Field Programmable Logic is not procedural, as Petri net evidently is, but rather *declarative*. The designed control part of a digital system is treated as a reasoning system implemented in the regular structure (for example, array) that is built from logic cells. The decision rule specification, as a textual equivalent of Petri net, is easily represented by using HDLs, and then mapped into hardware. The manual or automatic translation is straightforward.

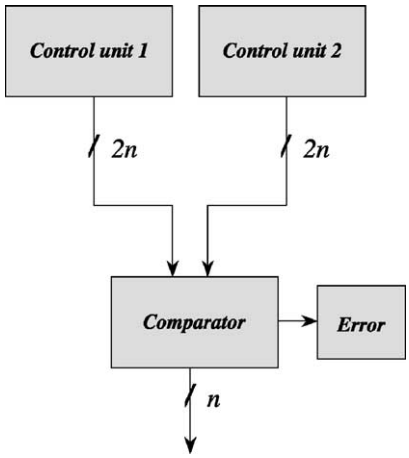


Fig. 3. General architecture of the controller.

The simplest and most effective technique for state encoding of state machines for FPGAs implementation is one-hot state assignment. On the other hand, as a Petri net place encoding there is used one-to-one mapping of places onto flip-flops (Wegrzyn et al., 1998).

## 3. Programmable logic-based design

### 3.1. General architecture of the controller

Safety critical applications require very rigorous design process and its realization. Each step should be formally verified. For this purpose, two models are used: Petri net model and HDL model. Implementation of those compatible blocks (Fig. 3) is based on such models. For better testing of correctness each output is in complementary form, i.e. affirmation $Y$ and negation $\overline{Y}$. Comparator compares outputs from two control units; the same values on both $Y$ and $\overline{Y}$ outputs indicate error.

The final outputs are provided from the comparator. Higher reliability could be met by using separate chips for each control unit.

### 3.2. Example

Fig. 4 depicts the controlled part of a designed simple reactive system. The controlled system consists of two carriages **1** and **2** that go on the left (***L***) and right (***R***) sides. The carriages start on signals $M1$ and $M2$, respectively. When both carriages go concurrently, they can reach points $D$ and $E$. Because, the carriage **1** has higher priority, it goes to point $B$, and then goes back to $A$, where it waits for next pressing of bottom $M1$. In this time, carriage **2** waits in point $E$ until carriage **1** reaches point $D$ in return way. Then the carriage **2** goes to $B$ and back to $C$, where it waits for next pressing of bottom $M2$. At the time, on the common way only one carriage can be located.

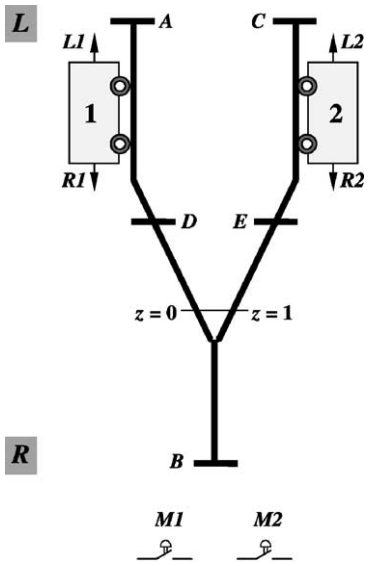


Fig. 4. The controlled parts of reactive system.

### 3.3. Controller design

In this section, the specification of industrial controller that is related with interpreted Petri net (Fig. 5), and state or decision table (Table 1), is developed.

There is one fundamental difference between a state diagram and a state table, a difference that makes state diagram design simpler but also more error prone (Wakerly, 1994):

- A state table is an exhaustive listing of the states for each state/input combination. No ambiguity is possible.
- A state diagram contains a set of arcs labeled with transition expressions. Even when there are many inputs, only one transition expression is required per arc. However, when a state diagram is constructed, there is no guarantee that the transition expressions written on the arcs leaving a particular state cover all the input combinations exactly one.

This difference is very important for safety critical systems.

In the considered Field Programmable Logic implementation, transitions between local states occur simultaneously. It is realized by means of using the same clock for all logic blocks (synchronous state machine). To implement concurrent state machine, a global state register can be used to reflect all current, concurrently related, local states. As an alternative option it is possible to distribute the local state register into several single logic blocks. In such a case, each Petri net place may be implemented separately as two local states: active—if the place is marked, passive—if the place is empty.

The direct mapping of Petri net into FPL is based on correspondence between a transition and a simple combinational circuit and correspondence between a place and a clearly defined subset of state register. In dealing with concurrency, the designer encounters some problems that will not arise in the logic synthesis of sequential systems. One of them is the concurrent local state encoding.

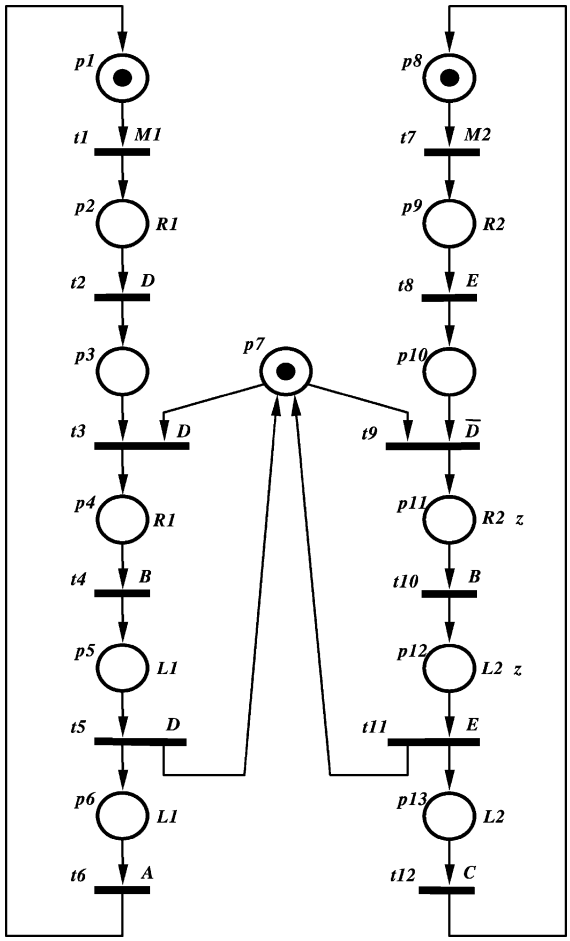


Fig. 5. Petri net model.

Table 1
Decision table for control unit

| Transition | Current local states | Conditions | Next local states |
|---|---|---|---|
| $t1$ | $p1$ | $M1$ | $p2$ |
| $t2$ | $p2$ | $D$ | $p3$ |
| $t3$ | $p3$, $p7$ | $D$ | $p4$ |
| $t4$ | $p4$ | $B$ | $p5$ |
| $t5$ | $p5$ | $D$ | $p6$, $p7$ |
| $t6$ | $p6$ | $A$ | $p1$ |
| $t7$ | $p8$ | $M2$ | $p9$ |
| $t8$ | $p9$ | $E$ | $p10$ |
| $t9$ | $p7$, $p10$ | $!D$ | $p11$ |
| $t10$ | $p11$ | $B$ | $p12$ |
| $t11$ | $p12$ | $E$ | $p7$, $p13$ |
| $t12$ | $p13$ | $C$ | $p8$ |

The one-hot-like method produces fast designs with a simple combinational part, especially for implementations in FPGA (Wegrzyn et al., 1998).

It is not possible to assume that all flip-flops attached to places, are set to 0 since several places can be marked

```
.clock CLK

.inputs M1 M2 A B C D E
.comb_outputs L1 L2 R1 R2 z

.part CarriageControl
.places p1 p2 p3 p4 p5 p6 p7
.places p8 p9 p10 p11 p12 p13
.transitions t1 t2 t3 t4 t5 t6
.transitions t7 t8 t9 t10 t11 t12
.net
 t1: p1 * M1 |- p2;
 t2: p2 * D |- p3;
 t3: p3 * p7 * D |- p4;
 t4: p4 * B |- p5;
 t5: p5 * D |- p6 * p7;
 t6: p6 * A |- P1;
 t7: p8 * M2 |- p9;
 t8: p9 * E |- p10;
 t9: p10 * p7 * !D |- p11;
 t10: p11 * B |- p12;
 t11: p12 * E |- p13 * p7;
 t12: p13 * C |- p8;
.MooreOutputs
 p2 |- R1;
 p4 |- R1;
 p5 |- L1;
 p6 |- L1;
 p9 |- R2;
 p11 |- R2 * z;
 p12 |- L2 * z;
 p13 |- L2;

.marking p1 p7 p8
.e
```

Fig. 6. A part of PNSF2 specification.

simultaneously. Some efficient state assignment techniques are referred to or presented in (Adamski, 1999). They are suited to FPGAs, that are register-reached. The encoding procedure assigns to each place or macroplace a Boolean term. This is combined from all Boolean terms, which are assigned to higher-level macroplaces it belongs to, and a particular 'private' part. The state variables could be eventually shared among subnets but all concurrent subnets should not have orthogonal codes (Adamski, 1999).

The use of concurrent state machine techniques in FPL design has several advantages. They create nearoptimal controller implementations (in terms of performance), tailored to the exact requirements. Practical designs often fit in a single FPGA package since there are no redundant elements. Any particular function, which is not included in the library, can be effectively implemented. Controller outputs respond to inputs with high speed, in predictable and repeatable way, without glitches. The behavioral specification, which is mapped in the programmable logic can be formally verified or validated by means of advanced CAD tools.

The standard CAD tools have been combined with dedicated design environment PeNCAD (Wegrzyn et al., 1998). As an intermediate format the rule-based textual specification, called PNSF2, is used (Fig. 6). The design experiments have been developed by means of using HDL simulators and synthesis tools and Xilinx implementation tools, and tested on Xilinx FPGA demoboard with XC4005E device.

Application Specific Logic Controllers are dedicated devices embedded within an application and implemented with programmable logic (Mandado et al., 1996; Wegrzyn et al., 1998). To define the behavior of reactive system, it is necessary to specify the set of allowed sequences of conditions, actions and events. The importance of this approach lies in its portability to any VHDL simulator, for example, Aldec Active-CAD. It makes possible to validate the indented behavior of the designed microsystem in an user-friendly environment. For example, VHDL description of the logic controller is validated using "testbenches", which models roughly the controlled subsystem behavior.

The Reprogrammable Logic Controller is considered as an abstract reasoning system (rule-based system) implemented in reconfigurable hardware (FPGA). The mapping between inputs, outputs and local internal states of the system is described in a formal manner by means of logic rules (represented as sequents) with some temporal operators, especially with operator *next* (Adamski, 1999). The correctness preserving synthesis, based on Gentzen calculus, is treated as a formal transformation of the initial set of compound rules (*Specification*) into another set of compound rules (*Implementation*). On the other hand, the decision table can be prepared (Table 1).

The table described all transitions between local states of the controller. Each row of the table is implemented as the *if-then* conditionals:

$$t3 : \text{if } ((p3 \times p7) \text{ and } (D)), \quad \text{then } p4;$$

Outputs of the controller are shown in Table 2.

Example implementation of the output $R1$ is based on the rule:

$$\text{if } (p2 \text{ or } p4), \quad \text{then } R1;$$

Such approach is more suitable for safety critical systems because it is simpler and easy to verify on either RTL or gate levels. In addition, there is a simple transformation for implementation of a system described by a cause/effect table. For such specification, a ROM-based approach is performed and easy realized by using look-up table (LUT) based logic block FPGAs (Xilinx, 2003).

Directly mapping into netlist format, which is based on Petri net model or decision table, has an additional advantage. Setting of the controller into the stable and safe state (e.g. on an accident) should be realized by resetting of all flip-flops. That action is easily implemented in such approach in FPGA devices.

The *one-to-one* (*one-hot* like) encoding of Petri net is treated as the simplest case of more general mapping. The

Table 2
Outputs table

| Local state | Outputs | Local state | Outputs |
|---|---|---|---|
| $p2$ | $R1$ | $p12$ | $L2, z$ |
| $p4$ | $R1$ | $p13$ | $L2$ |
| $p5$ | $L1$ | $p9$ | $R2$ |
| $p6$ | $L1$ | $p11$ | $R2, z$ |

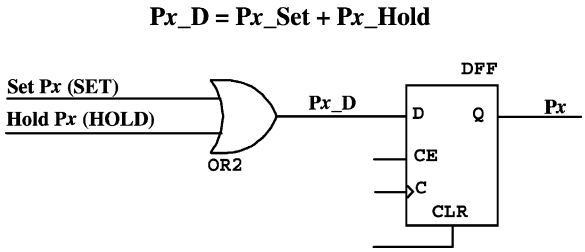


Fig. 7. Implementation of a place.

one-hot method (Fig. 7) produces fast designs with a simple combinational part, especially for implementations in FPGA. It is not possible to assume that all flip flops, except one, are set to 0 since several places can be marked simultaneously.

### 3.4. HDL modeling and verification of controllers

VHDL and Verilog are the standards of electronic hardware description languages. They are increasingly used to describe digital hardware designs that are applicable for both simulation and synthesis. In addition, these HDLs are commonly used as an exchange medium, e.g. between tools of different vendors. Today many simulation systems and other tools (synthesis, verification and others) based on HDL are available. The HDL users community is growing fast.

HDL modeling of Petri net is a current research topic presented in papers, e.g. VHDL modeling in (Adamski, 1999). Petri nets, as a logic control program of safety-critical system, can also be modeled using well-known hardware description languages (HDLs), particularly with Verilog or VHDL. There is only a simple direct mapping (Wegrzyn & Adamski, 1999) from the introduced behavioral models into VHDL and RTL model in Verilog. In Fig. 8, a part of Verilog model is shown.

Security and reliability are two of the most significant factors for the controller. In the proposed approach, two models are prepared in different languages, i.e. VHDL and Verilog. In addition, there are two different description levels applied: behavioral and RTL. The correspondence of the results of independent simulations is essential for this stage of analysis to be approved. However, some additional methods of verification are required.

A schema of simulation with different controller models is shown in Fig. 9.

### 3.5. Functions and functional blocks implementation

There are additional advantages by using FPGAs. In FPGAs there are possibilities for realization either control unit or data path (Mandado et al., 1996; Halang & Adamski, 1997). Examples of standard functions and functional blocks are shown in Table 3. Easy-to-read models are

```
`define No_Places 13
`define P1 CurrentState[1]
`define P2 CurrentState[2]
//...
`define P12 CurrentState[12]
`define P13 CurrentState[13]

`define T1 M1
`define T2 D
//...
`define T12 C

 module controller (clk, reset,
                    M1, M2, A, B, C, D, E,
                    L1, L2, R1, R2, z);
   input  clk, reset;
   input  M1, M2, A, B, C, D, E;
   output L1, L2, R1, R2, z;

   reg L1, L2, R1, R2, z;
   reg [`No_Places:1] CurrentState;

 //Set marking
 // Place 1 - initial marking
   always @(posedge clk)
      begin
        if (reset)  `P1 <= 1;
        else `P1 <= (`P1 & ~`T1)
                    | (`T6 & `P6);
      end
//...
// Place 13
   always @(posedge clk)
      begin
        if (reset)  `P13 <= 0;
        else `P13 <= (`P13 & ~`T12)
                    | (`T11 & `P12);
      end

 //Set outputs
 always @(CurrentState)
   begin
      if (`P2==1'b1) R1=1'b1;
      if (`P4==1'b1) R1=1'b1;
      if (`P5==1'b1) L1=1'b1;
     //...
   end
 endmodule
```

Fig. 8. A part of Verilog model of controller.

suitable for documentation and simulation purpose. However, for synthesis purpose a more detailed model should be prepared in order to make possible a verification of the realization.

Table 3
Examples of functions and functional blocks

| Function | Description |
|---|---|
| NOT | Inverter |
| MUX8 | 8-Input multiplexor |
| SHL | Left shift |
| ROL | Left shift with rotation |
| CTUD | Counter up/down |

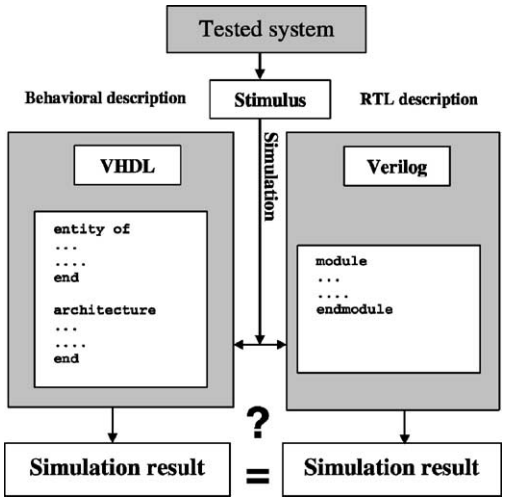


Fig. 9. Schema of different approaches to simulation.

## 4. Conclusions

This paper has been presenting an approach for modeling and design of discrete logic controllers, which is based on Petri nets, and decision table. The proposed methodology combines the well-known theory of Petri net and its formal verification, with mapping of the net into reprogrammable logic devices. The Petri net is directly mapped into PL without explicit enumeration of all possible global states. The specification is given in terms of local state changes. The proposed method of Petri net-based realization of Logic Controllers considers explicit hierarchy included in netlist formats, or hierarchical constructs of VHDL or Verilog. The syntactic and semantic compatibility of Petri net descriptions and decision rules with structured netlist or HDLs statements are as close as possible.

The alternative method of HDL modeling, verification and synthesis of controllers for safety critical systems has been considered. Such an approach gives opportunity for correctness checking at each design stage.

The standard software, like HDL-based, could be more flexible and useful than old fashionable dedicated PLC development systems. HDL models, for simulation on several related levels of design, are applied.

## References

Adamski, M. (1999). Application specific logic controllers for safety critical systems. *Proceedings of the 1999 IFAC Triennial World Congress* (Vol. Q, pp. 519–524). Beijing, China.

Cullyer, W. J., & Pygott, C. H. (1987, May). Application of formal methods to the VIPER microprocessor. *IEE Proceedings*, *134*(Pt. E, No. 3), 133–141.

Halang, W. A., & Jung, S.-K. (1994). A programmable logic controller for safety critical systems. *High Integrity Systems*, *1*(2), 179–193.

Halang, W. A., & Adamski, M. (1997). A programmable electronic system for safety related control applications. In *Proceedings of the International Conference on Safety and Reliability, ESREL'97* (pp. 349–356), Lisbon, Portugal. European Safety and Reliability Association (ESRA), July 17–20, 1997.

Mandado, E., Marcos J., & Perez, S. A. (1996). *Programmable logic devices and logic controllers*. London: Prentice-Hall.

Wakerly, J. F. (1994). *Digital design principles and practices*. Englewood Cliffs, NJ: Prentice-Hall.

Wegrzyn, M., Adamski, M., Monteiro, J.L. (1998). The application of reconfigurable logic to controller design. *Control Engineering Practice (Pergamon)*, *6*, 879–887.

Wegrzyn, M., & Adamski, M. (1999). Hierarchical approach for design of application specific logic controller. In *Proceedings of the IEEE International Symposium on Industrial Electronics ISIE'99* (Vol. 3, pp. 1389–1394), Bled, Slovenia, July 12–16, 1999.

Xilinx. (2003). *The Programmable Logic Data Book*, http://www.xilinx.com/partinfo/databook.htm.