jet INTL JOURNAL OF ELECTRONICS AND TELECOMMUNICATIONS, 2013, VOL. 59, NO. 4, PP. 351–356
Manuscript received November 3, 2013; revised December, 2013. DOI: 10.2478/eletel-2013-0042

Partial Reconfiguration in the Field of Logic Controllers Design

Michał Doligalski and Arkadiusz Bukowiec

***Abstract*—The paper presents method for logic controllers multi context implementation by means of partial reconfiguration. The UML state machine diagram specifies the behaviour of the logic controller. Multi context functionality is specified at the specification level as variants of the composite state. Each composite state, both orthogonal or compositional, describes specific functional requirement of the control process. The functional decomposition provided by composite states is required by the dynamic partial reconfiguration flow. The state machines specified by UML state machine diagrams are transformed into hierarchical configurable Petri nets (HCfgPN). HCfgPN are a Petri nets variant with the direct support of the exceptions handling mechanism. The paper presents places-oriented method for HCfgPN description in Verilog language. In the paper proposed methodology was illustrated by means of simple industrial control process.**

***Keywords*—HCfgPN, UML state machine diagram, Verilog, logic controller**

I. Introduction

DEVICE implementation of reconfigurable logic controllers (RLCs) with the use of Field Programmable Gate Array devices (FPGA) is a quite commonly used solution [1]–[3]. Modern Integrated Circuits (ICs), including FPGA devices, provide opportunity for implement faster, bigger and safe control systems. Improved techniques for design should follow the growing technological capabilities, allowing for it full use. Dynamic Partial Reconfiguration is a good example of powerful functionality [4], [5]. It provides the possibility of multi context design, where contexts are switched when other blocks of the device are turned on.

The paper presents modular approach for Logic Controllers (LCs) developing based on UML state machines and Petri nets. Both models are accepted form of logic controllers specification [6]. Full and coherent modularity on each stage of developing process simplifies and streamlines the design of LCs oriented for partial reconfiguration [7], [8]. The application of Partial Reconfiguration in the field of LCs, improve its quality not only by improving the functionality but also by reducing power consumption and minimization of the allocated resources. The application of the modern specification technics, UML language in particular [9], will simplify partial reconfiguration oriented design. Petri nets give the possibility for logic controller formal verification [10], for example by means of Gentzen reasoning techniques [11]–[13].

The research was financed from budget resources intended for science in 2010–2013 as an own research project No. N N516 513939.

M. Doligalski and A. Bukowiec are with the Computer Engineering & Electronics Department, University of Góra, ul. Licealna 9, 65-417 Zielona Góra, Poland (e-mails: {m.doligalski; a.bukowiec}@iie.uz.zgora.pl).

Context switching techniques presented in the paper, provide adaptation of control algorithm of the production process. Context is the alternative specification (version) of particular function provided by logic control algorithm. Alternative context is required when production process has two or more alternative formulas, e.g. ingredients may be in the form of raw material or semi-finished product. It's not necessary to remodel whole control algorithm but only some parts should be adjusted to actual conditions.

II. Partial Reconfiguration

Classical design flow of LCs implemented in the FPGA devices results preparation of one final `.bit` or `.mcs` configuration file. In this approach full static reconfiguration requires reload of whole FPGA device configuration which causes interrupt control. There is no possibility for changing context of the control algorithm without direct context switching implemented within logic controller. It is very possible that at the specification stage, not all contexts will be known. Also alternative context may be introduced after the control system deployment. So it will be no possible to implement all alternative contexts in one control algorithm. The other issue of direct context implementation results from the size of the algorithm. Each context will increase space required for the implementation. Partial reconfiguration (PR) is the feature of modern FPGA devices that provides control switching. Some parts of the devices may be reconfigured, when others will be preserved. The paper will consider dynamic reconfiguration when changes in the configuration are provided in run-time. In the Difference based partial reconfiguration (DBPR) oriented developing process, two configurations are compared and, as a result, differential configuration file is generated. This approach is dedicated rather for small design changes e.g. Block Ram contents modification [14], [15].

Full benefits from the partial-oriented design comes with modular based partial reconfiguration (MBPR). At the beginning static (SP) and reconfigurable partitions (RP) are identified within the design. For each RP, set of alternative reconfigurable modules (RMs) is developed. The bottom-up synthesis is performed and each module is synthesized separately. Finally top-level module connect both static and reconfigurable modules. During dynamic module-based partial reconfiguration, structure of separated reconfigurable partition may be changed without affecting the operation of other modules. The paper presents revised approach for PR design of logic controllers. Previous methodology increase design complexity [16]. It requires the application of *bus macro*

between reconfigurable module and other modules. Bus macro was an interface between dynamic modules and other modules (both static and dynamic), implemented by means of three-state buffers. In the new modern FPGA devices and updated CAD software (Plan Ahead) it's not required to add bus-macro inside the design. The management of the partitions and its configurations (reconfigurable modules) is transparent and intuitive.

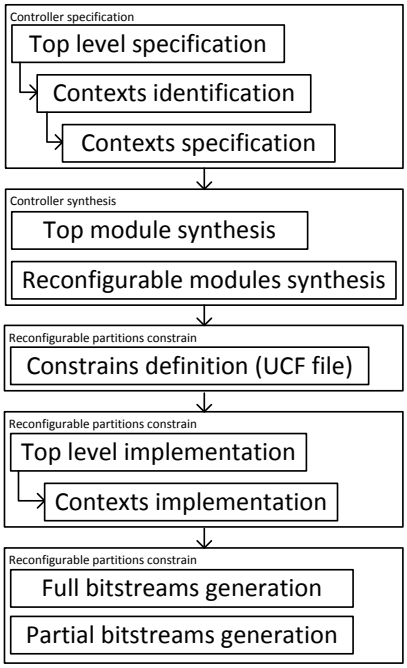

Fig. 1. Partial reconfiguration design flow.

Figure 1 presents general outline of Partial reconfiguration flow. At the controller specification stage top level specification is elaborated and static modules are specified as well. Next contexts are identified and specified. In the proposed approach controller is specified by means of UML state machine diagram. Specification is transformed into Petri net model (HCfgPN) and next synthesized using HDL language and CAD tools into EDIF or NGC format. Controller specification and synthesis was described in section III in details. Synthesis of top module and reconfigurable modules may be performed concurrently. Reconfigurable modules within top level module are described as black boxes at the synthesis stage. For each reconfigurable partition (RP) area constrains should be described by means of UCF file. It is necessary to guarantee that the biggest reconfigurable module (RM) will fit respective RP. Both static and reconfigurable parts of the logic controllers should be implemented. Static partitions should be implemented only once and next, to reduce time of development, promoted to other configurations (contexts). The PR flow results in set of `.bit` files both for full and partial configuration.

III. Specification and Synthesis

UML state machine and Hierarchical Configurable Petri Net are two component models of the Dual Specification (DS) [17], [18]. Modularity of the DS, simplify functional decomposition of the control system. Each composite state or macroplace is responsible for particular functionality implementation. Each functional block is implemented as separated module and next may be reconfigured independent.

The proposed design flow will be illustrated by simple production process. The process consists in measuring out two liquid substances in containers *A* and *B* and then mixing them in the reactor vat (Fig. 2). A finished product is poured through draining valve into the containers. During the execution of the production process there may appear both warning exceptions (*defect*) and critical ones (*failure*). A temporary freezing of the process performed from an operator's panel, for example, in order to carry out a visual inspection of the reactor, can be an example of a warning-type exception. This exception does not affect the manufacturing process and, this is why, it can be resumed (reactivated).

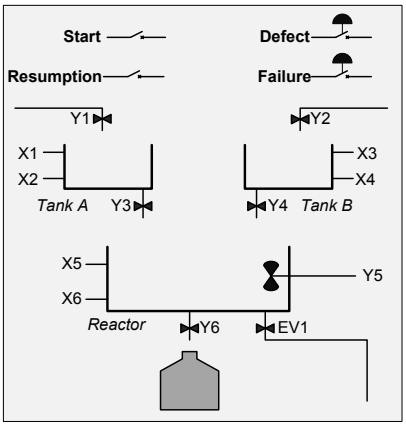

Fig. 2. Production process scheme with exception handling.

The difference between critical and noncritical exception handling is that after a critical exception preempted process can't be resumed and emergency procedure should be performed immediately. Exception that allows the resumption of non-critical control processes is noncritical, preempted part of control process can be resumed. This approach is natural in the field of control algorithms specified by means of UML state machine diagrams, however there is no direct support for exception handling in Petri net model.

UML state machine diagrams for given control process was presented in Fig. 3 and Fig. 4. There are two levels of hierarchy. First one includes main state machine, second one includes submachines included in *CS2* and *CS3* composite states. Composite state *CS2* consists of two concurrent regions. Each region consists one state [sub]machine. These submachines were marked by history pseudostate, it means that for state *CS2* noncritical exception handling mechanism was enabled.

After (noncritical) preemption control algorithm will handle the exception in state *Defect alarm* by action *do/Alarm*. Signal *Alarm* indicates abnormal situation by flashing lights on control panel. Substrates in *Tank A* and *Tank B* are not yet mixed, after visual inspection if there are no contaminants the process can be resumed. If submachine starts from history pseudostate, after resumption last active state will be reactivated.

The example of critical exception handling was presented in case of preemption form state *CS3*. Three concurrent submachines contain regions that have been marked by an initial pseudostate.

Critical exceptions thrown in *CS3* state will be handled in the *Emergency emptying* state. It is an emergency procedure – substrates were mixed partially and process cannot be continued. The reactor must be emptied from the contents by the emergency valve *EV1*. After proper emptying, state machine will change state to *Failure alarm* and wait for resumption.

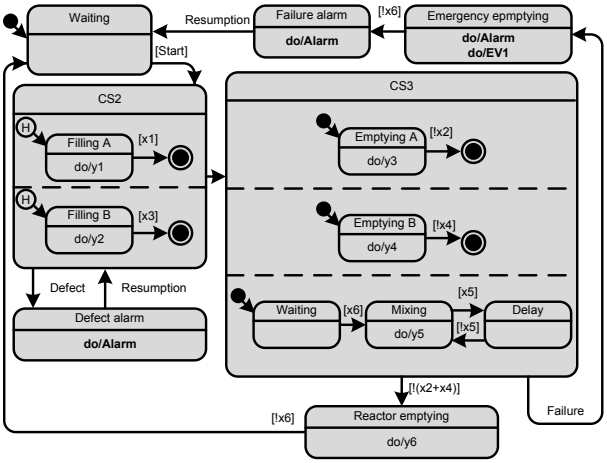

Fig. 3. Logic controller behavioral specification – context A for composite state 3.

In this example, composite state *CS3* has two contexts A and B, where measuring containers are emptied in a different order. Context A describes parallel and context b sequential emptying. The specification in the form of UML state machine diagram presented in the Fig. 3 and Fig. 4 can be directly transformed into hardware description language [19]. Such approach requires UML state machine formal verification, however Petri nets offer wider range of methods. In the proposed design methodology specification in the form of state machine is transformed into hierarchical configurable Petri net model (HCfgPN) [20].

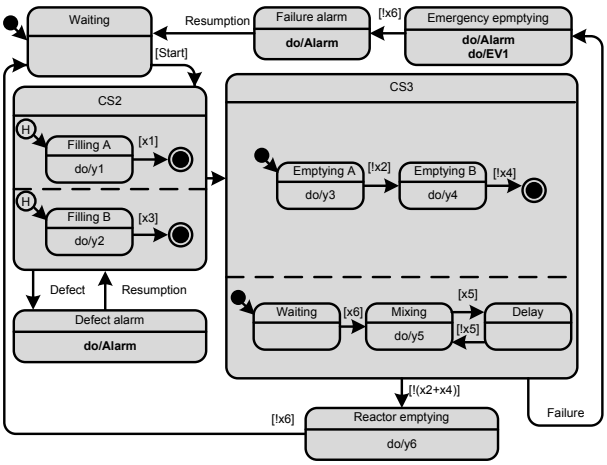

Fig. 4. Alternative context (B) for composite state 3.

Top level state machine elaborated at the specification state was transformed into *Subnet 1*. There is no exception handling at top level of state machine diagram so transitions T^{fin}, T^{i}, T^{a}, T^{w} can be omitted. For a better understanding of the net functioning, transitions in Fig. 5 have been kept and the condition false was assigned permanently.

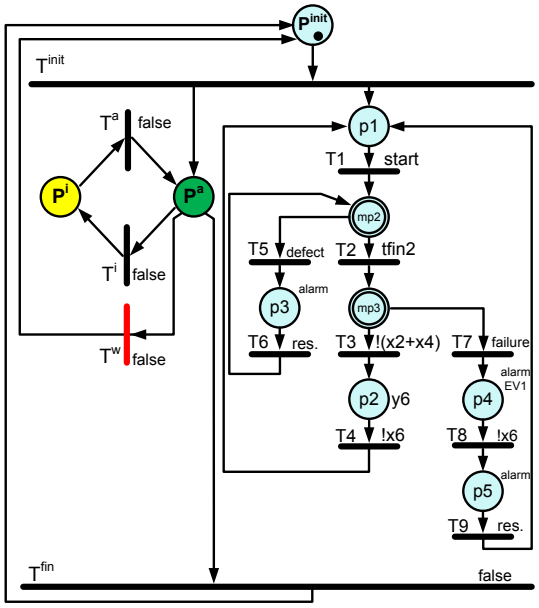

Fig. 5. Subnet no. 1 for top level state machine.

Subnet 1 contains two macroplaces: *mp2* and *mp3*. These macroplaces correspond to composite states *Cs2* and *Cs3* and are linked with *Subnet 2* and *Subnet 3* respectively. *Subnet 2* implements noncritical exception handling. Transition T^{init} firing activate *Subnet 2*. Preemption condition was assigned to transition T^{i}, resumption condition to transition T^{a}. Both conditions *t5* and *t6* are signals from supernet (*Subnet 1*) and in fact they are the logical condition of firing transitions *T5* and *T6*.

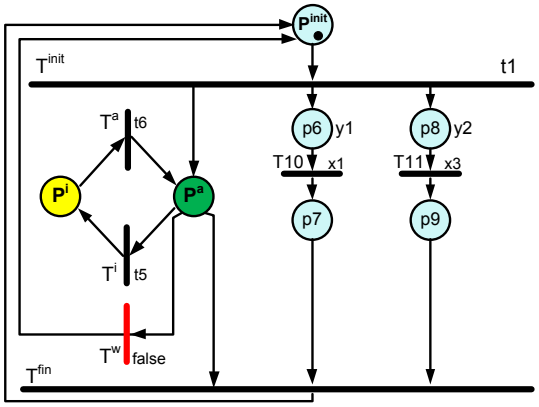

Fig. 6. Subnet no. 2 for CS2 composite state.

Subnet 3 implements critical exception handling mechanism by assigning *t7* condition to transition T^{w}. Enabling transition *T7* form *Subnet 1* also activates transition T^{w} from *Subnet 3*. Transition T^{w} provides token movement from place P^{a} to P^{init} and killing (removing) tokens from places *P10* to *P17*.

This behavior is not specific in comparison to classic Petri nets but simplify exception specification [17].

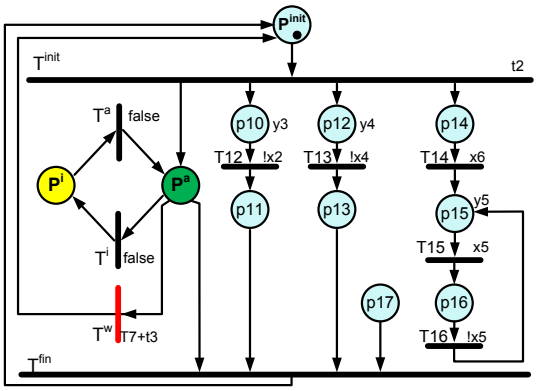

Fig. 7. Subnet no. 3 for CS3 composite state (context A).

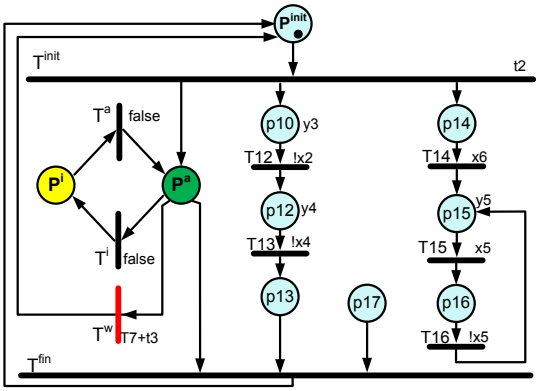

Fig. 8. Subnet no. 3 for CS3 composite state (context B).

Subnet 2 has only one context. This module will be preserved as static part of logic controller. If another context will be elaborated for *Subnet 2*, the design flow should restarted.

Figures 7 and 8 describe alternative versions of composite state *Cs3*. State *p17* has no input. In fact this state will block transition T^{fin} firing. This state was preserved intentionally in order to to be consistent with the UML state machine diagram and proposed method of the transformation. Alternative context of *Cs3* provides sequential tanks emptying, the base emptying was performed parallel way.

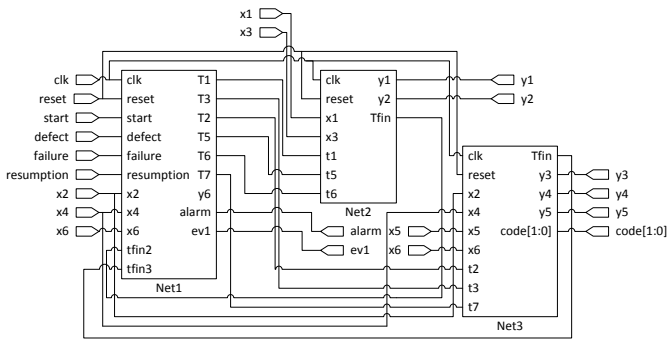

Fig. 9. Selected pblock for static and reconfigurable part.

```
module Net3(clk, reset, Tfin, x2, x4, x5, x6, t2,
    t7, y3, y4, y5 );
input  clk, reset, x1, x2, x4, x5, x6, t2, t7;
output Tfin, y3, y4, y5;
(...)
assign y3 = P10 & Pa;
assign y4 = P12 & Pa;
assign y5 = P15 & Pa;
(...)
assign Tinit= Pinit & t2 ;
assign Tfin = Pa & P11 & P13 & P17 & ~(Ti & Tw);
assign Ti = Pa & 0 & ~Tw;
assign Tw = Pa & t7;
assign Ta = Pi & 0;
(...)
always @(posedge clk)
begin
        if (reset) Pinit <= 1  ;
        else Pinit <= (Pinit & ~Tinit) | (Tw | Tfin)
            ;
end
always @(posedge clk)
begin
        if (reset) Pa <= 0  ;//0|1
        else Pa <= (Pa & ~(Ti | Tw | Tfin)) | (Tinit
            | Ta);
end
always @(posedge clk)
begin
        if (reset) Pi <= 0  ;
        else Pi <= (Pi & ~ Ta ) | (Ti);
end
(...)
always @(posedge clk)
begin
        if (reset) P10 <= 0  ;
        else P10 <= ~Tw & ((P10&~(T12 )) | (Tinit));
end
(...)
always @(posedge clk)
begin
        if (reset) P12 <= 0  ;
        else P12 <= ~Tw & ((P12 & ~(T13)) | (Tinit))
            ;
end
always @(posedge clk)
begin
        if (reset) P16 <= 0  ;
        else P16 <= ~Tw & ((P16 & ~(T16)) | (T15));
end
always @(posedge clk)
begin
        if (reset) P17 <= 0  ;
        else P17 <= ~Tw &((P17 & ~(Tfin)));
end
assign LocalConfig = Pa & ~ (Ti & Tw);

assign T13 = P12 & ~x4 & LocalConfig;
endmodule
```

Fig. 10. *Subnet 3* description in Verilog.

Each subnet was described in HDL language. In this particular case, Verilog language was used but VHDL description may be used as well. Each subnet was described separately and four functional block was an result. Figure 9 presents RTL level of the logic controllers. The communication (synchronisation) beetwen block is performed by dedicated signals. For example signal $tfin$ informs that subnet is finisched and outgoing transition from macroplace related to this net should be fired.

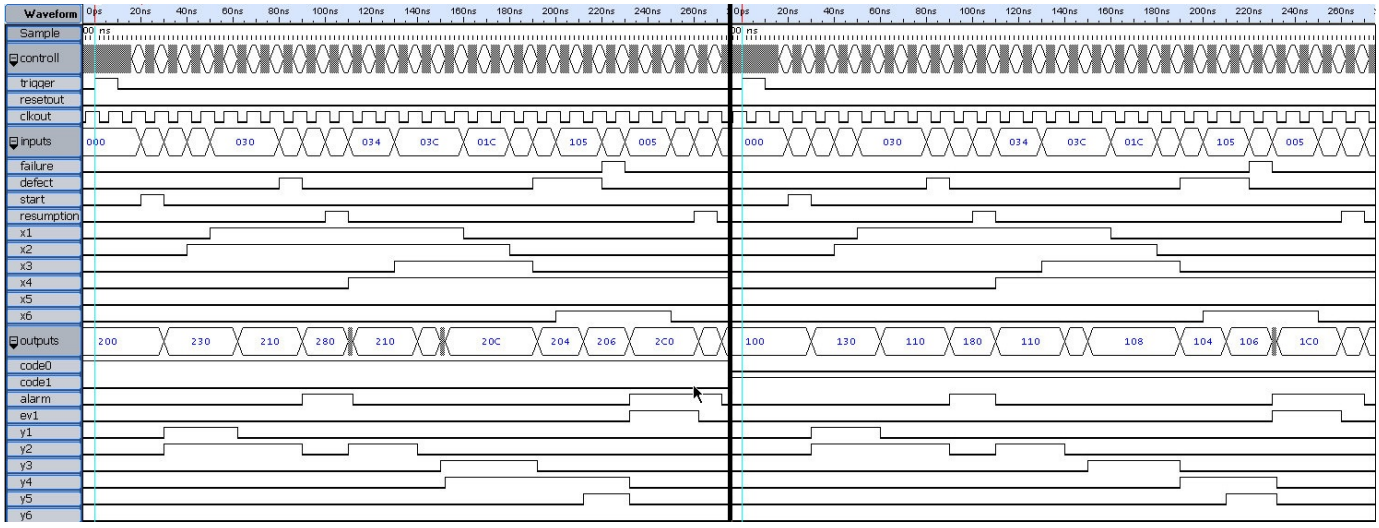

Fig. 11. In-circuit verification results. Left – context A, right – context B. Waveforms from Tektronix TLA 5204 logic analyser.

Till this step elaborated models are platform independent. The proposed specification method guarantee that the logic controller may be synthesized and implemented in FPGA device from any vendor. The logic controller was implemented into the Xilinx Virtex 5 FPGA device, subnets were synthesized by means of Xilinx ISE (13.2).

The further steps were executed by means of Xilinx Plan Ahead software. The area constrains for the reconfigurable partition was bounded. In this presented case the areal covers only *Slice* elements. If other elements like *BRAM* or *DSP* will be used for the implementation it's necessary to grantee that it will be inside selected partition *pblock*.

Figure 12 presents selected area constraints for both static and reconfigurable blocks. Reconfigurable partition was marked as *pblockU3* and contains only module *U3(Net3)*. The *PBlock_U1U3U3* contains other static part of the logic controllers including stimulus generator required for in-circuit verification. Fig. 12 also presents hierarchical structure of the design. Top level unit *DUT* contains two modules: logic controller *U1* and stimulus generator (*U2*). Logic controller contains three modules, where *U3* block contains two reconfigurable modules: *Net3a* and *Net3b*, for contexts A and B respectively.

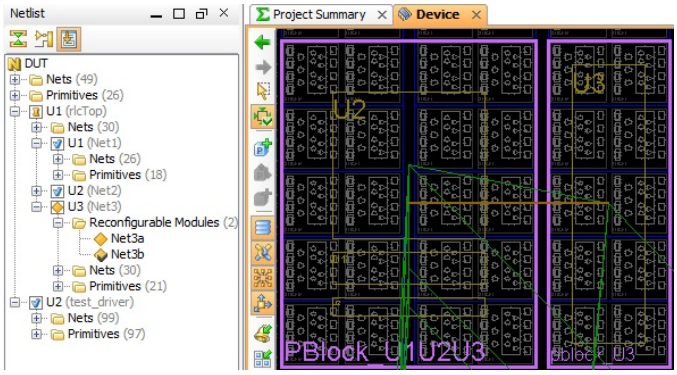

Fig. 12. Selected pblock for static and reconfigurable part.

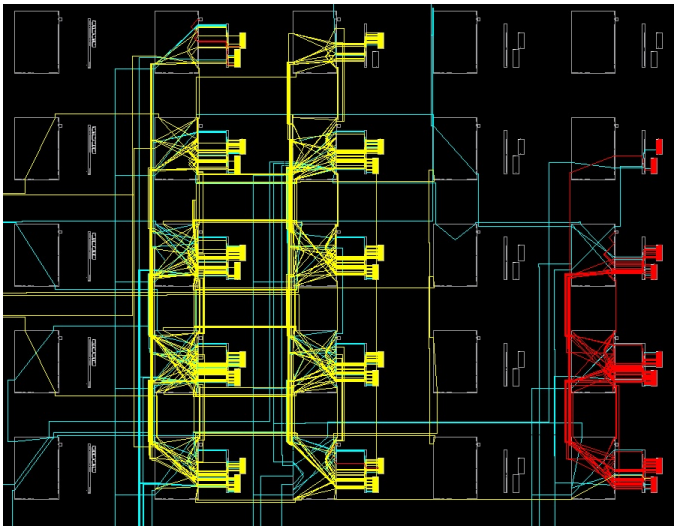

Fig. 13. View at the components and nets level.

IV. Implementation and Verification

Figure 13 presents the implementation results. Routed nets and placed components for the Net3 were marked in red. All other placed components and routed nets for Net1, Net3 and test_driver module were marked in yellow. Presented example is rather simple. Differences between configurations implementation at the components level are minor, therefore picture for the second configuration has been omitted intentionally.

In-circuit verification was performed by means of logic analyser. The additional module responsible for stimulus generation (SG) was implemented. The logic controller was driven by SG modlule, output were connected with the logic analyser.

The results of contexts verification were presented in Fig. 11. The waveform presents both input and output signals for the logic controllers. Simulation scenario covers both: critical and noncritical exception handling. The noncritical exception is triggered by signal *defect* and took place at a 75 ns. It results in the *P3* state activation. Subnet 2 is deactivated by

transition T^i firing. Resumption of control proces took place at a 105 ns, when signal *reactivation* becomes active. Subnet 2 is activated by transition T^a firing. Second scenario – critical exception took place at a 205 ns and was triggered by the signal *failure*. Token from place *MP3* was moved to *P4*. Subnet 3 is preempted and can't be resumed. All tokens from places *P10* to *P16* are deleted (killed) according to special rules of transition T^w triggering. Token from the place P^a is moved to place P^{init}. The differences beetwen context occurs at 150 ns. First configuration (context) describes parralel *Tank A* and *Tank B* emptying where both signals *y3* and *y4* are active. In case of context B (right waveform) tanks are emptyied in sequention: first *Tank A* and next *Tank B*.

Detailed verification by means of logic analyser guarantee that logic controller will work properly. Of course in case of large industrial proceses visual analysis is rather hard. The better solution is hardware in the loop verification (HIL). The HIL module can be implemented in the same FPGA device as the controller logic. The advantage of such solution is the ability of logic controller selft test between each production process or according to the schedule. Proper functioning of the device and the detection of defects is very important especially in the field of dependable systems.

V. Summary

In real industrial processes the difference between contexts will be significant. It will be reflected in controller complexity. In case of direct implementation of context switching, each context will increase required capacity of the FPGA device. It may result that the logic controller will not fit into selected FPGA device. Alternatively techniqes for logic elements optimisation can be used [3], [21]. The proposed methodology of the logic controller design will support multi context switching at system level. The paper presents way for logic controller implementation where selected functional blocks may be reconfigured. The advantage of this approach is the fact that at each stage from the specification to the synthesis and implementation functional decomposition is preserved. Further work will cover the implementation of the context switcher. It will be a supervisor of the contexts and, according to actual needs, will force reconfiguration of the logic controllers. The context switcher may be implemented as separate logic controller or algorithm dedicated to softprocessor embeded in FPGA device.

The ability of the FPGA devices partial reconfiguration enables the development of new functionality or the increase the broadly defined quality of logic controllers.

References

[1] M. Rawski, P. Tomaszewicz, G. Borowik, and T. Łuba, "5 logic synthesis method of digital circuits designed for implementation with embedded memory blocks of FPGAs," in *Design of Digital Systems and Devices*, ser. Lecture Notes in Electrical Engineering, M. Adamski, A. Barkalov, and M. Węgrzyn, Eds. Springer Berlin Heidelberg, 2011, vol. 79, pp. 121–144.

[2] R. Wiśniewski, A. Barkalov, L. Titarenko, and W. Halang, "Design of microprogrammed controllers to be implemented in FPGAs," *International Journal of Applied Mathematics and Computer Science*, vol. 21, no. 2, pp. 401–412, 2011.

[3] M. Adamski, M. Wiśniewska, R. Wiśniewski, and Ł. Stefanowicz, "Application of hypergraphs to the reduction of the memory size in the microprogrammed controllers with address converter," *Przegld Elektrotechniczny*, no. 8, pp. 134–136, 2012.

[4] A. Bukowiec, "Dynamic partial reconfiguration of petri net," in *Computer Aided Systems Theory - EUROCAST 2013 : 14th international conference; ISBN: 978-84-695-6971-9*. Las Palmas de Gran Canaria, Spain: Las Palmas de Gran Canaria, 2013, pp. 246–247 [abstr.].

[5] E. Hrynkiewicz, A. Milik, and J. Mocha, "Dynamically reconfigurable concurrent implementation of the binary control (in Polish)," *Electronics: Constructions, Technologies, Applications*, vol. 49,(11), pp. 187–190, 2008.

[6] F. Basile, P. Chiacchio, and D. Del Grosso, "A two-stage modelling architecture for distributed control of real-time industrial systems: Application of UML and Petri Net," *Comput. Stand. Interfaces*, vol. 31, pp. 528–538, March 2009.

[7] R. Harenstein, "Reconfigurable computing: A new business model and its impact on SoC design," in *Digital Systems Design, Euromicro Symposium on Digital Systems Design*. Los Alamitos, CA, USA: IEEE Computer Society, 2001, p. 103.

[8] Xilinx, *Partial Reconfiguration User Guide*, 12th ed., Xilinx, May 2010, uG702.

[9] G. Łabiak, M. Adamski, M. Doligalski, J. Tkacz, and A. Bukowiec, "UML modelling in rigorous design methodology for discrete controllers," *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 27–34, 2012.

[10] A. Karatkevich, *Dynamic Analysis of Petri Net-based Discrete systems*, ser. Lecture Notes in Control and Information Sciences. Berlin: Springer-Verlag, 2007, vol. 356.

[11] M. Adamski and J. Tkacz, "Formal reasoning in logic design of reconfigurable controllers," in *Proceedings of 11th IFAC/IEEE International Conference on Programmable Devices and Embedded Systems - PDeS 2012*. Brno, Czech Republic: Brno, 2012, pp. 1–6.

[12] J. Tkacz, "Deadlocks detection in logic controllers by means of using Gentzen reasoning method (in polish)," *PAK*, vol. 6, no. 6, pp. 11–13, 2006.

[13] T. Kropf, *Introduction to Formal Hardware Verification*. Berlin: Springer Verlag, 1999.

[14] R. Wiśniewski, A. Barkalov, and L. Titarenko, "Partial reconfiguration of compositional microprogram control units implemented on an FPGA," in *Proceedings of IEEE East-West Design & Test Symposium - EWDTS 08*, Kharkov National University of Radioelectronics. Lviv, Ukraine: Lviv, The Institute of Electrical and Electronics Engineers, Inc., 2008, pp. 80–83.

[15] M. Rawski, G. Borowik, T. Luba, P. Tomaszewicz, and B. Falkowski, "Logic synthesis strategy for FPGAs with embedded memory blocks," in *Mixed Design of Integrated Circuits Systems, 2009. MIXDES '09. MIXDES-16th International Conference*, 2009, pp. 296–301.

[16] M. Doligalski and M. Węgrzyn, "Partial reconfiguration-oriented design of logic controllers," in *Proceedings of SPIE : Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments*, vol. 6937, 2007, p. 10.

[17] M. Doligalski, *Behavioral specification diversification of reconfigurable logic controllers*, ser. Lecture Notes in Control and Computer Science. Zielona Góra: University of Zielona Góra Press, 2012, vol. 20.

[18] ——, "Uml state machine conversion in field of dual specification," *Przegld Elektrotechniczny*, vol. 85, no. 7, pp. 192–195, 2009.

[19] G. Łabiak, "From uml statecharts to fpga - the HiCoS approach," in *Forum on Specification & Design Languages - FDL '03; ISBN: 1636-9874*, University of Frankfurt. Frankfurt, Germany: Frankfurt, 2003, pp. 354–363.

[20] M. Doligalski and M. Adamski, "Petri net based specification in the design of logic controllers with exception handling mechanism," *International Journal of Electronics and Telecommunications*, vol. 58, no. 1, pp. 43–48, 2012.

[21] G. Borowik and T. Łuba, "Fast algorithm of attribute reduction based on the complementation of boolean function," in *Advanced Methods and Applications in Computational Intelligence*, ser. Topics in Intelligent Engineering and Informatics, R. Klempous, J. Nikodem, W. Jacak, and Z. Chaczko, Eds. Springer International Publishing, 2014, vol. 6, pp. 25–41.