Computer Engineering and Electronics Institute
Technical University of Zielona Góra

Proceedings

of the International Workshop on Discrete-Event System Design

DESDes'01

June 27÷29, 2001
Przytok near Zielona Góra

Edited by
Marian Adamski and Marek Węgrzyn

International Program Committee

Marian Adamski	Poland	*Chairman*
Wolfgang Fengler	Germany	*Co-Chairman*
Alain Amroun	France	
Zbigniew Banaszak	Poland	
Martin Bolton	UK	
Janusz Biernat	Poland	
Carlos Couto	Portugal	
Erik Dagless	UK	
Antonio Ferrari	Portugal	
Wolfgang A. Halang	Germany	
Edward Hrynkiewicz	Poland	
Stanley Hyduke	USA	
Monika Heiner	Germany	
Tadeusz Łuba	Poland	
Peter Neumann	Germany	
Joao L. Monteiro	Portugal	
Marek Perkowski	USA	
Alberto Proenca	Portugal	
Henry Selvaraj	USA	
Jerzy Sołdek	Poland	
Roman Stryjski	Poland	
Marek Węgrzyn	Poland	
Heinrich T.Vierhaus	Germany	
Arkadij Zakrevskij	Belarus	

Organizing Committee

Marek Węgrzyn	*Chair*
Janusz Jabłoński	
Leszek Jasiński	
Andrei Karatkevich	*Secretary*
Małgorzata Kołopieńczyk	
Joanna Kulińska	

All papers have been photographically reproduced exactly as received, on authors' own responsibility.

The papers have been reviewed.

Printing financed by: ADB Corporate UNIVERSITY

ISBN: 83-85911-62-6

© Technical University of Zielona Góra, 2001

Oficyna Wydawnicza Politechniki Zielonogórskiej
Zielona Góra, 2001

PREFACE

This volume contains the proceedings of the *International Workshop on Discrete-Event System Design*, *DESDes'01* held in Przytok near Zielona Góra, Poland, on 26-29 of June 2001. It contains selected 40 papers contributed by authors from Belarus, Czech Republic, Germany, Japan, Poland, Portugal, Spain, Turkey and Ukraine.

The aim of the *International Workshop on Discrete-Event System Design*, *DESDes'01* is to bring together researchers from universities and industry, and provide them with a platform to report on new topics in embedded microsystem design.

The Workshop focuses implementation of reactive, embedded, discrete-event systems in Field Programmable Logic (FPGA and CPLD), as well as related formal design methodologies and tools.

The scope of the Workshop includes Hardware Description Languages (VHDL or Verilog) in digital microsystem design, and hardware and software implementations of Petri net-based specifications. All others related aspects of digital microsystem design are also considered, especially specific integrated circuits (ASICs) and system-on-a-chip (SoC) applications. To complete the Workshop program there will be a Panel Discussion on new formal methodologies and influence of modern CAD tools into digital system design.

Przytok is a picturesque village near Zielona Góra. There is an excellent tourist and rest base. It consists of a monumental Renaissance palace, Special School Centre and International Youth Hostel. The social events offer a wide variety of opportunities to meet and exchange information about the possible collaboration.

Zielona Góra has around 120.000 inhabitants and is located 450 km west of Warsaw and 200 km east of Berlin. Two universities are located there: Technical University and Pedagogical University. The region of Zielona Góra is covered by beautiful wild forests and surrounded by attractive lakes.

We are very grateful to the members of the International Program Committee and all sponsors of our Workshop.

We expect that the readers of Proceedings will find this volume useful, informative and helpful in their related creative research works.

Finally, thank you for participating in the *DESDes'01* Workshop. We hope you will have an enjoyable and productive experience in Zielona Góra and Przytok.

Marian Adamski
Chairman

Zielona Góra, June 4th, 2001

About University, Faculty and Institute*

The origins of ***the Technical University*** date back to 1965 when the Higher College of Engineering was founded in Zielona Góra. It responded to the needs of the citizens of the ***Lubuski Region***, who wanted their own technical university to train and educate the people who would create and manage the industry of the area. The continuous development of the scientific community and the resulting high-quality research, along with the rapid development of faculties, the introduction of modern specialities, the building of new facilities, lecture rooms and modern laboratories, as well as computer-based learning and teaching, enabled the college to obtain a full academic status and to become the Technical University of Zielona Góra on September 1st, 1996. At present, the University has more 8,500 students in 7 different subject areas and 25 specializations. The students can choose from among many kinds of courses on various levels. The University has four faculties, which have the right to confer Ph.D. degrees. The University employs more than 440 members of academic staff.

Studying in a smaller academic centre is more attractive to a number of candidates because of everyday friendly contacts with professors, the possibility of individualising the courses, round-the-clock access to well-equipped computer laboratories enabling contact with global networks, the availability of accommodation in the University's halls of residence and lower costs of living. The University has the advantage that all faculties are on one campus, which is virtually traffic-free and very close to the centre of the town. The only facility outside the campus is the horse-riding centre in Raculka near Zielona Góra, which offers horse-riding therapy and horse-riding training.

The Faculty of Electrical Engineering was founded in 1967 and offers courses in two areas, for 2,511 students, in *Electrical Engineering* and *Computer Science*.

The Institute of Computer Engineering and Electronics is divided on: *Information Technology Division*, *Computer Engineering Division*, *Electronics and Microprocessor Systems Division*.
Main research areas include:

- synthesis and analysis of digital systems based on specific integrated circuits (ASIC) – implemented of Petri nets in the synthesis of concurrent digital systems;
- hardware-software co-design – Petri nets as an intermediate representation between HDL and the design models;
- computer networks – local area networks, operating systems, Internet technology;
- high performance multimedia communications systems;
- electronic devices and microprocessor systems.

Address

Technical University, Computer Engineering & Electronics Institute
65-246 Zielona Góra, ul. Podgórna 50, POLAND
Tel.: (+48 68) 3282 484; Fax: (+48 68) 3244 733
E-mail: *desdes01@iie.pz.zgora.pl*, WWW: *www.iie.pz.zgora.pl/desdes01*

* Source: J.Korbicz (Ed.), *Technical University of Zielona Góra*, Zielona Góra, 2000, (ISBN: 83-85911-31-6)

Table of contents

OPENING SESSION

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

SEQUENT MODEL FOR REPRESENTATION OF DIGITAL SYSTEMS BEHAVIOR

Arkadij ZAKREVSKIJ

Institute of Engineering Cybernetics NAS B, Surganov Str. 6, 220012, Minsk, Belarus
e-mail: *zakr@newman.bas-net.by*

Abstract. *A model of sequent automaton is proposed for description of digital systems behavior in the space of Boolean variables: input, output and inner ones. The rules of its equivalence transformations are formulated, leading to several canonical forms. Simple sequent automaton is introduced, represented in matrix form, which is intended for easing PLA implementation of the automaton. The problem of automata correctness is discussed.*

Key Words. *Digital system behavior, Boolean space of events, sequent automaton, canonical forms, checking for correctness, PLA implementation*

1. EVENTS IN THE BOOLEAN SPACE

Many complex engineering systems may be regarded as dynamic digital systems working in some surroundings. Very often their behavior can be expressed in terms of Boolean variables taking their values from the set $\{0, 1\}$ and defining in such a way the states of individual elements of the system. Usually, when a control system is constructed to ensure the proper interaction between system components, the set W of all variables is divided into three classes: X, Y and Z. Input variables (X) present information received from sensors situated in surroundings or in the system itself; output variables (Y), calculated inside the system, are intended for control purposes and used by executing elements; and inner variables (Z) may play both roles and could be considered as memory of the system.

$2^{|W|}$ different combinations of values of variables from W constitute the Boolean space over W ($|W|$ denotes the cardinality of set W). This Boolean space is designated below as BS(W). Each of its elements may be regarded as a global state of the system, or as the corresponding event that occurs when the system enters that state. Let us call such an event *elementary*. In the same way, the elements of Boolean spaces over X, Y and Z may be regarded as input states, output states and inner states, as well as corresponding events.

Besides that far more events of other types may be introduced into consideration. Generally, every subset of BS(W) may be interpreted as an event which occurs when some element from BS(W) is realized, i. e. when the variables from W take the corresponding combination of values. In this general case the event is called *complicated* and could be presented by the characteristic Boolean function of the regarded subset. So, the number of complicated events coincides with the number of arbitrary Boolean functions of $|W|$ variables.

From the practical point of view, the following two types of events deserve special consideration: basic events and simple events.

Basic events are represented by literals - symbols of variables or their negations - and occur when these variables take corresponding values. For example, basic event a occurs when variable a equals 1, and event c' occurs when $c = 0$. The number of different basic events is $2|W|$.

Simple events are represented by elementary conjunctions, and occur when these conjunctions take value 1. For example, event $ab'f$ occurs when $a = 1$, $b = 0$ and $f = 1$. The number of different simple events is $3^{|W|}$, including trivial event, when values of all variables are arbitrary.

Evidently, the class of simple events absorbs elementary events and basic events. So, elementary conjunction k_i is the general form for representation of events i of all three introduced types; it contains symbols of all variables in the case of an elementary event and only one symbol when a basic event is regarded. One event i can realize another j - it means that the latter always comes when the former comes. It follows from the definitions, that it occurs when conjunction k_i implicates conjunction k_j , in other words, when k_j can be obtained from k_i by deleting some of its letters. For example, event $abc'de'$ realizes events $ac'd$ and $bc'e'$, event $ac'd$ realizes basic events a, c' and d, etc. Hence, several different events can occur simultaneously, if only they are not orthogonal.

2. SEQUENT AUTOMATON

The behavior of a digital system is defined by the rules of changing its state. A standard form for describing such rules was suggested by the well-developed classical theory of finite automata considering relations between the sets of input, inner and output states. Unluckily, that model becomes inapplicable for digital systems with many Boolean variables - hundreds and more. That is why a new formal model was proposed in [3-5] called sequent automaton. It takes into account the fact, that interaction between variables from W takes place within comparatively small groups and has functional character. And it suggests means for describing both the control unit of the system and the object of control - the body of the system.

Sequent automaton is a logical dynamic model defined formally as a system S of *sequents* s_i. Each sequent s_i has the form $f_i \,|\!-\, k_i$ and defines the "cause-effect" relation between some complicated event represented by Boolean function f_i and a simple event k_i represented by conjunction term k_i ; |- is the symbol of the considered relation. Suppose function f_i is given in disjunctive normal form (DNF).

The expression $f_i \,|\!-\, k_i$ is interpreted as follows: if at some moment function f_i takes value 1, then immediately after that k_i must also become equal to 1 - by that the values of all variables in k_i are defined uniquely. In such a way a separate sequent can present a definite demand to the behavior of the discrete system, and the set S as a whole - the totality of such demands.

Note, that the variables from X may participate only in f_i and can carry information got from some sensors, the variables from Y present control signals and participate only in k_i, and the variables from Z are feed-back variables which can be presented both in f_i and k_i.

The explication of "immediately after that" depends greatly on the accepted time model. It is different for two kinds of behavior interpretation, which could be used for sequent automata, both of practical interest: synchronous and asynchronous.

We shall interpret system S mostly as a *synchronous* sequent automaton. In this case the behavior of the automaton is regarded in discrete time t - the sequence of moments $t_o, t_1, t_2, ..., t_l, t_{l+1}, ...$. At a current transition from t_l to t_{l+1} there are executed simultaneously all such sequents s_i for which $f_i = 1$ and as a result all corresponding conjunctions k_i turn to 1 (all their factors take value 1). In that case "immediately after that" means "at the next moment".

Suppose that if some of inner and output variables are absent in conjunctions k_i of executed sequents, they preserve their previous values. That is why the regarded sequent automata are called *inertial* [4]. Hence a new state of the sequent automaton (the set of values of inner variables) is defined uniquely, as well as new values of output variables.

Sometimes the initial state of the automaton is fixed (for moment t_o), then the automaton is called *initialized*. The initial state uniquely determines the set R of all reachable states. When computing it, it is supposed that all input variables are free, i. e. by any moment t_l they could take arbitrary combinations of values. Let us represent set R by characteristic Boolean function φ of inner variables which takes value 1 on the elements from R. In the case of non-initialized automata it is reasonable to consider that φ = 1.

Under *asynchronous* interpretation the behavior of sequent automaton is regarded in continuous time. There appear a lot of more hard problems of their analysis connected with races between variables presented in terms k_i, especially when providing the automaton with important quality of correctness.

3. EQUIVALENCE TRANSFORMATIONS AND CANONICAL FORMS

Let us say that sequent s_i is *satisfied* in some engineering system if event f_i is always followed by event k_i. And sequent s_i *realizes* sequent s_j if the latter is satisfied automatically when the former is satisfied.

Affirmation 1. Sequent s_i realizes sequent s_j if and only if $f_j \Rightarrow f_i$ and $k_i \Rightarrow k_j$, where $\Rightarrow$ is the symbol of formal implication.

For instance, sequent $ab \vee c$ |- uv' realizes sequent abc |- u. Indeed, $abc \Rightarrow ab \vee c$ and $uv' \Rightarrow u$.

If two sequents s_i and s_j realize each other, they are *equivalent*. In that case $f_i = f_j$ and $k_i = k_j$.

The relations of realization and equivalence can be extended onto sequent automata S and T. If S includes in some form all demands contained in T, S realizes T. If two automata realize each other, they are equivalent.

These relations are easily defined for *elementary sequent automata* S^e and T^e, which consist of *elementary sequents*. Left part of such a sequent presents an elementary event in BS($X \cup Z$), right part presents a basic event (for example, $ab'cde'$ |- q, where it is supposed that $X \cup Z = \{a, b, c, d, e\}$. S^e realizes T^e if it contains all sequents contained in T^e. S^e and T^e are equivalent if they contain the same sequents. It follows from here that elementary sequent automaton is a *canonical form*.

There exist two basic equivalencies formulated as follows.

Affirmation 2. Sequent $f_i \vee f_j$ |- k is equivalent to the pair of sequents f_i |- k and f_j |- k.

Affirmation 3. Sequent f |- $k_i k_j$ is equivalent to the pair of sequents f |- k_i and f |- k_j.

According to these affirmations any sequent can be decomposed into a series of elementary sequents (which cannot be decomposed further). That transformation enables to compare any sequent automata checking them for binary relations of realization and equivalence.

Affirmations 2 and 3 can be used for equivalence transformations of sequent automata by elementary operations of two kinds: splitting sequents (changing one sequent for a pair) and merging sequents (changing a pair of sequents for one, if possible).

Elementary sequent automaton is useful for theoretical constructions but could turn out quite non-economical when regarding some real control systems. Therefore two more canonical forms are introduced.

The *point sequent automaton* S^p consists of sequents in which all left parts represent elementary events (in BS($X \cup Z$)) and are different. The corresponding right parts show the responses. This form can be obtained from elementary sequent automaton S^e by merging sequents with equal left parts.

The *functional sequent automaton* S^f consists of sequents in which all right parts represent basic events in BS($Z \cup Y$) and are different. So the sequents have the form $f_i^1 \,|\!-\, u_i$ or $f_i^0 \,|\!-\, u_i'$, where variables $u_i \in Z \cup Y$, and the corresponding left parts are interpreted as switching functions for them: on-functions f_i^1 and off-functions f_i^0. S^f can be obtained from S^e by merging sequents with equal right parts.

Note that both forms S^p and S^f can be obtained also from arbitrary sequent automata by disjunctive decomposition of the left parts of the sequents (for the point sequent automaton) or conjunctive decomposition of the right parts (for functional one).

4. SIMPLE SEQUENT AUTOMATON

Consider now a special important type of sequent automata - a *simple sequent automaton.* It is defined formally as a system S of *simple sequents* - expressions $k_i' \,|\!-\, k_i''$ where both k_i' and k_i'' are elementary conjunctions representing simple events. This form has a convenient matrix representation, inasmuch as every elementary conjunction can be presented as a ternary vector.

Let us represent any simple sequent automaton by two ternary matrices: a *cause matrix* $\boldsymbol{A}$ and an *effect matrix* $\boldsymbol{B}$. They have equal number of rows indicating simple sequents, and their columns correspond to Boolean variables - input, output and inner ones.

Example. The two ternary matrices

$$\boldsymbol{A} = \begin{array}{cccccc} a & b & c & p & q & r \\ 1 & - & - & - & 0 & - \\ - & 0 & 1 & 1 & - & - \\ 0 & 1 & - & - & 1 & 1 \\ - & - & 0 & - & - & 0 \\ - & - & 0 & 1 & 0 & - \end{array}, \quad \boldsymbol{B} = \begin{array}{ccccccc} p & q & r & u & v & w & z \\ - & 1 & - & - & 1 & - & 1 \\ - & - & 0 & 1 & - & 0 & - \\ 1 & 0 & - & - & 1 & - & 0 \\ 0 & - & - & - & - & 1 & - \\ - & 1 & 1 & 0 & - & 1 & - \end{array}$$

represent the following system of simple sequents regarded as a simple sequent automaton:

$$\begin{array}{c} aq' \,|\!-\, qvz\,, \\ b'cp \,|\!-\, r'uw'\,, \\ a'bqr \,|\!-\, pq'vz'\,, \\ c'r' \,|\!-\, p'w\,, \\ c'pq' \,|\!-\, qru'w\,. \end{array}$$

Here $X = \{a, b, c\}$, $Y = \{u, v, w, z\}$, $Z = \{p, q, r\}$.

It has been noted [1] that, to a certain extent, simple sequents resemble the sequents of the theory of logical inference, which were introduced by Gentzen [2]. The latter ones are defined as expressions

$$A_1, \ldots, A_n \rightarrow B_1, \ldots, B_m$$

that connect arbitrary logic formulae $A_1, \ldots, A_n, B_1, \ldots, B_m$ and are interpreted as implications

$$A_1 \wedge \ldots \wedge A_n \rightarrow B_1 \vee \ldots \vee B_m .$$

The main difference is that any simple sequent $k_i' \mid\!- k_i''$ presents not pure logical but cause-effect relation: event k_i'' is generated by event k_i' and appears after it, so we cannot mix variables from k_i' with variables from k_i''.

But sometimes we may discard this time aspect and consider terms k_i' and k_i'' on the same level, for instance, when looking for stable states of the regarded system. In that case sequent $k_i' \mid\!- k_i''$ could be formally changed for implication $k_i' \rightarrow k_i''$ and subjected further to Boolean transformations, leading to equivalent sets of Gentzen sequents and corresponding sets of standard disjuncts usual for theory of logical inference.

For example, in such a way the system of simple sequents

$$ab \mid\!- cd', \quad a'b' \mid\!- cd, \quad a'b \mid\!- c$$

may be transformed into the following system of disjuncts:

$$a \vee b \vee d, \quad a' \vee b' \vee d', \quad a' \vee c' \vee d, \quad b \vee c' \vee d'.$$

5. APPLICATION IN LOGIC DESIGN

The model of simple sequent automaton is rather close to the well-known technique of disjunctive normal forms (DNF) used for hardware implementation of systems of Boolean functions [6]. Indeed, each row of matrix $\boldsymbol{A}$ may be regarded as a conjunctive term (product), and each column in $\boldsymbol{B}$ defines DNFs for two switching functions of the corresponding output or inner variable: 1s indicate terms entering ON-functions, while 0s indicate terms which enter OFF-functions. Note, that these DNFs can be easily obtained by transforming the regarded automaton into S^f-form and changing after that expressions $f_i^1 \mid\!- u_i$ for $u_i^1 = f_i^1$ and $f_i^0 \mid\!- u_i'$ for $u_i^0 = f_i^0$. For the same example

$p^1 = a'bqr$, $p^0 = c'r'$; $\quad q^1 = aq' \vee c'pq'$, $q^0 = a'bqr$; $\quad r^1 = c'pq'$, $r^0 = b'cp$;
$u^1 = b'cp$, $u^0 = c'pq'$; $\quad v^1 = aq' \vee a'bqr$; $\quad w^1 = c'r' \vee c'pq'$, $w^0 = b'cp$;
$z^1 = aq'$, $\quad z^0 = a'bqr$.

It is seen from here that the problem of constructing a simple sequent automaton with minimum number of rows is similar to the minimization of a system of Boolean functions in the class of DNFs known as a hard combinatorial problem. An approach to its solving was suggested in [7, 8].

The considered model turned out to be especially convenient for representation of programmable logic arrays (PLA) with memory on RS-flip-flops. It is used also in methods of automaton implementation of parallel algorithms for logical control described by expressions in PRALU [11].

Consider a simple sequent automaton shown in the above example. It is implemented by a PLA represented below. It has three inputs (a, b, c) supplied with inverters (NOT-elements) and four outputs (u, v, w, z) supplied with RS-flip-flops. So its input and output lines are

doubled. The six input lines are intersecting with five inner ones, and at some points of intersection transistors are placed. Their disposition can be presented by a Boolean matrix easily obtained from matrix ***A***, and determines the AND-stage of the PLA. In a similar way the OR-stage of the PLA is found from matrix ***B*** and realized on the intersection of inner lines with 14 output ones.

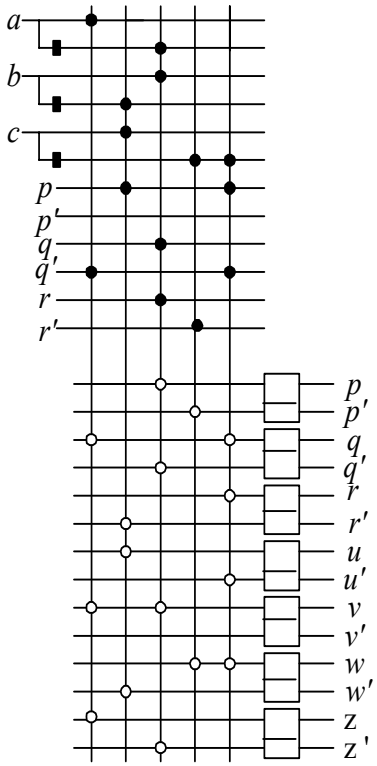


Figure 1. PLA implementation of a sequent automaton

6. CHECKING FOR CORRECTNESS

In general, correctness is a quality of objects of some type, defined as the sum of several properties, which are considered reasonable and necessary [10].

Let us enumerate such properties first for synchronous sequent automata.

Evidently, for any sequent s_i which carries some information inequalities $f_i \neq 0$ and $k_i \neq 1$ should hold, to avoid trivial sequents.

Sequents s_i and s_j are called *parallel* if they could be executed simultaneously. A necessary and sufficient condition of parallelism for non-initialized automaton is relation $f_i \wedge f_j \neq 0$, for initialized - relation $f_i \wedge f_j \wedge \varphi \neq 0$.

First of all, any sequent automaton should be *consistent*, that is very important. That means that for any parallel sequents s_i and s_j relation $k_i \wedge k_j \neq 0$ must hold. Evidently, this condition is necessary, inasmuch as by its violation there exists some variable that must take two different values which is impossible.

The second quality is not so necessary for sequent automata as the first one, but is also useful. It is *irredundancy*. A system S is *irredundant* if it is impossible to delete from it some sequent or if only a literal from a sequent without violating the functional properties of the system. For example, it should not have "non-reachable" sequents, such s_i for which $f_i \wedge \varphi = 0$.

It is rather easy to check a simple sequent automaton for consistency. An automaton represented by ternary matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ is obviously consistent if for any orthogonal rows in matrix $\boldsymbol{B}$ the corresponding rows of matrix $\boldsymbol{A}$ are also orthogonal. Note that this condition is satisfied in Example.

One more useful quality called *persistency* is very important for asynchronous sequent automata. To check them for this quality it is convenient to deal with the functional canonical form.

The point is that several sequents can be executed simultaneously and if the sequent automaton is asynchronous, these sequents (called parallel) could compete - the so called *race* could take place. The automaton is *persistent* if the execution of one of the parallel sequents does not destroy the conditions for executing other ones.

Affirmation 4. In a persistent asynchronous sequent automaton for any pair of parallel sequents

$$f_i^1 \text{ |- } u_i \text{ and } f_j^1 \text{ |- } u_j ,$$
$$f_i^0 \text{ |- } u_i' \text{ and } f_j^1 \text{ |- } u_j ,$$
$$f_i^1 \text{ |- } u_i \text{ and } f_j^0 \text{ |- } u_j' ,$$
$$f_i^0 \text{ |- } u_i' \text{ and } f_j^0 \text{ |- } u_j'$$

the corresponding relation should hold:

$$f_i^1 f_j^1 : u_i' u_j' \Rightarrow (f_i^1 : u_i' u_j)\,(f_j^1 : u_i u_j'),$$
$$f_i^0 f_j^1 : u_i u_j' \Rightarrow (f_i^0 : u_i u_j)\,(f_j^1 : u_i' u_j'),$$
$$f_i^1 f_j^0 : u_i' u_j \Rightarrow (f_i^1 : u_i' u_j')\,(f_j^0 : u_i u_j),$$
$$f_i^0 f_j^0 : u_i u_j \Rightarrow (f_i^0 : u_i u_j')\,(f_j^0 : u_i' u_j),$$

where expression $f : k$ means the result of substitution those variables of function f which enter elementary conjunction k for the values satisfying equation $k = 1$.

The proof of this affirmation can be found in [9].

ACKNOWLEDGMENT

This research was supported by ISTC, Project B-104-98.

REFERENCES

[1] M.Adamski, *Digital systems design by means of rigorous and structural method.* - Zielona Gora, 1990 (in Polish).

[2] G.Gentzen, "Untersuchungen über das logische Schließen", *Mat. Z., v. 39* (1934-35), pp. 176-210, 405-431.

[3] V.S.Grigoryev, A.D.Zakrevskij, V.A.Perchuk, "The sequent model of the discrete automaton",. *Vychislitelnaya tekhnika v mashinostroenii,* March 1972, Minsk, Institute of Engineering Cybernetics, pp.147-153 (in Russian).

[4] V.N.Zakharov, "Sequent description of control automata", *Izvestiya AN SSSR,* 1972, №2 (in Russian).

[5] V.N.Zakharov, *Automata with distributed memory.* Moscow: Energia, 1975 (in Russian).

[6] A.D.Zakrevskij, V.S.Grigoryev, “A system for synthesis of sequent automata in the basis of arbitrary DNFs”, *Problems of Cybernetics. Theory of relay devices and finite automata*. VINITI, Moscow, 1975, pp. 157-166 (in Russian).
[7] A.D.Zakrevskij, “Optimizing sequent automata”, *Optimization in digital devices design,* L., 1976, pp. 42-52 (in Russian).
[8] A.D.Zakrevskij, “Optimizing transformations of sequent automata”, *Tanul. MTA SeAKJ,* Budapest, 63/1977, p. 147-151 (in Russian).
[9] A.D.Zakrevskij, *Logical synthesis of cascade networks.* Moscow: Nauka, 1981 (in Russian).
[10] A.D.Zakrevskij, “The analysis of concurrent logic control algorithms”, *Lecture Notes in Computer Science, vol. 278,* Springer Verlag, 1987, pp. 497-500.
[11] A.D.Zakrevskij, *Parallel algorithms for logical control.* Minsk, Institute of Engineering Cybernetics, 1999 (in Russian).

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

ESTIMATION OF WCET USING A LITTLE LANGUAGE TO DESCRIBE MICROCONTROLLER AND DSP ARCHITECTURES

Adriano TAVARES, Carlos COUTO

Department of Industrial Electronics, University of Minho, 4800 Guimarães, PORTUGAL,
<atavares, ccouto>@dei.uminho.pt

Abstract. *A method for analysing and predicting the timing properties of a program fragment will be described. First a little language implemented to describe a processor's architecture is presented followed by the presentation of a new static WCET estimation method. The timing analysis starts by compiling a processor's architecture program followed by the disassembling of the program fragment. After sectioning the assembler program into basic blocks, call graphs are generated and these data are later used to evaluate the pipeline hazards and cache miss that penalize the real-time performance. Some experimental results of using the developed tool to predict the WCET of code segments using some Intel microcontroller are presented. Finally, some conclusions and future work are presented.*

Key Words. *Little Language, Worst-Case Execution Time, Machine Description, Language Paradigm, Timing Scheme, Timing Analysis*

1. INTRODUCTION

Real-time systems are characterized by the need to satisfy a huge timing and logical constraints that regulate their correctness. Therefore, predicting a tight worst case execution time of a code segment will be a must to guarantee the system correctness and performance. The simplest approach to estimate the execution time of a program fragment is for each arithmetic instruction, counting the number of times it appears on the code, express the contribution of this instruction in terms of clock cycles and update the total clock cycles with this contribution. Nevertheless, these approaches are unrealistic since they ignore the system interferences and the effects of cache and pipeline, two very important features of some processors that can be used in our hardware architecture. Some very elaborated methodology for WCET estimation, such as, Shaw [1] were developed in the past, but none of them takes into account the effects of cache and pipeline.

Theoretically, the estimation of WCET must skip over all the profits provided by modern processors, such as caches, and pipeline (i.e., each instruction execution suffers from all kind of pipeline hazards and each memory access would cause a cache miss) as they are the main

source of uncertainty. Experimentally, a very pessimistic result would be obtained, and so, making useless those processor's resources. Some WCET estimation schemes oriented to modern hardware features, were presented in the last years, and among them we refer to: Nilsen [2], Steven Li [3], Whalley [4], and Sung-Soo Lim [5]. However, these WCET estimators do not address some specificity of our target processors (microcontrollers and DSPs), since they are oriented to general-purpose processor. Therefore, we propose a new machine independent estimator, implemented as a little language for architecture description. Such a machine independent scheme, based on a little language was used before by Tremblay[6] to generate machine independent code, Proebsting and Fraser [7] to describe pipeline architectures and Nilsen [5] to implement a compiler, simulator and WCET estimator for pipeline processors.

2. LITTLE LANGUAGE

The purpose of any little language, typically, is to solve a specific problem and, in so doing, simplify the activities related to the solution of the problem. Our little language's statements are created based on the tasks that must be performed to describe processor's architectures in terms of structure and functional architecture of the interrupt controller, PTS (Peripheral Transaction Server), PWM (Pulse Width Modulation), WG (Waveform Generator), and HIS (High Speed Input), instruction set, instruction semantics, addressing modes, processor's registers, instruction coding, compiler's specificity, pipeline and cache resources, and so on. For our little language, we adopt a procedural and modular paradigm (language paradigm defines how the language processor must process the built-in statements), such that modules are independent from each other, the sequence of modules execution does not matter, but the register module must always be the first to be executed and within each module an exact sequence of instructions is specified and the computer executes them in the specified order. A processor's architecture program is written by modules, each one describing a specific processor's feature such as instruction set, interrupt structure and mechanism, registers structure, memory organization, pipeline, data cache, instruction cache, PTS, and so on. A module can be defined more than once, and it is a processor language job to verify the information consistency among them and concatenate all them into a single module.

The disassembler process implemented into four phases, has as input an executable file contains the code segment that one wants to measure and the compiled version of the processor's architecture program. The disassembler process starts at the start-up code address (startup code is the bootstrap code executed immediately after the reset or power-on of the processor) and follows the execution flow of the program:

1. starting at the start-up code address follows all possible execution paths till reaching the end address of the "main" function. At this stage, all function calls are examined and their entry code addresses are pushed into an auxiliary stack,
2. from the entry address of the "main" function, checks the main function code for interrupt activation,
3. for each active interrupt, gets its entry code address and pushed it into the auxiliary stack,
4. pops each entry address from the auxiliary stack and disassemble it, following the function's execution paths.

The execution of the simulation module is optional and the associated process is described by a set of operation introduced using the function "SetAction". For instance, the simulation process, including the flag register affectation, associated to an instruction are described by a

set of operation specified using "SetAction" calls. Running the simulation process before the estimation process, will produce a more optimistic worst case timing analysis since it can:

1. rectify the execution time of instructions that depend on data locations, such as stack, internal or external memory,
2. solve the indirect address problem by checking if it is a jump or a function call (function call by address),
3. estimate the iteration number of a loop.

The WCET estimator module requires a direct interaction with the user as some parameters are not directly measurable through the program code. Note that, the number of an interrupt occurrence and the preview of a possible maximum iterations number associated to an infinite loop are quite impossible to be evaluate using only the program code. The WCET estimation process was divided into two phases:

1. first, the code segment to be measured is decomposed into basic blocks,
2. for each basic block, it will be estimated the lower and upper execution time, using the shortest path method and a timing scheme [1].

The shortest path algorithm with the basic block graph as input is used to estimate the lower and upper bound on the execution time of the code segment. For the estimation of the upper bound, it is used the multiplicative inverse of the upper execution time of each basic block. A basic block is a sequence of assembler's instructions, such as, only the first instruction can be prefixed by a label and only the last one can be a control transfer instruction. The decomposition phase is carried out following the steps below:

1. rearrangement of code segment to guarantee the visual cohesion of a basic block. Note that, the ordering of instructions by address make more difficult the visualization of the inter basic block control flow, due to long jump instructions that can occur between basic blocks. To guarantee that visual cohesion, all sequence of instructions are rearranged by memory address, excluding those one located from long jump labels which are inserted from the last buffer index,
2. characterization of the conditional structure through the identification of the instructions sequence that compose the "if" and "else" body,
3. characterization of the loop structure through the identification of the instructions sequence that composes the loop body, control and transfer control. It is essential to discern between "while/for" and "do while" loop since the timing schemes are different,
4. After the identification and characterization of the control and loop structures, it will be built a basic block graph, showing all the execution paths between basic blocks
5. and for each basic block, find the lower and upper execution time.

2.1. Pipeline Modelling

The WCET estimator presented so far, considers that an instruction's execution is fixed over the program execution, i.e., it ignores the contribuition of modern processors. Note that, the dependence among instructions can cause pipeline hazards, introducing a delay in the instructions execution. This dependence emerges as several instructions are simultaneously executed and as the result of this parallelism execution among instructions, the execution time of an instruction fluctuates depending on the set of its neighbouring instructions. Our little language analyses the pipeline using the pipeline hazard detection technique suggested by

Proebsting and Fraser [7] and models the pipeline as a set of resources and each instruction as a process that acquires and consumes a subset of resources for its execution. Special purpose functions, such as, "setPipeStage(Mn)" and "SetPipeFunctionalUnit(Mn,num)" are used to define the pipeline stages and functional units, respectively. For each instruction, there is a set of functions to specify the pipeline stage each source operand must be available, the pipeline stage the output of the destination operand becomes available, each pipeline stage required to execute an instruction and the execution time associated to that stage, and the control hazard cost associated to a branch instruction.

The pipeline analysis of a given basic block must always take into account the influences of the predecessor basic blocks (note that, the dependence among instructions can cause pipeline hazards, introducing a delay in the instructions execution), otherwise, it leads to an underestimation of the execution time. Therefore, at the hazard detection stage of a given basic block, it will be always incorporate the pipeline's state associated to the predecessor basic blocks over the execution paths. The resources vector that describes the pipeline's state it will be iteratively updated by inserting pipeline stalls to correct the data and/or structural hazards when the next instruction is issued. If these two hazards happen simultaneously, the correction process start at the hazard that occurred first and after it will be checked if the second one still remains. The issuing of the new instruction will be always preceded by the updating of the previous pipeline's state, achieved by shifting the actual pipeline resource vector one cycle forward.

The pipeline architectures, usually, present special techniques to correct the execution flow when a control hazard happens. For instance, the delay transfer control technique offers the hardware an extra machine cycle to decide the branch. Also, special hardware is used to determine the branch label and value condition at the end of the instruction's decode. As one can conclude, the execution of delay instructions does not depend on the branch decision and it is always carried out. So, we model the control hazard, as being caused by all kind of branch instruction and by adding the sum of execution time of all instruction in the slot delay to the basic block execution time.

2.2. Cache Modelling

Cache is a high speed and small size memory, typically, a SRAM that contains parts of the most recent accesses to the main memory. Nowadays, the time necessary to load an instruction or data to the processor is much longer than the instruction execution time. The main rule of a cache memory is to reduce the time needed to move the information from and to the processor. An explanation for this improvement, comes from the locality of reference theory – at any time, the processor will access a very small and localized region of the main memory and the cache load this region, allowing faster memory accesses to the processor.

In spite of the memory performance enhancement, the cache makes the execution time estimation harder, as the execution time of any instruction will vary and depends on the presence of the instruction and data into the caches. Furthermore, to exactly know if the execution of a given instruction causes a cache miss/hit, it will be necessary to carry out a global analysis of the program. Note that an instruction's behaviour can be affected by memory references that happened long time before. Adversely, the estimation of WCET becomes harder for the modern processors, as the behaviour of cache and pipeline depend on each other. Therefore, we propose the following changes to the algorithm that takes into account the pipeline effects:

1. Classify the cache behaviour [4] for any data and instruction as cache hit or cache miss before the analysis of the pipeline behaviour,

2. Before the issuing of an instruction, verify if there is any cache miss related to the instruction, and if any, apply the miss penalty beforehand and then the detection and correction of pipeline hazards.

3. EXPERIMENTAL RESULTS

By the moment, we will present some results using the 8xC196 Intel microcontrollers as they are the only ones present all needed execution time information in the user's guide. But we hope soon to present results of experiments with modern processor such as, some Texas Instruments DSPs, Intel 8xC296, PICs and so on. At a first stage, the WCET estimator built the call graph given at the lower right quadrant of fig.2 and then, func() identified by the label C_2192 will be processed and providing a similar screen. At the upper right quadrant, information such as execution time of individual basic blocks, basic block control flow and function execution time are presented. At the lower right quadrant, can be presented the assembler code translated by the disassembler from the executable code, the call graph and the simulator state. The upper left quadrant presents parts our little language program describing the microcontroller architecture.

4. CONCLUSIONS

A very friendly tool for the WCET estimation was developed and the results obtained over some Intel microcontroller were very satisfactory. To a complete evaluation of our tool we will realize more test using other classes of processors such as DSPs, PICs and some Motorola microcontrolers. A plenty use of this tool requires some processors informations, such as, the execution time of each instructions composing the processor instruction set, sometimes not provided in the processor user's guide. In such case, to time an individual instruction, we recommended the use of the logic analyzer to trigger on the opcode at the target instruction location and on the opcode and location of the next instruction.

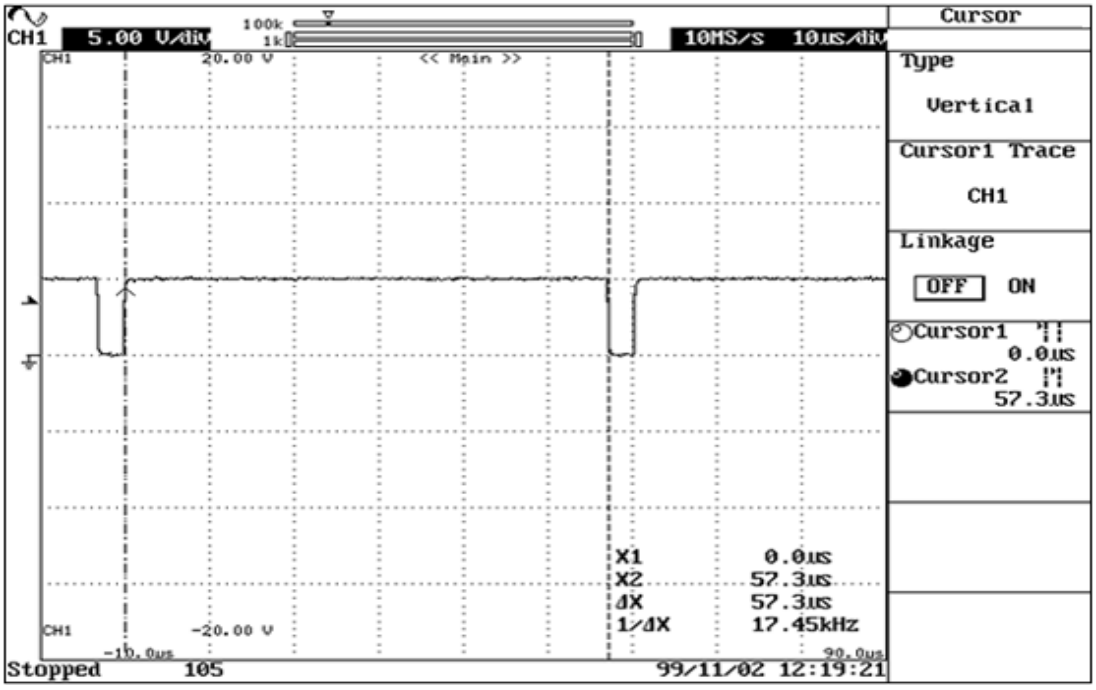


Fig. 1. Digital Oscilloscope triggers on the opcode of the first and last instructions of a code segment

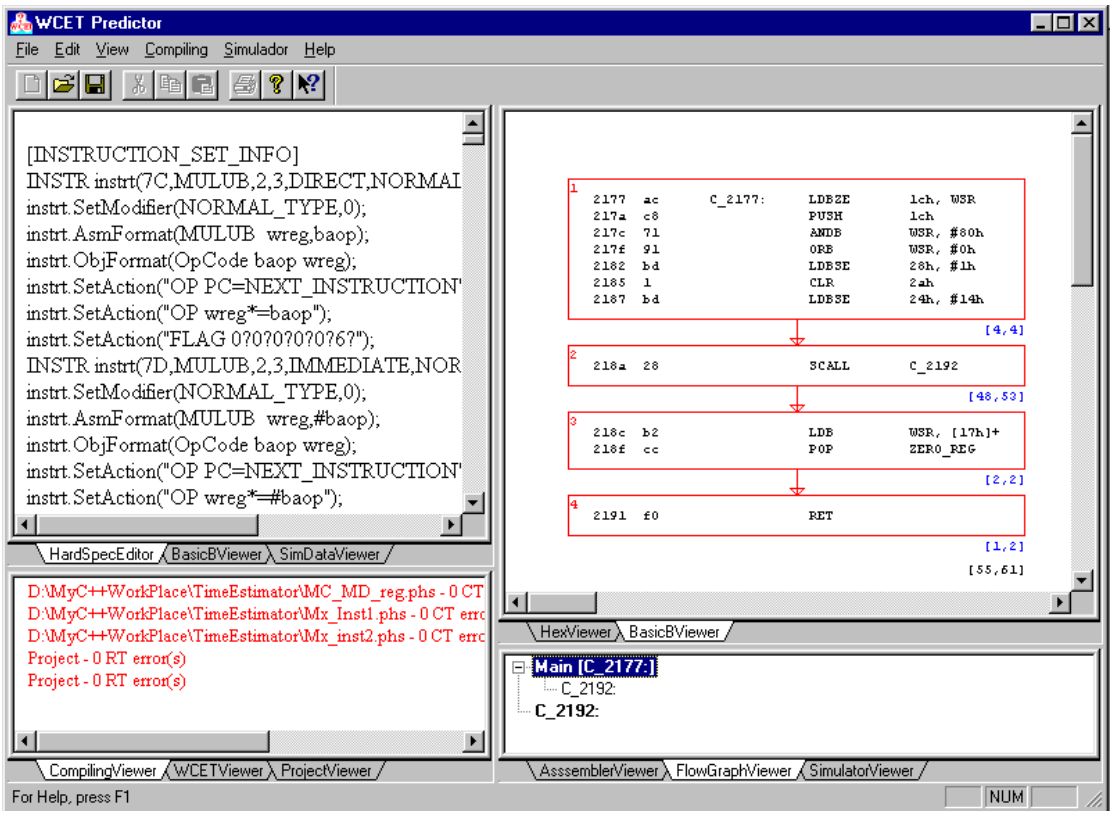


Fig. 2. WCET = 61μs was statically estimated using the developed tool over the same code segment

REFERENCES

[1] Alan C. Shaw, Deterministic Timing Schema for Parallel Programs, *Technical Report 90-05-06*, Department of Computer Science and Engineering, University of Washington, Seattle, 1990

[2] K. Nilsen & B. Rygg, Worst-Case Execution Time Analysis on Modern Processor, *ACM SIGPLAN Notices*, Vol. 30, No. 11, pp. 20-30, November 1995

[3] Y. Steven Li et al., Efficient Microarchitecture Modeling and Path Analysis for Real-Time Software, *Technical Report*, Department of Electrical Engineering, Princeton University

[4] C. Healy, M. Harmon & D. Whalley, Integrating the Timing Analysis of Pipelining and Instruction Caching, *Technical Report*, Computer Science Department, Florida State University

[5] Sung-Soo Lim, C. Yun Park et al., An Accurate Worst Case Timing Analysis for RISC Processors, *IEEE Transactions on Software Engineering*, Vol. 21, No. 7, pp. 593 – 604, July 1995

[6] P. Sorenson & J. Tremblay, *The Theory and Practice of Compiler Writing*, McGraw-Hill, ISBN 0-07-065161-2, 1987

[7] C. W. Fraser & T. Proebsting, Detecting Pipeline Structural Hazards Quickly, in *Proc. of the 21th Annual ACM SIGPLAN_SIGACT Symposium on Principles of Programming Languages*, pp. 280-286, January 1994

SESSION I:
PETRI NET-BASED DIGITAL DESIGN

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

IMPLEMENTING A PETRI NET SPECIFICATION IN A FPGA USING VHDL

Enrique Soto[1], Miguel Pereira[2]

[1] Dept. Tecnología Electrónica, Universidad de Vigo, Apdo. Oficial, 36200 Vigo, España, esoto@uvigo.es, http://www.dte.uvigo.es; [2] Intelsis Sistemas Inteligentes S.A. - R&D Digital Systems Department, Vía Edison 16 Polígono del Tambre 15890, Santiago de Compostela (La Coruña), mpereira@intelsis.es

Abstract*. This paper discusses how the FPGA architectures affect the implementation of Petri net specifications. Taking in consideration the observations from that study, a method is developed for obtaining VHDL descriptions amenable to synthesis, and tested against other standard methods of implementation. These results have relevance in the integration of access technologies to high speed telecommunication networks, where FPGAs are excellent implementation platforms.*

Key Words. *Petri nets, VHDL, FPGA, synthesis*

1. INTRODUCTION

In many applications it is necessary to develop control systems based on Petri nets [1]. When a complex system is going to be implemented in a small space, the best solution may be to use a FPGA.

FPGA architectures [2] are divided in many programmable and configurable modules that can be interconnected with the aim of optimizing the use of the device surface. It is necessary to remember that the main problem of PLDs, PALs and PLAs is the poor use of the device surface, that is, the low percentage of logic gates used. This occurs because this kind of programmable device has only one matrix for AND operations and other matrix for OR operations. FPGAs are different because they are composed of small configurable logic blocks (CLB) that work like sequential systems. CLBs are composed of a RAM memory and one or more macrocells. Each CLB RAM memory is programmed with the combinational system that defines the behavior of the sequential system. On each macrocell a memory element (biestable) and a configuration circuit are included. The configuration circuit defines the behavior of the macrocell.

VHDL is a standard hardware description language capable of representing the hardware with independence of its final implementation [3]. It is also widely supported by a number of simulation and synthesis tools.

2. FPGAS AND PETRI NETS

It is necessary to take into account the following points for implementing a sequential system on a FPGA:

- The system must be divided on low complexity subsystems for integrating them on each CLB of the FPGA.
- Usually, CLBs have only 4 or 5 inputs, and one or two outputs (macrocells) and sometimes it is necessary to achieve a strong division of the global system for integrating the subsystems into the CLBs.
- CLBs are interconnected between them through buses. These buses are connected to configurable connection matrices that have a limited capacity. It is necessary to bring near one another subsystems with a strong dependence between them to optimize the use of these connection matrices.

These points can be followed in many cases when implementing a Petri net. There are two kinds of elements in a Petri net: places and transitions. The circuit implementation of these elements is relatively easy, as shown in the schematics of a place and a transition in figure 1. Each one of these elements can be programmed in one or more CLBs following the model of figure 1. That would not be the most compact and efficient design, but it would be the simplest.

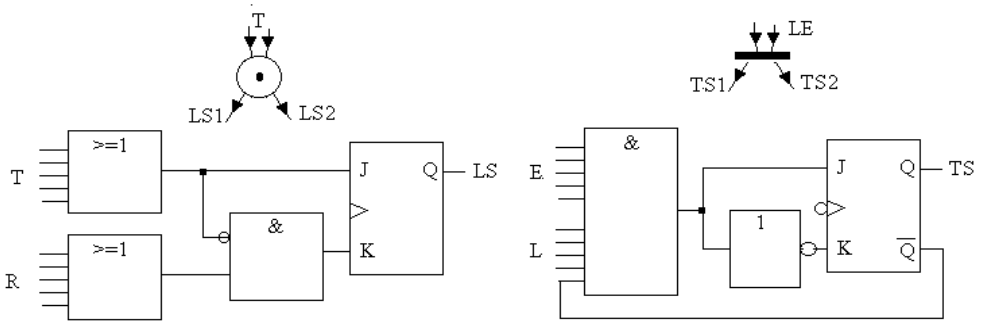


Figure 1. - Electrical schematics of a place and a transition. Each one of them can be implemented on a CLB. The main problem is the low number of inputs in a CLB. Sometimes it is necessary to use more CLBs for each element.
***T** inputs are the signals generated for the preceding transitions.*
***R** inputs are connected to the output of the next transitions.*
***LS** is the output signal of the place.*
***E** inputs are the system inputs involved in the transition.*
***L** inputs are the signals generated for the preceding places.*
***TS** is the output of the transition.*

For obtaining the most compact and efficient design, it is necessary to make the following transformations:

- In the models of figure 1, each place of the Petri net is associated to one bit-state (one-hot encoding). That is not the most compact solution because most of the designs do not need every combination of bits for defining all the states of the system. For instance, in many cases if a place is active implies that a set of places will not be active. Coding the place bits with a reduced number of bits will be a good solution because the number of

CLBs decreases. For instance, if a Petri net with six places has always only one place active, it is enough to use only 3 bits for coding the active place number (binary state encoding).

- Other transformation consists of implementing the combinational circuit of the global system, and dividing the final sequential system (combinational circuit and memory elements).

These transformations are used for making compact and fast designs, but they have some objections:

- When a compacted system is divided, may be too many CLBs have to be used, because of the low number of inputs on each CLB (4 or 5 inputs). This obstacle supposes sometimes to use more CLBs than dividing a non-compacted system.
- Verifying or updating a concrete signal of the Petri net, in a compacted system may be difficult. It is necessary to take into account the achieved transformations, and to supply the inverse transformations for monitoring the signal. This problem can be exposed in failure tolerant systems. This kind of systems needs to verify their signals, while they are running. This system may be more complex if it has been compacted previously.

To avoid the mentioned problems, this paper proposes a solution that consists in implementing the system using special blocks composed of one place and a transition. With this kind of blocks compact systems can be achieved, preserving the Petri net structure. Figure 2 shows an example of Petri net divided in 5 blocks. Each block is implemented in a CLB.

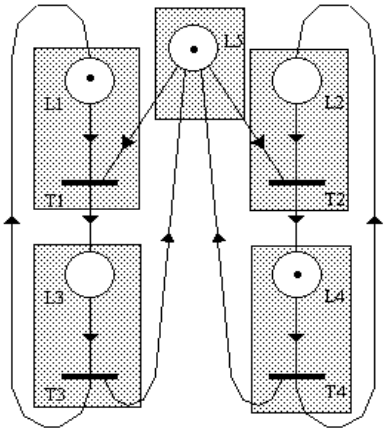


Figure 2. - Example of Petri net divided in blocks for its implementation in a FPGA.

3. IMPLEMENTATION

With this kind of implementation of Petri net based systems, every CLB is composed of a place connected to a transition. The place can be activated through its inputs connected to other transitions. It will be deactivated through reset inputs, or through the transition that is in the block (figure 3).

The transition will be active when the preceding place is active and the transition inputs have the appropriated values. Every block has two outputs, one state bit corresponding to the place, and a transition bit.

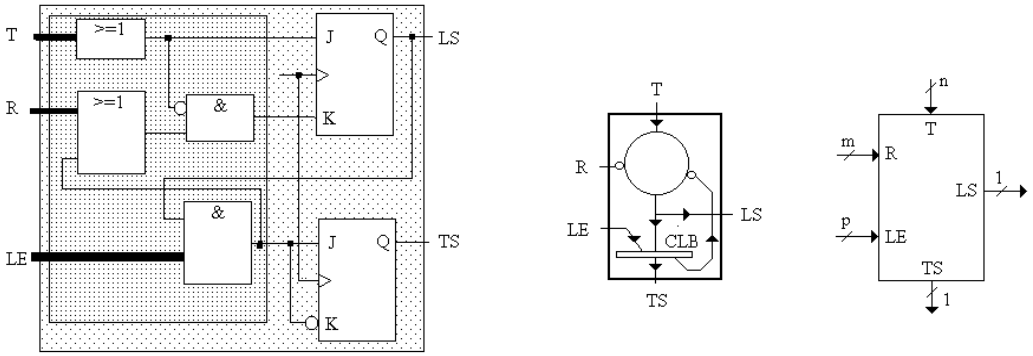


Figure 3. - Description of the new blocks that are developed in a configurable block in a FPGA. On the left, a simplified digital schematic of the block is shown.
*- **T** is the input bits set connected to other transitions for activating the place.*
*- **R** is the vector of signals for deactivating the place since other transitions.*
*- **LE** are the signals coming from other places and other input signals, that let the activation of the transition.*
*- **LS** is the output place.*
*- **TS** is the output transition.*

Figure 3 shows logical and electronic schematics of these blocks. The place and the transition are interconnected through two signals in the block. These signals are not always connected to the exterior. This detail allows a reduction of the CLBs connections in a FPGA. In many cases a concrete CLB has not enough inputs for including a block of this kind. In those cases, it is necessary to use auxiliary CLBs for implementing the block. However, it is unusual to find a Petri net on which every place is preceded by a high number of transitions (or to find a transition preceded by many places). Usually, most places and transitions in a practical Petri net are connected to one or two transitions or places, respectively (except common resources or synchronism points). Figure 4 shows some examples on which there is an element preceded by many others.

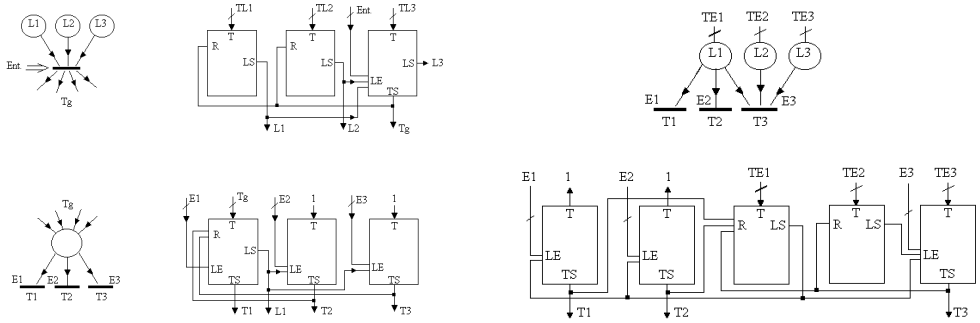


Figure 4. - Examples of different block interconnections for implementing places and transitions with multiple connections to other elements.

There are cases on which the number of CLB inputs is not enough to include a place or a transition in the CLB. Figure 5 shows a logical schematic for expanding the block inputs. In this figure, four CLBs are necessary for implementing the block. Three of them are auxiliary blocks and have the function of concentrating a number of inputs in one signal.

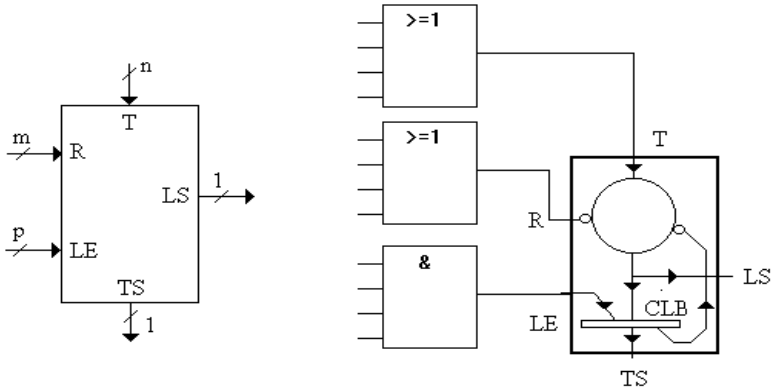


Figure 5. - Block schematic with a high number of inputs. The logic gates connected outside the block place-transition are used for incrementing the number of inputs.

4. VHDL HIGH LEVEL DESIGN

The methodology proposed uses those blocks described above as a set of parametrizable objects available in VHDL libraries. The implementation is simply the interconnection, according to the Petri net specification, of those objects whose correctness is guaranteed. The VHDL description represents correctly the specification as long the Petri net does it. The resulting architecture that is implemented within the FPGA is OHE (one-hot encoding).
This solution gives best results in the implementation of SRAM based FPGAs [4], at least as long as the number of places and the random logic associated with the transitions is not too complex relative to the combinational logic available in the FPGA.

5. EXAMPLE

Figure 6 shows the blocks interconnection of a Petri net based system on a FPGA. The example exposed corresponds to the net of figure 2.

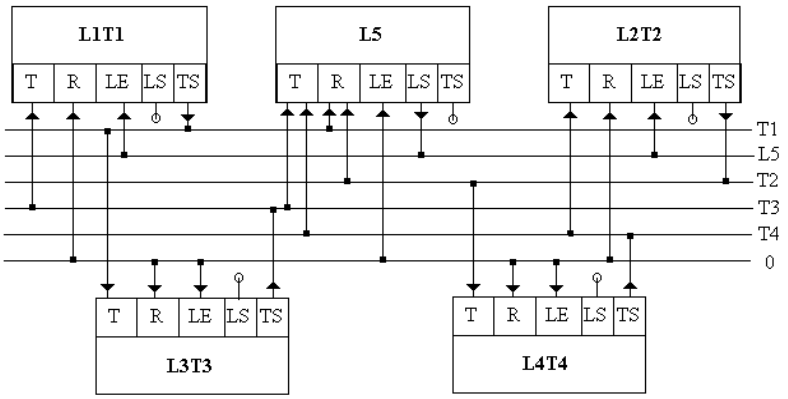


Figure 6. - Schematic of the connections for the Petri net of figure 1 with configurable blocks.

Each block is a CLB of the FPGA, and it is not necessary to include auxiliary blocks for incrementing the number of inputs of the elements of the net. There is only a place with two input signals in the net of figure 2. The rest of places have only one input signal. If some element had more than two inputs, it would be necessary to use the structures of figure 4, and then the number of CLBs would be increased. The results of different design methodologies using a sample FPGA are summarized in table 1.

Table 1

Design method	Design process	FPGA resources in use	Device Frequency Achieved
Schematic	Difficult	17 %	27,62Mhz
VDHL behavioral	Simple	21 %	16,75Mhz
This paper	Simple	12 %	63,69Mhz

6. CONCLUSIONS

In this paper, the implementation of Petri net based systems on FPGAs has been discussed. The main problem consists of using places and transitions with a different number of inputs, including the case when there are more inputs than a configurable block of a FPGA. For that, a method has been developed through two circuit models, one of them for places, and the other one for transitions. With these models a new block has been presented that contains a place interconnected with a transition. The object of this block consists in reducing the interconnections between CLBs in a FPGA, and therefore, reducing the number of inputs on each block (specially, feedback signals necessary to reset preceding places). This method is optimal for Petri nets on which most places and transitions are preceded for one or two (but no more) transitions or places. Furthermore, some possibilities have been exposed for the interconnection of blocks that increase the number of inputs in elements of a Petri net. The main purpose of this method is to integrate the maximum number of elements of a Petri net in a FPGA.

REFERENCES

[1] Zurawski, R., M.C. Zhou. "Petri Nets and Industrial Applications: a Tutorial". IEEE Trans. on Industrial Electronics, Dec. 1994.
[2] FLEX8000 HandBook ALTERA Corp. 1994.
[3] Soto, E., Mandado, E., Farina, J.. 'Lenguajes de descripcion hardware (HDL): el lenguaje VHDL'. Mundo Electronico, abril 1993
[4] Nemec, John. 'Stoke the fires of FPGA design. Keep an FPGA's architecture in mind and produce designs that will yield optimum performance and efficiency'. Electronic Design, October 25, 1994.

ACKNOWLEDGEMENTS

This work was financed by the European Commission and the Comisión Interministerial de Ciencia y Tecnología (Spain) through research grant TIC 1FD97-2248-C02-02 in collaboration with the company Versaware S.L. (Vigo, Spain).

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

DIGITAL HARDWARE IMPLEMENTATION OF PETRI NET BASED SPECIFICATIONS: DIRECT TRANSLATION FROM SAFE AUTOMATION PETRI NETS TO CIRCUIT ELEMENTS

Murat UZAM, Mutlu AVCI and M. Kürşat YALÇIN

Niğde Üniversitesi, Mühendislik-Mimarlık Fakültesi, Elektrik-Elektronik Mühendisliği Bölümü,
51100 NİĞDE, TURKEY. Tel: ++ 90 388 225 01 15, Fax: ++ 90 388 225 01 12,
e-mail: murat_uzam@hotmail.com, <mutluavci, kursat_yalcin>@yahoo.com

Abstract: *In this paper, a new method is proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper is to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions.*

Key Words. *Discrete event systems, Petri nets, hardware implementation, asynchronous controllers.*

1. INTRODUCTION

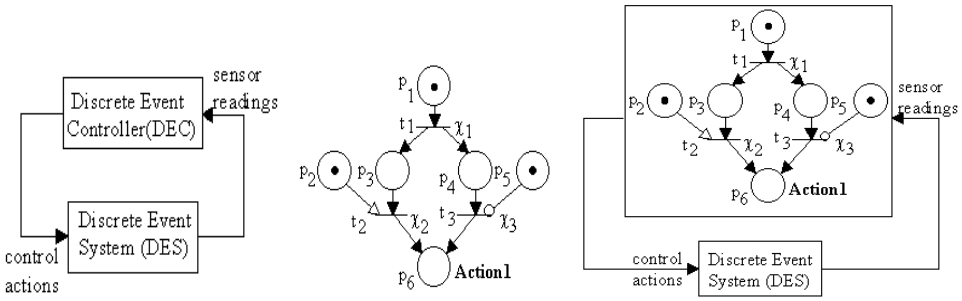
Discrete event systems (DES), examples of which include communication networks, manufacturing systems, robots, etc., exhibit properties such as non-determinism, conflict, and concurrency. The study (i.e. design, analysis, synthesis, etc.) of DES has been carried out mainly by using two modelling techniques: finite state machines (FSM) and Petri nets. FSM based studies suffer from so called state explosion problem. FSMs provide sequential models. When using FSMs graphical visiualisation of the modelled system can not be realised easily [2]. Petri nets have been used as an alternative formalism for the study of DESs due to their easily understood graphical representation in addition to their well-formed mathematical formalism. In this paper, we are concerned with the control of DES and we use Petri nets. The control of DESs is done firstly by modelling the controller as a Petri net model and then by implementing it in software or hardware. The implementation is carried out by simulating the Petri net model in terms of software or hardware structures. The software implementation has been done using either high level languages or low level languages. Synchronous or asynchronous controllers of Petri net based specifications have been obtained as hardware implementation. For a detailed information on Petri nets and digital hardware design, the reader is referred to [3]. In an asynchronous circuit, there is no global clock, i.e. they are self

timed. Asynchronous circuits can be viewed as hardwired versions of parallel and distributed programs. The program statements are physical components, i.e. logic gates, memory elements, etc. Asynchronous circuits are better than the synchronous counterparts in terms of performance, robustness, low power, low electromagnetic emission, modularity and re-use, and testability [3]. In this paper, we deal with asynchronous circuit implementation of Petri net based specifications. This type of implementation is carried out based on the idea of "physical simulation" and achieved by associating each place in the Petri net with a memory latch. Examples of this style can be found in [1, 4, 5, 6].

In this paper, a new method is proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper is to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions. The remainder of this paper is organised as follows: The next section defines safe automation Petri nets (SAPN). In the 3rd section, the digital hardware implementation of SAPN is explained. Finally, conclusions are given.

2. SAFE AUTOMATION PETRI NETS

Automation Petri nets (APN) have recently been introduced as a new formalism for the design of Discrete Event Control Systems [2]. Since ordinary Petri nets do not deal with sensors and actuators, the Petri net concepts are extended, by including actions and sensor readings as formal structures within the APN. These extensions involve extending the Petri nets to accommodate sensor signals at transitions and to assign level actions to places (and similarly to assign impulse actions to transitions). In this section, we define Safe (1-bounded) Automation Petri nets (SAPN) to be used for direct translation from Petri nets to circuit elements. A typical discrete event control system (DECS) is shown in Figure 1.(a). It consists of a discrete event system (DES), to be controlled and a discrete event controller (DEC). Sensor readings are regarded as inputs from the DES to the DEC, and control actions are considered as outputs from the DEC to the DES. The main function of the DEC is to supervise the desired DES operation and to avoid forbidden operations. To do this, the DEC processes the sensor readings and then it forces the DES to conform to the desired specifications through control actions. Petri nets can be used to design such DECs. However, ordinary Petri nets do not deal with actuators or sensors. Because of this, it is necessary to define a Petri net-based controller (Automation Petri net - APN), which can embrace both actuators and sensors within an extended Petri net framework. An SAPN is shown in Figure 1.(b). In the SAPN, sensor readings can be used as firing conditions at transitions. The presence or absence of sensor readings can be used in conjunction with the extended Petri net pre-conditions to fire transitions. In the SAPN, two types of actuations can be considered, namely impulse actions and level actions. Level actions are associated with places, while impulse actions are associated with transitions. With these additional features, it is possible to design Discrete Event Control Systems. Figure 1.(c) shows how an SAPN can be used as a DEC in a DECS.



(a) *(b)* *(c)*

Fig. 1. (a). A typical discrete event control system (DECS). (b). Safe Automation Petri Net (SAPN). (c). APN used as a controller in a DECS.

Formally, a Safe Automation Petri Net can be defined as follows:

$$\mathbf{SAPN = (P, T, Pre, Post, In, En, \chi, Q, M_0)}$$

Where,

- $P = \{p_1, p_2, ..., p_n\}$ is a finite, nonempty set of places,
- $T = \{t_1, t_2, ..., t_m\}$ is a finite, nonempty set of transitions, $P \cup T \neq \varnothing$ and $P \cap T = \varnothing$,
- Pre: $(P \times T) \rightarrow \{0,1\}$ is an input function that defines ordinary arcs from places to transitions,
- Post: $(T \times P) \rightarrow \{0,1\}$ is an output function that defines ordinary arcs from transitions to places,
- In: $(P \times T) \rightarrow \{0,1\}$ is an inhibitor input function that defines inhibitor arcs from places to transitions,
- En: $(P \times T) \rightarrow \{0,1\}$ is an enabling input function that defines enabling arcs from places to transitions,
- $\chi = \{\chi_1, \chi_2, ..., \chi_m\}$ is a finite, nonempty set of firing conditions associated with transitions,
- $Q = \{q_1, q_2, ..., q_n\}$ is a finite set of level actions that might be assigned to places or impulse actions that might be assigned to transitions,
- $M_0 : P \rightarrow \{0,1\}$ is the initial marking.

The SAPN consists of two types of nodes called *places*, represented by circles (○), and *transitions*, represented by bars (—). There are three types of arcs used in the SAPN, namely, *ordinary arcs*, represented by a directed arrow (⟶), *inhibitor arcs*, represented by an arc, whose end is a circle (—○), and finally *enabling arcs*, represented by a directed arrow, whose end is empty (—▷). Directed ordinary arcs connect places to transitions and vice versa, while enabling and inhibitor arcs connect only places to transitions. The number of tokens in places represent the current state of the system and firing of a transition represents the movement of the system from one state to another state. Each transition has a set of input and output places, which represent the pre-condition and post-condition of the transition. The level actions (Q), may be assigned to places, and the impulse actions may be assigned to transitions. Level actions may be enabled when there is a token at a place, while impulse actions may be enabled at the instant when a transition is fired. More than one action may be assigned to a place or a transition. Firing conditions in the SAPN are recognised by external events (signals) such as sensor readings, switch positions, etc. Six firing conditions, may be associated with a transition *t*: χ, $\overline{\chi}$, $\uparrow\chi$, $\uparrow\overline{\chi}$, $\chi\downarrow$ and, $\overline{\chi\downarrow}$. The firing condition χ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in

which case related transition *t* is allowed to fire if it is enabled, i.e. all input places have one token each. The firing condition $\overline{\chi}$ is the complement of the firing condition χ and is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. The rising-edge-firing-condition $\uparrow\chi$ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in which case related transition *t* is allowed to fire if it is enabled. The complement-rising-edge-firing condition $\overline{\uparrow\chi}$ is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. The falling-edge-firing-condition $\chi\downarrow$ is a Boolean variable that can be 1, in which case related transition *t* is not allowed to fire, or it can be 0, in which case related transition *t* is allowed to fire if it is enabled. Finally, The complement-falling-edge-firing-condition $\overline{\chi\downarrow}$ is a Boolean variable that can be 0, in which case related transition *t* is not allowed to fire, or it can be 1, in which case related transition *t* is allowed to fire if it is enabled. The marking of the SAPN is represented by the number of tokens in places. Tokens are represented by black dots (•). Movement of tokens between places describes the evolution of the SAPN and is accomplished by the firing of the enabled transitions. The following rules are used to govern the flow of tokens:

Enabling Rules:

1. If the input place p_1 of a transition t_1 is connected to t_1 with an ordinary arc $Pre(p_1,t_1)$, then t_1 is said to be enabled when p_1 contains a token, i.e., $M(p_1) = 1$.
2. If the input place p_1 of a transition t_1 is connected to t_1 with an enabling arc $En(p_1,t_1)$, then t_1 is said to be enabled when p_1 contains a token, i.e., $M(p_1) = 1$.
3. If the input place p_1 of a transition t_1 is connected to t_1 with an inhibitor arc $In(p_1,t_1)$, then t_1 is said to be enabled when p_1 contains no token, i.e., $M(p_1) = 0$.

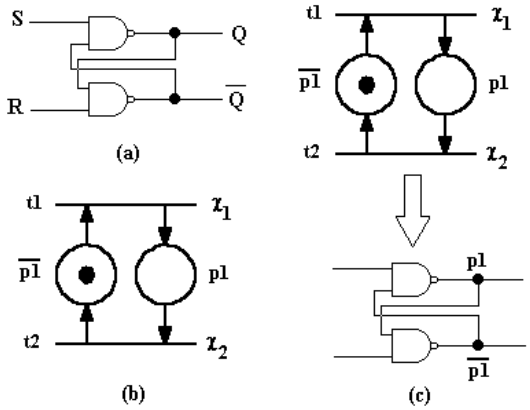
Firing Rules: In the SAPN, an enabled transition *t* can or can not fire depending on the external firing condition χ of *t*. These firing conditions can be either of the above mentioned firing conditions, i.e. χ, $\overline{\chi}$, $\uparrow\chi$, $\overline{\uparrow\chi}$, $\chi\downarrow$ or $\overline{\chi\downarrow}$, namely positive level, zero level, rising edge, complement rising edge, falling edge or complement falling edge of a (signal) sensor reading or a switch position. Broadly speaking, a firing condition of a transition *t* may include more than one sensor reading with 'AND', 'OR' and 'NOT' logical operators. When dealing with more than one sensor readings as firing conditions, the logical operators of firing conditions must be taken into account accordingly. In the special case, where $\chi = 1$, transition *t* is always allowed to fire when it is enabled. When an enabled transition *t* fires with a related firing condition χ, it removes one token from each input place p_i and deposits, at the same time, one token to each output place p_o. It should be noted that, the firing of an enabled transition *t* does not change the marking of the input places that are connected to *t* only by an enabling or an inhibitor arc.

3. DIGITAL HARDWARE IMPLEMENTATION OF SAFE AUTOMATION PETRI NETS : DIRECT TRANSLATION FROM SAPN TO CIRCUIT ELEMENTS

The direct translation method from SAPN to circuit elements, proposed in this paper, is based on the idea of physical simulation of every Petri net marking reachable from the initial marking in terms of the state of the circuit. In order to achieve the direct translation, the following three steps are followed: i) each place in the SAPN is associated with a memory element, i.e. an SR-flip-flop (SR-latch), ii) each transition in the SAPN is implemented with a

logical gate (NAND gate), iii) the initial marking is set-up by using an RC (Resistor and Capacitor) element. Let us now consider each of these three steps:

i) The first step in achieving the direct translation is to use a memory element to represent the presence or absence of a token in a place. If there is a token in a place then the output of the memory element is set to 1. In contrast, if there is no token in a place then the output of the memory element is reset to 0. To implement this operation we use an SR-flip-flop, as shown in Fig. 2.(a). An SR-flip-flop is constructed from two NAND gates connected back to back. The cross-coupled connections from the output of one gate to the input of the other gate constitutes a feedback path. Therefore, the circuit is classified as asynchronous. Each flip-flop has two outputs; Q and $\overline{Q}$, and two inputs; set (S) and reset (R). The truth table of the SR-flip-flop is given in Fig. 2.(d). The application of a momentary 0 to the S input causes output Q to go to 1 and $\overline{Q}$ to go to 0. The outputs of the circuit do not change when the S input returns to 1. A momentary 0 applied to the R input causes an output of $Q = 0$ and $\overline{Q} = 1$. The state of the flip-flop is always taken from the value of its normal output Q. When $Q = 1$, we say that the flip-flop stores a 1 and is in the *set* state. When $Q = 0$, we say that the flip-flop stores a 0 and is in the *reset* state. The SR-flip-flop manifests an undesirable condition if both inputs go to 0 simultaneously. When both inputs are 0, outputs Q and $\overline{Q}$ will go to 1, a condition which is normally meaningless in flip-flop operation. In Fig. 2.(b) an SAPN is shown, in which there are two places; p1 and its complement $\overline{p1}$, and two transitions; t1 and t2, with firing conditions χ_1 and χ_2 respectively. This is an explicit representation of the safe place p1. The implementation of places using the SR-flip-flop is shown in Fig. 2.(c). Output Q of the SR-flip-flop is used to represent place p1 and output $\overline{Q}$ is used to represent place $\overline{p1}$. When there is a token in p1, the SR-flip-flop is set, i.e. $Q = 1$ and $\overline{Q} = 0$. When there is a token in $\overline{p1}$, the SR-flip-flop is reset, i.e. $Q = 0$ and $\overline{Q} = 1$. It is assumed that the model will not permit both outputs becoming $Q = 1$ and $\overline{Q} = 1$. That is to say that the designer must take some action to assure that S = R = 0 will never occur.



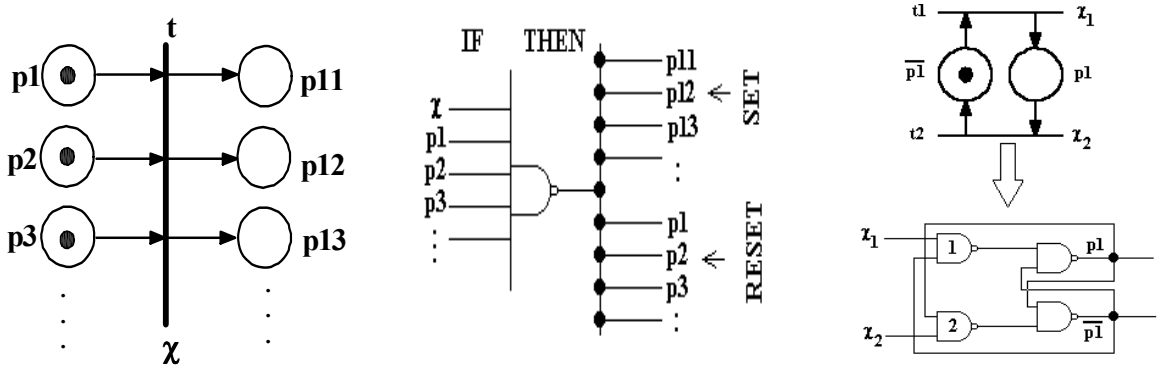
S	R	Q	$\overline{Q}$	Comment
0	1	1	0	Set
1	1	1	0	After S = 0 and R = 1
1	0	0	1	Reset
1	1	0	1	After S = 1 and R = 0
0	0	1	1	Not allowed

(d)

Fig. 2. a). An SR-flip-flop, b) an SAPN, c) the implementation of places. d). The truth table of the SR-flip-flop,

ii) The second step in achieving the direct translation is to use a NAND gate to implement transition in SAPN. The behaviour of a transition in SAPN may be summarised as follows: IF there is a token each in the input places of a transition t AND the firing condition χ of t occurs, THEN all the tokens are removed from the input places and one token each is deposited to the output places of t. To show how this behaviour is implemented, the SAPN shown in Fig. 3.(a) is used. In this case, the transition t fires when all input places p1, p2, p3,

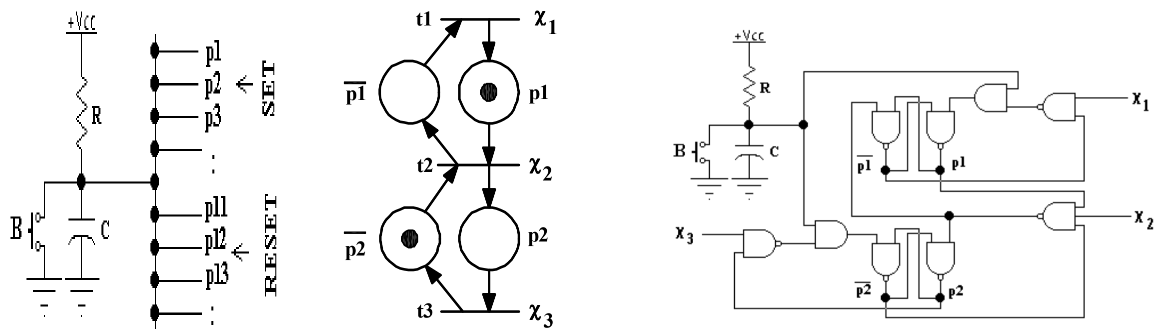
... have one token each and the firing condition χ occurs. When *t* fires it removes all the tokens from the input places p1, p2, p3, ..., and at the same time, it deposits one token each to the output places, p11, p12, p13,... . To implement the transition *t*, the structure shown in Fig. 3.(b) is used. In this case, when all input flip-flops are set and χ occurs *t* is fired by resetting all the input flip-flops and at the same time by setting all the output flip-flops. Please note that the difference between our approach and [1] is that, in [1] the removal of tokens from input places and adding tokens to output places has a duration and there is an intermediate state between the two operations, while in our approach the removal of tokens from input places and adding tokens to output places is instantaneous. Fig. 3.(c) shows the implementation of transitions t1 and t2 of Fig. 2(c). In the SAPN, t1 fires when there is a token in $\overline{p1}$ and χ_1 occurs. When fired, t1 removes the token from $\overline{p1}$ and deposits a token in p1. The NAND gate 1 implements t1 as follows: when output $\overline{p1} = 1$ and χ_1 occurs, i.e. χ_1 becomes 1, p1 is set to 1 by applying an instantaneous 0 from the output of the NAND gate 1 to the S input of the flip-flop and at the same time the output $\overline{p1}$ is reset, i.e. $\overline{p1} = 0$. The same applies to t2 in a similar manner.



(a) *(b)* *(c)*

Fig. 3. a). A transition in SAPN. b).Implementation of the transition, b). Implementation of transitions t1 and t2 of Fig. 2.(c).

iii) The third and last step is about setting-up the initial marking by using an RC (Resistor and Capacitor) element. It is necessary for proper functioning to set-up the initial marking before operating the circuit. It is a common practice to use an RC element to establish the power on reset (POR) and at the same time to use a button connected parallel to the capacitor such that at any time desired by pressing the button we are able to set the system back to the initial marking. The time delay $\tau = R.C$ defines how long the setting-up time will be for the initial marking. How this process is accomplished, is shown in Fig. 4.(a). When the power is first applied to the circuit, a 0 is applied, for the period of τ time, to the S inputs of flip-flops, which represent places with initial marking 1, i.e. all places p1, p2, p3, ... are set to 1. At the same time, a 0 is also applied to the R inputs of flip-flops, which represent places with initial marking 0, i.e. all places p11, p12, p13, ... are reset to 0. After the power is being applied to the circuit, at any time it is also possible to set-up the circuit back to the initial marking by pressing the button B. An example SAPN is shown in Fig. 4.(b). The hardware implementation of the SAPN shown in Fig. 4.(b) is given in Fig. 4.(c). In this circuit places and transitions are implemented as described before. The initial marking, i.e. $M_0(p1, \overline{p1}, p2, \overline{p2}) = (1,0,0,1)^T$, is implemented by setting the first flip-flop and by resetting the second flip-flop.



(a) (b) (c)

Fig. 4. a). Setting-up the initial marking. b). An example SAPN. c). Setting-up the initial marking of the SAPN shown in Fig. 4.(b).

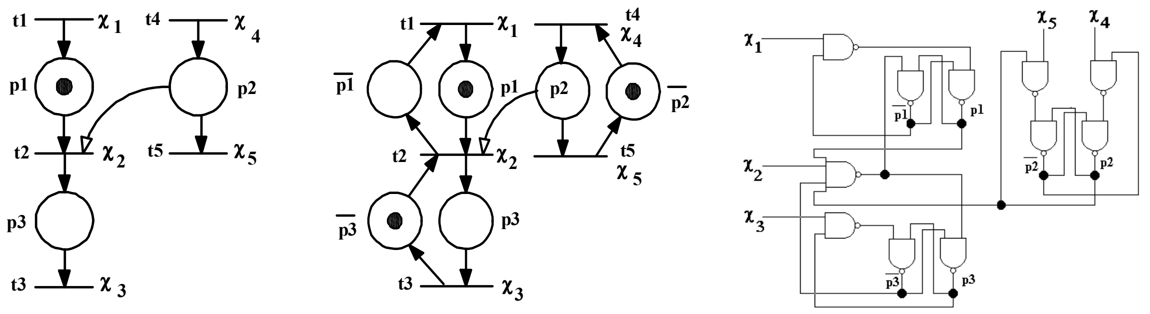
To show how our technique is applied to the hardware implementation of the SAPN, in this paper we consider the following SAPN structures:

- Enabling arc
- Inhibitor arc
- Actions

Please, note that for the sake of simplicity the implementation of the initial marking in the following SAPN structures are not shown.

3.1. Hardware Implementation of the Enabling Arc

The modelling power of Petri nets can be extended by adding the '*one testing*' ability, i.e., the ability to test whether a place has a token. This is achieved by introducing an *enabling arc*. The enabling arc connects an input place to a transition and is represented by a directed arrow, whose end is empty. The presence of an enabling arc connecting an input place to a transition means that the transition is only enabled if the input place has a token. The firing does not change the marking in the enabling arc connected places. In an SAPN, an enabling arc, *En(p*$_2$*,t*$_2$*)*, is shown in Fig. 5.(a). The transition t2 is fired if both p1 and p2 have one token each and firing condition χ_2 occurs. When t2 is fired, a token is removed from place p1 and a token is deposited into the output place p3, but the marking of enabling arc connected place p2 does not change. The transition t2 is not enabled to fire, if there is no token in place p2. Fig. 5.(b) shows the complement places of places p1, p2 and p3. The hardware implementation of the SAPN shown in Fig. 5.(b) is given in Fig. 5.(c). In this circuit places and transitions are implemented as described before.



(a) (b) (c)

Fig. 5. a). An enabling arc, En(p$_2$*,t*$_2$*), in an SAPN. b). The complement places of places p1, p2 and p3. c). Hardware implementation of the SAPN shown in Fig. 5.(b).*

3.2. Hardware Implementation of the Inhibitor Arc

The modelling power of Petri nets can be extended by adding the '*zero testing*' ability, i.e., the ability to test whether a place has no token. This is achieved by an *inhibitor arc*. The inhibitor arc connects an input place to a transition and is represented by an arc, whose end is a circle. The presence of an inhibitor arc connecting an input place to a transition means that the transition is enabled if the input place has no token. The firing does not change the marking in the inhibitor arc connected places. In an SAPN, an inhibitor arc, $In(p_2,t_2)$, is shown in Fig. 6.(a). The transition t2 is fired if place p1 has a token and p2 has no token and firing condition χ_2 occurs. When t2 is fired, a token is removed from the input place p1 and a token is deposited into the output place p3, but the marking of inhibitor arc connected place p2 does not change. The transition t2 is not enabled to fire, if there is a token in place p2. Fig. 6.(b) shows the complement places of places p1, p2 and p3. Note that the inhibitor arc $In(p_2,t_2)$, shown in Fig. 6.(a), can be replaced by the enabling arc $En(p_2,t_2)$. The hardware implementation of the SAPN shown in Fig. 6.(b) is given in Fig. 6.(c). In this circuit, places and transitions are implemented as described before.

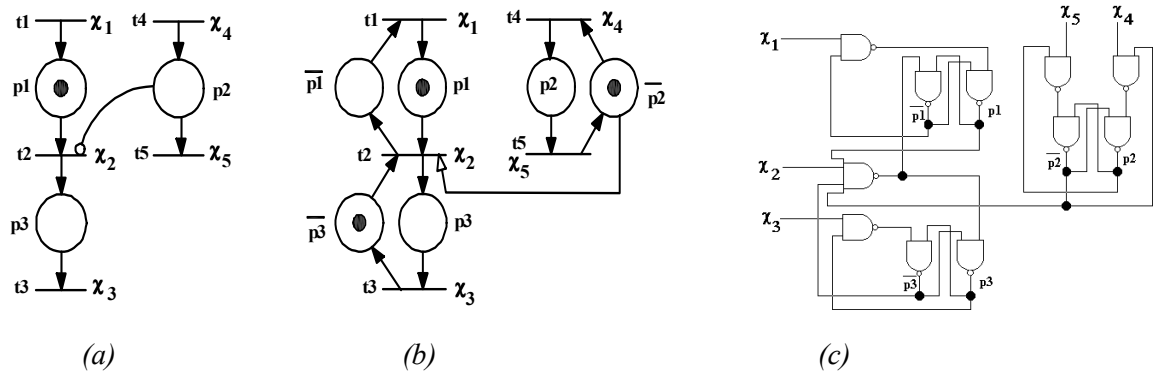


Fig. 6. a). An inhibitor arc, $In(p_2,t_2)$, in an SAPN. b). The complement places of places p1, p2 and p3. c). Hardware implementation of the SAPN shown in Fig. 6.(b).

3.3. Hardware Implementation of Actions

In the SAPN, two types of actuations can be considered, namely impulse actions and level actions. Impulse actions are associated with transitions and they are enabled at the instant, when the related transition is being fired. Level actions are associated with places and they are enabled when there is a token in the related place. More than one action may be assigned to a transition or a place. Fig. 7.(a) shows an SAPN in which there is an impulse action assigned to t2, and there is a level action assigned to p3. The hardware implementation of the SAPN shown in Fig. 7.(a) is given in Fig. 7.(b). In this circuit, places and transitions are implemented as described before.

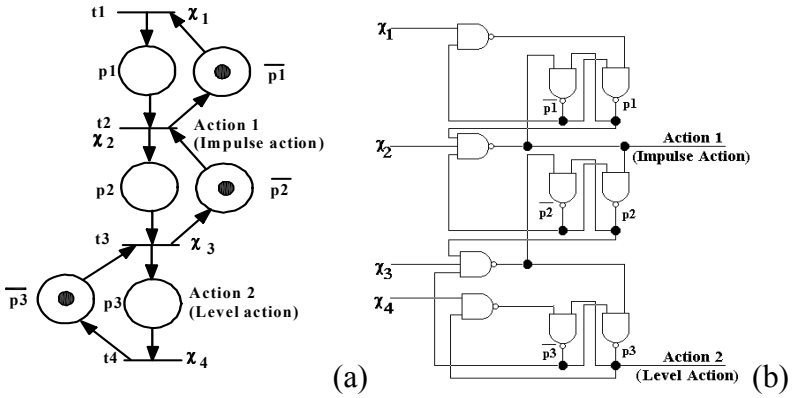


Fig. 7. a). Actions in an SAPN. b). Hardware implementation of the SAPN shown in Fig. 7.(a).

4. CONCLUSIONS

In this paper, a new method has been proposed for the digital hardware implementation of Petri net based specifications. The purpose of this paper has been to introduce a new discrete event control system paradigm, where the control system is modelled with extended Petri nets and implemented as an asynchronous controller using circuit elements. The results provided in this paper on the digital hardware implementation of Petri nets may be view as a better version of a previously introduced method [1], in terms of the implementation of transitions. Some Petri net structures, such as join, merge, fork, conflict, toggle, select, timed-transition have already been implemented by using our methodology, but due to the limited space they are not shown in this paper. Although the implementations considered in this paper are only for safe APNs, it is also possible to apply our method to general APNs and use up/down counters to represent places, instead of flip-flops. The results reported in this paper have already been applied to the control of an experimental manufacturing system. Our forthcoming publications will be about these results.

REFERENCES

1. V. Varshavsky & V. Marakhowsky, "Hardware Support for Discrete Event Coordination", *Proc. of Int. Workshop on Discrete Event Systems (WODES'96)*, Edinburg, UK, pp. 332-340, IEE, August 1996.
2. M. Uzam, *Petri-Net-Based Supervisory Control of Discrete Event Systems and Their Ladder Logic Diagram Implementations*. The University of Salford, UK, PhD. Thesis, 395 pages, 1998.
3. A.V. Yakovlev & A.B. Koelmans, "Petri Nets and Digital Hardware Design", *Lecture Notes in Computer Science, Lectures on Petri Nets II: Applications, Advances in Petri Nets, W. Reisig & G. Rozenberg (Eds.)*, Springer, pp. 154-236, 1998.
4. L.A. Hollaar, "Direct Implementation of Asynchronous Control Circuits", *IEEE Trans. On Computers, C-31(12)*, pp. 1133-1141, 1982.
5. V. Varshavsky, M. Kishinevsky, V. Marakhovsky, V. Peschansky, R. Rosenblum, A. Taubin, & B. Tzirlin, "Self-Timed Control of Concurrent Processes", *Kluwer Academic Publishers, Dordretch, The Netherlands*, 1990, V.I. Varshavsky, Ed.
6. V. Varshavsky & V. Marakhowsky, "Asynchronous Control Device Design by Net Model Behaviour Simulation", Lecture Notes in Computer Science, Vol. 1091: Proc. of the 17th Int. Conf. On Applications and Theory of Petri Nets, Osaka, pp. 497-515. Springer Verlag, 1996.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

ON ALGORITHMS FOR DECYCLISATION OF ORIENTED GRAPHS

Andrei KARATKEVICH

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *A.Karatkevich@iie.pz.zgora.pl*

Abstract. *The task of decyclization of the oriented graphs is considered. It can be applied to analysis of Petri nets, evaluation of circuits and programs etc. The known methods are reviewed; but they are not effective for big graphs. For the applications exact minimisation of the number of removed arcs usually is not necessary, so a quick heuristic algorithm allowing obtaining an approximate solution would be useful. Such algorithm is described in the paper. The exact methods are also discussed.*

Key Words. *Oriented graphs, circuits, verification, combinatorial tasks*

1. BACKGROUND AND THE PROBLEM STATEMENT

Oriented acyclic graphs are widely used in applications (in logical design algorithms, for example), being a convenient way of specifying partial order. If partial order is required, it is necessary to be sure whether the graph is acyclic or it has oriented cycles (circuits). If it has circuits, it may be necessary to transform it into an acyclic graph by removing some arcs; and usually it is better to remove minimal number of arcs enough to destroy all the circuits. So the task [1] is formulated as follows.

For given oriented graph remove minimal number of arcs to obtain acyclic graph.

An alternative formulising: for given oriented graph find maximal acyclic spanning subgraph.

This operation is called *minimal decyclisation* of oriented graph. Finding any acyclic spanning subgraph is called *decyclisation*.

Decyclisation can be applied to analysis of electronic circuits (the minimal set of arcs removing all the circuits of graph is known in electrical engineering as *minimal feedback arc set*). Feedback is a difference between combinational and sequential logical circuits, and the algorithms of logical design and analysis are very different for those two kinds of circuits; of course the algorithms for combinational circuits are much simpler. So selecting of maximal combinational sub-circuit of given circuit can be useful for circuit minimisation, test generation and other CAD and analysis tasks. Decyclisation also can be used for analysis of algorithms represented by graph models, such as PERT (*program evaluation and review technique*), graph-schemes or Petri nets; for disjunction of feedback in the control systems. The author encountered this problem developing algorithms of optimal simulation of Petri

nets [6]. One more area of application is deductive logic, where circuit in graph means a vicious circle.

2. MAIN DEFINITIONS

An *oriented graph*, or *orgraph G=(V, E)* is a set *V* of *nodes* together with a set *E* of *arcs* which are ordered pairs of the elements of *V*. The arc *e* is said to be *directed from* its *initial node* init(*e*) *to* its *terminal node* ter(*e*). An orgraph can be specified by its *adjacency matrix* **R** that is a square Boolean matrix n×n, where n=|*V*|; for that matrix a_{ij}=1 if and only if there is an arc from node *i* to node *j*.

Number of incoming arcs for node *v* is its *input degree* and is denoted as id(*v*); number of outgoing arcs is its *output degree* and is denoted as od(*v*) [2].

A *circuit* in an orgraph is a path along the arcs of the graph originating and terminating at the same node. An *elementary circuit* is a circuit coming through every node no more than once. Below saying "circuit" we always mean "elementary circuit". A *circuit matrix* **C** is a Boolean matrix m×p, where m – number of circuits, p=|*E*|; for that matrix a_{ij}=1 if and only if the arc *j* is in circuit *i* [5].

3. THE METHODS OF MINIMAL DECYCLISATION

The known methods of decyclisation of orgraphs are summarised in [1] (see also [8]). Also the effects of removing arcs from orgraphs are discussed in details in [4], but the tasks considered there are slightly different from the task we are interested in. The methods of minimal decyclisation exist, but they cannot be implied to big graphs because they need exponential time. The main approaches are listed below.

3.1. Method Using Adjacency Matrix

In [1] it is shown that an orgraph is acyclic if and only if its nodes can be ordered in such a way that its adjacency matrix becomes strictly triangular (without non-zero elements on the diagonal; such ordering can be obtained by topological sorting of the graph [10]). One of the methods of decyclisation is based on this affirmation. Its main idea follows: transform the adjacency matrix by exchanging columns (and corresponding rows) to make it as close to a strictly triangular matrix as possible (i.e. with minimal number of non-zero elements below the main diagonal or on it). Obtaining the optimal solution here is a difficult combinatorial task; of course some heuristics can be used here to obtain an approximate solution. Advantage of this method is that it doesn't require finding of all the circuits in the graph.

3.2. Method Using Boolean Equations

Another method is described in [1]: let's find all the circuits in the given graph, and let's associate a Boolean variable with each arc. Then for every circuit build disjunction of the correspondent variables (without negation) and construct conjunction of those disjunctions. The obtained Boolean formula is in the conjunctive normal form. Transform it into disjunctive normal form by usual transformations and simplifications of Boolean equations. Every conjunction in the resulting DNF represents a set of arcs, removing of which is enough to destroy all the circuits. The shortest conjunction represents the minimal set of arcs to be removed.

3.3. Reducing to the Task of Boolean Matrix Covering

It is easy to see that the method described above requires in fact finding all the possible sets of arcs enough for decyclisation (minimal in the sense that no element can be removed from those sets). That means that the method is applicable only for very small graphs. But since the set of all circuits in the graph is obtained, a better approach can be used here. As far as there is a set of circuits and it is necessary to select a minimal set of elements belonging to all the circuits, the task is directly reduced to the task of covering of Boolean matrix. So the general algorithm may consist of the next steps:

Algorithm 1 *(exact)*

1. Find all the circuits in the orgraph.
2. Construct the circuit matrix **C**.
3. Find the minimal covering of the matrix (minimal subset of columns such that every row has at least one non-zero element in one of those columns).
4. Remove from the graph all the arcs corresponding to the rows of the covering.

Finding the minimal matrix covering is a complex combinatorial task, but this approach is much more effective than the previous one, because there is no necessity to exhaust all the combinations of rows to find the optimal solution. However if finding optimal solution takes too much time, some heuristics may be used; exact and approximate methods of matrix covering are explained in details in [13].

The algorithm finding all the circuits in an orgraph is described in [11] (firstly in [3]). Both tasks (finding all circuits and minimal matrix covering) are NP-complete (the number of circuits in a graph exponentially depends on its size), so the only practical way of solving the task for a big graph is using heuristics.

4. THE HEURISTIC ALGORITHM

4.1. Analogy between the Tasks of Decyclisation and Finding of Spanning Trees

The task of finding of spanning tree [11] is usually considered for non-oriented graphs. The analogy between this task and decyclisation exists because if non-oriented cycles are considered, an ayclic graph turns to be a tree. In this sense, finding maximal acyclic spanning subgraph for an orgraph can be considered as a generalisation of the task of finding spanning tree for a non-oriented graph. But all the spanning trees for given graph have the same number of arcs; so the question of optimisation arises if the arcs are weighted and their total weight should be minimised or maximised. For the orgraphs that is not the case; different spanning subgraphs with different number of arcs may be maximal in the sense that no arc can be added to them without creating a circuit (Fig. 1; dashed arrows denote the removed arcs). But, as it is shown below, the approach used for finding minimal (or maximal) spanning tree can be used with some changes for finding maximal acyclic spanning subgraph of an orgraph.

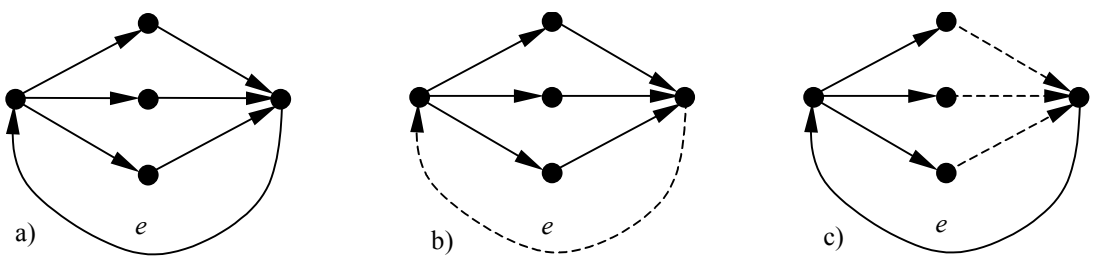


Figure 1. An orgraph (a) and its different acyclic subgraphs (b,c)

4.2. The Main Idea and Heuristic

An effective heuristic algorithm for decyclisation cannot require finding all the circuits in the given graph (a NP-complete task). So it is intuitively clear that a heuristic algorithm can build gradually an acyclic subgraph of the given graph. The same approach is used in [7] (Kruskal algorithm); but for a weighted non-oriented graph and the task of minimal spanning tree finding it allows obtaining the optimal solution, and for our task the solution will be approximate. The idea is, to start from any node of the given graph and to build an acyclic graph by adding the arcs such that the subgraph remains acyclic. But to direct this process and to avoid some non-optimal variants, the weights should be associated with the arcs of the given graph. We need a simple formula of calculating weights. And the evident way to do it is to use input and output degrees of initial and terminal nodes for each arc.

How to use these data? Let's consider some examples. It is easy to see that we should avoid adding to the spanning subgraph the arcs like the arc *e* at Fig. 1. That is an arc belonging to many circuits, so its presence in the subgraph would mean that several other arcs cannot simultaneously be there, and the obtained solution will be far from optimal. We don't know however the number of circuits each arc belongs to. (If it would be known the approach of removing the arcs belonging to the maximal number of circuits would be equivalent to one of the heuristic methods of matrix covering; but building the circuit matrix would require finding all the circuits). But what can be said about it by analysing input and output degrees?

If the terminal node has many outgoing arcs, and the initial node has many incoming arcs, that means that it belongs to many paths (and maybe circuits), and probably that it is the only arc common to all those paths (or one of the few ones). That means that it is undesirable to add this arc to the spanning subgraph. On the other hand, if the terminal node has many incoming arcs, and the initial node has many outgoing arcs, we may suppose that there are many circuits and if they have common arcs then somewhere else; so the arc under consideration can be added to the subgraph.

This intuitive reasoning can be expressed by the next formula:

$$w(e) = id(init(e)) - od(init(e)) + od(ter(e)) - id(ter(e)) \qquad (1)$$

where *e* is an arc and *w(e)* is its weight.

Then it is enough to create a spanning subgraph starting from the arc with minimal weight and adding also the arcs with minimal weights such that the subgraph remains acyclic. When no arc can be added to the subgraph, removing the rest of arcs from the given orgraph solves the task of decyclisation.

Here it is necessary to check at each step whether the subgraph is acyclic. It is an easy task for both oriented and non-oriented graphs; a graph has a cycle if and only if its depth-first-search tree contains a feedback arc. It can be checked by transitive closure of the relation specified by the given graph [12].

4.3. Formal Description of the Algorithm

Algorithm 2 *(heuristic)*

Let $G=(V, E)$ be the given graph. $G'=(V', E')$ such that $V'=V$, $E'=\varnothing$. $S=\varnothing$.

1. While $\exists\, v\in V$: $(\mathrm{id}(v)=0 \vee \mathrm{od}(v)=0)$

   1.1. $E \leftarrow E\backslash\{e\in E: v\in e\}$.*

---

* Such operation is useful also for exact algorithm; the removed arcs certainly don't belong to any circuit.

2. Calculate for every arc $e \in E$ its weight according to (1).
3. While ($E \setminus S \neq \varnothing$)

   3.1. $e \leftarrow \min(w(e): e \in (E \setminus S))$;

   3.2. $S \leftarrow S \cup \{e\}$;

   3.3. $E' \leftarrow E' \cup \{e\}$;

   3.4. If G' has a cycle then $E' \leftarrow E' \setminus \{e\}$.
4. ($E \setminus E'$) is a set of arcs, removing of which is enough for destroying all circuits in G. The end.

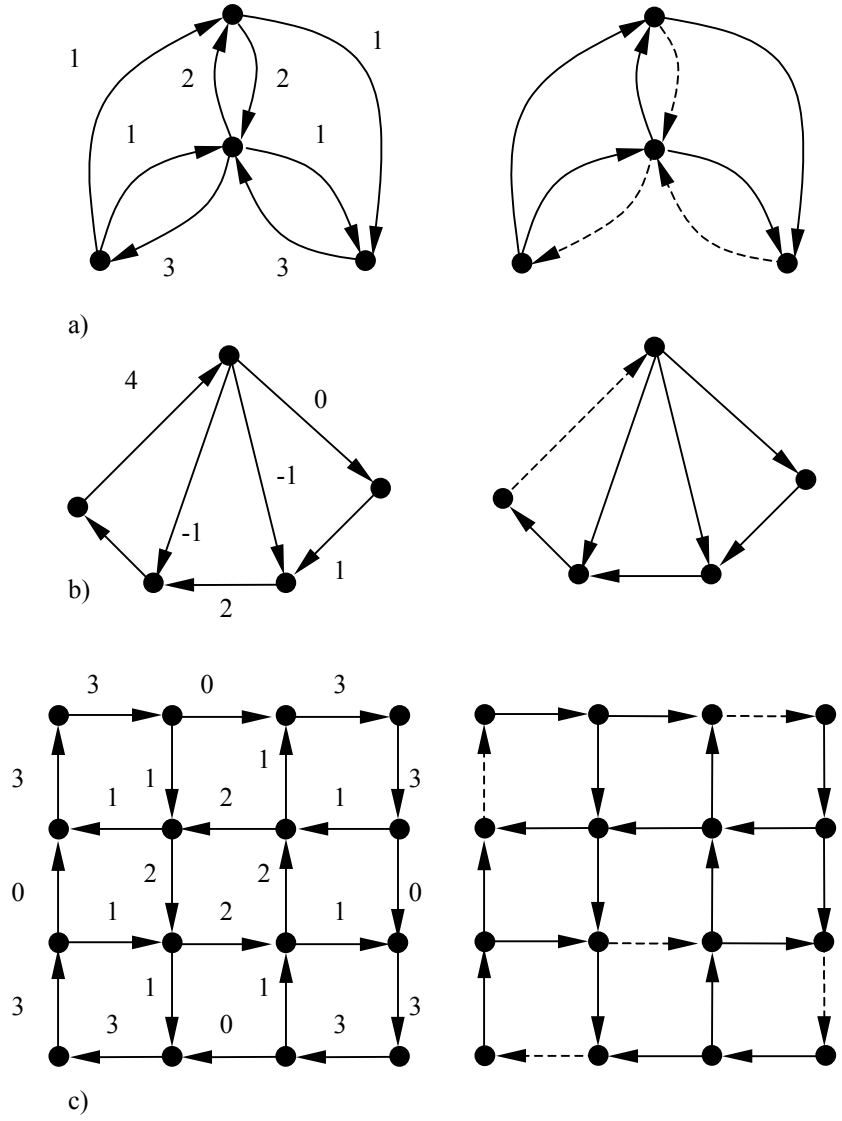


Figure 2. Examples of decyclisation

5. EXAMPLES

Let's consider some examples (Fig. 2). The graphs are taken from [4]. In all those cases the approximate algorithm finds the same solution as the exact one, i.e. the optimal solution. Weights of the arcs are shown near the arcs at Fig. 2. Note that we calculate weight for an arc not counting the arc itself for defining input and output degrees; so all the weights at Fig. 2

are 2 less than according to (1). That seems to be more convenient and doesn't affect to the results.

We can see that arcs with maximal weights can be used as the important indication for decyclisation. Consider Fig. 2a. Gradual building of acyclic spanning subgraph goes on as follows: at first the arcs with weight 1 are added to *G'*; it remains acyclic. Then one arc with weight 2 is added there; *G'* is still acyclic but no more arc can be added to it. So the decision is found, and it is optimal. Similar situation is with graphs shown at Fig. 2b and 2c; they are not explained in details because of lack of space.

6. CONCLUSIONS

The suggested approach allows quick obtaining of good approximate solutions of the task of decyclisation of oriented graphs. It can be applied to various problems such as Petri net analysis, minimisation and test generation for sequential circuits (by decyclisation these tasks can be reduced to the similar tasks for combinational circuits) and others, mentioned in *Background and the Problem Statement*. Statistical analysis of the method effectiveness is a topic of future work.

REFERENCES

[1] N.Deo, *Graph theory with applications to engineering and computer science*. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, USA, 1974.

[2] R.Diestel, *Graph Theory.* Electronic Edition, Springer-Verlag, New York, 2000.

[3] D.B.Johnson, *Finding All the Elementary Circuits of a Directed Graph.* SIAM J. Comput, 4, 1975, pp. 90-99.

[4] F.Harary, R.Z.Norman, D.Cartwright, *Structural Models: An Introduction to the Theory of Directed Graphs.* New York: John Wiley, 1965.

[5] E.J.Henlog, R.A.Williams, *Graph Theory in Modern Engineering: Computer Aided Design, Control, Optimization, Reliability Analysis.* Academic press, New York, 1973.

[6] A.Karatkevich, "Optimal Simulation of α-Nets". *Proceedings of the Polish-German Symposium on SRE'2000*, Zielona Góra, 2000.- P. 217-222.

[7] J.B.Kruskal, *On the shortest spanning subtree of a graph and the travelling salesman problem.* Proc. Amer. Math. Soc., 7, 1956, pp. 48-50.

[8] A.Lempel, "Minimum Feedback Arc and Vertex Sets of a Directed Graph". *IEEE Trans. Circuit Theory*, 1966 Dec., Vol. CT-13 No 4, pp. 399-403.

[9] E.M.Reingold, J.Nievergelt, N.Deo, *Combinatorial Algorithms: Theory and Practice.* Prentice-Hall, Inc., New Jersey, 1977

[10] R.Sedgewick, *Algorithms.* Addison Wesley, Inc., USA, 1984.

[11] M.M.Syslo, N.Deo, J.S.Kowalik, *Discrete Optimization Algorithms with Pascal Programs*. Prentice-Hall, Inc., 1983.

[12] S.Warshall, *A Theorem on Boolean Matrices.* J. ACM, 9, 1962, pp. 11-12.

[13] A.D.Zakrevskij, *Logic synthesis of cascade circuits*. Moscow, Nauka, 1981 (in Russian).

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

MODELING AND VERIFICATION OF SEQUENTIAL CONTROL PATHS USING PETRI NETS

Werner ERHARD, Andreas REINSCH and Torsten SCHOBER

Friedrich-Schiller University Jena, Dept. of CS, Ernst-Abbe-Platz 1-4,
D-07740 Jena, Germany, <*erhard, reinsch, schober*>*@informatik.uni-jena.de*

Abstract. *In this paper a design methodology based on interpreted Petri nets is applied to the functional verification of complex sequential control paths. Starting from a Petri net model with free choice structure and control engineering interpreted (CEI) net components, the verification of structural and behavioural properties as well as functional simulation is performed. A series of analysis strategies is derived to enable an automatic functional verification of a Petri net model. From the results of functional verification a set of modeling guidelines could be provided.*

Keywords. *Petri net, digital design, functional verification, sequential control path.*

1. PETRI NET BASED DIGITAL SYSTEM DESIGN

To model digital systems using Petri nets, one has to make two important choices concerning the net specification and the net interpretation. The variety to model a system in a structural way is determined by the net specification. By means of ordinary Place/Transition Petri nets it is simple to express both sequential and concurrent events. In particular the structural sub class of Free-Choice Petri nets (FCPN) that covers state machine and marked graphs provides clear structural modeling facilities. A comprehensive overview regarding net classes and its properties can be found in [7]. On the other hand the chosen net specification should enable the application of analysis methods for structural and behavioural net analysis. In this respect FCPN are particularly suitable too. Many results of Petri net theory can be efficiently applied to analyze properties of FCPN.

1.1. Modeling digital systems using CEI Petri nets

Due to net interpretation, the components of a Petri net are assigned to components of a digital system. Hence every net interpretation of a Petri net creates a Petri net model. There exist several net interpretations in different technical domains that either map Petri net components to elementary digital components or small sub nets to digital modules [8, 6, 1]. In [5] Control Engineering Interpreted (CEI) Petri nets have been introduced. This net interpretation performs a direct mapping between a system of communicating finite state machines and a hardware realization as shown in figure 1.

A CEI Petri net N_{SIPN} is defined as a tuple $N_{SIPN} = (P, T, F_{pre}, F_{post}, M_0, G, X, Y, Q_t, Q_p)$. Every transition t_i is assigned to an AND-gate and every place p_i is assigned to a memory cell. Consequently, a marked place $p_i = (e_i, a_i)$ symbolizes an active memory cell. Transition activating and place marking are realized by two mappings Q_t and Q_p. So, in addition to the usual firing rule, transitions are activated by guards G_i. These guards arise from logically conjuncted input signals x_i and express signal processing in a digital system. Output signals a_i can be used to enable state transition $p_i \rightarrow p_j$

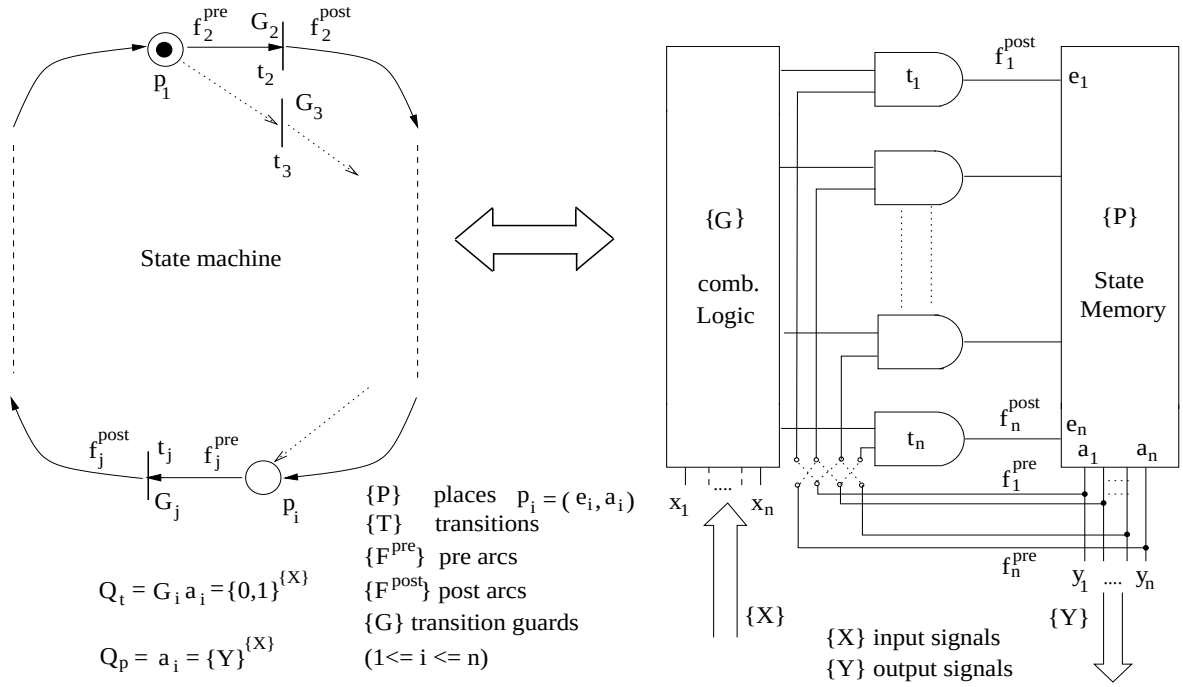


Figure 1: *CEI Petri net and its hardware realization*

or for output signal processing. In case of state transition, $a_i = 1$ is conjuncted with G_j at an arbitrary transition t_j. Then p_i gets low before p_j gets high and hence a new state is activated. Concurrent behaviour is represented by different state memory modules, each of sequential behaviour, communicating through shared transitions. The signal transition in a CEI Petri net shows that the token flow of the Petri net equivalently represents the signal flow of the digital system.

This is one of the major objectives in the hardware design methodology proposed in [2]. By means of CEI Petri nets with Free-Choice structure it is possible to model digital systems in a *transparent* way, such that system behaviour is equivalently reproduced by a simple structured Petri net model. Furthermore, behaviour and structure of the Petri net model is extensively analyzable if the Petri net model is retransformed into a FCPN. Therefore all transition guards G_i are eliminated.

To summarize at this point the approach of this work can be schematized as in figure 2. Starting with net specification FCPN and net interpretation CEI Petri net the modeling process can be performed. Afterwards the Petri net model is unannotated to a FCPN to enable net analyses. Results obtained by net analysis should be interpreted for the Petri net model to conclude a statement concerning its functional verification.

1.2. Petri net based design flow

Owing to high acceptance of commercial CAE-Systems and hardware description languages (HDL) it is not preferable to create a design flow that solely is based on Petri nets. Therefore in the design flow two entities enable a transformation between constructs of a HDL and Petri net components and vice versa. Consequently, there are several opportunities to specify a digital system. On top of figure 3 HDL input as well as graphical or textual Petri net input is proposed. For large designs high level Petri nets (HLPN) such as hierarchical Petri nets or colored Petri nets can be incorporated if it is possible to unfold the HLPN to a low level Petri net as FCPN. Once a Petri net model is created simulation and analysis methods can be applied to verify the design functionally. At this point Petri nets have two great advan-

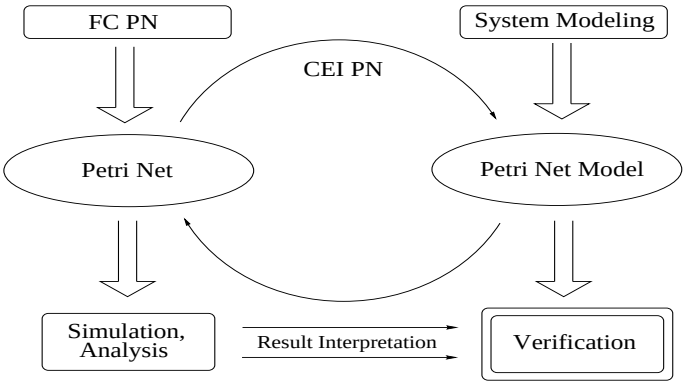


Figure 2: *Modeling and verification of digital systems using Petri nets*

tages. First because of its graphical concepts rough design errors can be detected easily by functional simulation. Beyond that an exhaustive analysis of structural and behavioural properties leads to a formal design verification rather than simulation of test pattern. A functionally verified design is transformed to dedicated HDL constructs to enable logic synthesis by means of conventional CAE tools. After logic synthesis simulation and analysis methods can be applied again to simulate and verify the timing behaviour of the design. Now deterministic and stochastic timed Petri nets are applied.
Thus it is shown that the design flow is based on Petri nets but nevertheless is embedded in a conventional design flow. To put the new design steps into practice the tool development environment "Petri Net Kernel (PNK)" [4] and the "Integrated Net Analyzer (INA)" [9] are used.

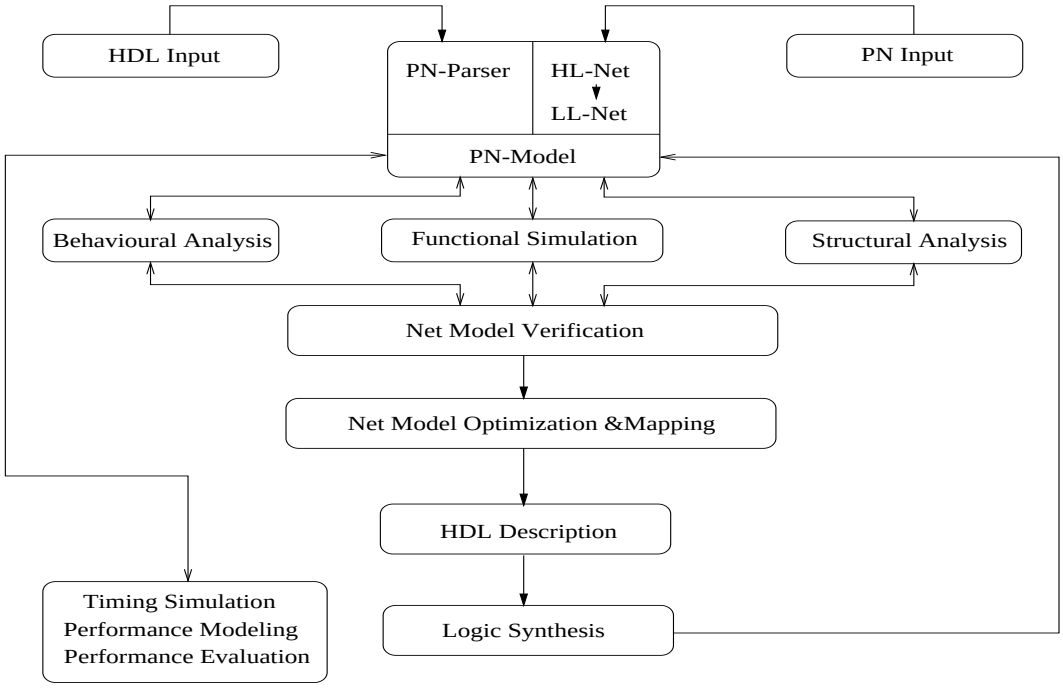


Figure 3: *Design flow for Petri net based digital system design*

2. FUNCTIONAL VERIFICATION OF PETRI NET MODELS

For functional verification of a modeled digital system a set of properties are studied by means of Petri net analysis. The analysis results are interpreted as properties of Petri net model. In a more practical way it is necessary to propose some requirements and goals that must be obtained for functional verification. Then a series of analysis strategies is derived to enable an automatic detection, localization and removal of modeling errors. As an outcome of applied analysis strategies some modeling guidelines could be derived. Adhering to these modeling guidelines enables approaching a functionally correct design already at early modeling cycles. In this work analysis strategies and modeling guidelines for functional verification of sequential control paths are suggested.

2.1. Requirements and Goals

Requirements for functional verification are expressed as Petri net properties of the analyzed Petri net. Net analysis is performed with the aid of tool INA as structural analysis and as reachability analysis. At first Petri net analysis has to ensure boundness and especially 1-boundness. In the Petri net model boundness determines that the modeled design has a finite state space. Control paths with an infinite number of states are not close to reality. Moreover 1-boundness is claimed, since the digital system should be modeled in a transparent way. If every place of th Petri net contains at most one token, logical values “high” and “low” are distinguishable for the memory cells that are represented by places. An unbounded Petri net is not analyzable any further. Liveness is the next necessary Petri net property for functional verification and it is interpreted as the capability to perform a state transition in any case. Every transition of the Petri net can be fired at any reachable marking. So if liveness is preserved the Petri net and thus the Petri net model are not deadlocked. If in a live Petri net the initial state is reachable then the Petri net is reversible. To reflect the structure of a sequential control path the Petri net should have state machine structure. In this case not any transition of the Petri net is shared. Every transition has exactly one predecessor place and one successor place. Therefore in a state machine generation and consumption of tokens is avoided. Complex sequential control paths such as control path of a sequential microprocessor consist of a system of strongly connected state machines. That includes decomposability into small partial nets with state machine structure. When a Petri net is 1-bounded, live and it has state machine structure then a very transparent Petri net model has been created. Every state in the Petri net model can be assigned to a state of the control path. Thus the reachability tree is rather small and represents exactly the number of control states. From the implementation point of view this state encoding is called one hot encoding.

Places with more than one successor transition generate conflict situations. If several post-transitions of a marked place are enabled and one transition fires, then all other transitions are disabled. Hence the Petri net is not persistent and also it is not predictable what transition will fire. Transition guards are able to solve conflicts because it represents an additional firing condition that is required to perform a state transition. So transitions in conflict can get unique using transition guards and no behaviour ambiguity remains. When a pre-place of a transition also appears as its post-place then there is a self loop in the Petri net. Self loops can give a structural expression to model external signal processing distinctly. It has to be clarified in the modeling process if any and which self loops are desired to emphasize external signal processing. Thus self loops can be detected and assigned to that situations and others are marked as modeling errors and should be removed.

2.2. Analysis Strategies

Table 1 summarizes all derived analyses strategies regarding detected modeling errors and affected Petri net properties. An automatic verification of Petri net models using $S1 \ldots S9$ can be performed according to figure 4. Using PNK and INA the analysis is efficiently implementable. It is supposed, that at “Start” a

reachability tree is derived or at least a part of it to determine boundness or unboundness of the analyzed Petri net. Strategy $S1$ detects shared transitions with more than one post-place or transitions without pre-place that cause unbounded places and hence an unbounded Petri net.

Strategy S1:

1. If Petri net N is unbounded, then
   (a) Determine all shared transitions t_i, $\mid t_i\bullet \mid > 1$ by means of $\bullet p_j$, $\forall p_j \in P \Rightarrow$ generate list $\{t_P^i\}$.
   (b) Determine transitions without a pre-place Ft_i0 by evaluating $\bullet t_i$, $\forall t_i \in T$.
2. Check $t_i = \bullet p_j$, $\forall p_j \in P$: if $t_i \in \{t_P^i\} \vee t_i = Ft_i0 \Rightarrow p_j$ is unbounded. Pre-transition t_i of place p_j produces tokens if $t_i \in \{t_P^i\}$ or $t_i = Ft_i0$ and hence unboundness of p_j is caused by t_i.

$S2$ and $S3$ are applied to detect and localize dead transitions that caused by lack of strong connectedness or by shared transitions with more than one pre-place. It is possible that the analyzed Petri net is 1-bounded and live but it shows no state machine structure. In this case all generated tokens are consumed within a marked graph, that is a Petri net structure in which no place is shared. Strategy $S4$ is applied to localize such shared transitions. By means of strategy $S5 \ldots S7$ dead transitions are detected and localized caused by one sided nodes as mentioned in the table. In the last two analysis strategies $S8$ and $S9$ transition guards are used to interpret conflicts and self loops within the Petri net model. A detailed description of all analysis strategies and its application within a case study is given in [3].

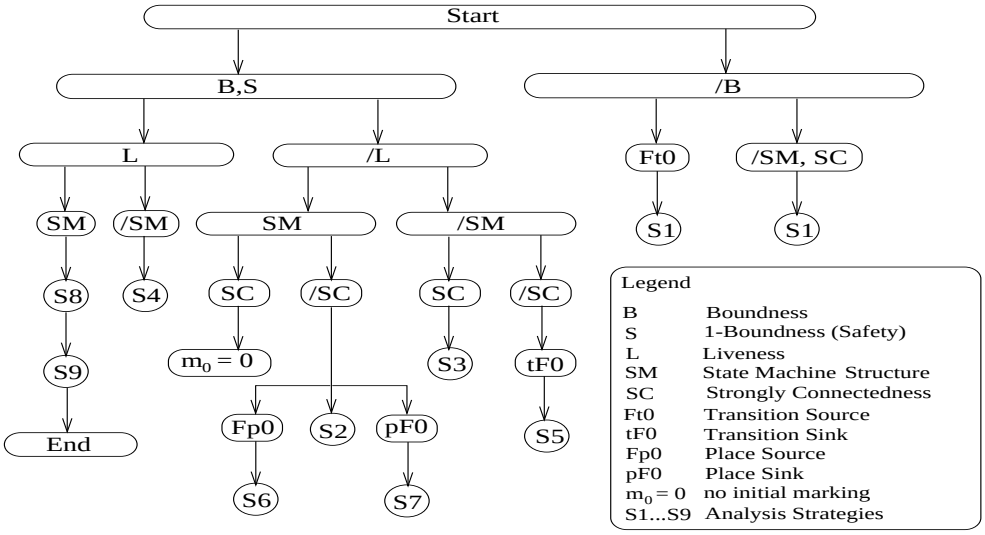


Figure 4: *Application scheme of derived analysis strategies*

According to the proposed analysis strategies it is possible to derive modeling guidelines that affect the modeling process and assist the designer to create a system model reflecting the desired functionality.

Modeling Guidelines:

1. Avoidance of shared transitions.
2. Avoidance of one-sided nodes.
3. Ensuring strong connectedness.
4. Removal of conflicts using transition guards:
   (a) All post-transitions of a shared place are provided with guards.
   (b) Every post-transition of a shared place is provided with an unique guard.

analysis strategy	modeling error	affected property
S1	generation of tokens or transition without pre-place	boundness
S2	not strongly connected	liveness
S3	consumption of tokens	state machine structure, liveness
S4	generation and consumption of tokens	state machine structure
S5	transition without post-place	state machine structure, liveness
S6	place without pre-transition	liveness
S7	place without post-transition	liveness
S8	self loops	pureness
S9	conflicts	static & dynamic conflict free

Table 1: *Summary of analysis strategies*

3. CONCLUSIONS

This work introduces a Petri net based hardware design methodology for modeling and functional verification of digital systems. System modeling is performed by means of Free-Choice Petri nets (FCPN) and control engineering interpreted (CEI) Petri nets. Using Petri net analysis techniques functional verification of Petri net models is obtained by analysis of Petri net properties and a suitable interpretation concerning the Petri net model. It is shown that a Petri net based design flow can be embedded in a conventional hardware design flow. For the functional verification of sequential control paths a series of analysis strategies is provided. Using these analysis strategies it is possible to detect and localize modeling errors automatically. Finally a set of modeling guidelines is determined to avoid modeling errors at early design cycles.

References

[1] J. Cortadella et. al. Hardware and Petri Nets: Application to Asynchronous Circuit Design. *Proc. of the* 21^{st} *ICATPN, LNCS 1825, Springer Verlag, 2000.*

[2] W. Erhard, A. Reinsch und T. Schober. Petri-Netz-basierter Entwurf asynchroner rekonfigurierbarer Systeme. *Tagungsband 15. GI/ITG-Fachtagung - Architektur von Rechensystemen ARCS'99, 4.-7.10.1999, Jena*.

[3] W. Erhard, A. Reinsch and T. Schober. Formale Verifikation sequentieller Kontrollpfade mit Petrinetzen. *Berichte zur Rechnerarchitektur, Band 7, Nr. 2, 2001*.

[4] E. Kindler and M. Weber. The Petri Net Kernel: An Infrastructure for Building Petri Net Tools. *Proc. of the* 20^{th} *ICATPN, LNCS 1643, Springer Verlag, 1999.*

[5] R. König und L. Quäck. *Petri-Netze in der Steuerungstechnik.* Verlag Technik Berlin, 1988.

[6] D. Misunas. Petri-Nets and Speed-Independent Design. *Communication of the ACM, 16(8):474-479, 1973.*

[7] T. Murata. Petri Nets: Properties, Analysis and Applications. *Proceedings of the IEEE 77(4):541-580, 1989.*

[8] S. Patil. Coordination of Asynchronous Events. *ScD Thesis, Dept. of Elec. Eng., MIT, Cambridge, Mass., May 1970.*

[9] P. H. Starke and S. Roch. Integrated Net Analyzer INA, Version 2.2. *LS Automaten und Systemtheorie, Institut für Informatik, Humboldt Universität zu Berlin, Dezember 2000.*

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

USING HIERARCHICAL STRUCTURING MECHANISMS WITH PETRI NETS FOR PLD BASED SYSTEM DESIGN

Luis GOMES, João-Paulo BARROS

Universidade Nova de Lisboa & UNINOVA,
P-2825-114 Monte de Caparica, PORTUGAL, *<lugo, jpb>@uninova.pt*

Abstract. *This paper addresses the use of hierarchical model structuring mechanisms for the design of embedded systems (in the sense of reactive real-time systems), using Reactive Petri nets. Relevant characteristics of Reactive Petri nets are briefly presented and their main roots are identified, namely Coloured Petri nets, Interpreted and Synchronised Petri nets.*
Two structuring techniques will be presented. The first one concerns with the integration of bus signals representation into non-autonomous Petri net models and the second one with a graphical structuring mechanism named by horizontal decomposition. This mechanism relies on the usage of macro-nodes, which have sub-nets associated with them. Three types of macro-nodes were used: macro-place, macro-transition and macro-block. The execution of the model is accomplished through the execution of the flat representation of the model.
At the end, a case study is presented around a controller of a 3-cell FIFO system. High-level and low-level Petri net models were used and compared for that purpose. The implementation of the referred controller was done using programmable logic devices, namely PALs and CPLDs.

Key Words. *Petri nets; Hierarchical structuring mechanisms; Programmable Logic Devices*

1. INTRODUCTION

It is widely acknowledged that the support for specific model structuring mechanisms is of the most importance for a complex system designer. In this sense, the use of techniques to build up compact models through the use of a "divide to conquer" strategy are commonly accepted as necessary. Common techniques use different levels of abstraction enabling the construction of the model in an incremental way.

From the point of view of the hardware design, the concepts of modular modelling have been widely used for several decades. Also from the point of view of software design, the effects of structured programming, modular programming and of object-oriented programming have been supporting the reusability and robustness of code. Complementarily, the use of data structures could help towards compactness and expressiveness of the produced specification.

These concepts and tendencies have been ported for the Petri net family of formalisms through a large number of proposals integrating hierarchies and modularity into Petri nets [4] [3] [1] [2]. Some of them support translation into simpler models (which means that they have the same modelling power), while other ones do not.

From our point of view, the concepts proposed in [4] are of major importance, namely substitution transitions (probably the most used hierarchical structuring mechanism; it is also used in Coloured Petri nets [1] and in Design/CPN, the most used Petri net tool), and substitution places (this mechanism plays a dual role related with substitution transitions; the execution of the model is accomplished through the flat model obtained after the fusion of the two-levels of nets. It is important to stress that it is not allowed to directly connect a substitution place to a substitution transition; although already identified in [4], it was considered there not to be a serious restriction. We disagree with that assumption.

This paper does not present, or use, some other important model-structuring mechanisms. Among them, one can mention the concept of depth, extensively used in Statecharts, and integrated in Reactive Petri net class [2] (which led to the concept of vertical decomposition structuring mechanism).

2. ABOUT REACTIVE PETRI NETS

This section briefly presents a non-autonomous high-level Petri net model, referred as Reactive Petri net model (RPN) [2]. In the following paragraphs, the model characteristics will be briefly presented in a non-formal way, around two components: the autonomous and the non-autonomous parts. In what concerns with the autonomous part of the model, Coloured Petri nets is the reference model [1]. The transition firing semantics is also similar.

The non-autonomous part in its turn can be divided in two main parts: input-output modelling and other execution issues. In the second group, one can found priorities associated with transitions, automatic conflict resolution and time modelling. From the point of view of this paper, only the input modelling is relevant. Interpreted and synchronised Petri nets [5] constitute the references to the input and output modelling. In this sense, input is modelled through event conditions associated to transitions. Output actions can be associated to markings or to transition firings.

The transitions firing rule was changed so as to consider the new input dependency. As such, for a transition to fire it must fulfil two conditions: it must be enabled, by the existence of a specific binding of tokens presented in the input places, and it must be ready, the external input evaluation must be true. Every transition enabled and ready will be fired. This means that the maximal step is always used. It is the followed approach in the synchronised Petri nets and interpreted Petri nets models.

3. PROPOSED HIERARCHICAL STRUCTURING MECHANISMS

Most of the known hierarchical structuring techniques emphasises the autonomous part of the model. Also, most of them use the flat model for the execution of the model; in this sense, the hierarchical structure is not reflected at the implementation level. This is also the case for the structuring mechanisms proposed in this paper.

3.1. From signals to buses support

Explicit modelling of interdependencies between the autonomous and non-autonomous parts of the model can be obtained in two situations:

- case I): binding variables can be used in event conditions;
- case II): input signals can be used in arc inscriptions, which implies that input signals can determine characteristics of coloured tokens.

From the designer point of view, this characteristic could be of major interest in terms of the compactness of the produced model. It has to be stressed that it is possible to find a flat model that is behaviourally equivalent to the model with mutual-dependent attributes, as far as the cardinality of the variables involved in those cross-references is finite. This will also be shown in the following paragraphs.

Case I situation is presented in Figure 1(a). There, a vector of input signals *a[i]* is associated with one transition. The specific binding of the tokens may select a specific input signal from the array. The flat model can be found replicating the transition as well as the arcs and inscriptions by a factor equal to the number of elements of the vector (two in the figure). The Figure 1(b) shows the flat model behaviourally equivalent to the model of Figure 1(a).

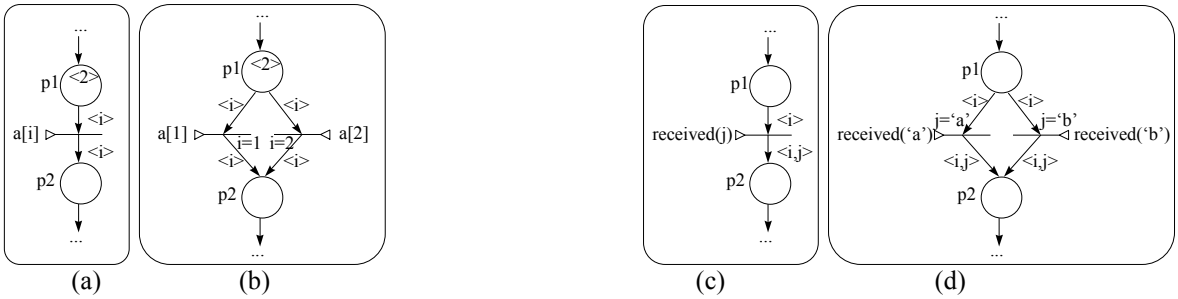


Figure 1 - Translation of specific inscriptions into flat models.

Case II situation is presented in Figure 1(c), where an external event will impose part of its own characteristics into generated tokens coloured attributes. The example models the synchronisation between a local activity (represented by p1 marking) and an external activity that emits an event with some attached information (in the example, two possible values are considered). The Figure 1(d) shows the behaviourally equivalent flat model. It has to be stressed that one needs to know in advance the type and cardinality of the external information in order to be able to produce the behaviourally equivalent flat model.

3.2. Horizontal decomposition

The horizontal decomposition mechanism is defined in the "common" way used by top-down and bottom-up approaches, supporting refinements and abstractions, and is based on the concept of module. The module is modelled in a separated net, stored in a page; every page may be used several times in the same design. The pages with references to the modules are referred as super-page (upper-level pages), while the pages containing the module model are referred as sub-pages (lower-level pages).

The execution of the model is accomplished through the flat model, which was the motivation for the naming of this decomposition type. This type of hierarchical decomposition only uses autonomous characteristics of the model. So, in this sub-section, non-autonomous characteristics, like signals, are irrelevant.

The nodes of the net model related with hierarchical structuring are named by macro-nodes. Three types of macro-nodes are foreseen: macro-place, macro-transition (also used by Hierarchical Coloured Petri nets [1]) and macro-block. Every macro-node will have an associated sub-page that can be referred as a macro-net. Distinctive graphical notations are used for the representation of macro-nodes: macro-places are represented by a double circle

(or ellipse), macro-transitions by a rectangle and macro-blocks are represented half-macro-place and half-macro-transition. Due to space limitation, detailed examples are not presented, nor associated detailed semantics.

A macro-place corresponds to a (macro-)node where all the arcs are connected to transitions, which means that the boundary of the macro-net (i.e. of the sub-page) is composed by a set of places. A macro-transition corresponds to a (macro-)node where all the arcs are connected to places. This means that the boundary of the sub-net should be composed by transitions; however, due to the necessary coloured binding evaluation for those transitions, the actual connections to input places and arc inscriptions are needed. So, in the sub-page associated with the macro-transition, references to places of the super-page have to be included. However, from the point of view of the execution of the model, the places connected to a macro-transition exist only at the super-page level; the places presented at the sub-page are void and will be merged with the associated places at the super-page.

As far as one macro-transition can not be executed like an ordinary transition and also that a macro-place cannot be considered as an ordinary place (which means that it cannot hold a marking), they act just as a graphical modelling convenience enabling the designer to structure the graphical model in an expressive way. In this sense, it is interesting to consider the use of graphical modules that can have arcs to/from places and transitions at the same page. They correspond to modules in a more general way and we name them macro-blocks. As far as this representation is not an executable specification (at this level), the bipartite intrinsic characteristic of Petri nets is not violated.

The following steps compose the merging process of the sub-page into the super-page:

- References of places and transitions used by the sub-page will be eventually changed in order to produce unique labels;
- One copy of the sub-page is inserted at the super-page; the macro-node reference is removed;
- The arcs connected with a macro-place semantics are connected to the referred boundary place; for arcs connected with a macro-transition semantics, void boundary places of the sub-page are merged with the associated places at the super-page (arcs and associated arc inscriptions in the sub-page are kept).

The interface of the macro-net (i.e. of the sub-page) is composed by a set of boundary places. As referred, this set of places will constitute the glued points between the sub-page and the super-page. However, when the sub-page boundary place is associated with a macro-place (or with a connection to a place in a macro-block), the boundary place is a common place (in the sense, that it can be marked, for instance). If the sub-page boundary place is associated with a macro-transition (or with a connection to a transition in a macro-block), the boundary place is a void place (in the sense, that it can not be marked, for instance). This characteristic allows the direct interconnection of macro-nodes, which can be of most interest in real-world applications, as we will try to emphasise in the following section.

4. CASE STUDY

An example illustrating the application of the proposed macro-nodes is presented. It is a 3-cell FIFO (first-in-first-out) system controller for assembly activities, referred in [2]. The assembly cell system has a conveyor to transport objects to the different cells. Each cell has presence sensors to detect objects on its in- and outputs, connected to the variables *in[1..3]* and *out[1..3]*. Each cell also has conveyor movement control, through the variables *move[1..3]*.

Figure 2 shows a simplified characterisation of the system controller, and two Reactive Petri net models as well. At the right hand side model, coloured characteristics support an easy modification of the model to accommodate a different number of cells. In the simplified model at the centre, a module-centric model based on the cell concept is presented; each cell is modelled by a macro-block. Associated detailed low-level model can be found in Figure 3. The usage of macro-blocks enables a more intuitive modelling and direct interpretation of the corresponding systems presented at the plant.

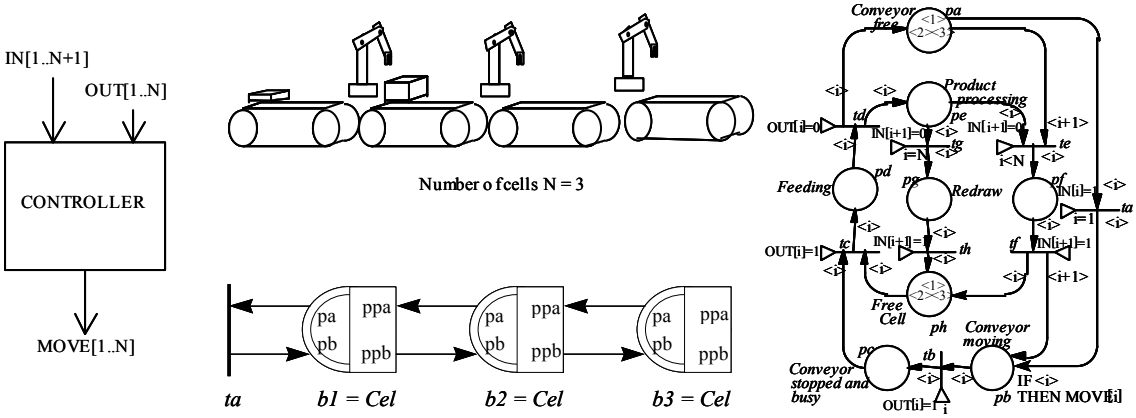


Figure 2 - N-cell FIFO system model.

The final goal is to implement the referred controller through a programmable logic device (PALs or CPLDs). As far as it is not possible to use a systematic direct translation procedure from high-level nets to hardware implementation, several possibilities were analysed to accomplish the goal.

Using the horizontal decomposition mechanism, it is possible to model the system in different ways, using high-level or low-level nets. Due to space limitations, it is not possible to present different examples. One is briefly presented considering the use of low-level Petri nets, where an equivalent model was obtained; it is presented at Figure 3. Three areas are identified associated with the different cells (which means to a different token in terms of the initial model).

Although the example complexity is relatively low, and so no general conclusions could be taken, different possible solutions were implemented and tested using PLDs (simple PLDs like PALs and CPLDs 9536 and 95108 from Xilinx). Possible implementation strategies rely on the following attitudes:

- Start with the coloured model and build up the associated space state, which is behaviourally equivalent to the original model; then implement the associated state machine in hardware using well-known techniques;
- Unfold the coloured model into low-level nets (as the one presented in Figure 3), using automatic tools; with the low-level model, different approaches are possible:
  - direct implementation based on direct translation of each node into hardware structure (translating places by memory elements and transitions by combinatorial logic);
  - indirect implementation based on the associated state space;
  - partitioning of the model, based on the net characteristics, and parallel implementation of the several parts.

Among the different solutions tested and associated with the last referred method, for instance the development of a hardware module associated with the macro-blocks presented in Figure 2 (centre), one other was based on place invariant computation. In the referred example, based

on the model of Figure 3, one can get four invariants with very interesting characteristics: i) the four place invariants cover all the places of the net, and ii) each invariant can be seen as a state machine (as far as only one and only one place of the invariant is marked at a time). In this sense, the implementation of the system can be based on the partitioning of the model into four concurrent state machines, which were trivially implemented in a PLD.

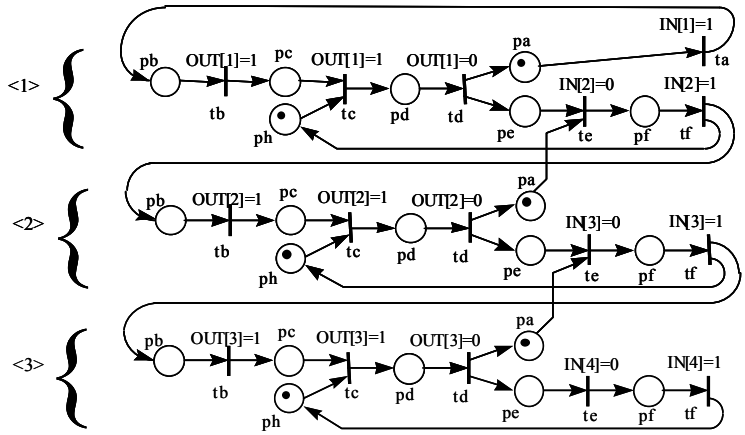


Figure 3 - Low-level representation of the 3-cell FIFO system.

5. CONCLUSIONS

In this paper, Petri net-based digital systems design was addressed. Two techniques were presented towards a better model structuring. Their usage was successfully validated through an example of a controller for a low-to-medium complexity system, which was implemented based on programmable logic devices (PALs and CPLDs). The first technique introduces the representation of arrays of signals into the Petri net model, while the second one uses the concept of module, which can be used associated with special kind of nodes, named by macro-nodes. Three kinds of macro-nodes were proved to be of interest.

ACKNOWLEDGMENTS

The presentation of the work at the DESDes'01 Workshop is supported by the IST project 10258 DOTS – "Distributed Object Telecontrol Systems and Networks".

REFERENCES

[1] K. Jensen; 1992; "Coloured Petri Nets: Basic Concepts, Analysis Methods and Practical Use, Volume 1"; Springer-Verlag; ISBN 3-540-55597-8

[2] L. Gomes; 1997; "Redes de Petri Reactivas e Hierárquicas - integração de formalismos no projecto de sistemas reactivos de tempo-real"; in Portuguese; PhD Thesis; Universidade Nova de Lisboa, Faculdade de Ciências e Tecnologia; 318 pages

[3] Luca Bernardinello, Fiorella De Cindio; 1992; "A Survey of Basic Net Models and Modular Net Classes"; in "Advances in Petri Nets 1992"; Lecture Notes in Computer Science; G. Rozenberg (Ed.); Springer-Verlag

[4] P. Huber, K. Jensen, R.M. Shapiro; 1990; "Hierarchies in Coloured Petri Nets"; in "Advances in Petri Nets 1990", Lecture Nores in Computer Science LNCS 483, G. Rozenberg (Ed.); pp. 313-341

[5] R. David, H. Alla; 1992; "Petri Nets & Grafcet; Tools for modelling discrete event systems"; Prentice Hall International (UK) Ltd; ISBN 0-13-327537-X.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

A RIGOROUS DESIGN METHODOLOGY FOR REPROGRAMMABLE LOGIC CONTROLLERS

Marian ADAMSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *M.Adamski@iie.pz.zgora.pl*

Abstract *In the paper it is shown how to implement parallel (concurrent) controllers in Field Programmable Logic (FPL). The main goal of the proposed design methodology is to preserve the direct, self-evident correspondence among a control interpreted Petri net and its possible hardware implementations. The symbolic specification of Petri net is considered in terms of the local state changes, which are distinguished by means of separated transitions, with their input and output places. Decision Rules are given as a set of Gentzen logic sequents (formal behavioural assertions). The initial specification, which is given in the form of symbolic logic expressions, may be verified, and then transformed into an intermediate format, which is accepted by industrial, VHDL- based, CAD tools. The logic (Boolean) expressions, which are suitable for direct mapping into FPGA or CPLD, can be also derived.*

Keywords*. Petri Nets, Logic Controllers, Hardware Description Languages, Field Programmable Logic, Sequential Function Chart*

1. INTRODUCTION

The well-structured formal specification, which is represented in the human-readable language, has a direct impact on the validation, formal verification and implementation of digital microsystems in Field Programmable Logic (FPL). The declarative, logic-based specification of Petri net can increase the efficiency of the Concurrent (Parallel) Controller design [1,4,8,14].

The model of *concurrent state machine* [4] is considered here inside the concept of *sequent automaton* and *parallel automaton* developed by Zakrevskij [16]. In the presented view, a control automaton, with distinguished, discrete and composite states, is also treated as a *dynamic inference system*, based on Gentzen logic [1]. The symbolic sequents-axioms may include some elements, taken from temporal logic [1,14]. The state space graph of controller is considered as a description of a discrete transition system [10]. Statements about the functionality of the designed system (behavioural axioms) are represented by means of *sequents-assertions*, forming the *rule-based decision system*. Complex sequents are formally

transformed into the set of equivalent *sequent-clauses* (*simple sequents* in [16]), which are very similar to the *production rules* [14].

At the beginning of design process, we use the control-oriented Petri net-based initial specifications of dedicated, reactive discrete-event systems (or Sequential Function Chart [3]). After analysis of some behavioural and structural properties of Petri net, a discrete-event model is related with a knowledge-based, textual, descriptive form of representation. The syntactic and semantic compatibility between Petri net descriptions and symbolic conditional assertions are kept as close, as possible. The formally transformed decision rules are directly mapped into VHDL statements. The Logic Controller is implemented in Field Programmable Logic, as a FPGA based reprogrammable unit. The automatic synthesis with VHDL oriented tools [2,3,10,15], as well as formal verification techniques and their efficiency [6,7,11,13,14,16], are out of the scope of the paper. The paper presents only the outline of formal methodology for Application Specific Logic Controllers (ASLC) design. The previous work on that subject has been recently summarised in papers [2,3,4].

2. DISCRETE EVENT CONTROL SYSTEM

As an example we have selected the simplified version of Logic Controller behaviour taken from papers [2, 3]. The chemical reactor V3 is fed with two kinds of liquids from measuring vessels V1 and V2. After the reaction between the liquids is completed, the reactor V3 is discharged. When the reactor V3 is empty, the process starts again from its initial state. To ensure complete reaction stirrer M agitates the process liquid in the reactor. Figure 1 shows a block diagram of the controlled system.

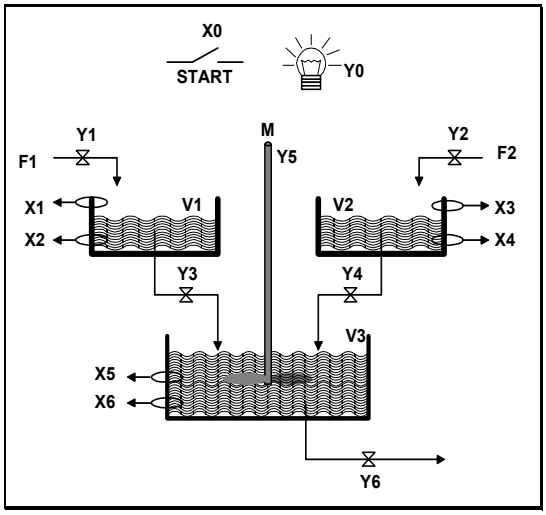


Fig. 1. The controlled part of discrete systems

3. BASIC THEORETICAL MODEL

Designing the discrete controller as a digital system involves a Petri net-based behavioural specification, taking into account the properties of the controlled objects (Figure 1). *Control Interpreted Petri Net* [6] has been shown to be a powerful tool to specify and model the behaviour of parallel (concurrent) controllers. The specification is given in terms of the local state changes. An event driven system can be abstracted as a concurrent state machine (CSM), in which several local states (represented as places in Petri net) may change, when event occurs (transition in Petri net fires). The marking (distribution of tokens among places) of a Petri net can be regarded as the current global state of the modelled system. From the present

global internal state (collection of simultaneously holding local states), the concurrent state machine goes to the next distributed internal global state, to generate the necessary combinational and registered output signals {y}. In such a way, an explicit local change of the marking, during the occurrence of transition, corresponds to an implicit global state change. The colours, which are attached to places [15], distinguish the intended sequential processes

The synchronous Petri nets [10,12] are introduced to model binary systems, which are synchronised by a global clock. Input signals of controller are associated with transitions as a Boolean guard. The most common static Moore type output signals are linked with places. Some static Mealy type output signals are related both with places and input signals. A part of Mealy type outputs frequently coincidence with the firing of transition. That makes it possible to label net transitions with some particular dynamic signal names.

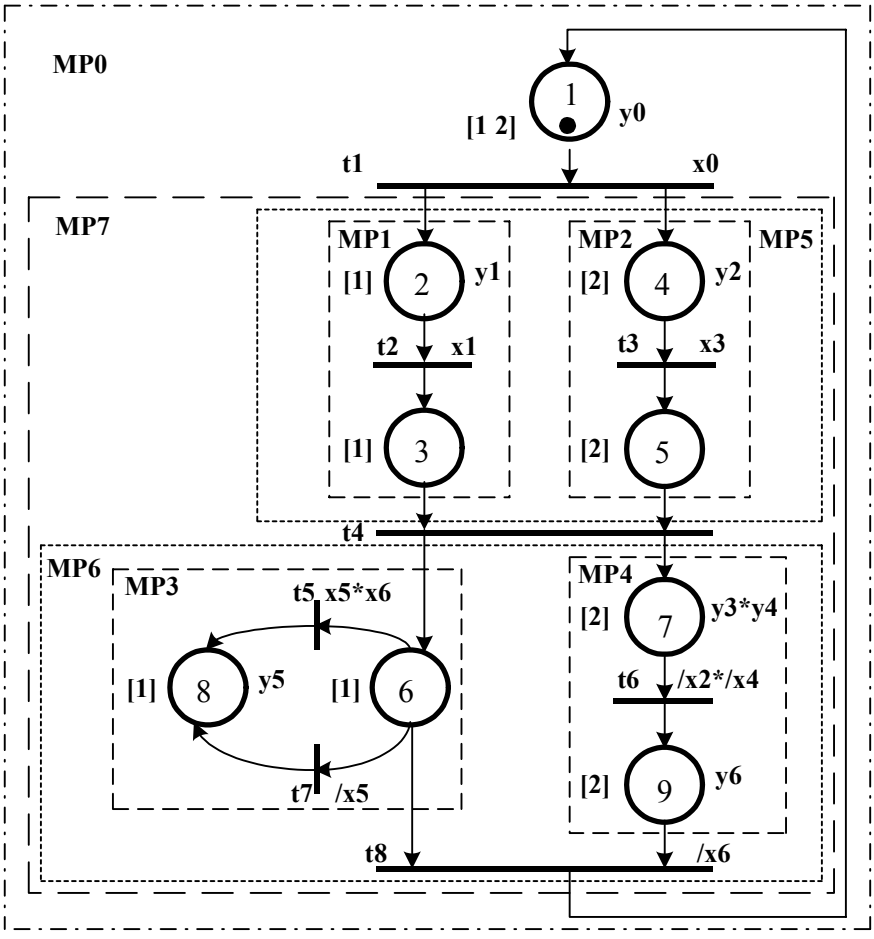


Fig. 2. Modular behavioural specification by means of Petri nets

To obtain the economical implementation and easy maintenance, the Petri net may be directly mapped into the Boolean equations without explicit enumeration of all possible global states and all possible global state changes. Since the specification is given only in terms of the local state changes (local transitions), the structured local state assignment (place encoding) is used [1,16].

Petri nets provide a unified method for the design of discrete-event systems from a hierarchical system description (Figure 2) to possibly hierarchical physical realizations. The hierarchically structured Petri net consists of subnets, which, except possibly the Base Net are well-formed blocks. The concurrency relation between subnets is depicted by means of colours, which are attached to the explicitly to the places, and implicitly to the transitions and

arcs as well as to the tokens [15]. The set of subnets is partially ordered. The coloured hierarchy relation tree (Figure 3) graphically represents the hierarchy and concurrency relations among subnets. The Base Net *MP0* is on the root of the tree. It contains the double-macroplaces *MP1-MP7*, which stand for the hierarchically structured subnets at the lower level of hierarchy. Each double macroplace (called shortly *double*) corresponds to a compound operation, which is itself a discrete sub-process described by the doubled block. The colours *[1]* and *[2]* are used for distinguishing some particular intended sequential processes, and continuously controlling the place invariants (P-subnets) and hierarchy tree during the composition or reduction of the net. The Petri net (Figure 3) is hierarchically encoded by means of state variables *Qi*, *i= 1,2,3,4*. The symbols *Qi* or */Qi,* attached to the particular path, which is directed from the root to the leave, form the unique encoding term for the considered macroplace or place.

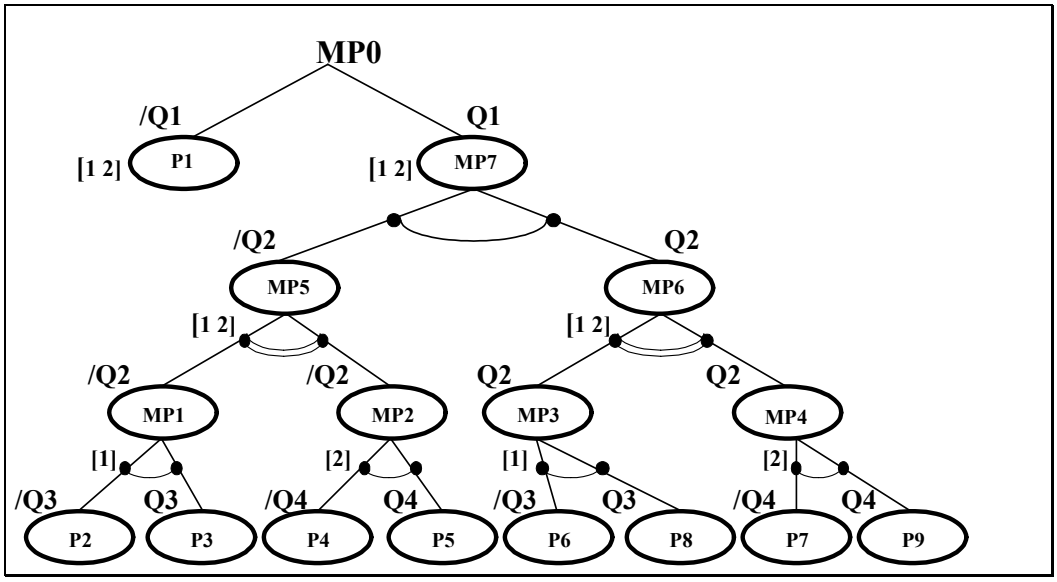


Fig. 3. Hierarchy tree

4. GENTZEN SEQUENT LOGIC

Petri nets can be viewed as a formal model for logic rule-based specification (interpretation structure). Transition rules are usually treated as production rules ('if-then' non-procedural statements). The principal design language used to specify the Logic Controller behaviour in extended nested *If-Then-Else* form in our design environment is Gentzen Sequent Logic [9].

While formulae may be regarded as a formal representation of compound propositions, sequents in our approach represent asserted statements. Sequents may formally describe all general forms of conditional assertions (rules). The Gentzen formal system naturally simulates and records human-like reasoning. The synthesis, based on Gentzen calculus, is treated as a formal symbolic transformation of the initial set of sequents (specification) into another equivalent set of sequents (implementation) [1,2]. The rules of inference are directly based on Gentzen Logic or they are previously proven, so the implementations are correct by construction.

Complex sentences are built up from propositions by application of the propositional connectives: *not (/)*, *and (*)*, *or (+)*, *if-then (->)*, *if-and-only-if (<->)* and *if-then-else (-> |*).

If F, *G*, and *H* are sentences then expression *F->G|H* (*if F then G else H*) is called *conditional*. The implication *F->G (if F then G)* is a special case of the conditional. We call *F*

the antecedent and *G* the consequent of implication. In our particular application (rule-based description) *F* is usually a conjunction of simple propositions, *G* and *H* are possibly nested conditionals, or *G* and *H* are conjunctions of simple propositions. The most frequently used form of simple decision rule is an *asserted implication* |- *F->G*, in which both *F* and *G* are elementary conjunctions. According to Gentzen the simple decision rule can be represented also as an equivalent sequent *F*|- *G*.

The complex conditional assertion (Complex Decision Rule) is described as a sequent:

|- *F->G | H;*

It encloses possibly nested components *G* and *H*. From the Gentzen logic point of view it is a shorter form of a complex sequent |-*(F->G)*(/F->H).*

Each Gentzen *inference figure* (*inference rule)* applied to the sequents (transition rules) preserves the meaning of the specification, but changes the number of the sequents and a form, in which sequents are expressed. If a specification is correct, transformation by means of natural inference guarantees correctness by construction.

As a simple example of formal transformation we consider the declarative assertion:

|- *(F- >G)* (/F->H);*

It can be transformed into an equivalent set of two simpler sequents (assertions) by splitting the right side

|- (F- >G) (/F->H);*

———————————— (Rule |- *)

|- (F->G); |- (/F->H);

The final assertions (simple Decision Rules) are as follows:

F |-G;

/F |-H;

5. PETRI NET SPECIFICATION IN SEQUENT LANGUAGE

The Logic Controller is considered as an abstract reasoning system (rule based system) implemented in reconfigurable hardware. 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' @ [1,11,14]. 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)*.

As a basic form of Petri net specification in rule based format the *transition-oriented declarative specification* is presented. It describes all possible active events in concurrent state machine, when local states are changed, and the guard (Boolean label) associated to transition is true.

```
T1: P1 * X0            |-    @P2 *@P4;
T2: P2 * X1            |-    @P3;
T3: P4 * X3            |-    @P5;
T4: P3 * P5            |-    @P6 * @P7;
T5: P6 * X5*X6         |-    @P8;
T6: P7 * /X2*/X4       |-    @P9;
T7: P8 * /X5           |-    @P6;
T8: P6 *P9 * /X6       |-    @P1;
```

The static (level) Moore type outputs depend directly on places:

```
P1 |- Y0;   P2 |- Y1;   P4 |- Y2;   P7 |- Y3 * Y4;
P8 |- Y5;   P9 |- Y6;
```

The total discrete state space (9 global states), which could be also possibly given in hierarchical manner, should be always consistent with all local state changes:

|-**P1***/P2*/P3*/P4*/P5*/P6*/P7*/P8*/P9, /P1***P2***/P3***P4***/P5*/P6*/P7*/P8*/P9,…,
/P1*/P2*/P3*/P4*/P5*/P6***P7****P8***/P9, /P1*/P2*/P3*/P4*/P5*/P6*/P7***P8****P9**

The presented form of description is very closed to well-known production rules, whose are a principal forms of Petri net description in LOGICIAN [1], CONPAR [8,10], PARIS [12], and PeNCAD [3,15,].

The dynamic (pulse or registered) output signal can be included directly to the decision rule, when it changes its value together with the occurrence of transition. On the other hand, all changes of the place making could be also explicitly included into the sequent, for example:

```
T1: P1 * X0  |-   @P2 @P4*/@P1*@/Y0*@Y1*@Y2;
```

In some cases, like implementations with D flip-flops in FPGA, the declarative, *place oriented specification* is taken into account:

```
P1:   P1    |- X0 -> @P2 * P4        | @P1;
P2:   P2    |- X1 -> @P3             | @P2;
P3:   P3    |- P5 -> @P6             | @P3;
P4:   P4    |- X3 -> @P5             | @P4;
P5:   P5    |- P3 -> @P7             | @P5;
P6:   P6    |- P9 * /X6 -> @P1       | (X5*X6 -> @P8 | @P6);
P7:   P7    |- /X2*/X4 -> @P9        | @P7;
P8:   P8    |- /X5 -> @P6            | @P8;
P9:   P9    |- P6*/X6->@P1           | @P9;
```

In this kind of specification, if the next value of the temporal variable, for example *@P1*, cannot be proved in the current marking (global state) as *true,* it is considered that it takes the value *false.*

When control outputs are immediate (combinational), they may be associated with the appropriate places and later transformed into registered ones. It should be noted that in many cases registered outputs could be used not only as names of places, but also directly as *codes* of the associated places

The sequents with transition symbols *{T1, T2, … , T8}*, after mapping the Petri net into VHDL statements according to M. Bolton's style, give economical implementations in FPGA [8]:

```
P1 * X0 |- T1
P2 * X1 |- T2
........
T1+P2*/T2 |- @P2
```

6. PETRI NET MODELLING AND SYNTHESIS WITH VHDL

The direct mapping of a Petri net into Field Programmable Logic (FPL) is based on a self-evident correspondence between a place and a clearly defined bit-subset of a state register. The places of the Petri net are assigned to the particular flip-flops in the Register Block. VHDL supports conditional-statement constructs, which can be used to describe Petri net. The proper local state assignment (encoding) makes it possible to map a given Interpreted Petri net directly into FPGA or CPLD without its transformation into an equivalent global State Machine. The simplest technique for Petri net place encoding is to use one-to-one mapping of places onto flip-flops in the style of a one-hot state assignment. In that case, a name of the place becomes also a name of the related flip-flop. The flip-flop is set into 1 if and only if the

particular place holds the token. Some of the recent developments involving modelling and analysis such constructs in VHDL were reported, for example in [2,3,8,10,15].

In general, places after encoding are distinguished by conjunctions, which are formed from state variables from the set *{Q1, Q2, ... , Qk}*. The local states, which are active simultaneously, have non-orthogonal codes. They are represented by places holding the tokens concurrently and belonging to the same vertex from the implicitly or explicitly given reachability graph of Petri net. The local states, which belong to the different, but sometimes overlapping sequential processes (P-invariants, SM-components) have orthogonal codes. One particular method of place encoding is based on hierarchical decomposition of the net. The example of an efficient heuristic hierarchical local state assignment `[Q1, Q2, Q3, Q4]` is as follows:

```
P1  = 0 - - -                    P1 = /Q1
MP7 = 1 * * *                    MP7= Q1
MP5 = 1 0 * *                    MP5= Q1*/Q2
MP6 = 1 1 * *                    MP6= Q1*Q2
MP1 = 1 0 * *                    MP1= Q1*/Q2
MP2 = 1 0 * *                    MP2= Q1*/Q2
MP3 = 1 1 * *                    MP3= Q1*Q2
MP4 = 1 1 * *                    MP4= Q1*Q2

P2  = 1 0 0 *                    P2= Q1*/Q2*/Q3
P3  = 1 0 1 *                    P3= Q1*/Q2*Q3
P4  = 1 0 * 0                    P4= Q1*/Q2*/Q4
P5  = 1 0 * 1                    P5= Q1*/Q2*Q4
P6  = 1 1 0 *                    P6= Q1*Q2*/Q3
P7  = 1 1 * 0                    P7= Q1*Q2*/Q4
P8  = 1 1 1 *                    P8= Q1*Q2*Q3
P9  = 1 1 * 1                    P9= Q1*Q2*Q4
```

The global state encoding is correct if all vertices of the reachability graph have different codes. The total code of the reachability graph vertex would be obtained by merging the codes of the simultaneously marked places. The code of the particular place or macroplace is represented by means of the vector composed from {0, 1, - , *}, or it is given as a related Boolean term. The symbols 0, 1, - ('don't care') have the usual meanings, but the symbol * in vector denotes 'explicitly don't know' (0 or 1, but not 'don't care'). For practical applications (CAD tools) it is usually sufficient to manipulate with Boolean expressions (product terms) [1,2,3,15].

7. CONCLUSIONS

Formal logic language, which is complementary with Petri nets, is suitable in specifying system level designs of logic controllers, implemented in FPL. Simulating of Petri net model and its hardware implementation can be simplified by translating of rule-based description to VHDL. The simulation results, at circuit level and algorithmic level, can be compared immediately. To simulate the pair consisting of the controller and discrete object under control, the test bench must include, in addition to the Reprogrammable Controller description, a second VHDL program, which model the controlled subsystem behaviour. The next design step concentrates on the automatic synthesis of Reprogrammable logic Controllers from their VHDL description. The paper presents the hierarchical Petri net approach for synthesis, in which the modular net is mapped into the Field Programmable logic as structured, but a flat netlist. The hierarchy levels are conserved and related with some particular local state variable subsets, and clearly distinguished by the encoding vectors (encoding terms). A concise, understandable specification can be easily locally modified. The

experimental Petri net to VHDL translator has been implemented on the top of standard VHDL design tools, like ALDEC Active-HDL.

REFERENCES

[1] M. Adamski, "Parallel Controller Implementation using Standard PLD Software". In: *FPGAs*, W.R. Moore, W. Luk (Ed.), Abingdon EE&CS Books, Abingdon, England, 1991, Chapter 5.5, pp.296-304.

[2] M. Adamski M., J. L. Monteiro, "From Interpreted Petri Net Specification to Reprogrammable Logic Controller Design", *Intern. Symposium on Industrial Electronics ISIE'2000*, 4-8 Dec. 2000, Puebla (Mexico), Vol. 1, pp.13-19.

[3] M. Adamski, "SFC, Petri Nets and Application Specific Logic Controllers". *Proc. of the IEEE Int. Conf. on Systems, Man, and Cybern.*, San Diego, USA, 1998, pp.728-733.

[4] M. Adamski, A.D.Zakrevskij, "Rule-Based Specification of Reactive Logical Control Devices", *Proceedings of the Polish-German Symposium on Science Research Education*, SRE'2000, Zielona Gora, 28-29.Sept.2000, Vol.1, pp. 199-204.

[5] K. Bilinski, M. Adamski, J.M. Saul, E. L. Dagless, "Petri net based algorithms for parallel controller synthesis", *IEE Proceedings-E, Computers and Digital Techniques,* Vol. 141, No. 6, Nov. 1994, pp. 405-412.

[6] R. David, H. Alla, *Petri Nets & Grafcet. Tools for modelling discrete event systems*, Prentice Hall, New York, 1992.

[7] W. Fengler, A. Wendt, M. Adamski, J.L. Monteiro, "Petri Net based Program Design and Implementation for Controller Systems", *1996 IFAC Triennial World Congress*, San Francisco, CA, USA, Vol. J, Identification II, Discrete Event Systems, pp.425-429.

[8] J. M. Fernandes, M. Adamski, A.J. Proença, "VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controllers", *IEE Proc.-E, Computer and Digital Techniques*, Vol.144, No.2, 1997, pp.127-137.

[9] M. E. Szabo (Ed.), *The collected papers of Gerhard Gentzen*, North Holland Publishing Company, Amsterdam, 1969.

[10] A. Yakovlev, L. Gomes, L. Lavagno (Ed.), *Hardware Design and Petri Nets*, Kluwer Academic Publishers, Boston, 2000.

[11] M. Heiner, "Petri Net Based System Analysis without State Explosion", *Proc. High Performance Computing'98*, Boston, April 1998, SCS Int., San Diego, 1988, pp.394-403

[12] T. Kozlowski, E. L. Dagless, J.M. Saul, M. Adamski, J. Szajna "Parallel controller synthesis using Petri nets", *IEE Proc.-E, Computers and Digital Techniques*, Vol. 142, No. 4, 1995, pp. 263-271.

[13] T. Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol.77, No. 4, April 1989, pp. 541-580.

[14] J.S. Sagoo, D.J. Holding, "A comparison of temporal Petri net based techniques in the specification and design of hard real-time systems", *Microprocessing and Microprogramming*, Vol. 32, No. 1-5, 1991, pp. 111 - 118.

[15] M. Wegrzyn, P. Wolanski, M. Adamski, J.L. Monteiro, "Field Programmable Device as a Logic Controller", *Proc. of the 2nd Conf. on Automatic Control - Control'96*, Oporto, Portugal, 1996, Vol. 2, pp.715-720.

[16] A.D. Zakrevskij, "Parallel algorithms for logical control". Inst. of Engineering Cybern. of NAS of Belarus, Minsk, 1999 (book in Russian).

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27-29, 2001; Przytok near Zielona Gora, Poland

Automatic HDL Generation for A DES Codec for an Encrypted NFS Server based on an Extended Petri Net

Shin'nosuke Yamaguchi, Katsumi Wasaki, Yasunari Shidama, Pauline Naomi Kawamoto

Faculty of Engineering, Shinshu University
4-17-1 Wakasato Nagano-city, Nagano 380-8553, Japan, *xrei@cs.shinshu-u.ac.jp*

Abstract. *In this paper, we propose a method for automatically generating the Hardware Description Language (HDL) source for a parallel pipelined DES enciphering circuit for the Network File System based on an extended Petri net. For this work, the authors have proposed an extended Petri net, called a "Logical Coloured Petri Nets (LCPN)", suitable for the design of complex control system processes and discuss its methods of evaluation. First, the logical verification of a circuit and its timing checks based on zero–delays are done at the HDL level. Then, delay information for real devices are used to verify the timing and operation speeds at the FPGA level. It is the purpose of this research to improve the reliability of circuit operation at the time of design by verifying the correctness of pipeline and parallel operations with the proposed new technique.*

Keywords. *extended Petri Net, Hardware Description Language*

1 Introduction

There has been a rapid expansion recently in computer systems making it necessary to develop flexible microprocessor units (MPUs). In general, MPU sequence controllers have to control multiple executing units which work together concurrently and synchronously while maintaining operation synchronization. The sequence controller for the parallel/pipelined ALU has been used as an important block in MPUs for advanced computer systems and is usually designed based on sequence ladder languages/diagrams, state charts, decision tables or circuit diagrams. However, these methods have the following problems [14]:

(1) A variety of designing errors and bugs are easily introduced into parallelized behavior.

(2) Verification of the correctness of specifications is difficult.

(3) Tracing control flow/software is difficult by anyone other than the original designers.

A descriptive model of sequence control by Petri nets has become attractive due to its simplicity [1][2]. Petri net is a graphical model which makes understanding the control flows easy. The mathematical nature of the Petri net can be used to obtain information about the behavior of systems which operate in a dynamic environment. Especially when time or safety factors make actual simulations of a system unfeasible, a study of the relationships between the mathematical objects of a Petri net can reveal such conditions as deadlocks, traps, reachability of marking states, etc. which aid in the verification of system operations. Examples of applying conventional Petri nets(PNs) and Colored Petri

nets(CPNs) to system design and analysis have been given in such areas as hardware design by Jensen [3], deadlock verification of Ada programs by Murata et al [4], FA control by Nagao et al [5][10][11][12][13], and computer system models by Miller [14].

The following problem exists when PNs and CPNs are actually applied to the work of describing control systems. The operation of these nets (firing conditions of transitions and the movements of tokens) is uniquely fixed. It is necessary to use many transition and place elements to represent the branching of conditions in a process and as a result, the net scale increases. In this paper, we describe the automatic HDL generation for a parallel pipelined DES enciphering circuit based on an extended Petri net. For this, the authors proposed an extended Petri net called a"Logical Coloured Petri Net (LCPN)", suitable for the design of complex control systems processes and discuss its methods of evaluation in [6][7][13].

2 Logical Coloured Petri Net - (LCPN)

In this section, we describe the definition of a logical coloured Petri net (LCPN) which can be used to improve the description capabilities of conventional Petri nets and colored Petri nets. Specifically tokens in LCPN have data (colours) and firing conditions are given by an arbitrary logical expression which is written in terms of the presence of tokens in input places and the data values of tokens. This feature greatly simplifies the work of expressing diverging conditions in a system. Once an LCPN model is developed, it can be analyzed by using reachability trees and simulations [8][9].

Definition 1 *LCPN : A logical coloured Petri net is a tuple of sets* $N_E = (S_E, T_E, F_E, M_E)$ *which fulfills each of the following conditions.*

(1) Let $S_E = \{s_1, s_2, \cdots, s_n\}$ *and* $T_E = \{t_1, t_2, \cdots, t_m\}$ *be the sets of the place and transition elements, respectively. Let* $F_E = \{f_1, f_2, \cdots, f_l\} \subseteq (S_E \times T_E) \cup (T_E \times S_E)$ *being a set of arcs from places to transitions and transitions to places.*

(2) The mark of each place $s_i \in S_E$ $(i = 1, 2, \cdots, n;\ \ n = \mid S_E \mid)$ *can take the value of a natural number from 0 to N. This is denoted by* $\mu(s_i)$*. We assume* $\mu(s_i) = 0$ *if there is no(empty) marking on* s_i*. Here, we define the function* μ *as* $\mu : S_E \longrightarrow \{0, 1, 2, \cdots, N\}$

(3) The capacity (maximum number of marks) of each place s_i *is 1. The set of all possible marks from (1) and (2) is represented as a mapping from* S_E *to* $\{0, 1, 2, \cdots, N\}$*. Thus, we define the cartesian product set* M_E *as* $M_E = \{0, 1, 2, \cdots, N\}^{S_E}$

(4) ${}^*t_j$ *represents the set of all places (input places) which have an arc extending to* $t_j \in T_E$ $(j = 1, 2, \cdots, m; m = \mid T_E \mid)$*. Similarly,* $t_j{}^*$ *represents the set of all places (output places) with an arc extending from* t_j*.*

$$\begin{aligned} {}^*t_j &= \{s \in S_E :\ (\exists f)(f \in F_E, f = (s, t_j))\} \\ t_j{}^* &= \{s \in S_E :\ (\exists f)(f \in F_E, f = (t_j, s))\} \end{aligned}$$

(5) The firing evaluation of a transition t_j *for an arbitrary marking* $\mu \in M_E$ *examines the firing condition* Φ^j*.* $\Phi^j(\mu \mid^* t_j)$ *is described by a logical expression in terms of the state* $\mu \mid^* t_j$ *of the places which belong to* ${}^*t_j$ *and is used to determine the next marking* $\mu' \in M_E$*.*

t_j *is called firable if* Φ^j *is evaluated to be true. It* t_j *is fired, a mark is removed from each place in* ${}^*t_j - t_j{}^*$*. Places in* $t_j{}^*$ *depend on the state of* ${}^*t_j$ *and are modified by the following* C^j*.*

$$C^j : \{0,1,2,\cdots,N\}^{*t_j} \longrightarrow \{0,1,2,\cdots,N\}^{t_j*}$$

$$if \quad \Phi^j(\mu|^* t_j) \; is \; true \quad then$$

$$\mu' = \begin{cases} 0 & : on \quad {}^*t_j - t_j{}^* \\ C^j(\mu|^* t_j) & : on \quad t_j{}^* \\ \mu & : otherwise \end{cases}$$

On the other hand, t_j is not firable if Φ^j is false. At this time, the state $\mu' = \mu$ is unchanged.

We can show the next marking resulting from a transition evaluation as a mapping f^j from state $\mu \in M_E$ of the places to $\mu' \in M_E$.

3 Modelling of Parallel Accumulators by LCPNs

3.1 Outline of FPGA development environment

In this section, we describe the design and analysis for the parallel accumulator with a Petri net tool made for trial purposes. Fig. 1 shows the data flow of the FPGA devel-

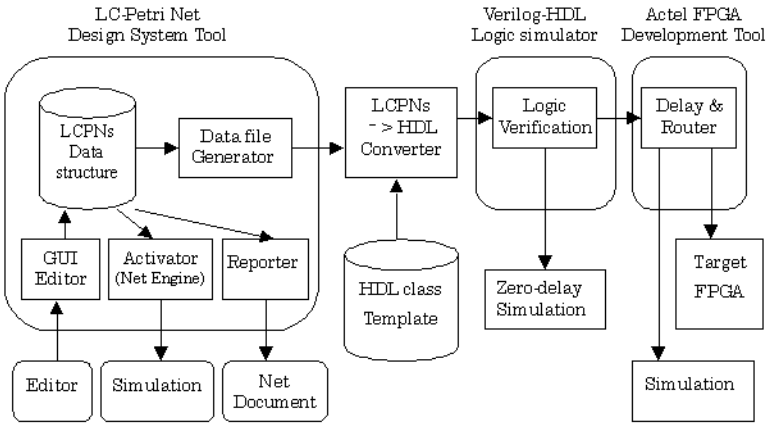


Figure 1: The FPGA development environment.

opment environment. This development environment is composed of the LCPN Design System Tool (see Fig. 2)[10], LCPN to HDL Converter, Verilog-HDL Logic Simulator, and Actel FPGA Development Tool.

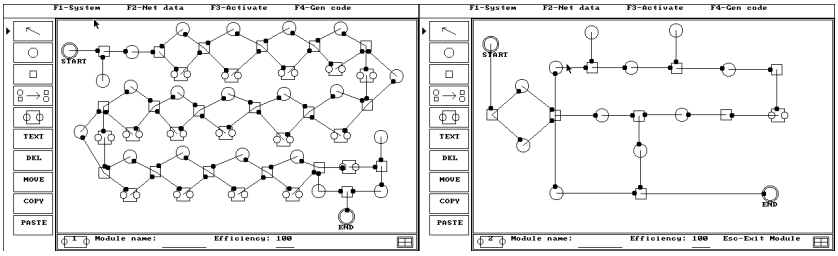


Figure 2: LCPN Design System Tool

At this stage, the tools examine the design's defects (validity of operations, presence of deadlocks, synchronous relation, etc. of the pipeline operation) in the accumulator

model. By using the Petri net, we can efficiently detect the presence of pipeline hazards and correct them. We can also verify the stability of the synchronous relations of parallel circuits from the net.

Next, the tools convert the net model which has been verified into existing hardware description language (Verilog-HDL). Verilog-HDL defines well each function module of the circuit (comparators, full adders, etc.) as a "Class" and associates the Petri net model from the "Class" of a sub-net and the circuit. The HDL source code files are all automatically generated with the HDL generator made by our group.

Finally, the HDL of the parallel operation machine generated automatically is mounted on the gate array with an FPGA layout tool which verifies the timing and operation speed at the circuit level. We can decrease the labor of verifying stability with regard to such problems as pipeline hazards, because we need only the checks of timing, operation speed, and operation results using the delay information of real devices.

3.2 Method of Converting LCPNs to HDL

We show the method of conversion from the LCPN structural data file (generated from the Petri net tool) to HDL below.

STEP 1 Expand sub-nets defined by the user
STEP 2 Replace sub-nets with simpler sub-nets (adder, multiplier modules, etc.)
STEP 3 Class generation of Verilog-HDL from sub-nets and transitions
STEP 4 Replace place elements with corresponding latches.
STEP 5 Replace arcs with wire instances.
STEP 6 Resolve information for the class template and connection wiring
STEP 7 Generate the target HDL source of Verilog-HDL

Figure 3: *LCPN to HDL Converter*

The LCPN to HDL converter performs the conversion sequence [STEP 1]-[STEP 7] from the net data file to HDL source file automatically. This conversion engine and the template database for trial purposes are made with the Borland Delphi5.0 development environment based on the Microsoft Windows95/98/NT operating system. This implementation has reduced the cost of HDL design and programming and gate array testing on site to one third of what development would have cost using conventional procedural language-based methods.

4 Example of Parallel DES Accumulator with LCPNs

In this work we used the LCPN to automatically generate HDL source code for a parallel accumulator in an actual DES enciphering circuit. The circuit was embedded into

a network file system (NFS) server in order to evaluate its efffctiveness. We codesigned the NFS with LCPN in [15].

The NFS described in RFC-1813, allows a local file system to mount a file system through a network. The NFS server and client distinguish files in a server by a unique file handle that is an integer number in 64 bits containing information of the file (name, i-node, update time, etc.)

The Data Encryption Standard(DES) cryptogram is an encoding algorithm announced in 1975. The DES cryptogram encodes a plain sentence in 64 bits by a key in 56 bits. Fig. 4 shows the one step operation in the DES cryptogram.In function f, the sentence is replaced by eight arrays (S box). The DES cryptogram respectively carries out its operations 16 times.

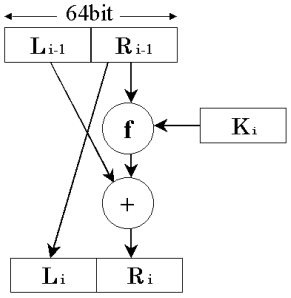


Figure 4:One step of DES cryptogram

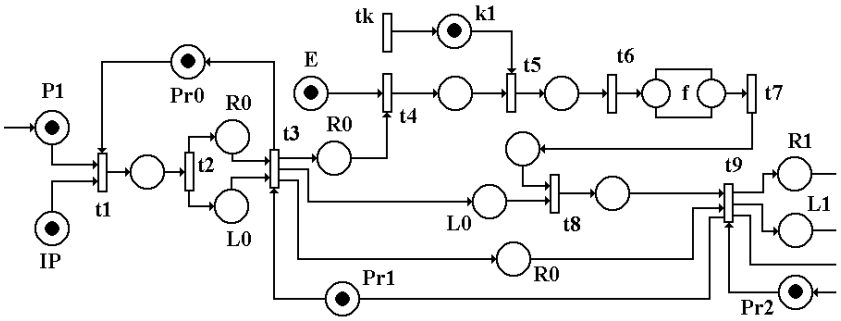


Figure 5:LCPN Example of Parallel DES Accumulator(subnet)

Next, the processing for execlusive OR is redesigned as a pipeline structure of 16 stages. Fig. 5 shows an example of a parallel DES accumulator with designed LCPNs based on this sequence. Here, we put the data divided into 64 bits in place P_1. Then, it is transposed by IP and then divided into L_0 and R_0. After the operation with the key schedule K_1 and the extension transposition E, etc. is executed, the exclusive OR of R_0 and L_0 is taken and the output is placed in R_1. In the first step, L_1 takes the value of R_0 as it is. We are attempting pipelining of 16 stages by carrying out exclusive processing controlled by place $Pr_n(n = 1, 2, \cdots, 16)$. While the file contents are being encoded, information on the file is handled in a different route.

In this pass, data is coded according to the character exchange style cryptogram. In this way, it is possible to keep shared file system information concealed while recording coded data.

To read back the encoded data, we repeat the f and exclusive OR operations 16 times, using the key schedule in reverse order. The net for one 16 stage DES pipeline for encryption file system code/decoding required 832 places, 400 transitions, and 1426 arcs.

5 Conclusion

We proposed and defined the concept of a logical coloured Petri net (LCPN) which improves the description capability of current PNs and CPNs. We modelled hardware for parallel pipelined DES accumulators for a network file server system by using LCPNs and analyzed the net model with a Peti net development tool environment. We implemented the accumulator hardware generated automatically from a net whose stability was verified and confirmed its operation on the simulator of the system.

In the future, we plan to study further the operation of LCPN and improve it as we apply it to CASE tools for developing system control software.

References

[1] T.Murata : "Petri Nets: Properties, Analysis and Applications ", Proc. IEEE, Vol. 77, No. 4, pp.541-580, 1989.

[2] J.L.Peterson : "Petri Net Theory and the Modelling of Systems", Prentice-Hall, Inc., 1981.

[3] K.Jensen : "Coloured Petri Nets, Basic Concepts, Analysis Methods and Practical Use", Vol. 1, Springer–Verlag, 1992.

[4] T.Murata, B.Shenker, and S.M.Shatz : "Detection of Ada Static Deadlocks Using Petri Net Invariants", IEEE Trans. Softw. Eng., Vol. 15 , No. 3 , pp.314–326, 1989.

[5] Y.Nagao, H.Ohta, H.Urabe and S.Kumagai : "Petri Net Based Programming System for FMS" IEICE Trans. Fundamentals. , Vol. E75–A, No. 10, pp.1326-1334, 1992.

[6] K.Wasaki, Y.Fuwa, M.Eguchi, and Y.Nakamura : "Extended Petri Nets for Control System Software", Technical Report of IEICE, COMP93-12, SS93-6, pp. 37-44, 1993.

[7] K.Wasaki, Y.Fuwa, M.Eguchi and Y.Nakamura : "Capacity of the Logical Colored Petri Net (LC-net) as a CASE Tool", Technical Report of IEICE, CAS93-69, pp. 101-108, 1993.

[8] P.N.Kawamoto, Y.Fuwa and Y.Nakamura: "Basic Concepts for Petri Nets with Boolean Markings", Journal of Formalized Mathmatics, Vol.4, No.1, 1993.

[9] P.N.Kawamoto, M.Eguchi, Y.Fuwa and Y.Nakamura : "The Detection of Deadlocks in Petri Nets with Ordered Evaluation Sequences", Technical Report of IEICE, SS92-29, KBSE92-50, pp. 45-52, 1993.

[10] M.Kitazawa,P.N.Kawamoto, Y.Fuwa, Y.Nakamura: "Petri Net Software Development for Parallel/Distributed Systems", *Proc. Parallel and Distributed Computing and Systems(PDCS'96)*, pp.418-420, 1996.

[11] T.Nakao, T.Yamagishi, P.N.Kawamoto, Y.Fuwa, Y.Nakamura: "A Petri Net Based Protocol Design Tool for Wireless Data Communication", *Proc. Multi-Dimensional Mobile Communications 96*, pp.307-311, 1996.

[12] A.Deodhar, E.C.Tan, A.Wahab : "FPGA Implementation of Discrete–Time Systems", *Fourth International Conference on Control, Automation, Robotics and Vision Proc.*, TE-2-2, pp.1328-1332, 1996.

[13] K.Wasaki, Y.Fuwa, M.Eguchi, Y.Nakamura : "Logical Coloured Petri Net Expanded to be Suitable making the Control System Model", *Fourth International Conference on Control, Automation, Robotics and Vision Proc.*, TA-2-2, pp.708-713, 1996.

[14] R.E.Miller : "A Comparison of Some Theoretical Models of Parallel Computations", *IEEE Trans. on Computers*, Vol. C-22, No. 8, pp.710-717, 1973.

[15] S.Yamaguchi, K.Wasaki, S.Shidama : "A process design for the Network File System model based on the Logical Coloured Petri Net", *Sixth International Conference on Control, Automation, Robotics and Vision (ICARCV'2000)*, December 2000.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Góra, Poland

STATE SPACE CALCULATION ALGORITHM OF HIERARCHICAL PETRI NETS WITH APPLICATION OF DECISION DIAGRAMS

Piotr MICZULSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *P.Miczulski@iie.pz.zgora.pl*

Abstract. *The state space of a hierarchical Petri net can be presented as a hierarchical reachability graph. However hierarchical reachability graph can be described with the help of logic functions. From the opposite direction, binary decision diagrams (BDD) are efficient data structures for representing logic functions. Because of the exponential growth of the number of states in Petri nets, it is difficult to process the whole state space. Therefore the abstraction method of the selected macromodules (macroplaces and macrotransitions) gives the possibility of analysis and synthesis for more complex systems. The goal of the paper is to show the representing method of the state space in the form of connected system of binary decision diagrams as well as its calculation algorithm.*

Key Words. *Hierarchical Petri nets, Binary Decision Diagrams, Hierarchical reachability graph*

1. INTRODUCTION

Hierarchical interpreted Petri nets are a structural method of describing concurrent processes [6]. They enable to design more complex systems through abstracting some parts of the net, at the moment. It is possible when the abstract part of net is formally correct i.e. it is safe, living and persistent [4].

One of possibilities of representing digital circuit state space is hierarchical reachability graph [5]. It describes the state space at various hierarchy levels. Hierarchical reachability graph can be represented in the form of a logic function, where the logic variables correspond to places of Petri net. The number of variables equals to the number of places. However the efficient ways of representing logic functions are decision diagrams, e.g. Binary Decision Diagrams (BDD) or Zero-suppressed Binary Decision Diagrams (ZBDD).

In this paper, the calculation's method of hierarchical state space with the help of operations on logic functions and decision diagrams is presented. The symbolic traversal method of space state, for "flat" Petri nets, was presented in [1], and the application of this method for

hierarchical Petri nets as well as the description's method of hierarchical reachability graph with the form of system of connected decision diagrams are the new ideas.

2. HIERARCHICAL PETRI NETS

A hierarchical Petri net is a directed graph, which has three types of nodes called: places (represented by circles), transitions (represented by bars or boxes) and macroplaces (represented by double circles). The macroplaces include another places, transitions and also macroplaces and they signify lower levels of hierarchy (*fig. 1*). When a hierarchical Petri net is used to model a parallel controller, each place and macroplace represents a local state of the digital circuit. Every marked place or macroplace represents an active local state, and the set of places, which are marked at the same time, represents the global state of the controller. However the transitions describe the events, which occur in the controller. The controller also can receive signals (inputs) coming from a data path as well as from another control unit. It produces, using this information, control signals, which determine the behavior of the system. Input signals can be assigned to transitions in the form of logic function. This function is called a transition predicate. If the predicate is satisfying and all input places of the transition have markers, the transition will fire.

The figure below presents the example of hierarchical Petri net, which consists of some levels of hierarchy.

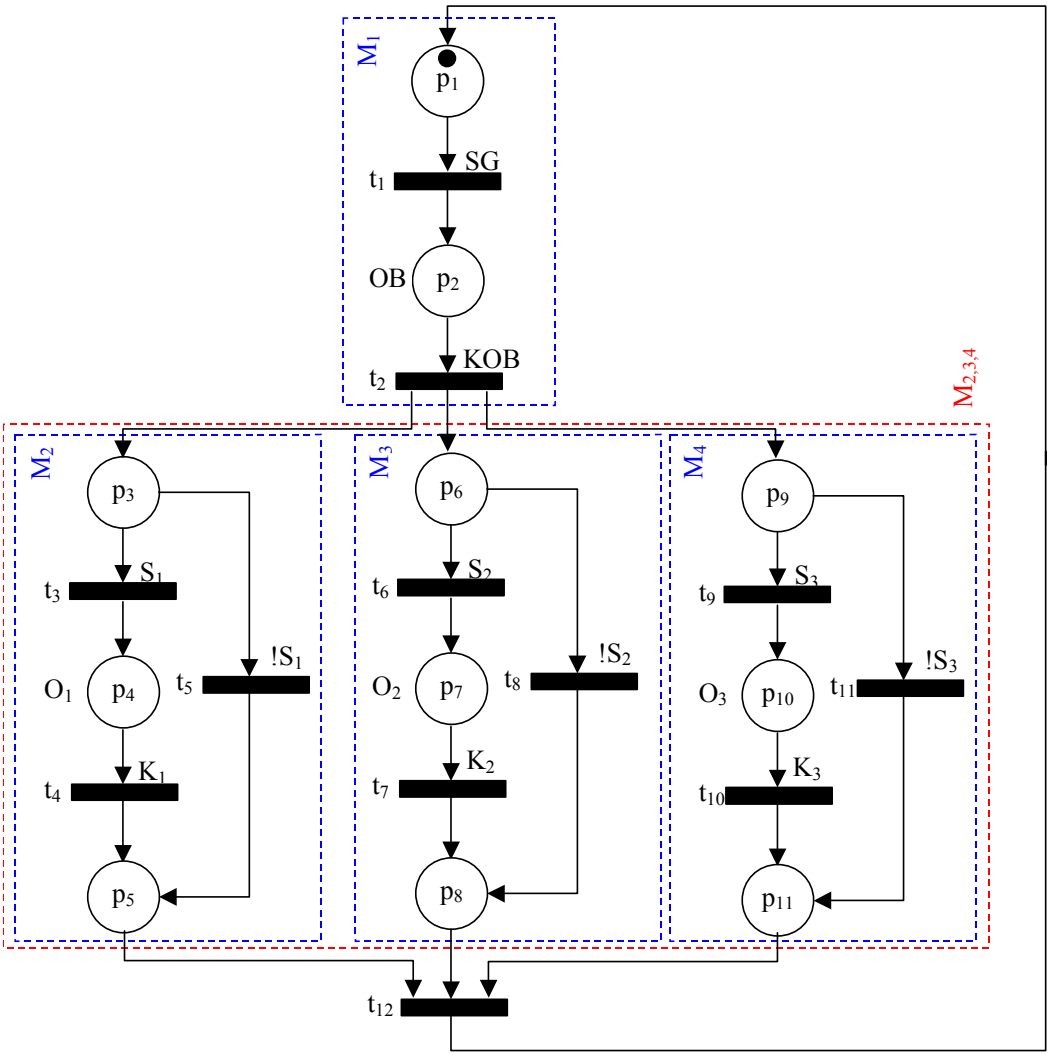


Fig. 1. The example of hierarchical Petri Net

The top hierarchy level is composed of macroplaces M_1 and $M_{2,3,4}$. The state space of it can be described in the form of the logic function: $\chi(M_1, M_{2,3,4}) = M_1\overline{M}_{2,3,4} + \overline{M}_1 M_{2,3,4}$.

The lower level of abstraction is composed of three parallel macroplaces M_2, M_3 and M_4 which are parts of macroplace $M_{2,3,4}$. However the macroplaces M_1, M_2, M_3 and M_4 form the lowest level of hierarchy. Similarly to the top level of hierarchy, every remaining abstraction level can be described with the help of logic function.

First thing which the designer has to do (after the controller was modeled with the help of Petri net) is to create graphical or textual description of Petri net. With this end in view he can use, for example the textual format of hierarchical Petri net specification (PNSF2) [6]. Next, the computer program loads this specification to internal data structures. The hierarchical or "flat" Petri net is stored in object data structures (*fig. 2*).

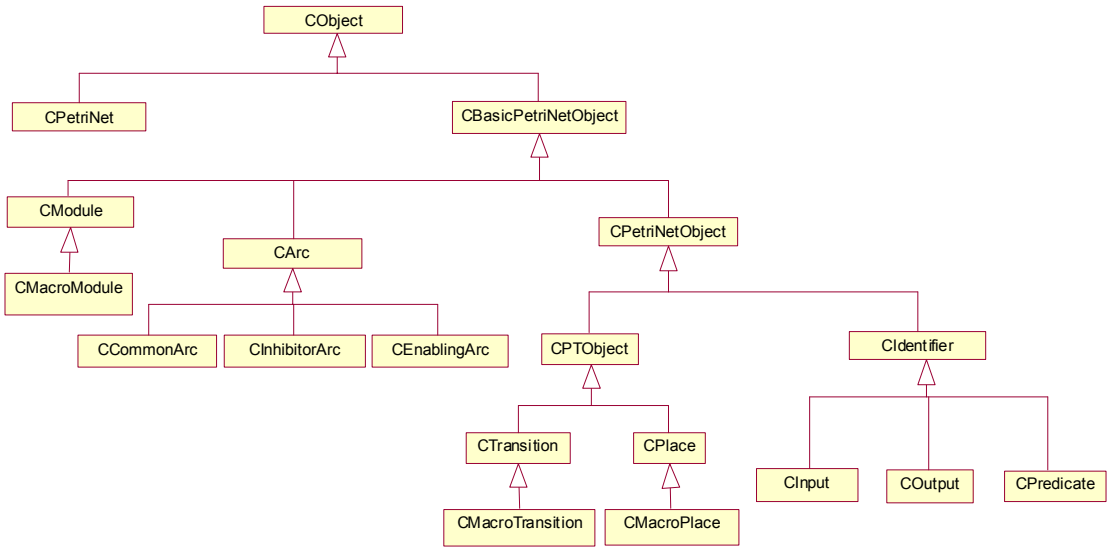


Fig. 2. The object data structure for representing of hierarchical Petri nets in computer memory

In the next step we can calculate the state space of digital controller using internal data structure and the logic operations which are performing using decision diagrams.

3. THE STATE SPACE AND BINARY DECISION DIAGRAM

A binary decision diagram is a rooted, directed, acyclic graph, which has two sink nodes labeled 0 and 1, representing Boolean function 0 and 1, and non-sink nodes, each labeled with a Boolean variable. Each non-sink node has two output edges labelled 0 and 1 and represents the Boolean function corresponding to its 0 edge or the Boolean function corresponding to its 1 edge. The construction of a BDD for the function is based on its Shannon expansion [2, 3].

An ordered binary decision diagram (OBDD) is a BDD in which all the variables are ordered and every path from the root node to a sink node visits the variables in the same order. A reduced ordered binary decision diagram (ROBDD) is an OBDD in which each node represents a distinct logic function. The size of a ROBDD strictly depends on variable ordering. Many heuristics have been developed to optimize the size of BDDs [2, 3]. In this paper all consideration binary decision diagrams are reduced and ordered.

The whole state space of the presented hierarchical Petri net (*fig. 1*) can be described as logic function:

$$\chi_{|M_0\rangle} = p_1\overline{p}_2\overline{p}_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1p_2\overline{p}_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8p_9\overline{p}_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9p_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9p_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7\overline{p}_8p_9\overline{p}_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}p_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9p_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6\overline{p}_7p_8\overline{p}_9\overline{p}_{10}p_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6\overline{p}_7\overline{p}_8\overline{p}_9\overline{p}_{10}p_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9\overline{p}_{10}p_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6\overline{p}_7p_8\overline{p}_9p_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9\overline{p}_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6\overline{p}_7p_8\overline{p}_9\overline{p}_{10}p_{11} + \overline{p}_1\overline{p}_2p_3\overline{p}_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9p_{10}\overline{p}_{11} +$$
$$\overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6\overline{p}_7p_8\overline{p}_9\overline{p}_{10}p_{11} + \overline{p}_1\overline{p}_2\overline{p}_3p_4\overline{p}_5\overline{p}_6p_7\overline{p}_8\overline{p}_9p_{10}\overline{p}_{11} + \overline{p}_1\overline{p}_2\overline{p}_3\overline{p}_4p_5\overline{p}_6p_7\overline{p}_8\overline{p}_9\overline{p}_{10}p_{11}$$

This means, that modeled controller may be in one of the twenty-nine states. The BBD diagram, for this function, has 24 non-sink nodes. We can reduce the number of nodes by creating connected system of binary decision diagrams. In this case, we are giving the six decision diagrams, which have nineteen non-sink nodes. This situation follows from that the size of decision diagram depends, among other things, on the number of logic variables of the function.

4. CALCULATION ALGORITHM OF STATE SPACE

In this chapter there is the description of state space calculation algorithm (*fig 3*). After parsing a description of Petri net, written in PNSF2 format and loading Petri net to internal data structures, the algorithm checks a structure of Petri net. If it is a "flat" Petri net, the algorithm splits it into hierarchical structure of macroplaces. In the next step, for each macroplace, the algorithm (recursively) calculates characteristic functions. This function, represented in the form of decision diagram, describes state space of each macroplace. However the calculated decision diagram is joined to the connected system of decision diagrams, which represent the whole state space of hierarchical Petri net.

One of more important steps of this algorithm is calculating characteristic function, describing the state space of the macroplace (*fig 4*). The symbolic traversal algorithm was gathered from [1]. In this method, next marking is calculated using their characteristic function and transition function. The transition functions ($\Delta : \Omega \to \Omega$) are logic functions associated with places and defined as a functional vector of Boolean functions:

$$\Delta = [\delta_1(P,X), \delta_2(P,X), ..., \delta_n(P,X)],$$

where $\delta_i(P,X)$ is a transition function of place p_i; P and X are sets of places and input signals respectively. The function δ_i has value 1 when place p_i will have a token in the next iteration, otherwise it equals 0. Every function δ_i consists of two parts:

- a part describing the situation when the place p_i will receive a token,
- a part describing the situation when the place will keep a token.

For example: place p_7 (*fig. 1*) will have a token in the next iteration, if place p_6 have token and input signal S_2 is active (transition t_6 will fire) or place p_7 has already got a token and

either input signal K_2 is inactivate (transition t_7 is disabled), thus the function δ_7 can be defined as follows:

$$\delta_7 = p_6 * S_2 + p_7 * \overline{K}_2 .$$

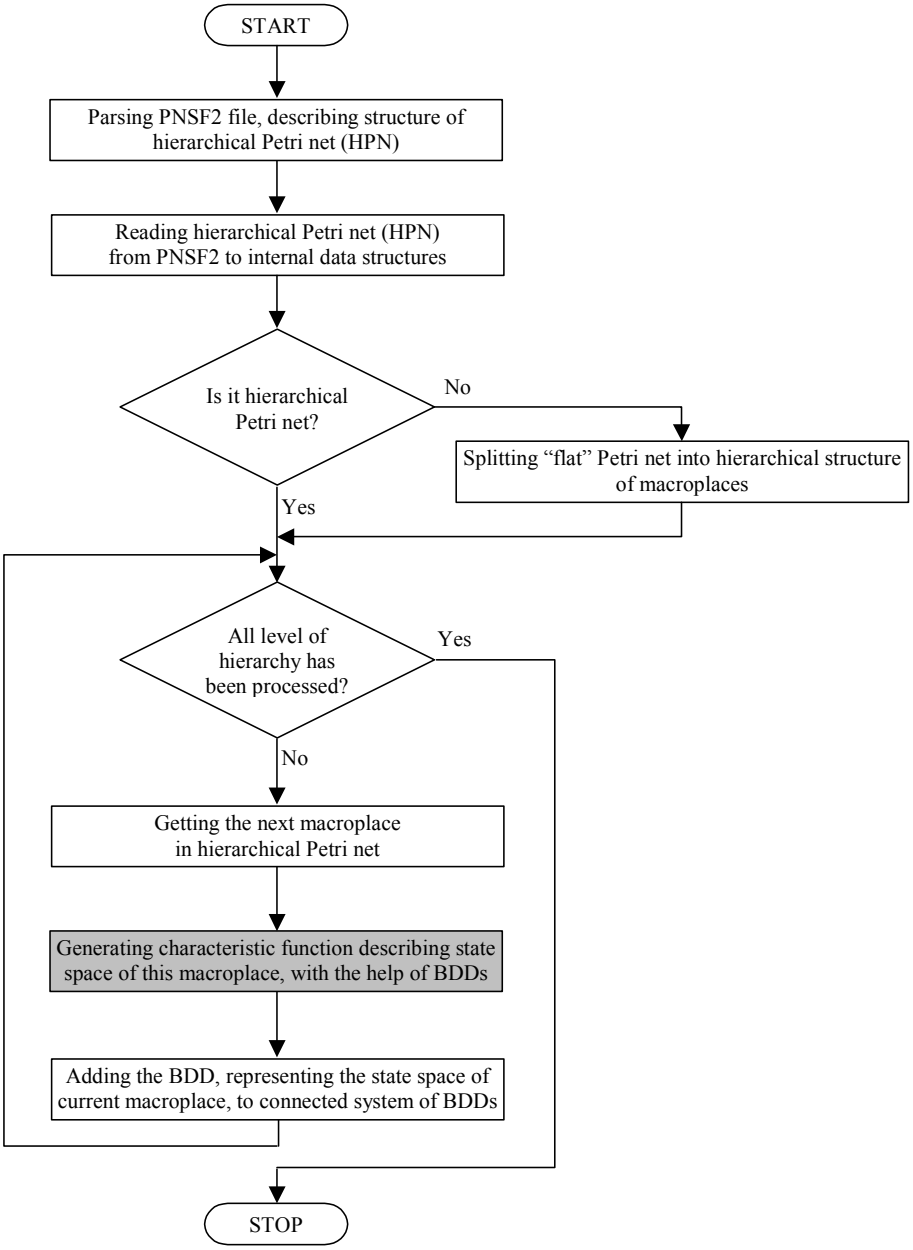


Fig. 3. Calculation algorithm of state space of hierarchical Petri net

The computation operation of a set of marking which can be reached from the current marking (*current_marking*) in one iteration according to the following equations:

$$next\_marking = \exists_{p}\exists_{x}(current\_marking * \prod_{i=1}^{n}[p_i' \odot (current\_marking * \delta_i(p,x))])$$

where p, p', x denote the present state, the next state and the input signal; $\exists_{p}$ and $\exists_{x}$ represent existential quantification of the present state and the input signal variables; symbol $\odot$ and * represents logic operators XNOR and AND respectively.

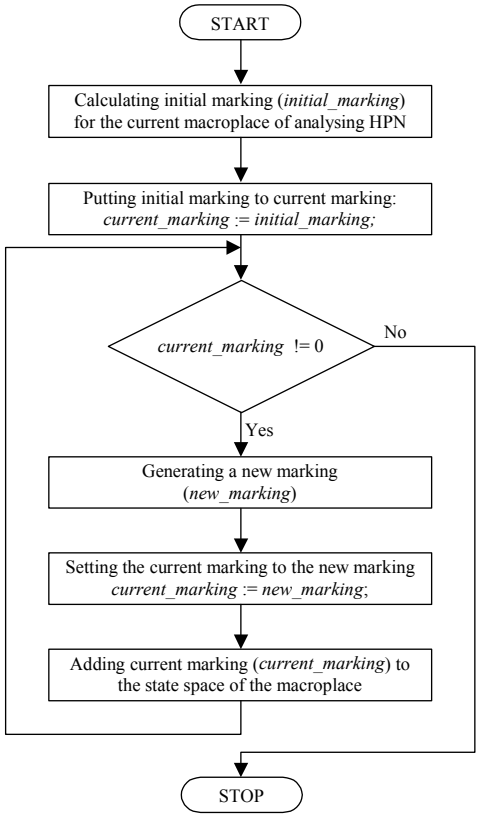


Fig. 4. Symbolic traversal algorithm for hierarchical Petri net

5. SUBMISSION

The application of connected system binary decision diagrams enables to reduce the number of nodes of decision diagrams. From the opposite the application hierarchical Petri nets makes easier designing parallel digital controller easier. It means, that we can process digital circuits, described by hierarchical Petri nets, on various abstraction's levels unnecessarily processing the whole state space. The next step will be working out the rules and the algorithm of transforming a flat Petri net into a hierarchical one.

The paper was prepared under the guidance of Professor M. Adamski.

REFERENCES

[1] K. Biliński, "Application of Petri Nets in parallel controller design", *PhD. Thesis*, University of Bristol, Electrical and Electronic Department, 1996

[2] R. Drechsler, "Binary Decision Diagram. Theory and Implementation", *Kluwer Academic Publishers*, 1998

[3] S. Minato, "Binary Decision Diagrams and Applications for VLSI CAD", *Kluwer Academic Publishers*, 1996

[4] T. Murata, "Petri Nets: Properties, Analysis and Applications". *Proceedings of the IEEE*, 77(4) ss. 541 – 580, 1989

[5] M. Notomi, T. Murata, "Hierarchical Reachability Graph of Bounded Petri Nets for Concurrent-Software Analysis". *Proceedings of IEEE Transactions on Software Engineering*, Vol. 20, No 5, 1994

[6] M. Węgrzyn.: "Hierarchical implementation of concurrent digital controllers with application FPGA". *PhD. Thesis*, Warszawa 1998

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

TIMED PETRI NETS FOR SOFTWARE APPLICATIONS

Grzegorz ANDRZEJEWSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *g.andrzejewski@iie.pz.zgora.pl*

Abstract. *This paper presents a model of interpreted timed Petri nets in an abstract program environment called Virtual Decision System (VDS). There are described possibilities of software implementation with technical limitations of mentioned method. It shows a practical example with elements of deviation analysis. Possibilities of interpretation of dynamic parameters with special taking automatic decomposition algorithm into Hardware/Software Co-Design systems are described.*

Key Words. *Timed Petri Nets, Digital Microsystems, Software Applications*

1. INTRODUCTION

There exist a lot of ways for formal specification of digital control systems. The most of all (at present) are various modifications of Finite State Machine (e.g. CFSM, HCFSM), flow graphs (e.g. DFG, CDFG), hardware description languages (e.g. VHDL, Verilog) and Petri nets [7]. The latter have a special importance in designing process for the sake of rich formal verification apparatus and naturalness of concurrency aiding [5].
In designing process of control system very often a situation occurs in which specific after-effects of processes are strongly dependent on time. There is a simple example showing this problem on a simplified control system of initial washing in automatic washer (Fig.1).

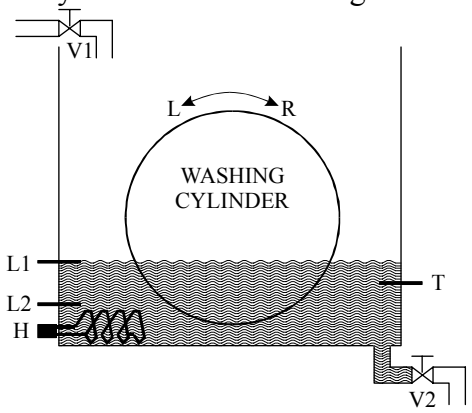


Fig. 1.An example of automatic washer control system

After turning on washing program valve V1 is opened and water is infused. Infusing process lasts to moment achieving of L1 level. In the same time after exceeding L2 level (total sinking of heater H) if the temperature is below required (TL1) heater system is turned on. After valve

V1 closing and water heating to temperature TL2 washing process is started in which the washing cylinder is turned alternate left and right for 10 sec. with 5 sec. break between. Keeping of temperature process is active for whole the washing cycle. The cycle is turned off after 280 sec. and cylinder is stopped, the heater is turned-off and the valve V2 is opened for water removing.
So, this problem is possible to describe by a hardware description language using *wait* and *after* instructions. But synthesis of them can give a lot of problems because these language constructions are not supported by synthesis tools. The other formal specification models (except Petri nets) don't allow a timing dependence in general.
In this article a new model of interpreted timed Petri nets for software applications is proposed, with its synthesis method. The method allows easy and cheap a practical implementation for shown dynamic system.

2. VIRTUAL DECISION SYSTEM

Program implementation of Petri net is possible in various ways. One is defining an abstract program environment making possible efficient implementation of control systems.
The proposed program environment is defined in terms of net places, names of input/output signals and program decision system called the Virtual Decision System VDS (Fig.2).

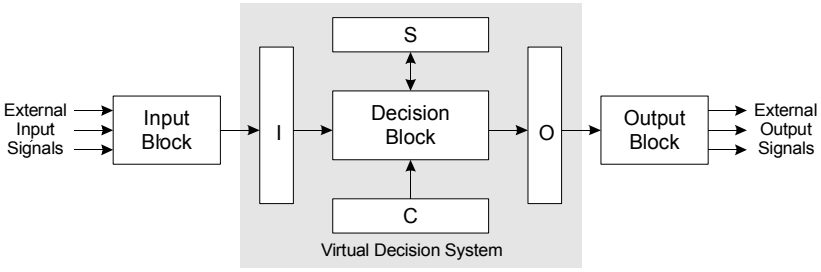


Fig. 2. Virtual Decision System

Where S – is a block storing information about system state, C – is a specification block, I – is a block storing the names of active input signals: $I = \{x \in X : x = 1\}$, O – is a block storing the names of active output signals: $O = \{y \in Y : y = 1\}$. It was broadly described in [8].

3. PROGRAM INTERPRETED PETRI NET

In [4] were introduced definitions describing program Petri net with most important kind of its: interpreted net, net with enabling and prohibit arcs. There is given a definition of program interpreted Petri net as a basis for further theoretical studies.

Program interpreted Petri net is a 5-tuple:

$$IPPN = (P, C, S_0, X, Y) \tag{1}$$

where: P is a non-empty set of names of net places, S_0 – is an initial net state, X is an input alphabet, Y is an output alphabet and C – is a specification structure composed of tables C^1 and C^2, such that: C^1 has $n \times 3$ size (n describes a number of decision rules) and C^1_{i1} i C^1_{i2} fields include sets of names of net places, which are correspondingly the conditions and the results of decision rule, C^1_{i3} field includes conditions in logical formulas form b(X) described on input alphabet; C^2 has $m \times 1$ size ($m = ||P||$) and each of fields includes set of names of output signals assigned to given place.

The conditions of rule *i* enabling:

$$\forall p \in C^1_{i1} : p \in S \tag{2-a}$$

$$b \in C^1_{i3} \Rightarrow b(X)=1 \tag{2-b}$$

The actions associated with rule *i* firing:

$$\forall p \in C^1_{i1} : S = S - p \quad (3\text{-a})$$

$$\forall p \in C^1_{i2} : S = S + p \quad (3\text{-b})$$

$$\forall (p \in C^1_{i1}) \forall (y \in C^2_p) : O = O - y \quad (3\text{-c})$$

$$\forall (p \in C^1_{i2}) \forall (y \in C^2_p) : O = O + y \quad (3\text{-d})$$

That means that from block S the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted. As well as from block O the names of output signals included in C^2 table's fields corresponding with places belonging to C^1_{i1} are removed and the names of output signals included in C^2 table's fields corresponding with places belonging to C^1_{i2} are inserted.

4. TIMED PETRI NETS

Timed Petri nets were introduced earlier in literature [1,6]. And yet the subject matter wasn't developed for the sake of implementation difficulties mainly. A new program model of timed Petri net is proposed in this paper, that allows an efficient implementation in microprocessor modules.

A general program model of interpreted timed Petri net is shown as an ordered 6-tuple:

$$TIPPN = (P, C, S_0, X, Y, T) \quad (4)$$

where: P, S_0, X, Y are defined just like (1), C is a specification block defined according to a kind of described net, and T is discrete scale of time. The differences will be stressed also in conditions of rules enabling and actions connected with their firing.

There is modified block S, keeping information about system state. For timed nets it can be a structure composed of following elements:

$S_1 = \{p: p \in P\}$ – includes names of places currently marked, being outside a keeping state,

$S^1_2 = \{p: p \in P\}$ – includes names of places currently marked, being inside a keeping state,

$S^2_2 = \{t: t \in T\}$ – includes numbers assigned to discrete scale of time and describing state of timing actions in accordance with the places in S^1_2,

$S^1_3 = \{p: p \in P\}$ – includes names of places prepared for marking after closing adequate timing actions assigned to given rules firing,

$S^2_3 = \{t: t \in T\}$ – includes numbers assigned to discrete scale of time and describing state of timing actions in accordance with the places in S^1_3.

4.1. Program timed net of P-type

The time parameters are assigned to places in the nets of P-type and they are called keeping times. The keeping times can be perceived as times of staying marker in place *p* what is shown in Fig.3-a) (along with practical interpretation).

The condition of transition T2 enabling is a time (2 sec.) passing since moment of introducing marker into place P1. In practice this situation is interpreted as starting external timing action by signal c1. The end of action is indicated by set signal c2 conditioning transition T2.

A theoretical model is consistent with (4). Specification block C has differences to (1):

table C^2 has m×2 size, into $C^2_1(p)$ are included names of output signals assigned to place *p*, and into $C^2_2(p)$ are included numbers assigned to discrete scale of time T, describing the keeping time of marker in place *p*.

The condition of rule *i* enabling (just like 2-b) and:

$$\forall p \in C_{i1}{}^1 : p \in S_1 \quad (5)$$

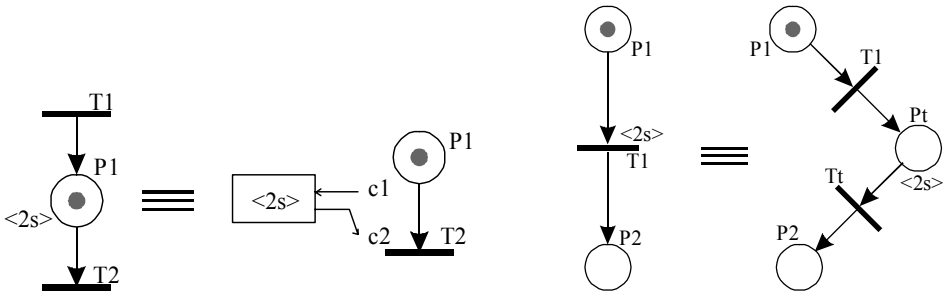
The actions associated with rule *i* firing:

$$\forall p \in C^1_{i1} : S_1 = S_1 - p \quad (6\text{-a})$$

$$\forall p \in C^1_{i2} : C^2_2(p) = 0 \Rightarrow S_1 = S_1 + p \quad (6\text{-b})$$

$$\forall p \in C^1_{i2} : C^2_2(p) \neq 0 \Rightarrow S^1_2 = S^1_2 + p \text{ i } S^2_2 = S^2_2 + C^2_2(p) \quad (6\text{-c})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if their keeping time equals zero ($C^2_2(p) = 0$), or to S^1_2 if not. Simultaneously a variable of timing action is added to S^2_2 (with initialization by $C^2_2(p)$ value).



a) *b)*

Fig. 3.An example of a) P-type net, b) T-type net

4.2. Program timed net of T-type

The time parameters are assigned to transitions in the nets of T-type and they are called execution times. The execution times can be perceived as times since moment of removing markers from input places to moment of inserting marker to output places given transition *t*. The situation can be interpreted with ideas P-type net introduced in 4.1. During transition T1 execution (Fig.3-b)) marker is moved to auxiliary place Pt. End of this execution (equivalent to keeping time of place Pt) allows auxiliary transition Tt execution. In consequence the marker is moved to output place P2.

A theoretical model is consistent with (4). Specification block C has differences to (1):
table C^1 has n×4 size, fields C^1_{i1}, C^1_{i2}, and C^1_{i3} just like (1), and into C^1_{i4} are included numbers assigned to discrete scale of time T, describing execution time of transition *t*.

The condition of rule *i* enabling just like (5). The actions associated with rule *i* firing just like (6-a) and:

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : S_1 = S_1 + p \quad (7\text{-a})$$

$$C^1_{i4} \neq 0 \Rightarrow \forall p \in C^1_{i2} : S^1_3 = S^1_3 + p \text{ i } S^2_3 = S^2_3 + C^1_{i4} \quad (7\text{-b})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if execution time of rule equals zero ($C^1_{i4} = 0$), or to S^1_3 if not. Simultaneously a variable of timing action is added to S^2_3 (with initialization by C^1_{i4} value).

4.3. Program timed net of PT-type

Timed Petri net PT-type is a superposition of nets defined in 4.1 and 4.2. There are the time parameters assigned with both places and transitions. A theoretical model is consistent with (4). Specification block C is composed of two tables: C^1 (just like in 4.2) and C^2 (just like in 4.1).

The condition of rule *i* enabling just like (5). The actions associated with rule *i* firing just like (6-a) and:

$$C^1_{i4} \neq 0 \Rightarrow \forall p \in C^1_{i2} : S^1_2 = S^1_2 + p \text{ i } S^2_2 = S^2_2 + C^1_{i4} \quad (8\text{-a})$$

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : C^2_2(p) = 0 \Rightarrow S_1 = S_1 + p \quad (8\text{-b})$$

$$C^1_{i4} = 0 \Rightarrow \forall p \in C^1_{i2} : C^2_2(p) \neq 0 \Rightarrow S^1_3 = S^1_3 + p \text{ i } S^2_3 = S^2_3 + C^2_2(p) \quad (8\text{-c})$$

That means that from block S_1 the names of places belonging to C^1_{i1} are removed and belonging to C^1_{i2} are inserted to S_1 if execution time of rule equals zero ($C^1_{i4} = 0$) and keeping time equals zero ($C^2_2(p) = 0$), or to S^1_2 if execution time of rule doesn't equal zero ($C^1_{i4} \neq 0$), or to S^1_3 if execution time of rule equals zero ($C^1_{i4} = 0$) and keeping time doesn't equal zero ($C^2_2(p) \neq 0$). The variables of timing action are adding to S^2_2 and S^2_3 simultaneously with (8-a) and (8-c).

5. RESULTS AND APPLICATION

So the defined environment and program implementation model of interpreted Petri net enables simple implementation with any high level language. Specification of blocks C and S can be declared as structures composed by required number of tables; blocks I and O – as global variables. Decision block can be declared as a system of functions operating on mentioned blocks. It was broadly depicted in [2,4].

This paper presents only methodology of initializing and performing of timing actions in simple systems with industrial standard microcontrollers.

The idea of executing timing actions relies on using interrupt system in microprocessor module. In general it must generate an interrupt with given constant frequency (e.g. 1 kHz). It is possible to realize by using a generator connected to external interrupt pin or by using an internal timer/counter (more recommended). The overflow that timer is indicated by calling right interrupt. The interrupt handling procedure is used to decrement of auxiliary registers storing information about actual state of timing actions.

At the initialization moment of timing action there is granted suitable variable with initial value $t_p = t_a * f_i$, where t_a is a required time of action, f_i is a frequency of calling interrupt *int*. The value of t_p is taken from right fields of specification block C (C^1_{i4} and $C^2_2(p)$).

During interrupt handling procedure microprocessor performs decrement operations all variables of timing actions. If any variable equals zero then it is taken operation assigned to its ending and in next step it is removed from variables list:

$$\forall (p \in S^1_2)\ S^2_2 = 0 \Rightarrow S^1_2 = S^1_2 - p \text{ and } S^2_2 = S^2_2 - S^2_2(p) \text{ and } S_1 = S_1 + p \quad (9\text{-a})$$

$$\forall (p \in S^1_3)\ S^2_3 = 0 \Rightarrow S^1_3 = S^1_3 - p \text{ and } S^2_3 = S^2_3 - S^2_3(p) \text{ and } S_1 = S_1 + p \quad (9\text{-b})$$

Contents of block O is updated according to contents of blocks S_1 oraz S^1_2:

$$\forall [p \notin (S_1 \cup S^1_2)]\ \forall [y \in C^2_1(p)]\ O = O - y \quad (10\text{-a})$$

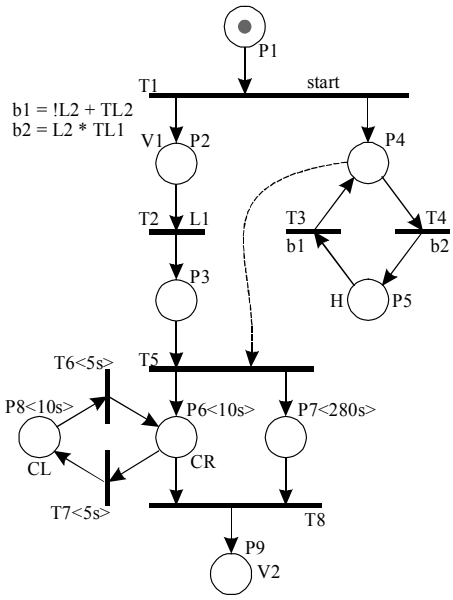
$$\forall [p \in (S_1 \cup S^1_2)]\ \forall [y \in C^2_1(p)]\ O = O + y \quad (10\text{-b})$$

A precision of timing action is dependent on a lot of factors in proposed system. The most important are: inaccuracy of generating interrupt system, delay of calling interrupt procedure, time of interrupt handling, number of active timing actions, time of updating state block S, reaction time of decision block and time of function I/O handling.

Suppose sufficient precision generating interrupt system there we can omit the deviation assigned to it. The other factors are strongly dependent among other things on used processor, its clock speed and quality of executable program code generated by given compiler.

In general the deviation of timing action is linear to number of decision rules and it's contained in given range. The range is possible to calculating in analysis process. A specific value of this deviation (in the range of course) for given rules or places is dependent on location the rules in specification block C. This is a result of sequential examining block C by decision block in permitted rules searching.

A net modeling that system and an obtained results are presented in Fig.4.



Parameter	Result
Total program memory occupation	968 B
Data memory occupation	148 B
Occupation of specification block C	100 B
Deviation of timing action (min.)	44 μs
Deviation of timing action (max.)	80 μs
Reaction time (min.)	0.6 ms
Reaction time (max.)	2,2 ms

Fig. 4. Fragment of the net modelling washer controller with the results

As a test system was adopted single-chip microcontroller PCF 80C51HB-3 with 12MHz clock. Program was compiled with μVision 2 tool (Keil Software).

6. SUMMARY

Application of the timed model Petri nets not only can effectively help on description level for reactive system strongly time depended but it can be used in system decomposition automation too (in Hardware/Software Co-Design sense [3]).
Further works and researches are to steer on studying generalized model for complicated hierarchical nets. We are going to consider not only simplifications of implementation for nets with complicated topology but also implementation of nets in distracted systems.

This work was supported by KBN grant: 7 T11C 010 20.

REFERENCES

[1] Adamski M.: *Digital systems design by formal transformation of specification*, Prep.35 Int. Wissen. Koll., TH Ilmenau, Germany, 1990, Heft 3, pp.62-65

[2] Andrzejewski G.: *Parallel controllers design with program model of interpreted Petri net*, OWD'2000, Istebna-Zaolzie, 22-25.10.2000, pp.63-68 (in Polish)

[3] Andrzejewski G.: *System decomposition in hardware/software co-design*, RUC'2001, Szczecin 7-8.05. 2001, pp.117-124 (in Polish)

[4] Andrzejewski G.: *Program model of Petri net*, accepted for publication in conference proceedings CAD DD'2001, Minsk 14-16.11.2001, Belarus

[5] Banaszak Z., Kuś J., Adamski M.: *Petri nets, Modeling, Control and Synthesis of Discrete Systems*, printed series of course lectures WSI in Zielona Góra, 1993 (in Polish)

[6] Bolton M.P.J.: *Digital systems design with programmable logic*, Addison Wesley Publ. Company, Wakingham, 1990

[7] Gajski D.D., Vahid F., Narayan S., Gong J.: *Specification and Design of Embedded Systems*, Prentice Hall, Englewood Cliffs, NJ, 1994

[8] F.Wagner: *The Virtual State Machine: Executable Control Flow Specification*, Rosa Fischer-Löw Verlag, 1994,ISBN3-929465-04-3

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

DERIVING PROGRAMS FROM PARALLEL ALGORITHMS OF LOGICAL CONTROL

Dmitrij . I. CHEREMISINOV

Institute of Engineering Cybernetics of National Academy of Sciences of Belarus,
Surganov str., 6, 220012, Minsk, Belarus, *cher@newman.bas-net.by*

Abstract. *A problem of program implementation of parallel algorithms of logical control is considered. Parallel algorithms are represented in PRALU as a set of linear algorithms that can be executed under the control of mechanism of Petri net type. The goal is to build the compiler from language PRALU. This compiler can be used as working tool to build programmable logic controller (PLC) or supervisory control application or Forth application.*

Key Words. *Parallel algorithms of logical control, Petri nets, program implementation, compilation*

1. INTRODUCTION

Although developed primary for logic control, PRALU [10] can be used to structure any application that involves sequences of operations and controlled flow of execution, and where it is important to be able to represent communication between the parts of system. This language has textual and graphical forms. The advantage of the graphical form is the simplicity and declarativeness. The textual form is used as formal tool or as intermediate representation.

A programmable logic controller (PLC) is digitally operating electronic system designed for the use in an industrial environment. It was originally developed to replace electro-magnetic relay circuits or solid-state logic blocks [7]. Relay Ladder Logic (RLL), the most common graphical language for PLC is symbolic representation of relay circuits. The core of PLC software is a program, which interprets RLL diagrams. This program continually scan RLL diagram. The time elapsed during a scan (proportional to the length of RLL) determines response time of the PLC. PLC with language PRALU have less response time.

Supervisory control applications such as set-point control, monitoring, fault detection and production optimisation are mainly operator support systems. There is a large industrial interest in applying AI (Artificial Intelligence) techniques to this type of the application [10]. These applications typically use the object-oriented knowledge base with the rule-based programming. A large problem with the rule-based system is their lack of structure. Language PRALU can be used to structure the set of rules in these applications.

Programming language Forth [8] was originally developed for small embedded control mini- and micro-computers. It has been used in a wide variety of applications, but his main area is

distributed real time control systems. Using PRALU as programming language of Forth systems extends their fighting chance by the logic control.

The common feature for all above-mentioned applications is the generic lack of capacity of previously used languages. Using PRALU in this case offers the capability to improve execution speed and maintenance of the application.

In this work the set of operations (Intermediate Language) is proposed, which is concise to represent any PRALU algorithm as a program for a single processor computer. To prove the sufficiency of the chosen set, all PRALU constructions are considered, and equal sequences of operations of intermediate language are shown. The implementation of this intermediate language is inexpensive. Any operation in this set can be implemented as a short sequence (average length is equal to 3 for Intel 86 family) of modern microprocessor commands. As a result the quality of target program is achieved.

The proposed method of program implementation of parallel algorithms of logical control can be used for deriving a program from any specification that can be mapped into the state based representation with arcs labeled with a symbol of events. In [5] a technique for converting behavior description into Petri net is described. Thereafter, as the graphical form of PRALU algorithm is the interpreted Petri net, it is possible to implement programmatically any state based representation.

To design and debug of PRALU algorithms there are programming environments on the most commonly used platforms – IBM PC, MS Windows [12] and MS DOS [2]. In [4] ActiveX component is described that can be used as control engine of supervisory control applications. This component constitutes its own interface through which the execution of PRALU algorithms can be animated.

2. MINIMAL SEMANTICS OF PRALU

In [10] for describing the rules of executions of PRALU algorithms the notion of parallel automaton is used. But this semantics of PRALU is oriented to hardware implementation. Many tasks (for example, optimal state encoding) are insignificant in program implementation. Parallel systems software design requires attention to detail beyond that normally required for hardware systems.

The process of implementation of PRALU can be viewed as the replacement of the PRALU semantics by another, more detail one. A language can have several semantics that distinguish of detail level. The minimal semantics is a formal system that states fundamental properties only, and other correct semantics must include these properties. The minimal semantics puts a problem of the validity of an implementation on the firm ground of the formalism.

In the minimal semantics a PRALU algorithm (see example 5.1) is viewed as formulae of logic calculus. This logic calculus combines the linear time temporal logic and the branching time temporal logic. The objects of this calculus are a time interval, *operation* and a time point, *event*. The operation can be active, *executing* or passive, *stopping*. The states of operation are given in a schedule of the process of execution of the PRALU algorithm. The time is the notion of the minimal semantics and must be understood as in temporal logic.

In the minimal semantics of PRALU following presumptions are supposed to be valid.

1. The operations are connected (there is no time gap between adjacent operations).
2. The execution sequence of operations is deterministic (the next time point is unique).
3. The execution of operation depends on the same set of operations (symmetry of time point).

4. The event can be either the result of operation or the reason of firing of operation.

If event is the reason of firing operation then this operation is called “wait” and is denoted as “-”. If event is a result of operation it is called “action” (is denoted as “->”). The operations of PRALU are orthogonal by the causal relation between events and operations. The formal description of minimal PRALU semantics is in [3].

The causal relation (partial order) between the operations is determined from control structure of PRALU algorithms (fig. 1). The structure of PRALU algorithms directly identifies the causal relations between operations. The full behavior includes the causal relation between events. Consequently the behavior of algorithms depends on an interaction of operation by events (information exchange). In minimal PRALU semantics the model of information exchange is the same as in CCS [6]. As a result PRALU is not implementable in minimal semantics.

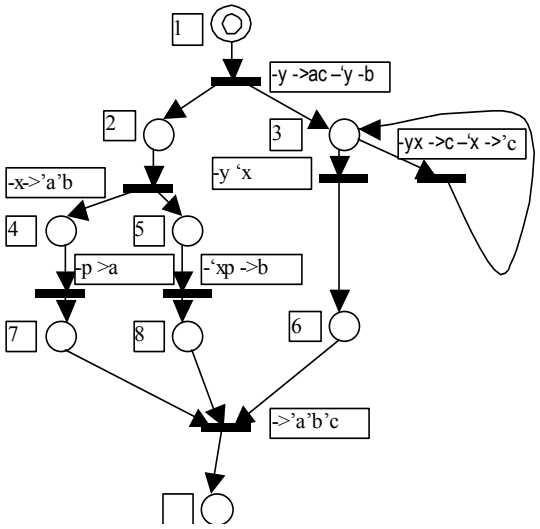


Fig. 1. Representation of PRALU algorithm as Petri net

The minimal semantics is primary a specification and it is useful because it lets us formally specify many different implementations of behaviour. Known methods of the PRALU algorithms validation [10] are true in minimal semantics, and consequently they are true in any correct elaboration of minimal semantics.

3. ELABORATIONS OF MINIMAL SEMANTICS

To be implemented semantics must predict the behaviour of PRALU algorithms unambiguously. The elaborated semantics must determine one-valued relation between the plan of operation execution and the causal structure of events. The information about the allowed orders and the times of events are captured in elaborated semantics.

Let us suppose that the wait operation starts at the same time point as event occurs that is the reason of firing this operation, and the action operation ends at the same time point as the event occurs that is the result of this operation (as it takes place in hardware implementation). In this case the relation between the plan of operation execution and the causal structure of events is determined by supposition about the duration of action in PRALU algorithms.

The hardware implementation of a PRALU algorithm has fully-specified behavior. The times of all operations are determined by the structure of a circuit. In formal terms the duration of all action operations is *constant* for every operation (it is not changed during execution of

algorithm). The hardware implementation has their communication scheduled statically in design time.

In CCS [6] the message exchange is determined with the concurrency relation. This relation fixes the set of couples of events. Each couple determines a single exchange event. But this relation is not a part of CCS model. Likewise language PRALU does not have explicit means to use this approach.

Other means is measured time, which is often used in concurrency determination. In PRALU measured time equals to the supposition about equality of times (continuance) of all action operations in the algorithm. The other form of this supposition is: the operation can not be executed in parallel to yourself.

4. INTERMEDIATE LANGUAGE

The converting Petri net into a program is considered in [9]. Our method produces a faster program. We propose the set of operations (Intermediate Language), which is concise to represent any PRALU algorithm as a program, and inexpensive implementation of this intermediate language. The proposed set of operations is the basis of algorithmic decomposition of the source algorithms. The wait and action operations are not unit actions and rather are the compositions in the proposed intermediate language.

To implement a concurrent algorithm on a single processor we must sort the plan of operation execution. This procedure is called scheduler. The scheduler of PRALU intermediate language is an object, which has properties and methods. The scheduler properties are a wait queue and a prepared queue. The scheduler methods are: thread start, thread stop, and thread interrupt. The scheduler methods are included into PRALU intermediate language.

Thread start operation includes into prepared queue the operation given in argument. Thread interrupt includes the next operation of algorithm into wait queue and takes the top operation from prepared queue and fires it. Thread stop stops current thread, takes the top operation from prepared queue and fires it. Thread stop operation has a conditional form.

The initial state of the scheduler is: prepared queue contains the first operation of the algorithm and wait queue is empty. If the scheduler has an empty prepared queue, then it copies content of wait queue into prepared queue and empties the wait queue. The realization of a queue is don't care for correct implementation of concurrency; this is the matter of the productivity of a program.

Prepared queue of a Forth system [8] is a data stack and wait queue is a call stack. In microprocessor program prepared queue is programmatically realized and wait queue is a call stack of microprocessor.

Besides this, there are operations of setting input or output buffers in PRALU intermediate language. To do information exchange, the output buffer is copied into the input buffer, when prepared queue is empty. At the same point of time the data output is executed from output buffer, and the external signals input into the input buffer. This guarantees that concurrently executing operations of the source algorithms have equal continuance.

Current marking of Petri net is represented by control vector. Each bit in this vector corresponds to the net place (the label of chain in source algorithm). The initial state of this vector is 0 in all bits. A mask vector is used to control the execution of dependent chains. The initial state of this vector is 0 in all bits. The conditional form of a thread stop operation tests this vector. If an argument is given, then this operation stops current thread, if mask vector bit is equal to 0.

Table 1.The instruction code of PRALU intermediate language.

Symbol	Function
@	Thread interrupt
%	Thread start
A	Thread stop
$	Set of output buffer
O	Test of input buffer
P	Set of control vector
R	Reset of control vector
M	Set of mask vector
C	Reset of mask vector

5. CONVERTING PRALU ALGORITHM INTO A PROGRAM

The compilation of PRALU is a substitution of PRALU operations by a sequence of intermediate language operations. The compiler PRALU uses the following patterns of substitution:
action operation (->x,…,y) => $(x,..y);
wait operation (-a,…,b) => @Aw1,..wn:O(a,…,b)Mw1,…,wn.
goto operation (->n,…,m) => Pw1,…wn:%n,…,%m
The pattern of wait operation and goto operation depends on parsing of source algorithm. This is general form.

5.1. Example.

1: -y ->ac –‘y ->b ->2.3	**1: @O(y) $(a,c) @O(‘y) $(b) %2%3 A**
2: -x ->’a’b ->4.5	**2: @O(x) $(’a,’b) %4%5 A**
3: -y’x ->6	**3: @Aw1O(y’,x)Mw1Mw2 P6 %7 A**
3: -yx ->c –‘x ->’c ->3	**4: @Aw1O(y,x) Mw1 $(c) @O(‘x) $(’c) @@ Pw1 %4%3 A**
4: -p ->a ->7	**5: @O(p) $(a) Mw2P7 %7 A**
5: -‘xp ->b ->8	**6: @O(‘x,p) $(b) Mw2P8 %7 A**
6.7.8: ->’a’b’c ->.	**7: @Aw2O(P6, P7,P8)Mw2 C(P6, P7,P8) $(a’b’c) A**

The result of compilation of algorithm on the left column is shown on the right one.

6. CONCLUSIONS

In this paper, we have discussed a systematic method to implement PRALU algorithm as a program. This method can be used for deriving a program from any specification that can be mapped into state based representation with arcs labeled with symbols of events.

In real life the proposed method has been used as a tool for designing programmable logic controllers, supervisory control applications, and Forth logic control capability.

REFERENCES

[1] Karl-Eric Arzen, “Grafcet for intelligent supervisory control application”, *Automatica*, Vol. 30, No. 10, pp. 1513 – 1525, 1994

[2] D.I.Cheremisinov, *The visualization of behavior PRALU algorithms*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, 1988 (in Russian)

[3] D.I.Cheremisinov, "The time model of PRALU algorithms", *Logic design automation of digital systems*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, pp. 46-55, 1991 (in Russian)

[4] D.I.Cheremisinov, "The engine of PRALU as ActiveX component", *Logic design*, Minsk, Institute of Engineering Cybernetics of Belarus Academy of Sciences, vol. 2, 1997 (in Russian)

[5] J.Cortadella, M.Kishinevsky, L.Lovagno, A.Yakokovlev, "Deriving Petri nets from finite transition system", *IEEE Trans. on Computers*, Vol. 47, No 8, pp. 859-882, 1998

[6] C.A.R. Hoare, *Communicating sequential processes*, Prentice-Hall, Englewood Cliffs, NJ, 1985

[7] G. Michel, *Programmable Logic Controllers*, Wiley, New York, 1994

[8] C.H. Moore, "FORTH: A new way to program a mini-computer", *Astr. and Astrophys. Suppl.*, Vol. 5, pp. 497-511, 1974

[9] R.A. Nelson, L.M. Haibt, P.T. Sheridan, "Casting Petri nets into programs", *IEEE Trans. Software Eng*., Vol. 9, No. 5, pp. 590-602, 1983

[10] A.D.Zakrevskij, *Parallel algorithms for logical control*, Minsk, Institute of Engineering Cybernetics of NAS of Belarus, 1999 (in Russian)

[11] A. Zakrevskij, B.Steinbach, "Sequent automaton - a model for logical control", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Republic of Belarus, Minsk, Sept. 5-6, pp. 211-215, 2000

[12] A.D. Zakrevskij, Y.V. Pottosin, V.I. Romanov, I.V. Vasilkova, "Experimental system of automated design of logical control devices", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Republic of Belarus, Minsk, Sept. 5-6, pp. 216-222, 2000

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

ON OPTIMAL STATE-ASSIGNMENT OF SYNCHRONOUS PARALLEL AUTOMATA*

Yury POTTOSIN

Institute of Engineering Cybernetics, National Academy of Sciences of Belarus,
Surganov str., 6, 220012, Minsk, BELARUS, *pott@newman.bas-net.by*

Abstract. *Three algorithms for assignment of partial states of synchronous parallel automata are considered. Two of them are heuristic, the third one is exact, i.e. the number of coding variables obtained by this algorithm is minimum. It is based on covering a non-parallelism graph of partial states by complete bipartite subgraphs. One of the heuristic algorithms is based on the solving the same problem but it uses an approximate method for it. The other of them is known as iterative one. The results of application of these algorithms on some pseudo-random synchronous parallel automata and the method for generating such objects are given.*

Key Words. *Parallel Automaton, State-Assignment, Problem of Covering, Generating Pseudo-Random Objects*

1. INTRODUCTION

The parallel automaton is a functional model of a discrete device and is rather convenient to represent the parallelism of interactive branches of controlled process [17]. The main distinction between a parallel automaton and a sequential one (finite state machine) is that the latter can be in only one state at any moment while the parallel automaton can be in several partial states simultaneously. A set of partial states a parallel automaton can be at simultaneously is called a *total state*. Any two partial states in which an automaton can be simultaneously are called *parallel*.

A parallel automaton is described by the set of strings of the form $\mu_i : -w_i \to v_i \to \nu_i$, where w_i and v_i are elementary conjunctions of Boolean variables that define the condition of transition and the output signals respectively, μ_i and ν_i are labels that represent the sets of partial states of the parallel automaton [17]. Every such a string should be understood as follows. If the total state of the parallel automaton contains all the partial states from μ_i and the event w_i has been realised in the input variable space, then the automaton is found to be in the total state that differs from the initial one by containing partial states from ν_i instead of those from μ_i. The values of output variables in this case are set to be such that $v_i = 1$.

*This work is supported by ISTC project B-104-98.

If $-w_i$ and $\rightarrow v_i$ are removed from the string it can be interpreted as a transition (μ_i, ν_i) in a Petri net. So, the set of such reduced strings can be considered as a Petri net being a "skeleton" of the given parallel automaton. Here we consider only those parallel automata whose skeleton is an α-net [17] that is a subclass of live and safe expanded nets of free choice which are studied in [6].

In state assignment of a parallel automaton, partial states are encoded by ternary vectors in the space of introduced internal variables that can take values 0, 1 or "–", orthogonal vectors being assigned to non-parallel states and non-orthogonal vectors to parallel states [2, 15, 16]. The orthogonality of ternary vectors means existence of a component having opposite values (0 and 1) in these vectors. It is natural to minimise the dimension of the space that results in the minimum of memory elements (flip-flops) in the circuit implementation of the automaton.

The methods to solve the state assignment problem for synchronous parallel automata are surveyed in [4]. Two heuristic algorithms are considered here. One of them is based on iterative method [3], the other reduces the minimisation of the number of memory elements to the problem of covering a non-parallelism graph of partial states by complete bipartite subgraphs [9]. To solve the problem of covering it uses a heuristic technique. The third algorithm considered here is exact, i.e. the number of coding variables (memory elements) obtained by this algorithm is minimum. It also finds a cover of a non-parallelism graph of partial states by complete bipartite subgraphs but using an exact technique [11]. These three algorithms were used to encode partial states of a number of synchronous parallel automata obtained as pseudo-random objects. The pseudo-random parallel automata with the given parameters were generated by a special computer program. The method for generating such objects is described. The results of this experiment allow one to decide about the quality of the algorithms. Similar experiments are described in [18] where another approach was investigated and the pseudo-random objects were Boolean matrices interpreted as partial states orthogonality matrices of parallel automata.

2. EXACT ALGORITHM

Below we refer to this algorithm as Algorithm A. It is based on covering a non-parallelism graph G of partial states by complete bipartite subgraphs. Let the complete bipartite subgraphs $B_1, B_2, \ldots, B_m$ form a shortest covering of G, and B_k for every $k = 1, 2, \ldots, m$ be associated with a Boolean coding variable z_k so that $z_k = 1$ for the states relative to one partite set of B_k and $z_k = 0$ for the states relative to the other. Then the values of coding variables $z_1, z_2, \ldots, z_m$ represent the solution sought for. The covering is considered here as every edge of G belongs to at least one B_k.

The first step to the solution is finding all maximum complete bipartite subgraphs of G. Three ways to do it are given in [10, 15]. Then one must obtain the shortest covering of edge set of G by those complete bipartite subgraphs. For decreasing the dimension of the covering problem the reduction rules for initial graph are used in construction the covering matrix. These rules are described in [15]. Another way to decrease the dimensions of the problem is given in [11]. It can be applied if G can be represented as $G = G_1 + G_2$, i.e. in the form of the result of join operation on two graphs [7]. The decrease is achieved if one of G_1 and G_2, say G_1, is a complete graph. This is typical for an automaton whose "skeleton" is α-net. Then the covering problem is solved only for G_2. When the cover is obtained the codes of the states which are associated with the vertices of G_2 are chosen as shown above. These codes form a Boolean space of coding variables. If they don't occupy all the space, the codes of the states associated with the vertices of G_1 are placed in the rest. The space is extended, if necessary, to the size enough for placing all state codes. In this case a non-redundant cover must be found rather than

a shortest one. Algorithm A realises this method. The idea of using decomposition as the means to reduce the dimension of the task is rather fruitful. For example, one can see in [8] another case of using decomposition to decrease the dimension of the problem of our field.

3. HEURISTIC ALGORITHMS

3.1. Algorithm B

The NP-hardness of covering problem [5] doesn't allow it always to be solved in acceptable time. Therefore the heuristic algorithm is proposed in [9] that obtains in many cases the shortest cover. We call it Algorithm B. It consists of two stages. At the first stage the sequence of graphs $G_2, G_3, \ldots, G_n = G$ is considered, where G is the non-parallelism graph of the given automaton with $V = \{v_1, v_2, \ldots, v_n\}$ as the set of vertices, and G_i is the subgraph of G induced by the set of vertices $V_i = \{v_1, v_2, \ldots, v_i\}$. Having the cover of G_i the transition from it to the cover of G_{i+1} is carried out. At the second stage the obtained cover is improved (if possible). This improvement consists in removing some complete bipartite subgraph from the covering and in the attempt of reconstruction the cover by adding edges to remained subgraphs. This procedure repeats for all elements of the cover. The complete bipartite subgraphs are obtained concurrently with constructing the cover.

3.2. Algorithm C

The other heuristic algorithm is based on the iterative method suggested in [3]. We refer to this algorithm as Algorithm C. The iterative method assumes the definition of parallelism relation and an initial coding matrix for partial states (the initial matrix may be empty). The matrix is extended in the process of coding by introducing additional coding variables that makes it possible to separate non-parallel partial states in certain pairs. To separate two states means to put opposite values (0 and 1) to some coding variable in the codes of these states. The method consists in iterative executions of two procedures: introducing a new coding variable and defining its values in codes of non-separated yet non-parallel partial states. These procedures are being executed until all non-parallel states have been separated. Minimising the number of introduced coding variables the method minimises the Hamming distance between codes of states related by transitions as well. The aim of this is the minimisation of the number of switchings of RS type flip-flops in circuit realisation of a parallel automaton.

Introducing a new coding variable is accompanied with separating the maximal number of non-separated yet non-parallel partial states by this variable. For this purpose at each step of the procedure of defining the values of the due variable, a state is chosen to encode by this variable. This state should be separated from the maximal number of states encoded already by this variable. The number of states that are not separated from the chosen one and have been encoded by this variable must be maximum. A new coding variable is introduced if the inner variables having been introduced don't separate all non-parallel partial states from each other.

4. GENERATING PARALLEL AUTOMATA

Any string of the form $\mu_i : -w_i \rightarrow v_i \rightarrow v_i$ in automaton specification we call a transition, and a set of transitions with the same μ_i a sentence. The algorithm for generating parallel automata is described in detail in [12] where a parallel automaton is constructed as a system of three pseudo-random objects. They are the skeleton of the automaton that is an α-net specified in the form of a sequence of pairs (μ_i, v_i), the ternary matrix $\boldsymbol{X}$ representing conjunctions w_i, and the

ternary matrix $\boldsymbol{Y}$ representing conjunctions v_i. In our task the α-net is enough, therefore we shouldn't describe the way of generating $\boldsymbol{X}$ and $\boldsymbol{Y}$ here.

The given beforehand parameters of every pseudo-random α-net generated by a special computer program are the number of places (partial states of the automaton) p, the number of transitions t, and the number of sentences s.

Generating pseudo-random parallel automata as systems of three mentioned above objects with given beforehand parameters would not be difficult if no correctness demands exist without which there is no sense to execute algorithms intended for such automata. Proceeding from the correctness properties of a parallel control algorithm that are named in [16], let us consider the following properties of a parallel automaton, that guarantee its correctness in our case. It must be irredundant (there is no transition that can be never done), recoverable (it can return to the initial total state from any other one), and self-coordinated (any transition cannot be started before it ceases).

Irredundancy, recoverability, and self-co-ordination of a parallel automaton corresponds to liveness and safety of the related α-net [16]. The characteristic properties of α-net are the initial marking of it consisting of one element, {1}, and the sets of input places of two different transitions coinciding or disjoining. In the Petri net theory the reduction methods for checking liveness and safety are well known [1], where the initial net is transformed according to certain rules with preserving these properties. The transformations reduce the dimension of a given net and so facilitate the checking liveness and safety of the net.

To check liveness and safety of α-nets the application of two rules is sufficient [16]. The first rule consists in deleting loops i.e. the transitions where $\mu_i = \nu_i$. The second one is as follows. Let a set of places π not containing place 1 be such that for every transition (μ_i, ν_i), $\pi \cap \mu_i \neq \varnothing$ implies $\pi = \mu_i$ and $\pi \cap \nu_i = \varnothing$, and $\pi \cap \nu_i \neq \varnothing$ implies $\pi \subseteq \nu_i$. Besides, there exists at least one transition with $\pi \cap \nu_i \neq \varnothing$. Then all transitions (μ_j, ν_j) with $\pi = \mu_i$ are removed and every transition (μ_k, ν_k) with $\pi \subseteq \nu_k$ is substituted by the set of transitions that are obtained from (μ_k, ν_k) by replacing π by sets ν_j from those transitions (μ_j, ν_j) where $\pi = \mu_j$. A live and safe α-net is proved in [14] to be completely reducible, i.e. the application of these rules leads to the net that consists of the only transition (1,1). This implies the way of generating live and safe α-nets that consists in transformations that are inverse to the above.

5. EXPERIMENTAL RESULTS

Algorithms A, B, and C are realised in computer programs and the corresponding modules are included as components into ISAPR that is a research CAD system [13]. The program for generating pseudo-random parallel automata is included into ISAPR as well. This program was used to generate several parallel automata. The results of partial state assignment are shown in Table 1. One of the automata whose partial states were encoded, RAZ, was not generated by the program mentioned above. It was obtained from a real control algorithm.

As it was noted, only the parameters of α-net, i.e. the number of places p, the number of transitions t, and the number of sentences s were considered. Besides those, the number of maximum complete bipartite subgraphs in the graph G of non-parallelism of partial states of the given automaton may be of interest. Algorithm A uses the method that decomposes graph G into two subgraphs, G_1 and G_2, G_1, being complete. So, the maximum complete bipartite subgraphs were found in G_2. The calculations were performed on a computer of AT type with the 386 processor.

6. CONCLUSION

The technique of investigation of algorithms for state assignment of parallel automata is described in this paper. The experimental data show that Algorithms B and C are quite competitive to each other, although the speed of Algorithm C is higher than that of Algorithm B. Algorithm A is intended to be applied for automata of small dimension. It can be used as a standard algorithm and helps one to appreciate the quality of solutions obtained by heuristic algorithms.

Table 1. Experimental results: p, t, and s are parameters of α-nets, b is the number of maximum complete bipartite subgraphs of G_2.

Name	*p*	*t*	*s*	*b*	*Algorithm A*		*Algorithm B*		*Algorithm C*	
					Code length	*Run time*	*Code length*	*Run time*	*Code length*	*Run time*
AP2	**20**	**18**	**18**	**75**	**6**	**13 min. 28 sec.**	**7**	**6 sec.**	**7**	**3 sec.**
APR1	**20**	**21**	**19**	**8**	**5**	**8 sec.**	**6**	**7 sec.**	**5**	**3 sec.**
APR2	**20**	**21**	**19**	**4**	**5**	**5 sec.**	**6**	**8 sec.**	**5**	**3 sec.**
APR3	**20**	**21**	**15**	**7**	**4**	**6 sec.**	**5**	**3 sec.**	**6**	**3 sec.**
APR6	**20**	**28**	**15**	**43**	**5**	**2 min. 23 sec.**	**6**	**8 sec.**	**6**	**3 sec.**
APR7	**20**	**30**	**15**	**55**	**5**	**49 sec.**	**6**	**8 sec.**	**6**	**3 sec.**
APR8	**20**	**15**	**15**	**49**	**5**	**1 min. 28 sec.**	**5**	**5 sec.**	**5**	**3 sec.**
RAZ	**20**	**21**	**19**	**1033**	**9**	**3 h. 46 m. 22 s.**	**9**	**8 sec.**	**10**	**4 sec.**

REFERENCES

[1] S.M.Achasova, O.L.Bandman, *Correctness of Concurrent Computing Processes*, Nauka, Siberian Division, Novosibirsk, 1990 (in Russian)

[2] M.Adamski, M.Wegrzyn, "Field programmable implementation of current state machine", *Proc. of the Third International Conference on Computer-Aided Design of Discrete Devices (CAD DD'99), Vol.1*, Institute of Engineering Cybernetics, NAS Belarus, Minsk, pp.4-12, 1999

[3] L.D.Cheremisinova, "State assignment for parallel synchronous automata", *Izvestia AN BSSR, Ser. fisiko-tehnicheskih nauk, No.1*, pp. 86-91, 1987 (in Russian)

[4] L.D.Cheremisinova, Yu.V.Pottosin, "Assignment of partial states of a parallel synchronous automaton, *Proceedings of the International Conference on Computer-Aided Design of Discrete Devices CAD DD'95"*. Wydawnictwo Uczelniane Politechniki Szczecinskiej, Minsk-Szczecin, pp. 85-88, 1995

[5] M.R.Garey, D.S.Johnson, *Computers and Intractability: A Guide to the Theory on NP-completeness*, W.M.Freeman & Company, San Francisco, Ca., 1979

[6] M.T.Hack “Analysis of production schemata by Petri nets”, *Project MAC TR-94*, Cambridge, 1972

[7] F.Harary, *Graph Theory*, Addison-Wesley Publishing Company, Reading, Mass., 1969

[8] A.Karatkevich, “Hierarchical decomposition of safe Petri nets”, *Proc. of the Third International Conference on Computer-Aided Design of Discrete Devices (CAD DD’99), Vol.1*, Institute of Engineering Cybernetics, NAS Belarus, Minsk, pp. 34-39, 1999

[9] Yu.V.Pottosin, "Covering a graph by complete bipartite subgraphs", *Design of Discrete Systems*, Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk, pp. 72-84, 1989 (in Russian)

[10] Yu.V.Pottosin, “Finding maximum complete bipartite subgraphs in a graph” *Automatization of Logical Design of Discrete Systems*, Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk, pp. 19-27, 1991 (in Russian)

[11] Yu.V.Pottosin, "A method for encoding the partial states of a parallel automaton with minimal-length codes", *Automatic Control and Computer Sciences*, *N 6*, Allerton Press, Inc., New York, pp. 45-50, 1995

[12] Yu.V.Pottosin, “Generating parallel automata”, *Methods and Algorithms for Logical Design*, Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk, pp. 132-142, 1995 (in Russian)

[13] N.R.Toropov, *Research CAD System for Discrete Control Devices: Materials on Software for Computers*, Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk, 1994 (in Russian)

[14] V.V.Tropashko, “Proof of the conjecture of complete reducibility of α-nets”, *Design of Logical Control Systems*, Institute of Engineering Cybernetics of Academy of Sciences of BSSR, Minsk, pp. 13-21, 1986 (in Russian)

[15] A.D.Zakrevskij, “Optimization of matrix of partial state assignment for parallel automata”, *Formalization and Automatization of Logic Design*, Institute of Engineering Cybernetics of Academy of Sciences of Belarus, Minsk, pp. 63-70, 1993 (in Russian)

[16] A.D.Zakrevskij, "Parallel logical control algorithms: verification and hardware implementation", *Computer Science Journal of Moldova, Vol.4, No.1*, pp. 3-19, 1996

[17] A.D.Zakrevskij, *Parallel Algorithms for Logical Control.* Institute of Engineering Cybernetics of National Academy of Sciences of Belarus, Minsk, 1999 (in Russian)

[18] A.Zakrevskij, I.Vasilkova, "A quick algorithm for state assignment in parallel automata", *Proc. of the Third International Conference on Computer-Aided Design of Discrete Devices (CAD DD’99), Vol.1*, Institute of Engineering Cybernetics, NAS Belarus, Minsk, pp.40-44, 1999

SESSION II:
SYSTEM ENGINEERING

The International Workshop
On Discrete-Event System Design, DESDes'01,
June 27-29; Przytok near Zielona Gora, Poland

SIMULATION AND TARGETING USING OORT*

Sérgio LOPES, João MONTEIRO

Industrial Electronics Departament, Engineering School, University of Minho,
Campus de Azurém, 4800-058 Guimarães, PORTUGAL,
<*sergio.lopes, joao.monteiro*>*@dei.uminho.pt*

Abstract. *The development of embedded systems requires both tools and methods which help the designer to deal with the higher complexity and tougher constrains due to the different hardware support, the often distributed topology and time requirements. Moreover, the last steps of each version of the design, namely, simulation and targeting, should be made easier and faster to execute, in order to facilitate the correction of problems and the issue of a new more correct version, thus increasing the frequency of an iterative engineering process. This has a major impact on the overall costs and final product quality, and therefore the use of a CASE tool that supports the chosen methodology becomes an obvious advantage. We have applied the Object-Oriented Real-Time Techniques (OORT) method, which is oriented towards the specification of distributed real-time systems, to the implementation of the Multiple Lift System (MLS) case study. This paper describes briefly the method and presents our experience in the simulation and targeting of developed system, namely the difficulties we had and the success we have achieved.*

Keywords*: Distributed Systems Specification, Software Engineering, Discrete-Event Systems Control, Simulation, Targeting.*

1. INTRODUCTION

Real-time systems are very complex because they are often distributed, run in different platforms, have temporal constraints, etc. The development of these systems demand high quality and increasing economic constraints, therefore it is necessary to minimise their errors and its maintenance costs, and deliver them in short deadlines.

To achieve these goals it is necessary to verify a few conditions: decrease the complexity of the systems through hierarchical and graphical modelling for high flexibility in the maintenance; protect the investments with the application of international standards in the development; to apply early verification and validation techniques to reduce the errors; and, reduce the delivery times by automating code generation and increasing the level of reusability. Finally, its necessary to have a tool that provides these conditions. The present work was developed with the *Object*GEODE[i] toolset, that supports the OORT method.

The OORT method [14] is organised according to the diagram of figure 1 and applies the Unified Modelling Language (UML), Message Sequence Chart (MSC) and Specification and Description Language (SDL). The UML language is a de jure standard (see [5] for details) and it is defined in [10]. The MSC was defined [8] as complement to SDL, both international standards by ITU-T. [13] provides an introduction to MSC. SDL is defined by [6], [7] and [9],

* This work was supported by the Fundação para a Ciência e a Tecnologia PRAXIS XXI program of the Ministério da Ciência e Tecnologia of the Portuguese Government.

however [11] is a more comprehensive reference, while [4] is a handy summary of the language. The use of both languages together is guided by [10].
In this work we have applied the OORT method to the modelling of a case study – the Multiple Lift System (MLS). A description of a MLS architecture using UML is presented in [2]. The analysis model uses UML to model the system's environment, and MSC to specify the behaviour of the system. The system's architecture is defined in SDL. The detailed design uses SDL for the concurrent objects specification and UML for the passive components description. The MSC language supports the test design activity. For the simulation of the designed system we used the *Object*GEODE simulator, and finally we use the C Code Generator for the targeting. Each of this steps in the systems engineering process is described in the following sections.

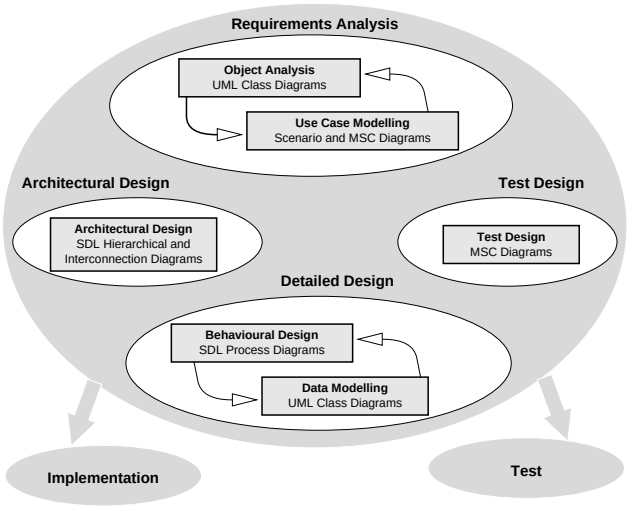


Figure 1. The OORT method.

2. REQUIREMENTS ANALYSIS

In the requirements analysis phase, the system environment is modelled and the user requirements are described. The analyst must concentrate on **what** the system should do. The environment where the system will operate is described by means of UML class diagrams – **object modelling**. The functional behaviour of the system is specified by MSCs organised in a hierarchy of scenarios - **use case modelling**.
The system is viewed from the exterior as a black box with which external entities (system actors) interact. Both the object model and use case model must be independent of the solutions chosen to implement the system.

2.1. Object Analysis

In the description of the system environment the class diagrams are used to express the application and services domains. This is done by identifying the relevant entities of the application domain (physical and logical), their attributes, and the relationships between them. It is also necessary, for the sake of simplicity and expressiveness, to group entities and their relationships in different modules that reflect different perspectives of the system, as is supported by [16]. Generally speaking, there is one module for each of the actors that interact with the system, one for some basic system composition and other to express certain environment relationships.

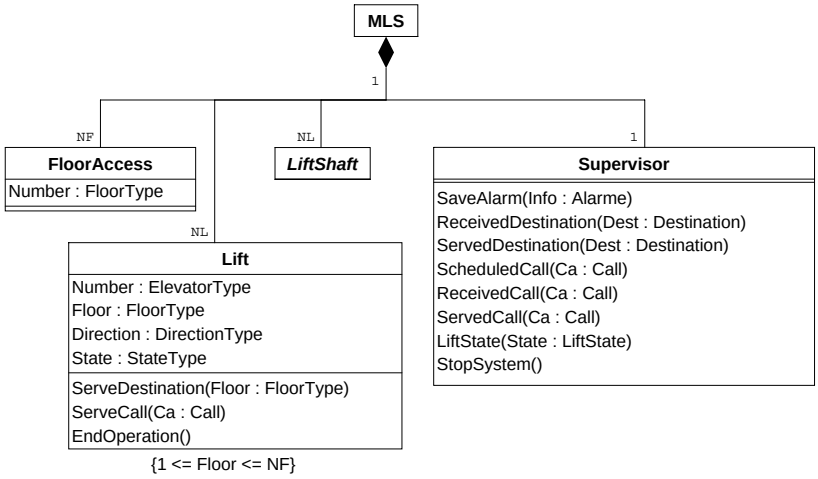


Figure 2. System Architecture UML Class Diagram.

The generic system architecture is modelled in figure 2. In order to keep simple modules, each of the component classes are refined in different diagrams.

2.2. Use Case Modelling

The use case model is composed by the scenario hierarchy and MSC diagrams. The scenario hierarchy should contain all the different expected scenarios of interaction between the system and its environment. The goal it is to model the functional and dynamic requirements of the system. First, the main scenarios are identified, and then they are individually refined in subsequent more detailed scenarios until the terminal scenarios can be easily described by a chronological sequence of interactions between the system and its environment.

One problem of this approach is the scenario explosion. To deal with that difficulty we apply composition operators that combine hierarchically the several scenarios. Nevertheless, the problem is only diminished but not completely solved. It is still necessary to choose well the scenarios, namely to chose those which are the most representative of the system behaviour.

The system operation is divided in phases that are organised by composition operators, and each phase is a branch in the scenario hierarchy. Figure 3 shows the `Trip` phase scenario hierarchy, in which we have a `Floor Crossing` terminal scenario which is illustrated in figure 4.

A constant concern must be the coherence between the use case and the object models. See [14] for more details.

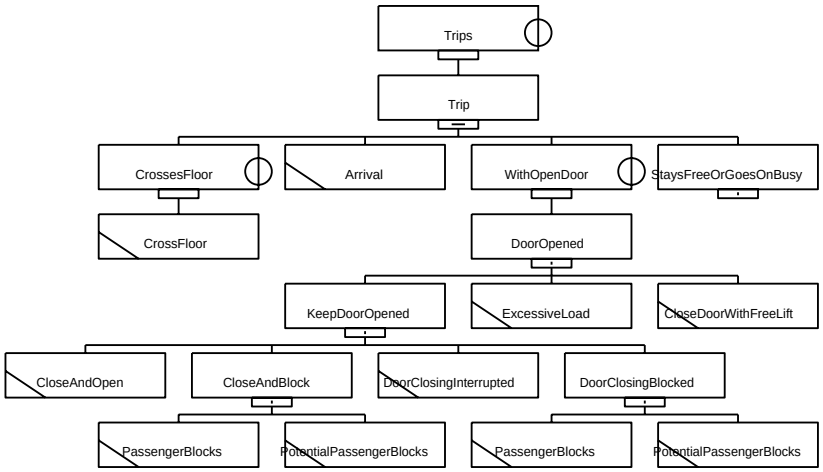


Figure 3. Scenario Hierarchy for the Trip Sub Scenario.

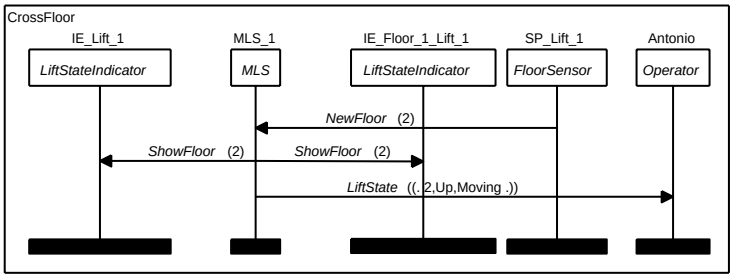


Figure 4. Abstract MSC for the Floor Crossing Scenario.

3. ARCHITECTURAL DESIGN

In this phase the system designers specify a logical architecture of the system (as opposed to the physical architecture). The SDL language covers all aspects of the architecture design.
The system is composed of **concurrent objects** (those which have an execution thread) **and passive objects** (those which implement a set of functions invoked by concurrent objects). In the architecture design phase, the concurrent objects that compose the system are identified and organised hierarchically. This is accomplished by a combination of refinement and composition. The refinement is a top-down process in which higher level objects are divided in smaller and more detailed objects, always trying to keep a good modularity. The composition is a bottom-up process in which designers try to group objects in such a way that favours reutilization and that maintains a good encapsulation of the architectural objects. Figure 5 illustrates the SDL object's hierarchy of the MLS.
In the architectural design, the real characteristics of the environment where the system will operate should be considered, as well as the efficiency aspects. On the other hand, the SDL model should be independent of the real object distribution on the final platform.

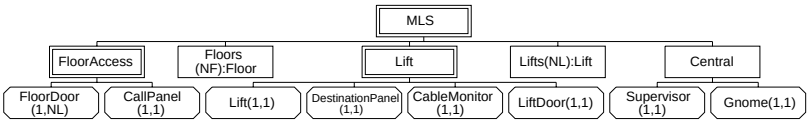


Figure 5. MLS SDL Hierarchy Diagram.

At the first level, the system actors are considered through their interfaces, and modelled as channels between the system top level objects and the outside world. Figure 6 shows the top level of the MLS architecture.

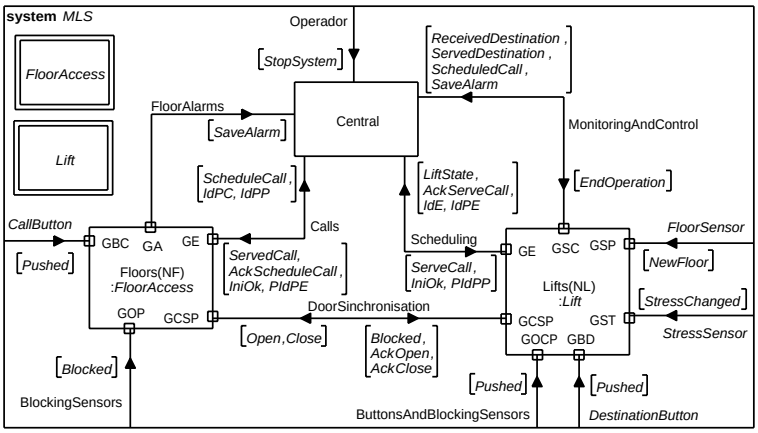


Figure 6. SDL Interconnection Diagram of the Top Level of the MLS Hierarchy.

Some passive objects are also defined, such as signals with complex arguments, Abstract Data Types (ADTs) associated with internal signal processing, and operators to implement the I/O communication with the outside world (instead of signals).
The use of SDL assures the portability of the system architecture, since the communication service is independent of the real object distribution, the communication channels are dynamic, and the objects can be parameterised.

4. DETAILED DESIGN

The description of concurrent and passive objects that constitute the system architecture is done in the detailed design phase. In other words, it is described **how** the system implements the expected services, and it should be independent of the final platform where the system will run.

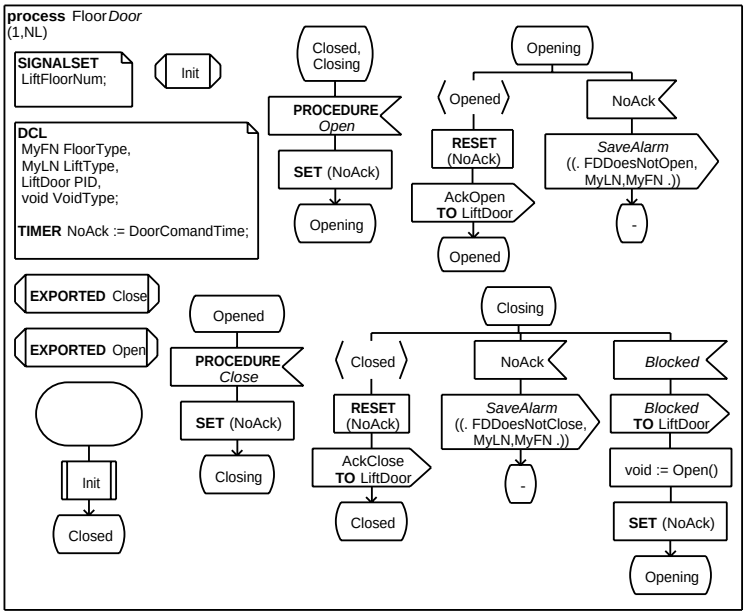


Figure 7. SDL Process Diagram of the Floor Door Process.

4.1. Concurrent Objects Design

The concurrent objects are the terminal objects of the SDL hierarchy. They are SDL processes and are a kind of Finite State Machine (FSM), with its states and state transitions, called **process diagrams**. The process diagrams are built by analysing the input signals of each process defined in the architecture model and how the answer to those signals depends on the previous states. The SDL has a set of mechanisms to describe the transitions that allow a complete specification of the process behaviour. In the figure 7 is shown a process diagram.
The reuse of external concurrent objects is supported by the SDL encapsulation and inheritance mechanisms.

4.2. Passive Objects Design

Some passive objects are identified during the analysis phase. Generally they model data used or produced by the system, and they are included in the detailed design to provide services to concurrent objects. There are also passive objects that result from design options, such as data management, user interface or equipment interface and inclusion of other design techniques.

Although the SDL ADTs provide a way to define passive objects they are better defined by UML classes. So the ADTs from the SDL detailed design model are translated to UML classes and organised in detailed design class diagrams.
The reuse of external passive objects is facilitated by the UML encapsulation and inheritance mechanisms. These characteristics of UML, and also SDL, allow for the use of other techniques of design in certain systems. For instance, in the case of embedded systems, it can be useful to use VHDL to design some physical parts.

4.3. Portability

The multi-tasking, the communication and the time management are implemented by the SDL virtual machine, and therefore are independent of the physical platform and RTOS on which the system will run. The system maintenance is kept at the SDL specification level, thus it is easier to correct and change the system. However, the portability depends largely on the language chosen to implement the passive objects.

5. TEST DESIGN

In this phase, the communication between all the elements of the system architecture is specified by applying detailed MSCs to describe the sequences of messages exchanged between them, in all the scenarios that compose the use case model. This is done by refining the abstract MSC of each terminal scenario from the analysis according to the SDL architecture model. Consequently, the test design activity can be done in parallel with the architecture design and serve as requirements to the detailed design phase.
In the intermediate architecture levels, the detailed MSCs represent integration tests between the concurrent objects. The last step of refinement correspond to unit tests that describe the behaviour of processes (the terminal SDL architecture level).
The process level detailed MSCs can be further enriched by including in each process behaviour detailed graphical elements such as states, procedures and timers.
Figure 8 shows the integration test corresponding to figure 4 abstract MSC, and figure 9 represents the respective unit test for one of the blocks.
This phase can be a very long and resource consuming, thus substantially increasing the system development cost. However, it is decisive to the system success.
The use case model reflects the user perspective of the system. The test design should be spread to cover aspects related to the architecture, such as performance, robustness, security, flexibility, etc.

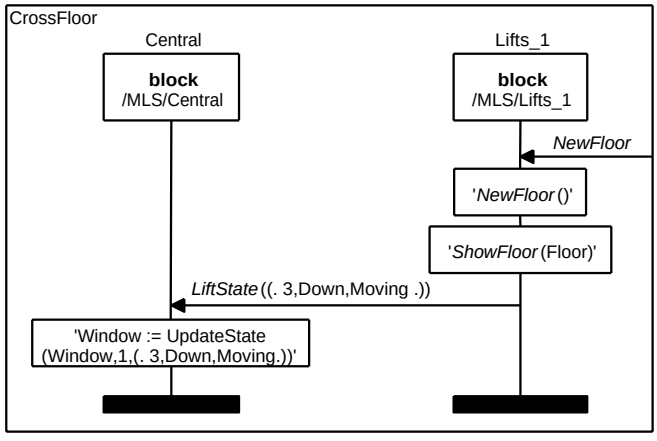


Figure 8 –Detailed MSC with Floor Arrival Integration Test.

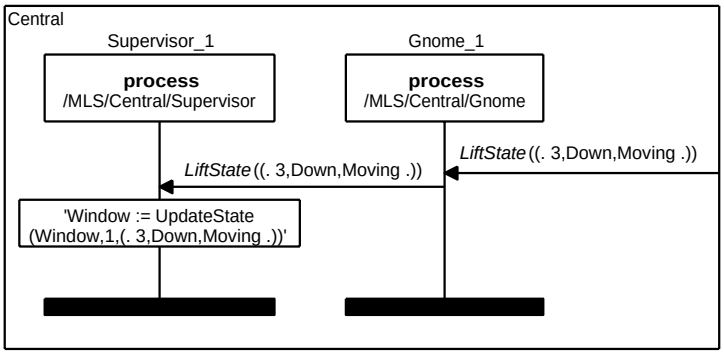


Figure 9. Detailed MSC with Floor Arrival Unit Test of Block Piso.

6. SIMULATION

With the *Object*GEODE simulator one can simulate SDL models, comparing them with MSCs that state the expected functionalities and error situations, and generating MSCs of the actual system behaviour. The execution of an SDL model is a sequence of steps, firing transitions from state to state.

The simulator has three operation modes: **interactive** - in which the user acts as the system environment and monitors the system's internal behaviour; **random** - the simulator executes the SDL model picking randomly one of the transitions possible to fire; **exhaustive** - the simulator automatically executes the model and explores all the possible system states.

The interactive model can be used to do the first tests to verify in a detailed way some important situations to correct and complete the overall behaviour of the system, i.e., to verify that the system really works. This mode was very useful to detect some flaws in ADTs whose operators were specified in textual SDL. For instance, the ADTs responsible for the calls dispatch, which are heavy computational, needed a little touch in the algorithms. As the SDL simulator has a granularity of one transition, we can not go step by step inside the operations executed during the transition from one state to another. But we can see that one transition does not follow the expected path or that some variable does not have the value it was supposed to have after that transition. Therefore, with that information, we can inspect more closely the operators called by the transition and verify their code correctness, but most of the times it is immediately evident which operator is wrong. This mode is specially suited for rapid prototyping.

Obviously, this is not an adequate way to simulate a large number of cases. After a certain level of confidence in the overall application behaviour is achieved, we can test for a larger number of scenarios, in order to detect dynamical errors such as deadlocks, dead code, unexpected signals, signals without receiver, overflows, *etc*. To do this we simulate in the random mode, to verify if the system is being correctly built. This mode allows to do the system verification.

However, we can do that with the exhaustive simulation, in fact we can do everything with the exhaustive simulation, but it would not be efficient? The exhaustive simulation requires a lot of computer resources and takes a lot of time. It's not something you can do everyday. The introduction of this mode between the interactive and the exhaustive is a very good solution because we can save a lot of time. We can detect most of the errors in a much quicker way.

The exhaustive simulation allows to make the validation of the system, *i.e.*, to verify if the system meets the requirements. We can check if it implements the expected services, by detecting interactions that do not follow some defined properties, or interaction sequences that

are not expected.

7. TARGETING

The implementation of the designed system is made easier by the code generator of the *Object*GEODE, which automatically translates the SDL specification to C code. The generated code is independent of the target platform in which the system will run. The SDL semantics, including the communication, process instance scheduling, time management and shared variables, is implemented by a dynamic library. That library is also responsible for the integration with the executing environment, namely the RTOS. By default the communications are implemented through TCP/IP sockets.

In order to generate the application, it is necessary to describe the target platform in which the system will be executed, This is done by means of a mapping between the architecture of the SDL specified system and the architecture of the C code implementation.

The SDL architecture consist in a logical architecture of structural objects (system, blocks, processes, etc...) in which the lower objects (the processes) implement the behaviour of the described system. The physical implementation of that description consist in a hierarchy of the following objects: **node** - all the software executed by one processing unit with multi-tasking OS; **task** - unit of parallelism of the OS. One task can correspond to one of the SDL objects: system - Task/System (TS) mapping; Block - Task/Block (TB) mapping; process - Task/Process (TP) mapping; Process Instance - Task/Instance (TI) mapping.

In the TI mapping the complete application is managed by the target OS. In the TP mapping, the OS is in charge of the interaction between processes, whilst the management of the several process instances inside the task is done by the SDL virtual machine of *Object*GEODE. In the case of TB mapping, the OS manages the communication between blocks, while the management of the SDL objects inside each block is done by the SDL virtual machine. Obviously, the TS mapping is the only one possible for operative systems without multi-tasking and in each node the SDL virtual machine manages all the application. For the MLS, the TP mapping was chosen.

After the code is generated, the user only has to supply the missing code for the parts that interact or depend directly on the platform. Therefore, the user can choose the language which best suits his needs and then link that code with the generated code. The ADT operators that do not interact with external devices, can be coded algorithmically in SDL, and thus the respective C code will be generated. By default, to each ADT operator corresponds one C function which interface is automatically generated. The figure 10 illustrates the application generation scheme in a very simplistic manner.

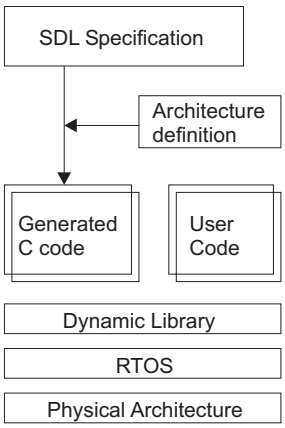


Figura 10. Simplified strategy for the application generation.

8. CONCLUSION

The simulation is a very important phase of the system's development because it allows the costs reduction by decreasing the number of missed versions, *i.e.*, it helps the designers to get closer to the "right at first time". The three simulation modes can be used by the order presented, *i.e.*, in the order of the increasing level of system correctness.
The code generated by the *Object*GEODE toolset is optimised for the target platform by means of a mapping between the SDL architecture and the physical architecture defined by the user. Any change in the application target it only requires a change in the mapping, so the system specification and its logical architecture remain the same. The user only has to supply the code which is target dependent.
Because SDL is a formal language it can be used to define rules in the partition and synthesis of a system specification into hardware and software, as is the case of a methodology presented in [1]. Furthermore, the implementation can be automatic, thus limiting the manual coding to the non real-time operations. The generated application is scalable, because the logical architecture is independent of the physical architecture. The mapping between objects and hardware is define in the implementation phase only.
The SDL specification, being a model expressed in a formal language, permits the automatic simulation of the system [3], to make early validations, and the automatic code generation. The simulation of a formal language is trustable since it is defined by a clear set of mathematical rules. Therefore, comparing to the non formalised development, the applications are better in terms of efficiency, less errors, flexibility and easy of maintenance.

REFERENCES

[1] J.M. Daveau, G.F. Marchioro, T. Ben-Ismail and A.A. Jerraya, “Cosmos: An SDL Based Hardware/Software Codesign Environment”, in: *Hardware/Software Co-design and Co-Verification*, eds. Bergé, J-M, Levia, O. and Rouillard, J., Kluwer Academic Publishers, 1997, 59-87.

[2] B.P. Douglass, Real-Time UML: Developing Efficient Objects for Embedded Systems (Addison-Wesley, 1998).

[3] V. Encontre, How to Use Modeling to Implement Verifiable, Scalable, and Efficient Real-Time Application Programs, *Real-Time Engineering*, Fall 1997.

[4] O. Faergemand and A. Olsen, Introduction to SDL-92, *Computer Networks and ISDN Systems*, 26(9), 1994.

[5] Cris Kobryn, UML 2001: A Standardization Odyssey, *Communications of the ACM*, 42 (10), 1991, 29-37.

[6] ITU-T Recommendation Z.100, Specification and Description Languge (SDL), March 1993.

[7] ITU-T Recommendation Z.100 Appendix 1, SDL Methodology Guidelines, March 1994.

[8] ITU-T Recommendation Z.120, Message Sequence Chart (MSC), ITU, October 1996.

[9] ITU-T Recommendation Z.100 Addendum 1, Specification and Description Languge (SDL) Addendum 1, October 1996.

[10] ITU-T Recommendation Z.100 Supplement 1, SDL + Methodology: Use of MSC and SDL (with ASN.1), May 1997.

[11] A. Olsen, O. Faergemand, B. Moller-Pedersen, R. Reed, J.R.W. Smith, Systems Engineering Using SDL-92 (North Holland, 1994).

[12] OMG Unified Modeling Language Specification, Version 1.3, June 1999.

[13] E. Rudolph, P. Graubmann, J. Grabowski, Tutorial on Message Sequence Charts, *Computer Networks and ISDN Systems*, 28(12), 1996.

[14] Verilog, ObjectGEODE Method Guidelines (Verilog SA, 1996).

[15] Verilog, ObjectGEODE SDL Simulator Reference Manual (Verilog SA, 1996).

[16] E. Yourdon, Object-Oriented Systems Design: An Integrated Aproach (Prentice Hall, 1994).

[i] *Object*GEODE is a registered trademark by Verilog.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

OPTIMIZING SW/HW ARCHITECTURE FOR PARALLEL EMBEDDED SYSTEMS - A CASE STUDY

Vaclav DVORAK

Dept. of Computer Science and Engineering, University of Technology Brno,
Bozetechova 2, 612 66 Brno, CZECH REPUBLIC, *dvorak@dcse.fee.vutbr.cz*

Abstract. *The paper addresses the issue of prototyping hw/sw architecture of application-specific multi-processor systems (recently on a chip). Performance prediction of these systems, either bus-based SMPs or message-passing networks of DSPs, is undertaken using a CSP-based tool Transim. Variations in processor count, clock rate, link speed, bus bandwidth, cache line, as well as in partitioning and mapping the resulting sw components to processors can be easily accounted for. The technique is demonstrated on parallel FFT on 2 to 8 processors.*

Key Words. *Parallel Embedded Systems, Multiprocessor Simulation, Hardware Description Language*

1. INTRODUCTION

The design of mixed hw/sw systems for embedded applications has been an active research area in recent years. Hw/sw co-synthesis and co-simulation have been mainly restricted to a single processor and programmable arrays attached to it, that were placed incidentally on a single chip (SoC). A new kind of systems, application-specific multi-processor SoC, is emerging with frequent applications in small-scale parallel systems for high-performance control, data acquisition, and analysis, image processing, wireless, networking processors, and game computers. Typically several DSPs and/or microcontrollers are interconnected with an on-chip communication network and may or may not use an operating system. In this paper, we want to concentrate on performance prediction of various configurations of hw and sw, since performance guarantees must be complied with before anything else can be decided. Other difficult problems such as system validation at the functional level and at the cycle-accurate level, software and RTOS synthesis, task scheduling and allocation, overall system testing, etc., are not considered.

In this paper we will study only application-specific multiprocessors and modeling their performance. As a suitable application for performance comparison, we have selected

a parallel FFT (1024) benchmark (1024 points, one dimension), in real-time environment, with the goal of maximizing the number of such FFTs per second.

2. ARCHITECTURES OF PARALLEL EMBEDDED SYSTEMS AND CMP

The performance race between a single large processor on a chip and a single-chip multiprocessor (CMP) is not decided yet. Applications such as multimedia point to CMP with multithreaded processors [1] for the best possible performance. The choice between application-specific (systolic) architectures or processors on one hand and CMP on the other is yet more difficult. CMP architectures may also take several forms such as:

- a bus-based SMP with coherent caches similar to Pentium Pro quad pack, with an atomic bus or a split-transaction bus;
- a SMP with a crossbar located between processors and a shared first-level cache which in turn connects to a shared main memory;
- a distributed memory architecture with a direct interconnection network (e.g. a hypercube) or an indirect one (the multistage crossbar).

As the number of processors on the chip will be typically lower than 10, at least in a near future, we do not have to worry about scalability of these architectures. Therefore the bus interconnection will not be seen as too restrictive in this context.

Some more scalable architectures such the SMP with processors and memory modules interconnected via a multistage interconnection network (the so called „dancehall“ organization) or a hw-supported distributed shared memory will not be considered as candidates for small-scale parallel embedded systems or SoCs.

Let us note, that the choice of architecture can often be also dictated by a particular application to be implemented in parallel. E.g. broadcasting data to processors, if not hidden by computation, may require a bus for speed, but on the contrary, all-to-all scatter communication of intermediate results will be serialized on the bus and potentially slower than on a direct communication network. Some decisions can be supported by back-of-the-envelope calculations, others are more difficult due to varying message lengths or irregular nature of communications. This is where simulation fits into.

For the sake of the presented case study, we will investigate the following (on-chip) communication networks:

1. fully connected network
2. SF hypercube
3. WH hypercube
4. Crossbar switch
5. Atomic bus.

The number of processors p = 2, 4, and 8. The problem size of a benchmark (parallel 1D-FFT) will be n = 1024 points.

3. THE SIMULATION TOOL AND THE DESCRIPTION LANGUAGE

A performance modeling has to take characteristics of the machine (including an operating systems, if any) and application and predict the execution time. Generally it is much more difficult to simulate performance of an application in shared address space than in message passing, since the events of interest are not explicit in the shared variable program. In the shared address space, performance modeling is complicated by the very same properties that

make developing a program easier: naming, replication and coherence are all implicit, i.e. transparent to the programmer, so it is difficult to determine how much communication occurs and when, e.g when cache mapping conflicts are involved [5].

Sound performance evaluation methodology is essential for credible computer architecture research to evaluate hw/sw architectural ideas or trade-offs. Commonly used shared-memory simulators rsim, Proteus, Tango, limes or MulSim [2], beside their sophistication, are not suitable for message passing systems. This made us to reconsider the simulation methodology for shared-memory multiprocessors. Here we suggest using a single CSP-based simulator both for message passing as well as for shared address space. It is based on simple approximations and leaves the speed vs. accuracy tradeoff on the user, who can control the level of detail and accuracy of simulation.

The CSP-based Transim tool can run simulations written in Transim language [3]. It is a subset of Occam 2 with various extensions. Transim is naturally intended for message-passing distributed memory systems. Nevertheless, it can be used also for simulation shared memory bus-based (SMP) systems - bus transactions in SMP are modeled as communications between node processes and a central process running on an extra processor. Transim also supports shared variables, which are used in modeling locks and barriers. Until now, only an atomic bus model has been tested; the split-transaction bus requires more house-keeping and its model is going to be developed in a near future.

The input file for Transim simulator tool contains descriptions of software, hardware and mapping to one another. In software description, control statements are used usual way, computations (integer only) do not consume simulated time. That is why all pieces of sequential code are completed or replaced (floating point) by special timing constructs SERV ( ). Argument of SERV( ) specifies the number of CPU cycles taken by the task. Granularity of simulation is therefore selectable from individual instructions to large pieces of code. Explicit overhead can be represented directly by WAIT() construct. Data-dependent computations can be simulated by SERV construct with a random number of CPU cycles. Some features of a RT distributed operating system kernel, originally supported by hw in transputers, are also built into the simulator, such as process management, process priorities (2 levels only), context switching, timers, etc.

The NODE construct in hardware description is used to specify the CPU speed, communication model and other parameters; otherwise the default values are used. The mapping between software and hardware, between processes and processors, is made through the MAP construct. Parallel processes on different processors, one process per processor, are created by PLACED PAR construct for MPMD or by replicated PLACED PAR for SPMD model of computation.

4. THE PARALLEL FFT BENCHMARK PROGRAM

We will illustrate the technique of optimization of hw/sw multiprocessor architecture on the problem of computing the 1D-, n-point-, discrete Fourier transform on p processors in O((n log n)/p) time. Let p divides n, $n = 2^q$ is a power of two and $n \geq p^2$. Let the n-dimensional vector x [n×1] be represented by matrix X[n/p×p] in row-major order (one column per processor). The DFT of the vector x is given by

$$Y = W_n X = W_p \left[S * W_{n/p} X \right]^T \tag{1}$$

where S [n/p× p] is the scaling matrix, * is elementwise multiplication and the resulting vector y [n×1] = W_n x is represented by matrix Y [n/p×p] in column major order form (n/p^2 rows per processor). Operation denoted by T is a generalized matrix transpose that corresponds to the usual notion of matrix transpose in case of square matrices [4].

The algorithm can be performed in the following three stages. The first stage involves a local computation of a DFT of size n/p in each processor, followed by the twiddle-factor scaling (elementwise multiplication by S). The second stage is a communication step that involves a matrix transposition. Finally, n/p^2 local FFTs, each of size p, are sufficient to complete the overall FFT computations on n points. The amount of computation work for the sequential FFT of an n-element real vector is n log n / 2 "butterfly" operations, where one butterfly represents 4 multiplications and 6 additions/subtractions (20 CPU clocks in simulation). In parallel implementation the computation work done by p processors is divided into stage 1 and 3, but the total amount of work is the same,

$$p \ [\frac{n}{2p}\log\frac{n}{p} + \frac{n}{p^2}\frac{p}{2}\log p \ ] = \frac{n}{2}\log n . \qquad (2)$$

Let us note, that the work done in stage 1 proportional to (log n - log p) is much larger than the work done in stage 3, proportional to log p. The only overhead in parallel implementation is due to a matrix transposition. The matrix transposition problem is equivalent to all-to-all scatter (AAS) group communication. Clearly, it requires 1 step in a fully connected topology, p/2 steps (a lower bound) in the SF-hypercube, p-1 steps in the WH-hypercube or a crossbar, and finally p(p-1) bus transactions on a bus.

FFT processing will be done continuously in real time. Therefore loading of the next input vector from outside and writing the previous results from processors to environment will be carried out in the background, in parallel with three stages of processing of the current input vector (with the first stage of processing only in the shared memory case).

5. PARAMETERS OF SIMULATED ARCHITECTURES AND RESULTS OF SIMULATION

Six architectures simulated in the case study are listed in Tab.1 together with the execution time. The CPU clock rate is 200 MHz in all 6 cases, the external channel speed of 100 Mbit/s (12 MB/s) is used for serial links in all message-passing architectures, whereas bus transfer rate for SMP is 100 MB/s. Downloading and uploading of input data and results were supposed to continue in the background in all processors simultaneously at 8-times higher rate than the link speed, which is almost equivalent to the bus speed in SMP case. In message-passing architectures the AAS communication was overlapped with submatrix transposition as much as possible. Optimum routing algorithm for SF hypercube and AAS communication requires p/2 steps and uses a schedule tables at Fig.1. In case of WH hypercube, dimension-ordered routing is used in every step i, i = 1, 2,…, p-1, in which src-node and dst-node with the relative addresses RA = src ⊕ dst = i exchange messages.

The small cluster of (digital signal) processors, referred to as COSP in Tab.1, uses a centralized router switch (Omega type) with sw/hw overhead of 5 μs, the same as a start-up cost of serial links, and WH routing. The algorithm for AAS was designed to avoid contention using cyclic permutations [e.g (01234567), (0246)(1357), …, (07654321)] for p=8.

Finally a bus-based shared memory system with coherent caches (SMP) has had 100MB/s bus bandwidth, 50 MHz bus clock, and the miss penalty of 20 CPU clocks. We will assume an

step	RA in dimension	
	0	1
1	3	2
2	1	3

step	relative addr. used in dimension		
	0	1	2
1	3	6	4
2	1	7	6
3	7	2	5
4	5	3	7

Fig. 1. . Optimum schedule for AAS in all-port full-duplex 2D- and 3D- SF hypercubes

atomic bus for simplicity and fair bus arbitration policy. Other types of bus arbitration (priority-based, random, etc.) are also feasible. The cache block size is 16 bytes and the size of the cache is assumed to be sufficient to hold input data (a real vector), intermediate data after the first stage of FFT (a complex vector) as well as the results (a complex vector). In the worst case (p=2) the size of all these vectors will be around 10 kB, if we use REAL32 format. We assume I/O connected via a bus adapter directly to the cache. To avoid arbitration between CPU and I/O, the next input and previous results are transferred in/out during the first stage of the FFT algorithm.

The results summarized in Tab.1 and plotted in Fig.2 deserve some comments. A fully connected network of processors is the fastest architecture for 8 processors, but the slowest for 2 processors. The reason is that communication is mostly seen as an overhead, but gets better overlapped with communication when p increases. The cluster of DSPs (COSP row in Tab.1) starts with p=4 and increasing the number of processors from 4 to 8 does not make much sense because it has small influence on speed.

Tab.1. Parallel FFT execution times in μs for six analyzed architectures

p =	2	4	8
full	436,8	180,8	138
COSP		230,4	173,1
SMP	363,5	304,7	321,6
SF cube		272	182,8
WH cube		230	174,4

In the SMP with shared bus, processors write the results of the n/p-point FFT computed in stage 1 into the local caches and do the transposition at the same time. This means that consecutive values of FFT will be stored with a stride required by the rule of matrix transposition. The following read requests by other processors at the start of stage 3 will generate read misses: at cache block size 16 bytes, one miss always after 3 hits in a sequence. Fresh cache blocks will be loaded into requestor's cache and simultaneously into the shared memory. A prefetch of cache blocks has been simulated without an observable improvement in speed, most probably due to bus saturation. This is even worse for 8 processors than for 4, see Fig.2.

As for hypercubes, the WF hypercube is superior and gives the same results as a cluster of DSPs. Slightly worse performance than that of a fully connected processors is balanced by much simpler interconnection and by a lower number of communication ports.

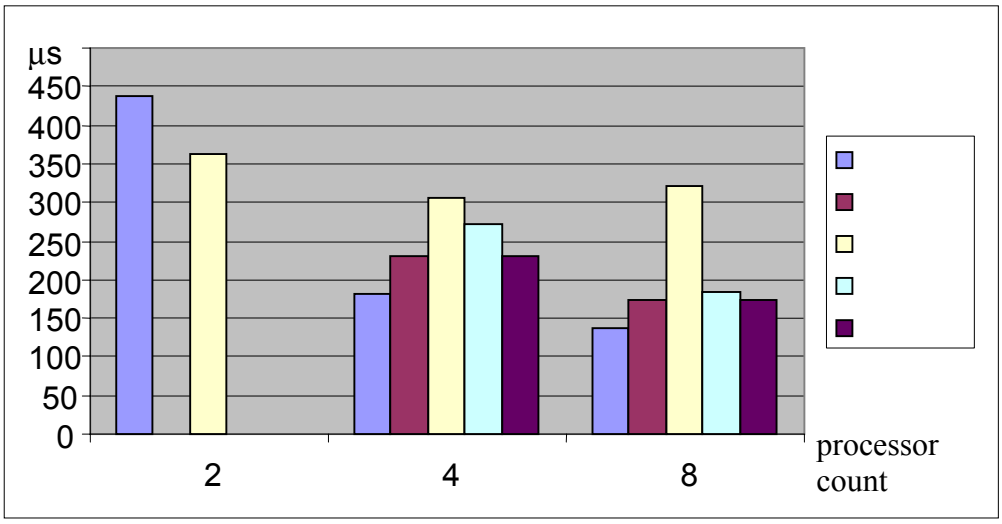


Fig.2. Comparison of execution times [µs] for six architectures.

6. CONCLUSIONS

The performance study of the parallel FFT benchmark on a number of architectures using Transim tool proved to be a useful exercise. Even though the results of simulations have not been confronted with real computations, they can certainly serve to indicate serious candidate architectures that satisfy certain performance requirements. The approximations hidden in simulation are limiting accuracy of real-time performance prediction, but the level of detail in simulation is given by the user, by how much time he or she is willing to spend on building the model of hw and sw. For example, modeling the split-transaction bus or the contention in interconnection network for WH routing could be quite difficult. The latter was not attempted in this case study since the FFT benchmark requires only regular contention-free communication. This, of course, will not be generally the case. Nevertheless, simulation enables fast varying of sw/hw configuration parameters and studying the impact of such changes on performance, free from the second-order effects. In this context, the CSP-based Transim simulator and language proved to be very flexible, robust and easy to use. The future work will continue to include other benchmarks and analyze the accuracy of performance prediction.

REFERENCES

[1] Silc, J.- Robic, B.-Ungerer, T.: Processor Architecture: From Dataflow to Superscalar and Beyond. Springer-Verlag, 1999, ISBN 3-540-64798-8.

[2] http://heather.cs.ucdavis.edu/~matloff/mulsim.html

[3] Hart, E.: TRANSIM - Prototyping Parallel Algorithms. London, University of Westminster Press (2nd edition), 1994.

[4] Zomaya, A.: Parallel and Distributed Computing Handbook. McGraw Hill, 1996, p.344.

ACKNOWLEDGEMENT

This research has been carried out under the financial support of the Research intention no. CEZ: J22/98: 262200012 - Research in information and control systems.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

UML EXTENSIONS FOR MODELING REAL-TIME AND EMBEDDED SYSTEMS

Sławomir SZOSTAK[1], Silva ROBAK[2], Roman STRYJSKI[1],
Bogdan FRANCZYK[1],

[1] Pedagogical University of Zielona Gora, Technical Institute,
ul. Wojska Polskiego 69, 65-762 Zielona Gora, POLAND, *S.Szostak@wsp.zgora.pl*,
Stryjski@post.pl
[2] Technical University of Zielona Gora, Institute of Computer Science and Management,
ul. Podgorna 50, 65-246 Zielona Gora, *Robak@pz.zgora.pl*

Abstract. *The process of modeling and developing of real-time and embedded systems should be supported by suitable methods and notations. In the paper we examine different approaches for customizing standard modeling language UML to model such systems in object-oriented analysis and design. We propose the use of UML standard lightweight extensibility mechanisms (stereotypes) without changing the UML metamodel. Our approach allows joining advantages of extended sequence diagrams and timing diagrams with UML and provides traceability of a concept throughout system development. The examples illustrate our approach. Applying lightweight UML extension mechanism allows existing standard UML modeling tools to be used without any adaptations.*

Key Words. *Real-time and embedded systems, UML extensibility mechanisms, extended sequence diagrams*

1. INTRODUCTION

The Unified Modeling Language (UML) adopted by OMG [8] as its standard modeling language has emerged as the software industry's dominant language. UML is a general-purpose graphical language for specifying, constructing, visualizing, and documenting workproducts that are modified, or used by software-intensive systems [1]. The UML needs to be extended for proposes of modeling real-time and embedded systems. It can be done either by using UML lightweight extensibility mechanisms (such as stereotypes, constraints and tagged values) or by heavyweight extension mechanisms - metaclasses. Metamodel level is a one layer of the UML's four-level model architecture based on metamodel architectural pattern [5]. The metamodeling offers significant advantages. It allows formal specification of all modeling concepts (together with their attributes, constraints and relationships), defines a base for unified exchange format and makes possible the extendibility of UML, i.e. instantiation of new metamodel classes as subclasses of the existing metamodel classes. Although changing the metamodel underlying the UML offers the highest degree of

flexibility, we have not taken it into consideration because the metamodel is not accessible or difficult to modification in existing UML modeling tools.

In the paper we present an approach for modeling real-time and embedded systems using UML. We present a concept for distinguishing model elements with stereotypes and then we examine known approaches for extending UML such as extended sequence diagrams for modeling real-time systems and also the use of timing diagrams.

2. EXTENDS OF STANDARD UML FOR REAL-TIME SYSTEMS

2.1. Stereotypes

Lightweight extension mechanisms are represented in UML metamodel as metamodel's classes named Stereotype, Constraint and TaggedValue. Stereotypes are a way of extending the basic metamodel to create a new model element as a subclassification of an existing model element. Stereotypes are used to mark, classify, or introduce new model elements in metamodel class hierarchy. Every model element may be marked with at most one stereotype, which is depicted in front of an element's name enclosed in double angle brackets, and/or represented graphical as an icon.

To model an element, which corresponds to a feature of real-time and embedded systems, we may introduce new stereotypes i.e. the objects such as a processor can be divided into the processors <<cisc>> and <<risc>> by using a stereotype, and thus give them the different features. The UML already predefines some stereotypes for classes, messages, objects, and etc [7]. The instance class containing the stereotype <<active>> is shown in Figure 1.

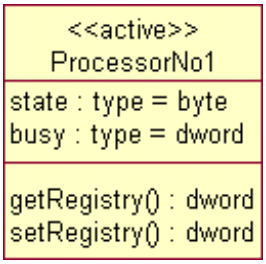


Fig. 1. Example class with stereotype <<active>>

Examining the real-time systems at building the stereotypes, it should take the characteristic features which derive from a given application domain into consideration. Suppose our system consists of the typical elements of industrial automation such as: processors, drivers, sensors, actuators, networking, monitoring, etc. For these and similar systems, i.e. embedded systems., safety-critical systems, the instance stereotypes can be distinguished in the UML for various elements. They are presented in Table 1-3.

Table 1. Stereotypes for nodes

UML Type	*Stereotype*	*About stereotype*
Node	<<processor>>	represents device that executes software
	<<other device>>	device that can not executes any software
	<<sensor>>	device that monitors course of external processes

<<actuator>>	Device that aktuates external process or other internal device
<<display>>	device that displays information for external actor (user)
<<knob>>	input device for external user
<<button>>	input device for external user
<<switch>>	input device for external user
<<watchdog>>	sensor that waits for fail-safe behaviour

Table 2. Stereotypes for messages (communications)

UML Type	*Stereotype*	*About stereotype*
message	<<synchronous>>	association realized as simple method call (directly)
	<<asynchronous-local>>	association that crosses a thread boundary and put the message in target thread's queue
	<<asynchronous-remote>>	association that crosses a processor boundary and put the message in target thread's queue without waiting for answer
	<<synchronous-remote>>	association that across a processor boundary and block sender until receiver returns answer
	<<periodic>>	message is sent periodically
	<<episodic>>	message is sent when event occurs
	<<epiperiodic>>	message is sent periodic and when event occurs

Table 3. Stereotypes for classes

UML Type	*Stereotype*	*About stereotype*
class	<<active>>	class is the root of an operating system thread

2.2. Scenarios

In each system some processes, which range the definite objects of this system, occur. Each of these processes consists of the elementary entities (i.e. external and internal calls, messages, interacts with actors, between objects etc.), whose chronological set composes a certain path or a branching tree through the system behavior. Such a path (or a branching tree) is called a scenario. Each scenario is based on a set of the objects and actors. The system behavior is composed of many completely independent and/or partly correlated paths. Only many such as scenarios produce a full image of the use-case system. Scenarios contain information about events both important and incidental for a system, but mostly scenarios are constructed basing on the most important elements. If scenarios differ only in the incidental elements, they are ignored. There are some various methods to describe a scenario: textual description, sequence diagrams and state diagrams [2]. The first method is not interesting because of its informality.

The state diagrams do not distinguish themselves anything specific for the real-time systems. Therefore, we study the extended sequence diagrams.

As we know, a sequence diagram shows the flow of messages between the objects of the system and the actors (Figure 2).

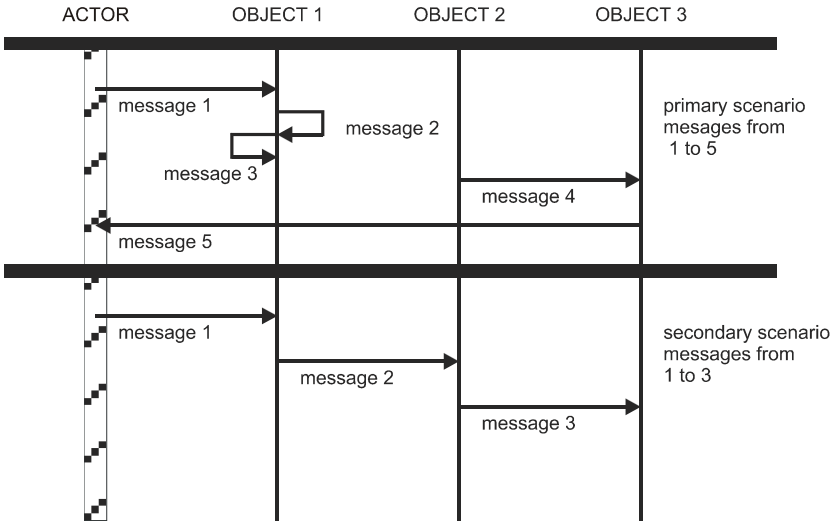


Fig. 2. Simple sequence diagram example

This diagram does not regard any following requirements for the real-time systems:

- execution time of event or message
- rise and fall time
- initiation and dwell time
- slack time
- deadline
- period
- leading and trailing jitter.

Therefore, some additional elements of a sequence diagram have been introduced for the real-time systems:

- timing marks: simple and conditioned
- state marks
- event mark.

Timing marks definite the duration of the time of a single event or message. This time is indicated between the start and the end of message (i.e. {< 20 ms}), or as the interval between the events (i.e. $\{t_1 - t_2 <= 10$ ms$\}$). It is called a simple timing mark. We may also indicate the duration of the time of a greater number of events (i.e. $\{t_5 - t_1 < 0.5$ s, but $t_3 - t_2 < 100$ ms$\}$).

Event marks represent the events that give rise to the message on the time line referred to the relevant object. The letters or symbols (shown on Figure 3 as indexed) on the time line written at the opposite ends of the message arrow, respectively indicate them.

State marks are to bridge a certain gap between the sequence and state diagrams. State diagrams do usually not depict the time dependencies between the states, and sequence diagrams do not show the present state of the system. State marks are the rounded rectangles placed on the time line off to the relevant object.

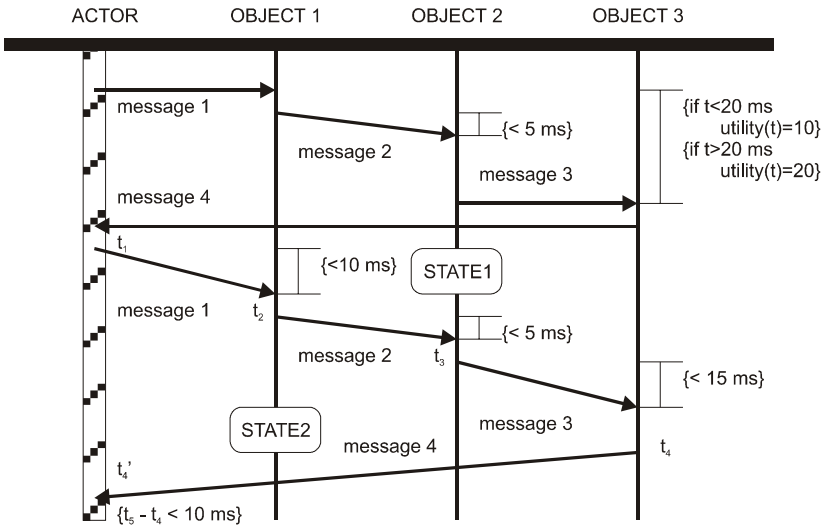


Fig. 3. Extended for real-time sequence diagram example

Timing diagrams are another way of representing a path of the system behavior. They have been known to electrical engineers for a long time (as well as to people who focus on programming the industrial controllers) as the diagrams being used in designing the electrical state machines (digital). They are, however, extended: the axis X depicts the time, but the axis Y can represent more than two states: on and off (or 1 and 0, H and L). Along the axis X there are some gaps, which separate the different states. If the system is in the defined state, a line (function) will be drawn in that state. On the axis Y along the state line there are special names of the states as well as the indicated events that give rise the relevant state (Figure 4).

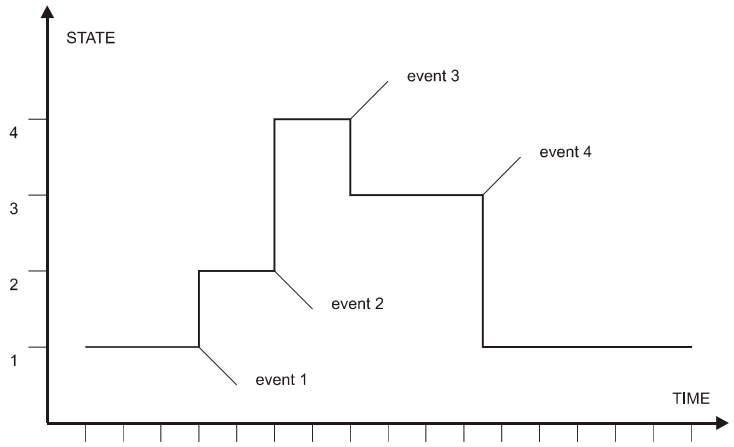


Fig. 4. Timing diagram for one scenario

This diagram depicts a development of events in time in a simplification, however. This considers only the timing of the particular states and enables it to show the only one scenario (for the only one object). It is possible to place more than one scenario in the timing diagrams, regarding the respective periods of the duration of the states (Figure 5 and 6).

3. SUMMARY

The approach described above allows combining the advantages of standard modeling with UML diagrams adopted for real-time and embedded systems. Traceability of a concept throughout system development is provided. Using only lightweight UML extension mechanisms (stereotypes) means, that existing standard UML modeling tools can be used without any extensions or adaptations.

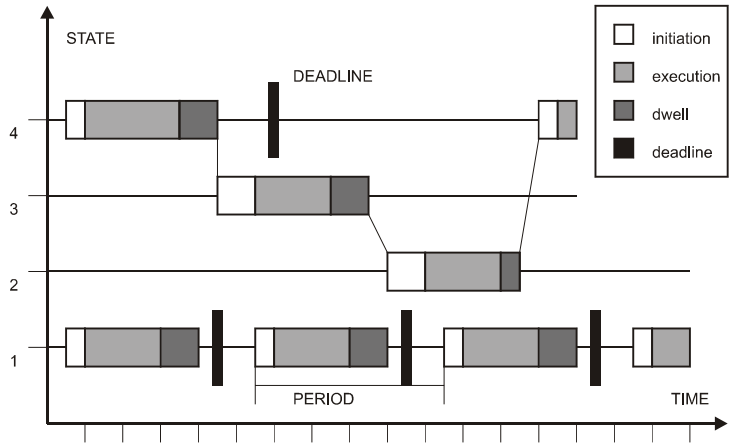


Fig. 5. Timing diagram for multiple scenarios

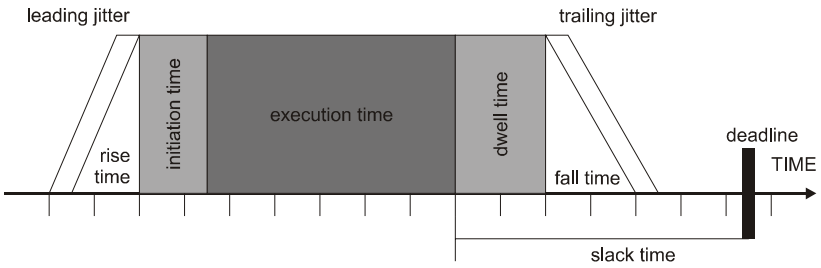


Fig. 6. Complex timing diagram for one event of scenario

In order to reach more comprehensive support for real-time and embedded systems modeling the next step is the integration of their features into the UML and specification of appropriate constructs to be defined as a special real-time and embedded systems UML profile.

REFERENCES

[1] G.Booch, J.Rumbaugh and I.Jacobson, The Unified Modeling Language User's Guide, *Addison Wesley*, 1999

[2] B.P.Douglass, "Doing Hard Time", *Addison-Wesley*, 1999

[3] B.P.Douglas, "The UML For Systems Engineering, *I-Logix whitepapers*, *www.i-logix.com*

[4] N.Hilary, "Bringing the Gap between Requirenments and Design With Use Cases and Scenarios". *I-Logix whitepapers*, *www.i-logix.com*

[5] C.Kobryn, "UML 2001: A standarization odyssey. Communication of the ACM", *Vol.42, No. 10, pp. 29-37*, 1999

[6] R.J.Muller, "Bazy Danych. Język UML w modelowaniu danych", *Mikom*, Luty 2000

[7] "OMG Unified Modeling Language Specification v 1.3", *Object Management Group*, March 2000

[8] OMG, Object Management Group, *http://www.omg.org*, 2000

[9] R.Rinat, "A Framework-Based Approach to Real-Time Development with UML", *I-Logix whitepapers*, Israel, June 2000

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

CORRECTNESS PROOF OF AN OPERATING SYSTEM KERNEL FOR HARD REAL TIME COMPUTING

Grzegorz HAMUDA, Wolfgang HALANG

Akademia Górniczo-Hutnicza w Krakowie, al.Mickiewicza 30, 30-074 Kraków,
POLAND, *gha@ia.agh.edu.pl*

FernUniversität Hagen, Faculty of Electrical Engineering, 58084 Hagen, GERMANY,
wolfgang.halang@fernuni-hagen.de

***Abstract**. An architecture (including both hardware and software solutions) for an operating system kernel to be employed in hard real time environments is shortly described, and its functionality is presented. By considering its application areas, which comprise safety critical systems, the need for correctness of such a kernel is pointed out. Ways to achieve this property are identified in the context of appropriate correctness criteria. It is discussed how proper formal methods are selected for verification, and to which particular task each method is applicable. Experiences and observations are presented. As one of the latter, the need to apply both theoretical (formal) and practical methods is underlined. Therefore, a simulator for the kernel was developed, whose functionality is described as well.*

***Key Words**. Hard real time computing, operating system kernel, correctness, formal methods, proofs, simulation*

1. THE ARCHITECTURE

The architecture of an operating system kernel considered here (see [2] for details) can be characterised by an asymmetrical dual processor configuration: applications consisting of independent, co-operating tasks execute on a general purpose processor (Slave), whereas all OS kernel activities are executed on its own dedicated co-processor (Master). The latter acts as supervisor of all activities of the entire system, including execution of user tasks, scheduling, time management, memory management, interrupt and input/output event servicing.

Task scheduling is based on the earliest deadline first algorithm. This algorithm can be employed, because estimations of the execution times of all tasks are known a priori. The task processor always executes the task having the shortest deadline. Unnecessary context switches are thus avoided. What is more important, however, is that the execution of system calls does not cause task pre-emptions (with the exception of activation or continuation of tasks with the shorter deadlines).

As shown on Fig. 1, the kernel is divided into three co-operating layers. The task interface to the OS Secondary Reaction Layer is organised in the form of system calls, which can be

divided into the different groups of tasking operations (activate, terminate, prevent, suspend, continue, resume, end), task scheduling (scheduler), task synchronisation (sync_test, sync_resume), task communication, and input/output operations.

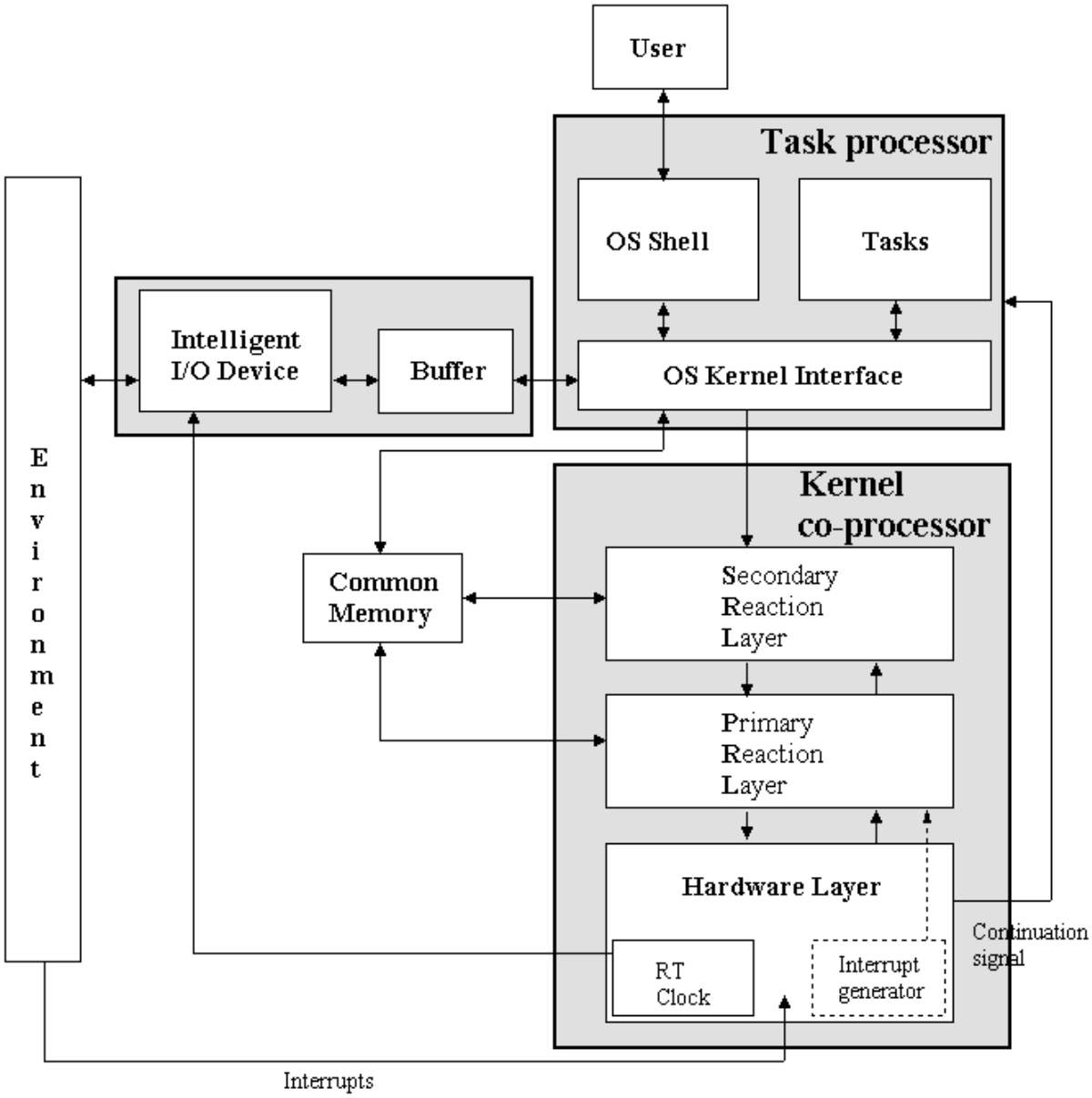


Fig. 1. Hardware architecture - functional diagram.

System calls concerning tasks, which are received by the Secondary Reaction Layer, contain time (or event) conditions for actions to be performed. The actions are stored in one of the system tables, while the calls are passed to the Primary Reaction Layer, which notifies the Secondary Reaction Layer on the occurrence of the corresponding trigger events, such as the passage of a duration or an external interrupt (Fig. 2). Such events trigger the execution of the associated actions. The OS kernel was specified, mainly in the language PEARL, in the form of about 28 algorithms.

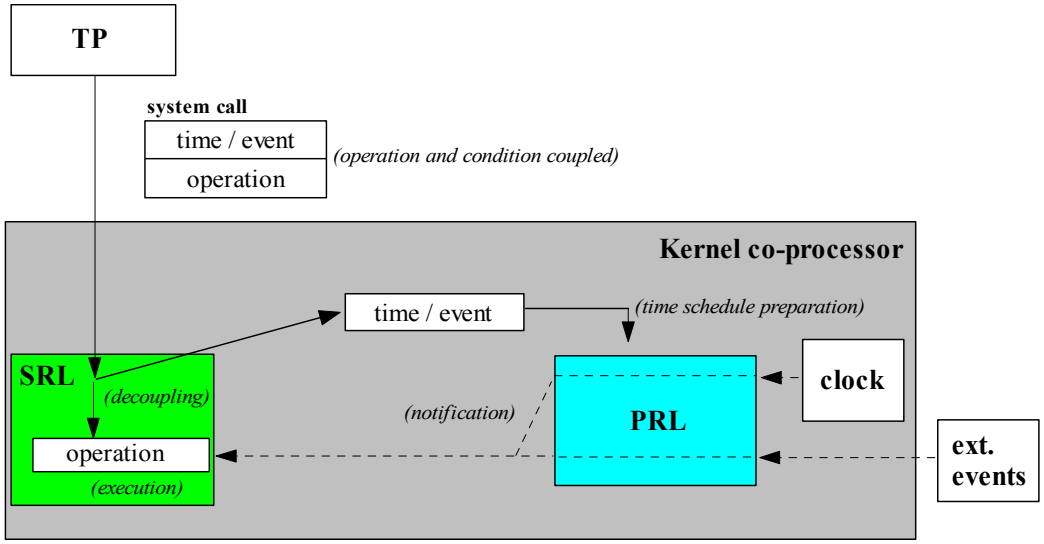


Fig. 2. System call.

2. THE PROBLEM OF CORRECTNESS

The kernel was designed to be used in hard real time environments. As it is broadly known, control systems for such environments may not crash or behave in a non-predictive manner. This concerns the control application itself (the code), the hardware it is running on, the hardware it controls, and the operating system as well. To be able to show that the proposed kernel can be used for such purposes, an attempt was made to proof - by the use of formal methods- that the kernel is reliable and dependable.

There is no universal prescription for the usage of formal methods in practice. The most useful advice and guidelines were found in [3]. There, the following phases of formal verification were mentioned (Fig. 3):

- Characterisation - that is to achieve a deep understanding of the application (and its domain area) to be verified
- Modeling - selection of a proper mathematical representation(s) (model) most suitable for the application, selection of a (formal) specification language and of appropriate tools (theorem prover, proof checker, model checker)
- Specification - decisions concerning the specification strategy (hierarchical levels, language, properties), writing the formal specification
- Analysis - interpretation of the specification prepared, proving the key properties etc.

From this short description at least four conclusions, having significance in practice, can be drawn:

1. Detailed (informal) specification of a system is very helpful (mainly in the characterisation phase),
2. Decisions concerning the selection of both a formal (mathematical) model and of tool(s) for verification purposes should be made in such a way that they fit to the system to be verified as closely as possible, and not the other way around,
3. System properties to be proven should be named (first, they should be discovered during the characterisation phase),

4. The selection of the formal method tool is (more or less) a function of the model selected and the system properties.

The formal verification of the kernel properties was based on the just described conclusions,

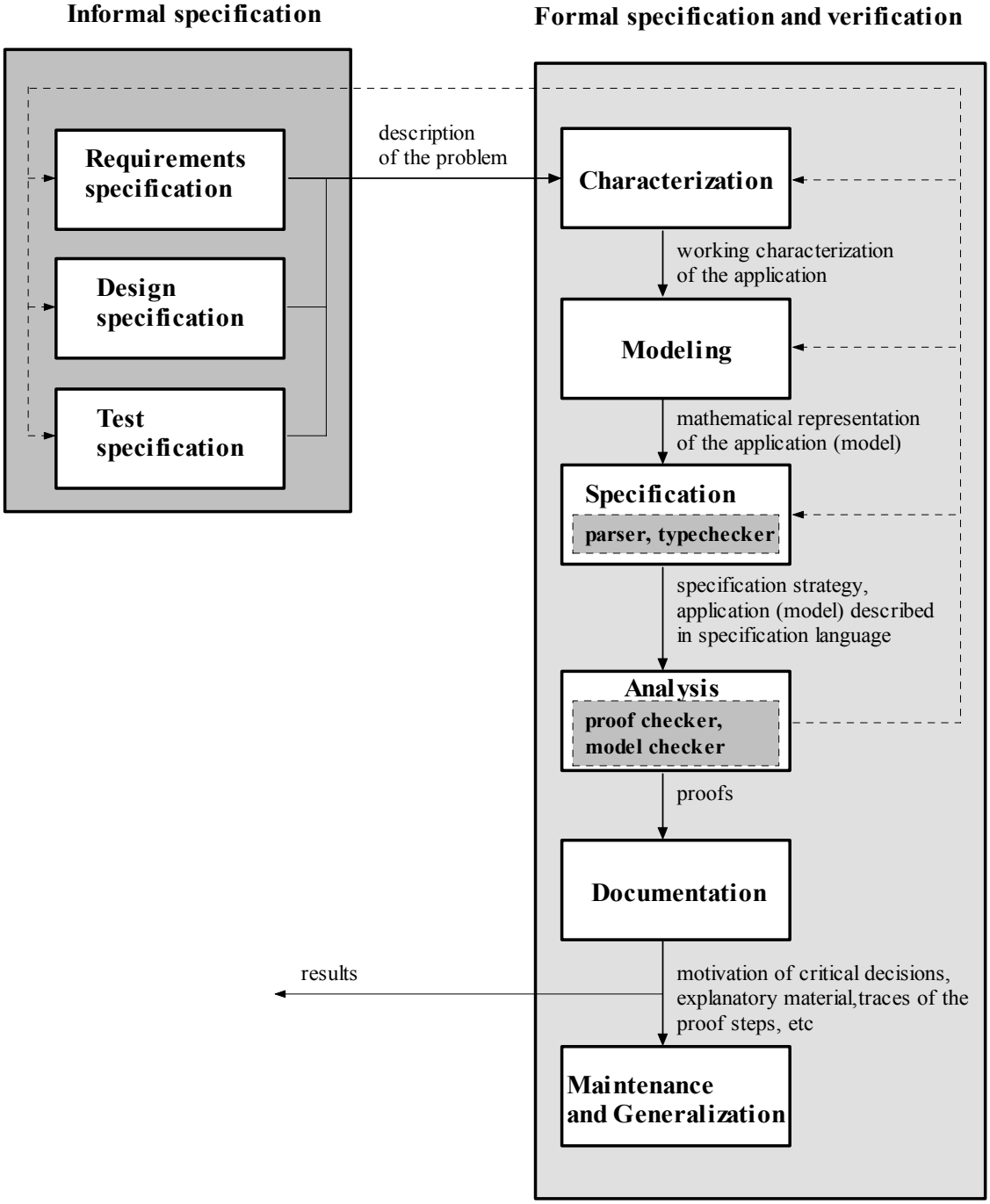


Fig. 3. Phases of formal verification.

treated as general directions. During the characterisation phase, more than 20 system properties were identified, the most fundamental on among them being:

The deadlines of all tasks listed as ready are met (normal situation), or the deadlines cannot be met (this fact is known a priori) and an overload signal is raised.

As another result of this phase of work, the following observations were made:

- Proving correctness of the hardware elements of the kernel would require the usage of a functional description.
- In case of the system calls (specified as algorithms) Hoare Logic is needed to show, for example, that the algorithms properly manipulate the system data structures.
- Some system properties will be having the form of theorems and axioms, while the others will be expressed as invariants.
- Hardware and software (of the kernel) cannot be analysed separately.

The mentioned observations have shown the need to select both a (formal) method and tools carefully. The latter should be able to be used both as theorem provers and model checkers.

There are many popular software tools supporting different formal methods. We have finally chosen the Prototype Verification System (PVS). It is a general purpose tool based on higher order logic. In PVS, the user is required to describe system properties to be proven in a specification language similar to a programming language. Thanks to its popularity and generality, many formal methods were successfully incorporated into PVS. The experience from the current stage of the work (specification of all system properties in PVS) confirms the decisions made earlier.

3. THE KERNEL SIMULATOR

Since the OS kernel was specified as a set of algorithms, there was a need to simulate it. It is common practice that teams applying formal methods build simulators to achieve deeper understanding of the functionalities to be verified. Our simulator is composed of several co-operating programs written in Java. The communication model is based on Remote Method Invocation (i.e., the client/server model). The kernel itself forms one program (Simulator), in which threads were used to model both the task processor and the kernel co-processor activities. To both the memory modules and fifos, which should be accessible for different kernel components, exclusive access can be ensured thanks to a thread synchronisation mechanism built in Java. This program acts as a (RMI) server, providing the other program (Monitor) with all the data needed for on-line system analysis. The third program (Control Panel) was designed to both enable control over the execution of the OS kernel (which can be started, stopped, re-run, executed step by step etc.) and emulation of the external environment. The monitor program mentioned above also acts as a (RMI) server for programs performing different kinds of visualisation. The basic one is Message Sequence Chart (MSC) Trace of the whole system. The program animates the diagram showing the exchange of data between elements of the kernel. Other visualisation tools can then be added easily. Each of the mentioned visualisation tools can be started or stopped independently of the simulator.

4. CONCLUSIONS

The usage of formal methods in practice requires planning and decision making. Important steps and decisions were shown for the example of a real time operating system kernel. The role of the traditional (informal) specification was underlined. The traditional methods (state charts, transition graphs, block diagrams etc.) turned out to be very useful, and form a sound basis for the formal verification of correctness. Both the PVS specification language and the tool PVS as a formal method served our purpose very well.

REFERENCES

[1] Myla Archer and Constance Heitmeyer *Human-Style Theorem Proving Using PVS* Naval Research Laboratory, Washington, DC 20375

[2] Wolfgang A. Halang and Alexander D. Stoyenko *Constructing Predictable Real Time Systems* Boston: Kluwer Academic Publishers 1991

[3] Sandeep Kulkarni, John Rushby, Natarajan Shankar *Formal Methods Specification and Verification Handbook for Software and Computer Systems Volume I: Planning and Technology Insertion Volume II: A Practitioner's Companion* Office of Safety and Mission Assurance, NASA-GB-002-95, Release 1.0

[4] Sam Owre, Natarajan Shankar, M. K. Srivas *A Tutorial on Using PVS for Hardware Verification* SRI International, Computer Science Laboratory, Menlo Park CA 94025 USA

[5] Sam Owre, Natarajan Shankar, J. M. Rushby *PVS Language Reference (Version 2.3)* SRI International, Computer Science Laboratory, Menlo Park CA 94025 USA

[6] Sam Owre, Natarajan Shankar, J. M. Rushby *PVS Prover Guide (Version 2.3)* SRI International, Computer Science Laboratory, Menlo Park CA 94025 USA

[7] John Rushby *Formal methods and digital systems validation for airborne systems,* NASA Contractor Report 1673, August 1995

[8] C. J. Walter, R. M. Kieckhafer, A. M. Finn *MAFT: A multicomputer architecture for fault-tolerance in real-time control systems,* IEEE Real-Time Systems Symposium, December 1985

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

ALGORITHM FOR OPTIMIZATION OF PARALLEL COMPUTATIONS ON THE BASIS OF GENETIC ALGORITHMS AND MODEL OF A VIRTUAL NETWORK

Raouf Kh. SADYKHOV*, Aliaksei V.OTWAGIN**

*-Prof. D.Sc., Byelorussian State University of Informatics and Radioelectronics, 6 P.Brovka st., 220600, Minsk, BELARUS

**- post-graduate student, Institute of Engineering Cybernetics, 6 Sourganov st. 220072, Minsk, BELARUS, *forlelik@mail.ru*

***Abstract**: The problem of the program modules assignment onto processors of the multiprocessor computing system is considered. The general statement of a task and initial data for its description is given. The common approach and number of algorithms, used for solution search, is considered and their basic features are analyzed. The algorithm of the solution search, using representation of parallel algorithm as virtual neural network with training, based on the genetic algorithms, is offered. The comparative estimations of clusterization algorithms and algorithm of a virtual network training are proposed.*

***Key Words**: parallel program, virtual neural network, clusterization, genetic algorithm.*

1. INTRODUCTION

The problem of the parallel program modules assignment onto processors of the multiprocessor computing system is related with class of so-called NP-complex tasks. The goal is such distribution of the interconnected modules set for the program onto parallel system processors, at which the most uniform loading of processors by computing is provided, and also the time and cost of information interchange between modules located on different processors is minimized. At the exact solution of such task on condition that module can be assigned on any processor it is necessary to consider N^M variants for an modules allocation on processors, where N is the number of the system processors, M – the number of the program modules. If N and M are great enough, the exact solution can not be found for acceptable time, especially when we have problem of real time control. Therefore, for the

decision of such tasks the certain heuristic algorithms, which generally are not ensuring the optimum decisions, are used.

In given report a number of algorithms to be used for distribution of the parallel program modules on processors of the parallel computing system is considered. The analysis of these algorithms, offered in [1], has allowed to estimate general heuristics and offer a search method of the optimum or suboptimum solution search for finite time.

2. STATEMENT OF A TASK

Let's consider the multiprocessor system, in which each processor has own RAM, the processors are non-uniform under the performance, and there are communication channels between all processors. The synchronization losses for information interchange between processors are not taken into account.

The parallel program, executed in system, is divided on modules, where the subset can be executed only on the certain processor (that have the certain resource), and other subset can be executed on the any processor. The module represents indivisible unit of the information processing (elementary operation). Thus, the module chosen for running on the processor at certain moment, occupies it without interruption on all time of it's execution.

In world practice of the decision of such tasks the methods from the graph theory and the network diagrams are widely applied. Let's number of modules of the program is equal to M, and number of processors in system is N, then parallel program is represented as directed acyclic weighted graph (DAG). The graph can be described as a tuple G = (V, E, C, T), where:

V - set of graph nodes representing separate modules of the program. Each of modules is characterized by execution time $t_{m,n}$ ($m \in M, n \in N$);

E - set of communication edges representing connection between modules on the information or control continuity. The edges are directed from the module - transmitter to the module - receiver of the information. Weight of an edge determines cost of the information transfer on an edge $c_{i,j}$ ($i, j \in M$) ;

C(M×-M) - matrix of cost determining cost of the information transfer between the certain modules of the program. If there is no connection between modules, then zero is put in the appropriate position;

T(M×N) - matrix of task execution times for each of processors in the system. If the module can not be allocated onto processor, zero is put in the appropriate position.

Each task can be executed only after the execution all of its predecessors are completed. After end of a task the data will transmit to all its followers. Thus, the certain similarity of this statement of a task with a task of a minimal cutsection finding for the network flow is observed. The decision of this task in the general case is decision of a task of linear programming.

3. THE GENERAL APPROACH TO THE ALLOCATION TASK DECISION

In [1] the given task is considered as a task of graph division on subgraphs (clustering), when each of subgraphs contains tasks nominated to the certain processor. There is a number of clustering algorithms, however they provide the optimum solutions on the certain subset of tasks, but do not provide the solutions generally. The basic ideas and characteristics of this group of algorithms are considered below in brief.

All clustering algorithms begin functioning from number of clusters, equal to number of graph nodes. In clustering procedure separate clusters are united, if this association promotes achievement of the goal for algorithm.

The clustering algorithms are based on so-called "edge zeroing". Thus the edges of information interchange between tasks on identical processor are considered as removed, and weight or cost of the communications of these edges is accepted equal 0. At definition of performance time of tasks graph these edges do not influence on estimations. On one step the algorithm can reject the certain number of edges depending on realization.

Each algorithm uses certain heuristics at clusterization. These of heuristics determine edges, which should be "vanished", i. e. the edges, on which two subgraphs of tasks are united in one cluster. Heuristics are be of two types:

1) Heuristics of performance - minimization of performance time, minimization of the communications cost, maximization of function of effective cost;

2) Non-performance heuristics - requirement of clusterization linearity (each task has even one edge connecting with one of tasks for given cluster), miss of cycles in cluster, conformity between clusters and data distribution in system. These heuristics can be taken into account with a various priority, thus the results of clusterization can be essentially differed. In a Fig. 1 the examples of graph division are given with account of various heuristics.

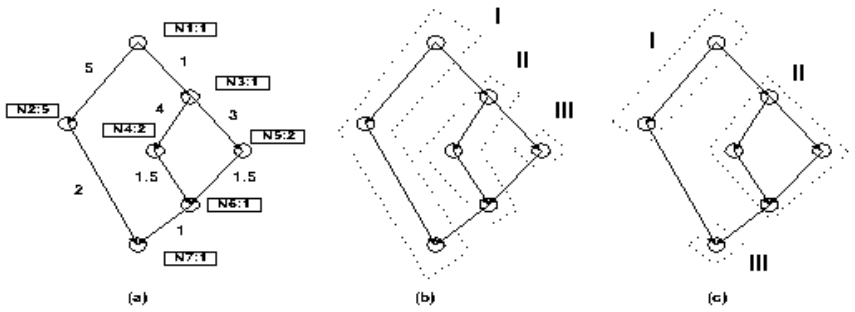


Fig. 1. Clustering examples

The Fig. 1a represents graph for the parallel program. The graph nodes have a designation Ni:c, where i - number of node, c - its performance time. The graph edges have weight indicating time of transition on this edge in a case, when the tasks, appropriate to its beginning and the end, are nominated to different processors. The Fig. 1b represents an example of linear clusterization, 1c – nonlinear clusterization. Performance time(PT) of the parallel program (in relative time units - RTU) for linear clusterization (1b) PT=12.5 RTU, for nonlinear (1c) PT=9 RTU.

Usually heuristics are defined as cost functions of the certain kind allowing its calculation for polynomial time. Thus some special cases of task graphs allowing formulate an estimation function are usually considered.

The clusterization algorithms also assume that technique of "search without return " (non-backtracking) are used.

Each of algorithms is estimated on expenses of time necessary for search of optimum clusterization. The estimation is made by quantity of conditional operations v and e, where v - operation of the analysis for one of nodes, e - operation of the analysis for one of edges. Some algorithms of "edge zeroing" are below listed.

1) Algorithm Kim and Browne's [2]. This algorithm is based on a critical path search in the graph and association of nodes of a critical path in one cluster. Critical path is the path, which has a greatest length or the sum of edges cost.

2) Algorithm Sarkar's [3]. The essence of algorithm in sorting all edges according decrease of their cost and consecutive zeroing of edges of maximal cost, while it does not contradict the algorithm goal. The goal consists in minimization of parallel time, but it is transformed to minimization of communications cost between units.

3) Algorithm of a dominant sequence [4]. Dominant sequence is a critical path in the graph calculated on each clustering step. Thus, later sequence can include edges from earlier sequence. If edge zeroing does not increase the time of start for a task, this task is added in same cluster, otherwise one concerns to new cluster.

The various statements of the optimization goals impose the features on clustering result. All above mentioned algorithms generally give various results, and are effectively applied only on the limited set of tasks. The algorithm for search of a minimal cut of a network flow [5] takes into account simultaneously execution time for a task and time of information interchanges between tasks. The algorithm uses for the search the modified graph of intermodular connections. In this graph except for edges determining the information transfer, there are edges determining execution time of a task on each of processors. Graph cut unequivocally defines distribution of modules on processors, and weight of a cut is a total cost of the program performance for the given distribution of modules on processors. The opportunity of consecutive application of algorithm for a cuts search for greater, than 2, number of processors, is specified. Thus the following difficulties were specified:

1. Unit nominated on n-th processor in multiprocessor system, will not be always nominated to same unit in dual-processor system.
2. The algorithm cannot be used, if the removal of one of processors of system is made.

4. VIRTUAL NETWORK MODEL

In the given report the algorithm of the minimal cut search of a network flow is considered from the point of view of virtual neural network model, which basic concepts are determined in [6]. In this model set of graph elements is divided on clusters. Each element of cluster is closely connected to other elements in this cluster, i.e. the cost of connections inside cluster should be higher, than cost for external intercluster connections for this cluster. The connection is considered internal for cluster, if the nodes of its beginning and ending belong the given cluster. Thus, the localization of the data exchange inside cluster is achieved. The given rule corresponds to a principle of zeroing most of "expensive" connections. The model of a virtual network is easily scaled on any number of processors and tasks. The given model assumes self-organizing, i.e. alignment of estimations for separate clusters and convergence them to some average size.

The ability of model to self-organizing during its definition or change allows at a correct choice of criteria of an estimation receive splitting the graph of tasks on processors with optimum cost of graph execution on the computing system.

For evaluation function of an estimation of the received model everyone cluster is considered, and the general estimation is defined by the sum of estimations for all clusters.

The goal of the graph clusterization in a virtual network is simultaneous convergence to high reliability for everyone cluster with the minimal loading of the appropriate processor. The loading is determined by the sum of performance times for tasks nominated to the given processor.

The solution search for this task of linear programming is fulfilled by the methods of combinatorics in a combination with methods of the theory of neural networks. Thus the method of construction of a training sequence with the help of genetic algorithms [7] is used. Such method allows simultaneously to consider the whole group of the possible solutions and

fulfill parallel search in various areas of decision space. Therefore probability of reception of the optimum solution is much higher.

At the appropriate coding of the possible solution for the task of chromosome analysis and development of genetic operators of its change (mutation and crossover) it is possible to construct genetic algorithm of search with properties " search without return ". Use of various techniques of search: search in N iterations, search before improvement of the decision within the limits of some $0<\varepsilon<1$, search with an interdiction of separate directions - will allow receive the optimum solution for acceptable time.

5. SEARCH ALGORITHM BASED ON A VIRTUAL NETWORK TECHNIQUE

The essence of algorithm for training of a virtual network with the help of genetic algorithms consists in application of neural networks training principles for training sets generated by genetic algorithm. Thus the virtual network is considered as multi-layered neural network with direct connections.

At training model of a virtual network the Hebb principle [8] have been used : increase of weight of connections promoting reception of the best solution. The weight matrix of a virtual network is initialized by random values. Then via genetic algorithm the most acceptable variant of clusterization is computed according to criterion functions. This variant can be used as a training set for a network.

For experiments the program model, which simulate training of a virtual network for the random graph, has been constructed. The results of modeling allow to make a conclusion about dependence them both from the size the graph, and from its density - number of edges between nodes. The comparison of the received results with results of Sarkar algorithm [3] is given below.

The experiments were spent for random graphs, containing from 10 up to 50 nodes. The average performance time of the parallel program was defined, and its modules were distributed according to clusterization results. Besides the loading of processors actually by calculations which have been not connected to transfer of the data or idle times was defined. For each algorithm 50 experiences have been executed.

The Fig. 2 demonstrates average loading of the processor for two algorithms, where VN - algorithm of a virtual network.

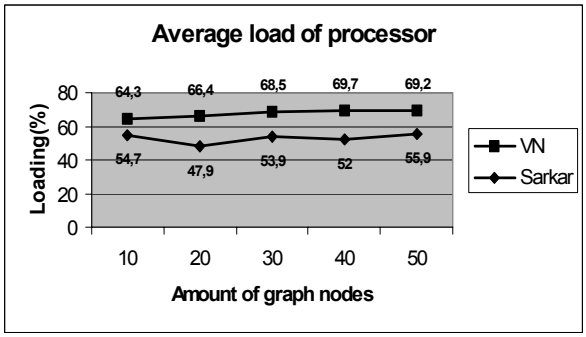


Fig. 2. Loading of system processors at clusterization.

The average loading of processors by calculations is more for algorithm of a virtual network. The uniformity of loading requires allocation of some tasks for processors which are not ensuring generally of the earliest performance a tasks. However, the result of Sarkar's algorithm usually assumes presence in system of the processor which executed considerably large (sometimes in 2-3 times) loading in comparison with others.

The Fig. 3. illustrates process of training for model of a virtual network for 30 tasks. The training was spent for 1000 variants of clusterization. The choice of variant by genetic algorithm was spent in 200 iterations.

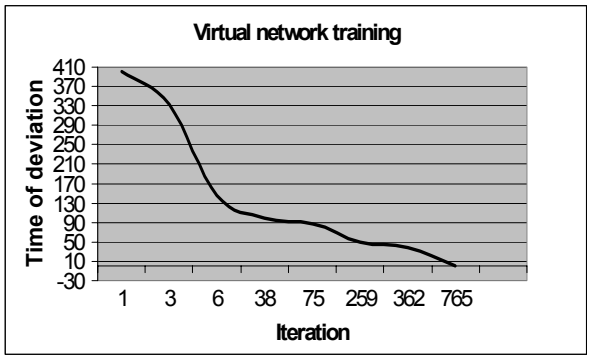


Fig. 3. Process of training of a virtual network

The increase of iterations number for training or choice of a training set increases time of solution search for a virtual network. However, the quality of the received solutions (performance time of the program and uniformity of loading) thus also are improved.

REFERENCES

[1] *A Comparison of Clustering Heuristics for Scheduling DAGs on Multiprocessors*. A.Gerasoulis and T. Yang. Department of Computer Science, Rutgers University, New Brunswick, NJ 08903., 1996

[2] Kim S.J., and Browne J.C. A General Approach to Mapping of Parallel Computation upon Multiprocessor Architectures, *International Conference on Parallel Processing*, vol. 3, 1988, pp. 1-8

[3] Sarkar V. *Partitioning and Scheduling Parallel Programs for Execution on Multiprocessors,* The MIT Press, 1989.

[4] Yang T., Gerasoulis A. A Fast Static Algorithm for DAGs on an Unbounded Number of Processors, *Proc. of Supercomputing ' 91*, *IEEE*, 1991, pp. 633-642.

[5] Trakhtengertz E.A. *The software of parallel processes*. M.: Science, 1987.(In Russian)

[6] Yufik Y. M., Sheridan T. B. Virtual Networks: New framework for operator modeling and interface optimization in complex supervisory control systems, *A Rev. Control*, vol. 20, pp. 179-195.

[7] Michalewicz Z. *Genetic Algorithms + Data Structures = Evolution Programs*. Second, Extended Edition. Springer-Verlag, 1994, 340 pp.

[8] V.A. Golovko. *Neurointelligence: the theory both application. V.1: Organization and training of neural networks with direct communications and feedback*. Brest, 1999.(In Russian).

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

STATE ASSIGNMENT OF ASYNCHRONOUS PARALLEL AUTOMATA

Ljudmila CHEREMISINOVA

Institute of Engineering Cybernetics of National Academy of Sciences of Belarus,
Surganov str., 6, 220012, Minsk, BELARUS, *cld@newman.bas-net.by*

Abstract. *A problem of race-free state assignment of asynchronous parallel automata is considered. The goal is to encode partial states of parallel automaton using minimal number of coding variables and excluding critical races during automaton operation. Requirements imposing on the partial states codes to eliminate the influence of races are formulated. An exact algorithm to find a minimal solution of the problem of race-free state assignment for parallel automata is suggested. The algorithm provides reducing the computational effort when searching state encoding.*

Key Words. *Parallel Automaton, Partial State Assignment, Races in Asynchronous Automata,*

1. INTRODUCTION

The success of the control of a multiple component system depends greatly on the efficiency of the synchronization among its processing elements. The functions of a control of such a system are concentrated in one block - logic control device that should provide a proper synchronization of interaction between the components. In order to represent clearly the interaction involved in concurrent engineering system it is necessary to describe formally its functional and structural properties.

As a functional model of a discrete control device to be designed a model of parallel automaton is proposed [1, 8, 9]. This model can be considered as an extension of a sequential automaton (finite state machine) to represent parallel processes. The parallel automaton is more complicated and less studied model in contrast with classical sequential automaton model. An essential difference from sequential automaton is that a parallel automaton can be in more than one state simultaneously. That is why the states of a parallel automaton were called as partial ones [8]. All partial states a parallel automaton is in at some moment form its global state. In that case any two of these partial states (forming a global state) are called parallel [8]. Any transition of automaton defines the partial state changes that cause the global state changes. The most of transitions (and all for asynchronous parallel automaton) are forced by changes of external signals.

The design of asynchronous automata has been an active area of research for the last 40 years. There has been a renewed interest in asynchronous design because of their potential for high-

performance and avoidance of clock. However, design of asynchronous automata remains a cumbersome problem because of difficulties to ensure correct dynamic behavior.

The important step on the way to control device hardware implementation is the state assignment. It is at the heart of the automaton synthesis problem (especially for its asynchronous mode of realization). Despite large effort devoted to this problem no satisfactory solutions have been proposed. A difference of this process for parallel automaton in comparison with the sequential one is that there are parallel states in the first one (they are compatible in the sense that the automaton can find itself in them at the same time). That is why it was suggested in [8] to code partial states with ternary vectors which should be non-orthogonal for parallel partial states but orthogonal for non-parallel ones. After having coded partial states it is possible provide them with their codes. In such a way an initial parallel automaton is transformed from its abstract form into a structural one – a sequent parallel automaton or a system of Boolean functions that can be directly hardware implemented.

The problem of state assignment becomes harder when asynchronous implementation of a parallel automaton is considered. The mentioned condition imposed on codes is necessary but is not enough for that case. The additional condition to be fulfilled is to avoid the influence of races between memory elements (flip-flops) during hardware operation. One of the ways to avoid that is to order switches of memory elements so as to eliminate critical races.

A problem of race-free state assignment of asynchronous parallel automata is considered. The goal is to encode partial states of parallel automaton using minimal number of coding variables and to avoid the critical races during automaton operation. An exact algorithm to find a minimal solution of the problem is suggested. The algorithm allows reducing the computational effort when searching state encoding. The same problem is considered in [5] where another approach was suggested. The method is based on covering a family of complete bipartite subgraphs defining constraints of absence of critical races by minimal number of maximal complete bipartite subgraphs of the state non-parallelism graph.

2. CONSTRAINTS OF ABSENCE OF CRITICAL RACES

The asynchronous sequential automaton behaves as follows. Initially, the automaton is stable in some state. After the input state changes the outputs change their values as specified in automaton description. An internal state change may be concurrently with the output change. After automaton achieving a new stable state it is ready to receive a new input. Throughout this cycle output and inner variables should be free of glitches. In summary asynchronous designs differ from those synchronous since state changes may pass through intermediate states.

The sequence of these intermediate states must be preserved in the case of multi-output change (when intermediate states involve the output change). It can be done with the proper state assignment. The 1-hot encoding [4] can ensure such a behavior, but it demands too many coding variables. That is why the methods of race-free state assignment are of interest.

In [6] the constraints to ensure hardware implementation of sequential automaton to be race-free are given. These constraints allow avoiding interference between automaton transitions that take place for the same input state. The codes satisfying these constraints ensure race-free implementation of the automaton. The encoding constraints can be represented in the form of dichotomies. A dichotomy is a bipartition $\{S_1; S_2\}$ of a subset $S_1 \cup S_2 \subseteq S$ ($S_1 \cap S_2 = \varnothing$). In considered state encoding a binary variable y_i covers dichotomy $\{S_1; S_2\}$ if $y_i = 0$ for every state in S_1 and $y_i = 1$ for every state in S_2 (or vice versa). A pair of transitions taking place at the same input is called below as competitive transitions. In [6] the following constraints of critical race-free encoding are given that are induced by competitive transitions of different types:

1) $s_i \to s_j$, $s_k \to s_l$ (i, j, k, l are pair-wise different) give rise to $\{s_i, s_j; s_k, s_l\}$;
2) $s_i \to s_j$, $s_j \to s_l$ (i, j, l are pair-wise different) give rise to $\{s_i, s_j; s_l\}$ and $\{s_i; s_j, s_l\}$;
3) $s_i \to s_j$, $s_k \to s_j$ (i, j, k are pair-wise different) give rise to $\{s_i, s_j; s_k\}$ if the output on the transition from s_k is different than that on the transitions from s_i and s_j (at the input considered).

A parallel automaton is described by a set of generalized transitions $(X_{kl}, S_k) \to (S_l, Y_{kl})$ between the subsets of partial states. Such a transition should be understood as follows: if the global state of the parallel automaton contains all the partial states from S_k and the variables in the conjunction term X_{kl} assume values at which $X_{kl} = 1$, then as the result of the transition the automaton goes to a new global state that differs from initial one by that it contains partial states from S_l instead of those from S_k. More than one generalized transition may take place in some moment when parallel automaton functions. These transitions define changing different subsets of parallel partial states. There are no races on such a pair of transitions.

In the case of parallel automaton we have generalized transitions instead of elementary ones. A generalized transition t_{kl}: $S_k \to S_l$ consist of $|S_k| \cdot |S_l|$ elementary transitions $s_{ki} \to s_{lj}$, where $s_{ki} \in S_k$ is nonparallel to $s_{lj} \in S_l$. Let us introduce the set $T(t_{kl}, t_{pq})$ of pairs of elementary transitions $s_{ki} \to s_{lj}$ and $s_{pi} \to s_{qj}$ between pair-wise nonparallel partial states taken from S_k, S_l, S_p and S_q generated by the pair of competitive transitions t_{kl}: $S_k \to S_l$ and t_{pq}: $S_p \to S_q$. For compatible pair t_{kl}, t_{pq} of generalized transitions we have $T(t_{kl}, t_{pq}) = \varnothing$.

In [2] it is shown that in order to avoid the influence of races on competitive generalized transitions t_{kl} and t_{pq} it is sufficient to avoid it on one pair of elementary transitions from the set $T(t_{kl}, t_{pq})$. Thus this statement gives the way of a parallel automaton partial states encoding. Besides this statement ensures any dichotomy constraint consists of pair-wise nonparallel partial states that implies the absence of a constraint forcing a coding variable to have orthogonal values in codes of parallel partial states.

Let distinguish elementary u_{ij}, simple u^p_{ni} and generalized U_n constraints. The first one is a single dichotomy constraint. The second one is associated with a pair of elementary transitions and can consist of one (cases 1, 3 of constraints) or two (case 2) elementary constraints. To avoid critical races on a pair of elementary competitive transitions one has to satisfy an appropriate simple constraint (one or two elementary ones). A generalized constraint U_n induced by a pair P_n of competitive generalized transitions consists of the simple constraints induced by pairs of elementary transitions from its generated set $T(P_n)$. To avoid critical races on P_n it is sufficient to satisfy one of the simple constraints from U_n.

Example 1. Let us consider the following parallel automaton in the form $X_{kl}\ S_k \to S_l\ Y_{kl}$:

1.	$'x_1$	$s_1 \to s_2 \cdot s_3$	$y_1 y_2$	5. x_3	$s_3 \to s_6$	y_4
2.	$'x_2 x_3$	$s_2 \to s_9$	$'y_2 y_3$	6. $x_1 'x_2$	$s_4 \to s_7$	$y_1 'y_2$
3.	$'x_3$	$s_9 \to s_2$	$y_2 'y_3$	7. $'x_2 x_3$	$s_5 \to s_8$	$'y_3$
4.	x_2	$s_2 \to s_4 \cdot s_5$	$'y_1 y_3$	8. $'x_3$	$s_6 \cdot s_7 \cdot s_8 \to s_1$	$'y_1 'y_4$

The partial states from $\{s_2, s_4, s_5, s_7, s_8, s_9\}$ and $\{s_3, s_6\}$ are pair-wise parallel as well as partial states from $\{s_4, s_7\}$ and $\{s_5, s_8\}$. One can see, for example, that the pair t_1, t_8 of generalized transitions is competitive. The generalized constraint U_{18} induced by that pair consists of 3 simple constraints: $u^p_1 = (\{s_1, s_2; s_7\}$ and $\{s_1, s_7; s_2\})$, $u^p_2 = (\{s_1, s_2; s_8\}$ and $\{s_1, s_8; s_2\})$ and . $u^p_3 = (\{s_1, s_3; s_6\}$ and $\{s_1; s_3, s_6\})$.

By analogy with the case of sequential automaton [7] the algorithm of critical race-free partial states assignment of parallel automaton has two steps: 1) generate and compress a set of encoding constraints; 2) solve these constraints to produce a partial state assignment.

3. GENERATING AND COMPRESSING A SET OF ENCODING CONSTRAINTS

Now an encoding problem formulation is presented that is based on a matrix notation similar to that used in [7] for sequential automata. A dichotomy constraint $\{s_i, s_j; s_k, s_l\}$ can be presented as a ternary (3-valued) vector called a constraint vector. Its length equals to the number of partial states, i-th and j-th entries are 1, k-th and l-th entries are 0 (or vice versa), and the other ones are "-" (don't care). For example the dichotomy $\{s_1,s_7; s_2,s_9\}$ corresponds to the vector "1 0 - - - - 1 - 0".

The constraint matrix $\boldsymbol{U}$ is a ternary matrix with as many rows as critical race-free constraints exist (for a given automaton) and columns as partial states. The matrix $\boldsymbol{U}$ has a complex structure – it consist of submatrices $\boldsymbol{U}_i$ defining generalized constraints the last ones are in turn 1 or 2 line sectioned (separating simple constraints).

Now we give some definitions having in view ternary vectors of the same length. A ternary vector $\boldsymbol{a}$ covers a ternary vector $\boldsymbol{b}$ if, whenever the i-th entry of $\boldsymbol{b}$ is $\sigma \in \{1,0\}$ i-the entry of $\boldsymbol{a}$ is σ too. $\boldsymbol{b}$ is an inversion of $\boldsymbol{a}$ ($\boldsymbol{b} = '\boldsymbol{a}$) if, whenever the i-th entry of $\boldsymbol{a}$ is 1, 0, "-" the i-the entry of $\boldsymbol{b}$ is 0, 1, "-" respectively. Vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal if for at least an index i the i-th entries of $\boldsymbol{a}$ and $\boldsymbol{b}$ are orthogonal (1 and 0 or vice versa). An elementary constraint u_i implicates an elementary constraint u_j if u_j as a ternary vector covers u_i or its inversion.

A simple constraint $u^p{}_n$ implicates:
– an elementary constraint u_j if u_j is implicated by one of the elementary constraints from $u^p{}_n$,
– a simple constraint $u^p{}_m$ if every $u_{mj} \in u^p{}_m$ is implicated at least by one of $u_{nj} \in u^p{}_n$,
A generalized constraint U_k implicates:
– a simple constraint $u^p{}_j$ (elementary constraint u_j) if every $u^p{}_{kj} \in U_k$ implicates it,
– a generalized constraint U_n if every $u^p{}_{kj} \in U_k$ implicates at least one of $u^p{}_{ni} \in U_n$.

For computational efficiency of procedure of searching an optimal encoding it is important to reduce the number of rows of constraint matrix $\boldsymbol{U}$ to the minimal number that represent an equivalent set of constraints on the encoding. It is trivial that duplicate generalized constraints can be deleted. Then the number of rows of $\boldsymbol{U}$ can be compressed further by discarding generalized constraints that are implicated by any other generalized constraint.

Example 2. For considered automaton we can see that generalized constraint ($\{s_1, s_7; s_2, s_9\}$ or $\{s_1, s_8; s_2, s_9\}$) induced by the pair t_2, t_8 of competitive transitions implicates the elementary constraint $\{s_1; s_2, s_9\}$ from the simple constraint ($\{s_1, s_2; s_9\}$ and $\{s_1; s_2, s_9\}$) induced by the pair t_1, t_2 of competitive transitions. Thus we have the following irredundant set of generalized constraints U_k (in the form of dichotomies) for this automaton:

1. $\{s_1,s_2; s_9\}$,
2. ($\{s_1,s_2; s_4\}$ and $\{s_1; s_2, s_4\}$) or $\{s_1; s_2, s_5\}$,
3. $\{s_1,s_2; s_5,s_8\}$,
4. ($\{s_1,s_2; s_7\}$ and $\{s_1, s_7; s_2\}$) or $\{s_1, s_8; s_2\}$,
5. $\{s_2,s_9; s_4,s_7\}$,
6. $\{s_2,s_4; s_9\}$ or ($\{s_2, s_5; s_9\}$ and $\{s_2, s_9; s_5\}$),
7. $\{s_1,s_7; s_2,s_9\}$ or $\{s_1,s_8; s_2,s_9\}$,
8. $\{s_1,s_7; s_2,s_4\}$ or $\{s_1,s_8; s_2,s_5\}$,
9. $\{s_1; s_4,s_7\}$,
10. ($\{s_1,s_3; s_6\}$ and $\{s_1; s_3,s_6\}$),
11. $\{s_4; s_7\}$,
12. $\{s_5; s_8\}$.

The last two constraints U_{11} and U_{12} are introduced since nonparallel partial states should be encoded with orthogonal codes (but constraint U_8 does not implicates neither U_{11} nor U_{12}).

4. FINDING ENCODING OF PARTIAL STATES

One can see that the matrix $\boldsymbol{U}$ is an encoding matrix $\boldsymbol{V}$, but the number of coding variables (equaled to the number of rows) is too big. The encoding matrix $\boldsymbol{V}$ is grown from an initial seed constraint matrix $\boldsymbol{U}$ by its compressing at the expense of combining some constraints and substituting them for one constraint implicating them.

Now we give some definitions and derive some useful properties from them. A constraint u is called an implicant of a set of rows of constrained matrix $\boldsymbol{U}$ if it implicates each of them taken separately. A set $U_j \in U$ is considered as compatible if there exists its implicant having no orthogonal entries associated with parallel states. For example the single $u_{11} \in U_1$ is compatible with $u_{31} \in U_3$ (its implicants are $\{s_1,s_2;\ s_5,s_8,s_9\}$, $\{s_1,s_2;\ s_4,s_5,s_8,s_9\}$, $\{s_1,s_2;\ s_4,s_5,s_7,s_8,s_9\}$), but not compatible with $u_{11} \in u^p{}_1 \in U_{10}$.

We can simply find whether two rows $\boldsymbol{v}_k \in \boldsymbol{V}$ and $\boldsymbol{u}_l \in \boldsymbol{U}$ (or both from $\boldsymbol{U}$) are compatible when using the notion of a boundary vector suggested in [3]. The boundary vector for any row $\boldsymbol{u}_k \in \boldsymbol{U}$ (or $\boldsymbol{v}_k \in \boldsymbol{V}$) is - 4-valued vector that gives an upper bound of its grows (extension) i.e. it determines the potential of verifying the components of $\boldsymbol{u}_k$. In [3] the operations over 3- and 4-valued vectors are given that help simply to find implicants.

When concatenating two rows $\boldsymbol{v}_k$ and $\boldsymbol{u}_l$ (constructing their implicant) we do minimal extension of $\boldsymbol{v}_k$ to implicate $\boldsymbol{u}_l$. In this way any i-th entry of the result of concatenation is equal to that of $\boldsymbol{v}_k$ and $\boldsymbol{u}_l$ (or '$\boldsymbol{u}_l$). $\boldsymbol{v}_k \in \boldsymbol{V}$ is an implicant for generalized constraint $\boldsymbol{U}_l \in \boldsymbol{U}$ if it is implicant for some $\boldsymbol{u}^p{}_{li} \in \boldsymbol{U}_l$. An implicant of a subset U' of generalized constraints is maximal if it is incompatible with all those others (it cannot implicate any more generalized constraints besides those from U'). For example the implicant $\{s_1,s_2;\ s_4,s_5,s_7,s_8,s_9\}$ is maximal, but $\{s_1,s_2;\ s_5,s_8,s_9\}$ is not.

An exact algorithm to find a minimum solution of the problem of race-free state assignment is based on building a set C of all maximal implicants for constraint matrix $\boldsymbol{U}$ and then searching a subset of $V \subseteq C$ of minimal cardinality such that for any generalized constraint $U_i \in U$ there exists an implicant in V implicating it. The second part of the problem is reduced to covering problem of Boolean matrix [7], as in the case with Quine's table.

4.1. Search of maximal implicants

We use branch and bound algorithm to build all maximal implicants. Constraints are processed one by one in predefined order choosing (at each step) one compatible with the current state of the implicant formed. If we exhaust such constraints we would start backtracking to a previous step to modify the solution and repeat searching.

The computational efforts can be reduced using a previously generated compatibility relation on the rows from $\boldsymbol{U}$. Taking into account that any maximal implicant may satisfy only one of the simple constraints from each generalized constraint they all can be regarded as pair-wise incompatible. At each step of the algorithm it is enough to consider as candidates for concatenating only those rows compatible with all concatenated in the current implicant. Further search reduction can be received by sorting the constraints according to the degree of their incompatibility: the greater it is the less is branching.

For the automaton considered there exist 17 maximal implicants.

4.2. Covering problem statement

Once a set C of maximal implicants is found the task is to extract from it a subset that satisfy all generalized constraints $U_k \subset U$. Every $U_k = \{u_{k1}{}^p, u_{k2}{}^p, \ldots, u_{kn}{}^p\}$ is satisfied as OR (though one of $u_{ki}{}^p$ should be satisfied) and $u_{ki}{}^p$ consist of one or two u_{ij} that are satisfied as AND (both u_{i1} and u_{i2} should be satisfied). These statements can be expressed logically (as it is suggested in [5]) by the formulas: $U_k = u_{k1}{}^p \vee u_{k2}{}^p \vee \ldots \vee u_{kn}{}^p$ and $u_{ki}{}^p = u_{i1} \cdot u_{i2}$. Substituting expressions $u_{ki}{}^p$ into U_k and using the distributive low one can receive conjunctive normal form $U_k = U_k{}^1 \cdot U_k{}^2 \cdot \ldots \cdot U_k{}^m$. Any $U_k{}^i$ is a union of separate elementary constraints. For example generalized constraint U_2 (from example 2) is represented as $(\{s_1,s_2;\ s_4\}$ or $\{s_1;\ s_2,\ s_5\}) \cdot (\{s_1;\ s_2,\ s_4\}$ or $\{s_1;\ s_2,\ s_5\})$.

Now the problem is stated in the form of Quine's table Q. Its rows correspond to implicants $C_i \in C$ and columns to conjunctive members U_k^i for all U_k. An entry (ij) of Q is 1 (marked) if C_i implicates j-th conjunctive member. The task is to find the minimal number of rows covering all columns (every column should have 1 at least in one position corresponding to rows chosen) [7]. The cover presenting encoding for automaton considered (examples 1,2) contains 5 rows. So we find 5 components codes of partial states that provide the absence of critical races when the automaton operates:

$$V = \begin{bmatrix} 1 & 1 & - & 0 & 0 & - & 0 & 0 & 1 \\ 1 & 0 & - & - & 0 & - & - & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & - & - \\ 1 & - & 1 & - & - & 0 & - & - & - \\ 1 & 1 & - & 1 & - & - & 0 & - & 0 \end{bmatrix}$$

It should be noted that some entries of matrix equal to 0 or 1 can be substituted with value don't care because of usage of maximal implicants.

5. CONCLUSION

Unfortunately the problems considered are computationally hard ones. The growth of the computation time as the size of the problem increases is a practical limitation of the method suggested to computer-aided design systems. It can be used for solving encoding problems of moderate size obtaining after decomposing the whole big problem. Besides the method can be useful for estimation of efficiency of heuristic encoding techniques [3].

REFERENCES

[1] M.Adamski, M.Wegrzyn, "Field programmable implementation of current state machine", *Proc.of the Third Int. Conf.on Computer-Aided Design of Discrete Devises* (*CAD DD'99*), Minsk, Institute of Engineering Cybernetics of the of Belarus Academy of Sciences, Vol. 1, pp. 4-12, 1999

[2] L.D.Cheremisinova, "Implementation of parallel digital control algorithms by asynchronous automata", *Automatic Control and Computer Sciences*, Vol. 19, No. 2, pp. 78 – 83, 1985

[3] L.D.Cheremisinova, "Race-free state assignment of a parallel asynchronous automaton", *Upravlyajushchie sistemy i mashiny*, No 2, pp. 51-54, 1987 (in Russian)

[4] L.D.Cheremisinova, "PLC Implementation of concurrent control algorithms", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Minsk, Republic of Belarus, Sept. 5-6, pp. 190-196, 2000

[5] Yu.V.Pottosin, "State assignment of asynchronous parallel automata with codes of minimum length", *Proc.of the Int. Workshop "Discrete Optimization Methods in Scheduling and Computer-Aided Design"*, Minsk, Republic of Belarus, Sept. 5-6, pp. 202-206, 2000

[6] S.H.Unger, *Asynchronous sequential switching circuits*, Wiley-Interscience, New York, 1969

[7] A.D.Zakrevskij, *Logical synthesis of cascade networks*, Nauka, Moscow, 1981 (in Russian)

[8] A.D.Zakrevskij, "Parallel automaton", *Doklady AN BSSR*, Vol. 28, No. 8, pp. 717 – 719, 1984 (in Russian)

[9] A.D.Zakrevskij, *Parallel algorithms for logical control*, Minsk, Institute of Engineering Cybernetics of NAS of Belarus, 1999. (in Russian)

SESSION III:
INTEGRATED CIRCUITS

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

FUNCTIONAL DECOMPOSITION – THE VALUE AND IMPLICATION FOR MODERN DIGITAL DESIGNING

Mariusz RAWSKI†, Tadeusz ŁUBA†, Zbigniew JACHNA‡,
Rafał RZECHOWSKI†

† Warsaw University of Technology, Institute of Telecommunications, Nowowiejska 15/19,
00-665 Warsaw, Poland, e-mail: *rawski@tele.pw.edu.pl*

‡ Military Academy of Technology, Kaliskiego 2, Warsaw, Poland,
e-mail: *Zjachna@wel.wat.waw.pl*

Abstract. *General functional decomposition has important applications in many fields of modern engineering and science. However, it is mainly perceived as a method of logic synthesis for implementation of Boolean functions into FPGA-based architectures. In this paper, an application of functional decomposition in other fields of modern engineering is presented. The experimental results demonstrate that a method of synthesis based on functional decomposition can help in implementing sequential machines using flip-flops or ROM memory. It also can be efficiently used as a method of multilevel logic synthesis for VLSI technology.*

Key Words. *Logic Synthesis, Functional Decomposition, FPGA, VLSI*

1. INTRODUCTION

Decomposition has become an important activity in the analysis and design of digital systems. It is fundamental to many fields of modern engineering and science [3], [6], [9], [16]. Functional decomposition relies on breaking down a complex system into a network of smaller and relatively independent co-operating sub-systems, in such a way that the original system's behaviour is preserved, i.e. function F is decomposed to subfunction G and H in the form described by formula F = H(A, G(B)).

Recently, new methods of logic synthesis based on functional decomposition are being developed [1], [4], [7]. One of the promising decomposition-based methods is the so-called balanced decomposition [11].

Thanks to the fact that the multi-level functional decomposition gives very good results in the logic synthesis of combinational circuits, it is viewed for the most part as a synthesis method addressed to implementation of combinational functions into FPGA-based architectures [13], [15]. However, a decomposition-based method can be used beyond this field. Since in the sequential machine synthesis after state code assignment the process of implementation is

reduced to the computation of flip-flops' excitation functions, the decomposition can be efficiently used to assist said implementation. Application of a balanced decomposition method allows the designer to decide what is the optimisation criterion – circuit area or circuit speed. Good results produced by decomposition-based logic synthesis methods in implementation of combinational circuits guarantee that this method will implement encoded sequential machines efficiently and effectively. The balanced decomposition gives the designer control over the process of excitation functions' implementation. Thanks to this, such undesirable effects as hazards can be avoided. Elimination of these effects can increase the speed of circuits.

Modern FPGA architectures contain embedded memory blocks. In many cases, designers do not need to use these resources. However, such memory blocks allow implementing sequential machines in a way that requires less logic cells than traditional, flip-flop implementation. This may be used to implement "non-vital" sequential parts of the design saving logic cell resources for more important parts. However such an implementation may require more memory than available in a circuit. To reduce memory usage in ROM-based sequential machine implementations decomposition-based methods can be successfully used [10].

Decomposition-like synthesis methods are not limited only to FPGA-based architectures [8]. The balanced functional decomposition can also be used to implement digital systems in CPLD and even VLSI technology (gate-array or standard cell). An appropriately chosen decomposition strategy allows circuit synthesis, with results comparable or even better than those achieved with classical multi-level synthesis methods based on algebraic transformation. Additionally, decomposition-based methods produce logic networks that do not require technology mapping.

In this paper, once some basic information has been introduced, the application of the decomposition to implementation of sequential machines is presented. Following that, an algorithm of implementation of function in VLSI technology, a variant of balanced functional decomposition, is presented. Subsequently, some experimental results, reached with a prototype tool that implements the balanced functional decomposition, are discussed.

The experimental results demonstrate that the decomposition is capable of constructing solutions of comparable or even better quality than the methods implemented in university or commercial systems.

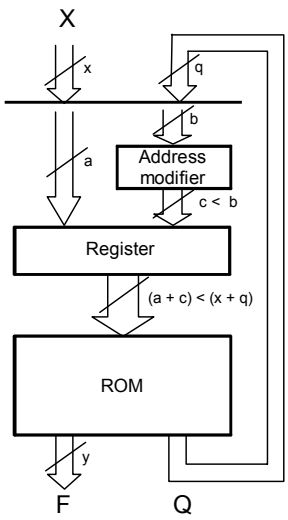


Fig. 1. Implementation of FSM using an address modifier

2. FINITE STATE MACHINE IMPLEMENTATION

The FSM can also be implemented through the use of ROM (*Read Only Memory*) [10].

The FSM defined by a given transition table can be implemented in a structure shown in Fig. 1 by means of an address modifier. The process may be considered as a decomposition of memory block into two blocks: a combinational address modifier and a smaller memory block. Appropriately chosen strategy of balanced decomposition may allow to reduce required memory size at the cost of additional logic cells for address modifier implementation. This makes possible to implement FSM that exceed available memory through using embedded memory blocks and additional programmable logic.

3. DECOMPOSITION INTO GATES

Different logic synthesis programs use different techniques to transform a synthesised network of logic gates into a form that is implementable in standard-cell or gate-array technology. These methods use either a multi-level representation of function for structural technology mapping (SIS system), i.e. a matching of the topology of synthesised network to the gate patterns defined in the technology library, or, for logical technology mapping, a calculation of the logical coverage of sub-nets of multi-level function structures by the functions defined by technology gates. However, if the multilevel structure does not respect technology constraints or is not built with a close relation to the actual target, both methods of technology mapping will disturb the optimally synthesised network, leading to losses in quality of the final product [5].

The obvious conclusion is that there is a need to use algorithms, which allow the designer to implement logic structures directly in gate level representation, with special focus on VLSI technology constraints. Attempts are being made to use evolutional algorithms or to use special properties of logic functions, such as, for example, symmetries of inputs. At the current stage, those algorithms have to be supported by the other techniques due to the maximum block size they can synthesise and the class of logic function on which they can operate [14].

Balanced functional decomposition, as used to synthesise gate-level circuits, is free of these disadvantages [16]. It permits the elimination of the whole technology mapping process and can be used to synthesise large circuits consisting of any gates available in the technology library. At the same time, functional decomposition remains a homogeneous synthesis method and does not need any supporting algorithms to perform its operations.

In the herein discussed method of functional decomposition, the technology mapping process is performed during serial decomposition, at stage of coding generation for G block's outputs. Coding of outputs of the G block allows for many variants and can be used to adjust synthesis parameters to meet technology targets and to satisfy the logical coverage of synthesised blocks by technology gates. This allows optimisation of not only the number of gates (area) of the implementation but also such implementation criteria as speed [13] and power consumption [2].

Experimental results show that such a solution gives very good results when used in stand-alone synthesis procedures, as well as in algorithms supporting other synthesis methods such as evolutional algorithms.

4. EXPERIMENTAL RESULTS AND CONCLUSIONS

The balanced decomposition was applied to implement in FPGA architectures several "real life" examples: combinational functions and combinational parts of FSMs. We used the following examples:

- bin2bcd1 – binary to BCD converter for binary values from 0 to 31,
- bin2bcd2 – binary to BCD converter for binary values from 0 to 355,
- rd88 – Sbox from Rijndael implementation,
- DESaut – combinational part of the state machine used in DES algorithm implementation,
- 5B6B – the combinational part of the 5B-6B coder,
- count4 – 4 bit counter with COUNT UP, COUNT DOWN, HOLD, CLEAR and LOAD.

Table 1

Example	***FPGA based architecture EPF10K10LC84-3***				
	DEMAIN	***MAX+Plus II***	***FPGA Express***	***Leonardo Spectrum***	***SIS***
bin2bcd1	6	41	6	6	6
bin2bcd2	39	505	225	120	136
rd88	167	332	341	245	248
DESaut	28	46	25	30	32
5B6B	41	92	100	49	51
count4	11	74	17	11	13

For the comparison following synthesis tools were used: MAX+Plus II ver. 10 Baseline, FPGA Express 3.5, Leonardo Spectrum ver. 1999.1, SIS and DEMAIN. Logic network produced by all synthesis tools were implemented in EPF10K10LC84-3, the FPGA device from FLEX family of Altera. For VLSI implementation comparison, examples from a standard benchmark set were used [17].

Application of decomposition methods in area of machine learning was demonstrated with use of examples from a standard benchmark set as well as functions from area of knowledge-based systems.

Table 1 shows the comparison of our method based on balanced decomposition as implemented in tool DEMAIN with other methods compared tools. The table shows the comparison of logic cells needed for implementation of given examples. Results of implementation in FPGA architecture show that the method based on balanced decomposition gives better results than other tools used in the comparison. The worst results are produced by the Altera MAXPlus+II.

Table 2

Example	***FF_MAX+PlusII***		***FF_DEMAIN***		***ROM***		***AM_ROM***	
	LCs/Bits	***Speed [MHz]***	***LCs/Bits***	***Speed [MHz]***	***LCs/Bits***	***Speed [MHz]***	***LCs/Bits***	***Speed [MHz]***
DESaut	46/0	41,1	28/0	61,7	8/1792	47,8	7/896	47,1
5B6B	93/0	48,7	43/0	114,9	6/448	48,0	– [3]	– [3]
count4	72/0 18/0 [1]	44,2 86,2 [1]	11/0 13/0 [2]	68,5 90,0 [2]	16/16384	– [4]	12/1024	39,5

[1] FSM described with special AHDL construction; [2] decomposition with the minimum number of logic levels, [3] decomposition not possible; [4] not enough memory bits to implement the project

Since, upon the encoding of the FSM's states, the implementation of such FSM architectures involves the technology mapping of the combinational part into target architecture, the quality of such an implementation strongly depends on combinational function implementation quality. In Table 2 a comparison of different FSM implementations are presented. Each sequential machine was described by a transition table with encoded states. We here present the number of logic cells and memory bits required (i.e. area of the circuit) and the maximal frequency of clock signal (i.e. speed of the circuit) for each method of FSM implementation. The columns falling under the FF_MAX+PlusII heading present results obtained by the Altera MAX+PlusII system in a classical flip-flop implementation of FSM. The columns under FF_DEMAIN show results of implementation of the transition table with the use of balanced decomposition. The ROM columns provide the results of ROM implementation; the columns under AM_ROM present the results of ROM implementation with use of address modifier. It can be easily noticed that the application of balanced decomposition can improve the quality of flip-flop as well as ROM implementation. Especially interesting is the implementation of the 4-bit counter. Its description with a transition table leads to strongly non-optimal implementation. On the other hand, its description when using a special Altera HDL (Hardware Description Language) construction produces very good results. However, utilization of balanced decomposition allows the designer to choose between whether area or speed is optimized. The ROM implementation of this example requires to many memory bits (the size of required memory block exceeds the available memory), thus it can not be implemented in given structure. Application of functional decomposition allows reducing the necessary size of memory, what makes implementation possible.

Results presented in Table 3 demonstrate the performance of decomposition-based method in implementing the digital circuits into two-input gates. The comparison was performed for a set of standard benchmarks [17]. The comparison of the results was carried out among the functional decomposition method, a classical, multi-level synthesis approach (represented by algorithms of the SIS system [5]), an evolutional algorithm supported by two different versions of decomposition (the Shannon decomposition and a balanced functional decomposition) [14], and decomposition supported by a function's information relationship measures [7].

Table 3

Example	***Functional decomposition algorithm***	***SIS (with modified rug script.)***	***Evolutional algorithm supported by***		***Gendec – decomposition supported by function's relationship measures.***
			Shannon decomposition + MUX	***balanced decomposition***	
xor5	4	8	–	–	4
shift	21	20	–	–	15
rd53	18	22	21+3	16	36
lion	14	13	–	–	15
alu2	61	60	512+63	71	–
adr4	27	21	–	–	–
9sym	36	188	144+31	37	–
5xp1	76	91	121+15	61	–

Balanced decomposition produces very good results in combinational function implementation in FPGA-based architectures. However, results presented in this paper show that balanced functional decomposition can be efficiently and effectively applied beyond the

implementation of combinational circuits. It can also be used in other fields of modern engineering.

ACKNOWLEDGEMENT

This paper was partially supported by project Grant KBN No. 8T11B08817.

REFERENCES

[1] Burns M., Perkowski M., Jóźwiak L.: "An Efficient Approach to Decomposition of Multi-Output Boolean Functions with Large Set of Bound Variables", *EUROMICRO'98 Conference*, Vasteras, Sweden, 1998

[2] Brzozowski I., Kos A..: "Minimisation of Power Consumption in Digital Integrated Circuits by Reduction of Switching Activity", *Proc of the Euromicro Conference*, pp.376-380. Vol. 1, Milan 1999.

[3] Brzozowski J. A., Łuba T.: "Decomposition of Boolean Functions Specified by Cubes", *Research Report CS-97-01*, University of Waterloo, Waterloo, Canada, 1997; REVISED October 1998.

[4] Chang S.C., Marek-Sadowska M., Hwang T.T.: "Technology Mapping for TLU FPGAs Based on Decomposition of Binary Decision Diagrams", *IEEE Trans. on CAD*, vol.15, No. 10, October, 1996, pp. 1226-1236.

[5] De Micheli G.: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York., 1994.

[6] Hartmanis J., Stearns R. E.: *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, 1966.

[7] Jóźwiak L., Chojnacki A.: "Functional Decomposition Based on Information Relationship Measures Extremely Effective and Efficient for Symmetric Functions", *Proc of the Euromicro Conference*, pp.150-159. Vol. 1, Milan 1999.

[8] Kravets V. N., Sakallah K. A.: "Constructive library-aware synthesis using symmetries", *Proc. Of Design, Automation and Test in Europe Conference*, 2000

[9] Łuba T.: "Multi-level logic synthesis based on decomposition". *Microprocessors and Microsystems*, 18, No. 8, pp. 429-437, 1994.

[10] Łuba T., Górski K., Wronski L.B.: "ROM-based Finite State Machines with PLA address modifiers", *EURO-DAC'92*, Hamburg, Germany, September, 1992

[11] Łuba T., Selvaraj H., Nowicka M., Kraśniewski A.: *Balanced multilevel decomposition and its applications in FPGA-based synthesis*. in G.Saucier, A.Mignotte (ed.), Logic and Architecture Synthesis, Chapman&Hall, 1995.

[12] Łuba T., Selvaraj H.: "A General Approach to Boolean Function Decomposition and its Applications in FPGA-based Synthesis". *VLSI Design, Special Issue on Decompositions in VLSI Design*, vol. 3, Nos. 3-4, pp. 289-300, 1995.

[13] Łuba T., Nowicka M., Rawski M.: "Performance-oriented Synthesis for LUT-based FPGAs", *Proc. Mixed Design of Integrated Circuits and Systems*, pp. 96-101, Lodz, 1996.

[14] Łuba T., Moraga C., Yanushkevich S., Opoka M., Shmerko V.: "Evolutionary Multilevel Network Synthesis in Given Design Style", *Proc. IEEE 30th Int. symposium on Multiple-Valued Logic*, pp.253-258, Portland,2000

[15] Rawski M., Jóźwiak L., Łuba T.: "Functional Decomposition with an Efficient Input Support Selection for Sub-functions Based on Information Relationship Measures", *Journal of Systems Architecture 47*, pp. 137-155, 2001.

[16] Rzechowski R., Jóźwiak L., Łuba T.: "Technology Driven Multilevel Logic Synthesis based on Functional Decomposition into Gates", *Proc. of the 25th EUROMICRO Conference*, pp. 368-375, Milan, Italy 1999.

[17] Collaborative Benchmarking Laboratory, Department of Computer Science at North Carolina State University, *http://www.cbl.ncsu.edu/*

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

IMPLEMENTATION OF THE FSM INTO FPGA

Hana KUBÁTOVÁ

Department of Computer Science and Engineering, Czech Technical University in Prague,
Karlovo nám. 13, 121 35 Prague 2, Czech Republic, *Phone*: +42 2 2435 7281
Fax: +42 2 2492 3325, *e-mail: kubatova@fel.cvut.cz*

Abstract. *This paper deals with the possibility of description and decomposition of the finite state machine (FSM). The aim is to obtain better placement of a designed FSM to the selected FPGA. It compares several methods of coding of the FSM internal states with respect to the space (number of the CLB blocks) and time characteristics. It evaluates the FSM benchmarks and looks for such qualitative properties to choose the best method for coding before performing all FOUNDATION algorithms because this process is time consuming. The new method for coding of the internal FSM states is presented. All results are documented by experiments.*

Key Words. *Finite state machine (FSM), Hardware Description Language (VHDL), state transition graph (STG), code method, decomposition, benchmark*

1. INTRODUCTION

Most research reports and other materials devoted to searching of the "optimal" coding of the internal states of *FSM* are based on minimal number of internal states and sometimes also on minimal number of used flip-flops in their hardware realization. The only method how to get the really optimal results is testing of all possibilities, [1]. But sometimes "wasting" of the internal states or flip-flops is better solution due to speed of the designed circuit. The most coding methods are not based on recently used structures, like different types of FPGA or CPLD. Therefore we try to compare several types of sequential circuit benchmarks to search the relation between the type of this circuit (number of the internal states, inputs, outputs, cycles, branching) and the coding method with respect to their implementation by *XILINX FPGA.*

We have worked with the CAD system *XILINX FOUNDATION* v2.1i during all our experiments. We have used the benchmarks from the Internet in KISS2 format, some coding algorithms from *JEDI* program and system *SIS 1.2* [7]. First of all we have classified the FSM benchmarks to know the quantitative characteristics of them: number of internal states, inputs, outputs, transitions (i.e. the number of arcs in the state transition graph - STG), maximal number of input arcs, maximal number of output arcs to and from STG nodes, etc. We have compared eight coding methods: "one-hot", binary, Johnson and Gray that are implemented in *FOUNDATION* CAD system; we have implemented Fan-in and Fan-out oriented algorithms

and the algorithm “FAN” connecting Fan-in and Fan-out ones [1], [5] and our original method called “own” that will be presented in this paper. The second group of our experiments has been directed to the decompositions of the FSM. The final results (number of the CLB blocks and maximal frequency) were obtained for concrete FPGA implementation (Spartan XCS05-PC84).

2. METHODS

2.1. Coding methods

“One hot” method uses the same number of bits as the number of internal states - the great number of internal variables is the main disadvantage of this method. The states, that have the same next state for a given input, should be given adjacent assignments ("Fan-out oriented"). The states, that are the next states of the same state, should be given adjacent assignments ("Fan-in oriented"). The states, which have the same output for a given input should be given adjacent assignments (this will help to cover the 1's in the output Karnaugh-maps; "output oriented"). Very popular and frequently used method is the binary code, that uses the minimum number of internal variables, and the Gray code with the same characteristics and adjacent codes for a sequence of states. First partial results based on these 7 methods and benchmarks characteristics were presented in [3]. We have found out, that binary coding is better than “one hot” coding for those FSM, which fulfil the following condition: STG that describes the FSM should be complete or nearly complete. If the ratio of average output degree of a node to the number of states is greater than 0.7, than it is better to use the binary coding. On the contrary, when this ratio is low, “one hot” coding is better. This qualitative characteristic property of the FSM benchmarks is defined as:

$$AN = \text{AverageOutEdges}/(\text{NumberOf States} - 1) \quad (1)$$

The value $AN = 0.7$ were verified on benchmarks and on our specially generated testing FSM [4].

2.2. Method “own”

Our original method combines the “one-hot” and binary coding methods. It is based on the partially FSM internal state decomposition. The global algorithm could be described as follows:

a) All FSM internal states Q_i are placed to the set S_0 – not yet classified states

b) From all S_0 elements select the state Q_i with the most number of transitions to the another disjoint states from S_0. This state Q_i is taken away from S_0 and becomes the first member of the new set S_{group}

c) Construct the set of neighbour internal states of all members of S_{group} – $S_{neighbour}$. Compute the score [4], that expresses the placement suitability for a state Q_j into S_{group}, for all states from $S_{neighbour}$. The state with the highest score add to S_{group} . The score is a sum of:

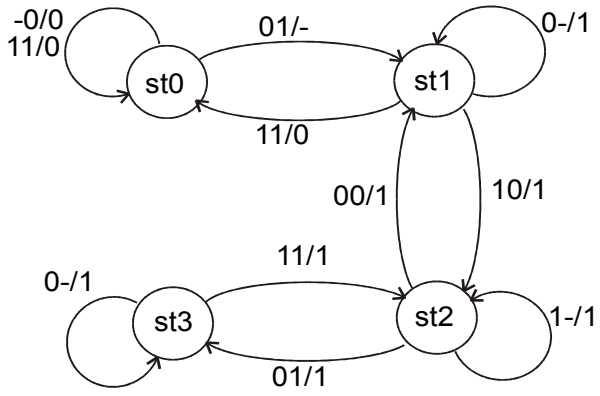
- The number of the transitions from Qj to all states from S_{group} multiplied by the constant 10;
- The number of such states from S_{group} the transition exists from Qj into those ones multiplied by the constant 20;
- The number of the transitions from Qj to all neighbour internal states from S_{group} (i.e. to all states from $S_{neighbour}$) multiplied by the constant 3;

- The number of such states from $S_{neighbour}$ the transition exists from Qj to those ones multiplied by the constant 6;
- The number of the transitions from all internal states from S_{group} to Qj multiplied by the constant 10;
- The number of such states from S_{group} the transition exists from those ones into Qj multiplied by the constant 20;
- The number of the transitions from all neighbour states of S_{group} (placed in $S_{neighbour}$) to Qj multiplied by the constant 3;
- The number of the neighbour states in $S_{neighbour}$ the transition exists from those ones to Qj multiplied by the constant 6;

d) Compute the AN (1) ratio for S_{group}. When this ratio is grater then the "border ratio" (the input parameter of this algorithm, according our experiments usually 0.7) the state Qj becomes the real member of S_{group}. Now continue by step c). When the ratio is less then the "border ratio", state Qj is discarded from the S_{group} and this set is closed. Now continue by step b).

e) When all internal states are placed into some set S_i and S_0 is empty, the internal state code can be constructed. It is connected from the binary part (serial number of the state in its set in binary notation) and the one-hot part (serial number of a set in one-hot notation). The number of binary part bits is equal to b where 2^b greater or equal to the maximum number of states in sets. The number of one-hot part bits is equal to the number of sets S_i.

Example (*lion* benchmark [7], border ratio 0.7):



Obr.1. STG of the *lion* benchmark

a) All FSM internal states Q_j are placed to the set S_0 – not yet classified states

$$S_0 = \{st0, st1, st2, st3\}$$

b) For all S_0 elements compute the number of transitions to the another disjoint states from S_0 (st0…1, st1…2, st2…2, st3…1). Choose the state with the highest value and construct the new set S_1:

$$S_0 = \{st0, st2, st3\}, S_1 = \{st1\}$$

c) Construct the set of neighbour internal states of all members of S_1 – $S_{neighbour}$:

$$S_0 = \{st0, st2, st3\}, S_1 = \{st1\}, S_{neighbour} = \{st0, st2\}$$

Compute the score for all states from $S_{neighbour}$:

$$st0_{score} = 1.10+1.20+2.3+1.6+1.10+1.20+2.3+1.6 = 84$$

$$st2_{score} = 1.10+1.20+1.3+1.6+1.10+1.20+1.3+1.6 = 78$$

Choose the state with the highest score and add it to S_1:

$$S_0 = \{st2, st3\},\ S_1 = \{st0, st1\},\ S_{neighbour} = \{st2\}$$

d) Compute the *AN* (1) ratio for the elements from S_1: $AN = 1.0$. *AN* is grater then 0.7, therefore the state *Qj* becomes the real member of S_1. Now continue by step c).

c) Try to add the state *st2* into S_1 and compute the *AN*. Because $AN = 0.66$ state *st2* is discarded from the S_1 and this set is closed. Now continue by step b).

At the end all internal states are placed into 2 groups:

$$S_1 = \{st0, st1\},\ S_2 = \{st2, st3\}$$

Now internal state code is connected from the one bit binary part and the two bits one-hot parts:

st0 … 0/01

st1 … 1/01

st2 … 0/10

st3 … 1/10

3. EXPERIMENTS

The conversion program between KISS2 format and *VHDL* was necessary to build - we have implemented the converter *K2V_DOS* (in *C++* by translator *GCC* for *DOS OS*) [3], [4]. The *K2V_DOS* program allows an acquisition of information about the FSM like e.g.: node degree, number of states, number of transitions, etc. The FSM in the *VHDL* description, that was created by the *K2V_DOS* program, can be described by different ways (with different results):

- one big process sensitive to the clock signal and to the input signals (one *case* statement is used in this process - it selects active state and in each branch of the *case* there are *if* statements, which define next states and outputs - this is the same method, like the *XILINX FOUNDATION* uses for conversion between STG and *VHDL* [8]);
- three processes (*next-state-proc* for implementation of the next-state function, *state-dff-proc* for asynchronous reset and using D flip-flops and *output-proc* for the FSM output function realization). To overcome the *XILINX FOUNDATION* optimization for the "one-hot" coding method we have used direct code assignment, too.

The *K2V_DOS* program system can generate our special testing FSM (for more precise setting of the "border ratio" *AN*). We have generated the Moore type FSM with the determined number of internal states and mainly the determined number of the transitions from the internal states. Our FSM has the STG with the strictly defined and the same number of transitions from all states. The resulting format is the KISS2 format – e.g. 4.kiss testing FSM has the STG with four edges from every internal state (node). Both the first and also the next state connections are generated randomly to overcome the *XILINX FOUNDATION* optimization for the counter design.

The *K2V_DOS* program can generate different FSM internal state coding by methods binary, Gray, Johnson, one-hot, Fan-in, Fan-out and FAN and "own". All benchmarks were

processed by *DECOMP* program to generate all possible types of decompositions (in KISS2 format due to using the same batch for FPGA implementation).

4. RESULTS

We have performed about 1000 experiments with different types of coding and decomposition methods for 50 benchmarks. We have obtained the great amount of the results processed to the visual graphs. One of them expressing the comparison of the "one-hot", binary and "own 0.7" coding methods with translation of them into three VHDL processes and direct code assignment is presented on Fig. 2.

We can present the following conclusions based on our experiment results:

- the binary coding method gives the best results for FSM with few internal states (5) and for FSM with $AN > 0.7$ (the state transition graph with many cycles)
- "one-hot" coding methods is better for other cases and mostly generates the faster circuits (but the *XILINX FOUNDATION* uses optimization methods for "one-hot" coding)
- the original "own" method is universal one because it combines the advantages of both "one-hot" and binary methods (see Fig. 2)
- for such FSM implementation where the majority of the CLB blocks are used (e.g. 90%) the "one-hot" methods gives better results mainly with respect to the maximum working frequency due to easier wiring
- all FSM decomposition types are not advantageous to use in most cases due to great information exchange - the parallel decomposition is the best one (when it exists)
- the different strategy for looking for the partitions – the best FSM partition is not that one with minimal number of internal states but that one with the minimal sets of input and output symbols – could be used for FPGA implementation

5. ACKNOWLEDGEMENTS

This paper is based on the results of several student projects supervised by the author during two years. This research was in part supported by the internal CTU grant.

REFERENCES

[1] Ashar, P., Devadas, F., Newton, A.R.: "Sequential Logic Synthesis*", Kluwer Academic Publishers*, Boston/Dordrecht/London 1992

[2] Hrdý, T.: "Influence of the FSM decomposition to their implementation", *Diploma thesis*, CTU Prague, FEL, 2001 (in Czech)

[3] Kubátová, H., Hrdý, T., Prokeš, M.: "Problems with the Encoding of the FSM with the Relation to its Implementation by FPGA", *ECI2000 Electronic Computers and Informatics, International Scientific Conference*, Herl'any 2000.

[4] Prokeš, M: "Influence of the FSM internal states codind to their implementation", *Diploma thesis*, CTU Prague, FEL, 2001 (in Czech)

[5] Studenovský, J.: Coding of internal states of synchronous sequential circuit. *Diploma thesis*, CTU Prague, FEL, 1999 (in Czech)

[6] Výmola, V.: Decomposition of a finite state automaton. *Diploma thesis*, CTU Prague, FEL, 2000 (in Czech)

[7] Benchmarks test
ftp://ftp.mcnc.org/pub/benchmark/Benchmark.dirs/LGSynth93/LGSynth93.tar
[8] The Programmable Logic Data Book. XILINX Tenth Anniversary, 1999
http://www.xilinx.com

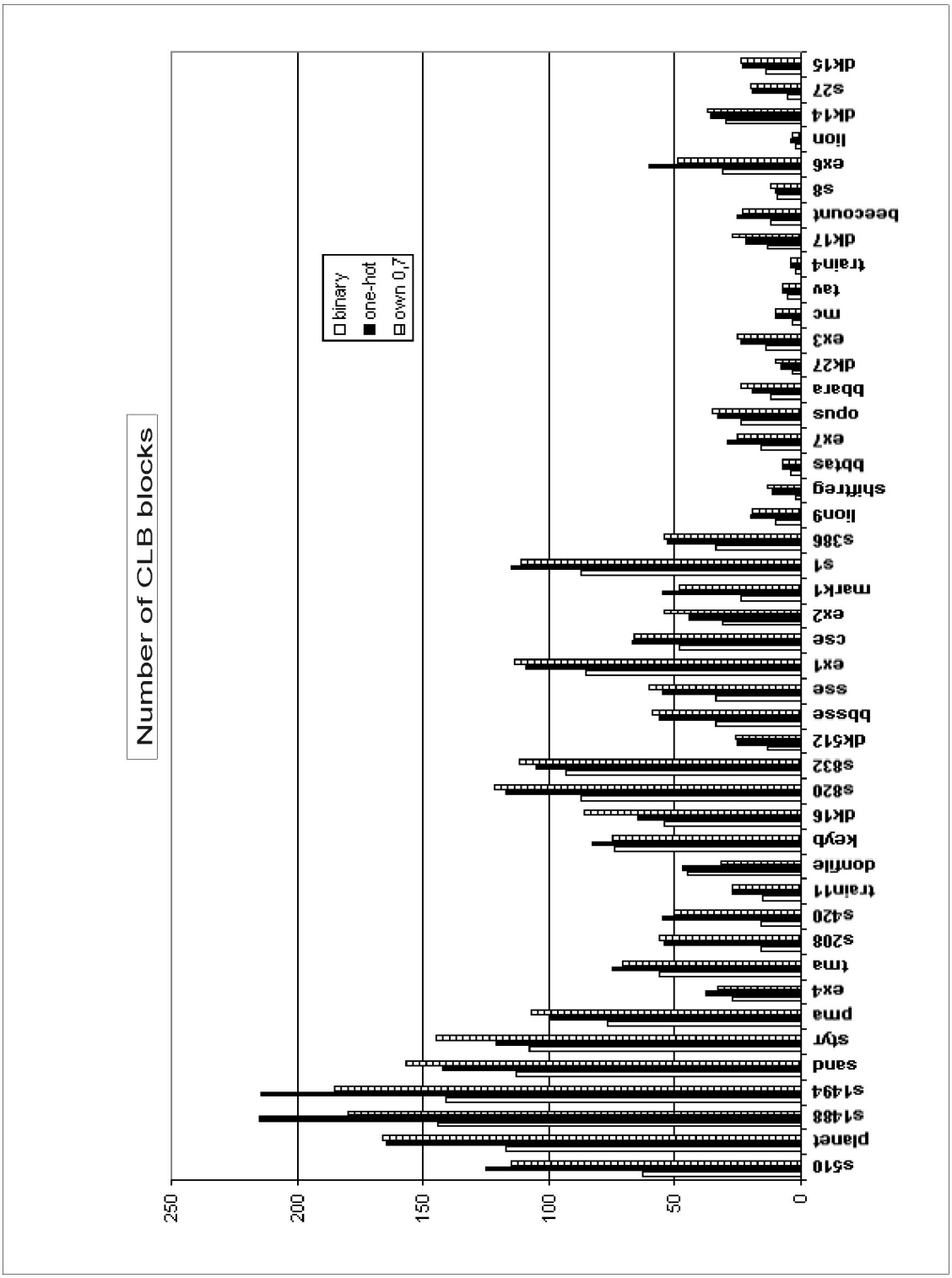


Fig. 2. Number of used CLB blocks for all processed benchmarks

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

REMARKS ON PARALLEL BIT-BYTE CPU STRUCTURES OF PROGRAMMABLE LOGIC CONTROLLERS

Mirosław CHMIEL, Edward HRYNKIEWICZ

Institute of Electronics, Silesian University of Technology, Akademicka 16
44-100 Gliwice, Poland, tel. (+48 32) 2371316, (+48 32) 2372461
e-mail: *chmiel@boss.iele.polsl.gliwice.pl*, *eh@boss.iele.polsl.gliwice.pl*

Abstract. *The paper presents some hardware solutions for the bit-byte CPU of a PLC, which are oriented for maximum optimisation of data exchange between the CPU processors. The optimisation intends to maximum utilisation of the possibilities given by the two-processor architecture of the CPUs. The key point is preserving high speed of instruction processing by the bit processor, and high functionality of the byte processor. The optimal structure should enable the processors to work concurrently for as much of the tome as possible, and minimise the situations, when one processor has to wait for the other.*

Key Words. *CPU, Logic Control, PLC*

1. INTRODUCTION

One of the main parameters (features) of Programmable Logic Controllers (PLC) is execution time of one thousand of control commands. This parameter evaluates the quality of PLC. Due to this fact it is important to design and construct a CPU with a structure enabling fast control program execution. The most developed CPUs of PLCs of many well-known manufacturers are constructed as the multiprocessor units. Particular processors in such units execute the commission for them tasks. In this way, we obtain a unit, which makes possible parallel operating of several processors. For such CPU the main problem is the way of task assuming to particular processors and finding a structure of CPU capable to solve such assigning task in practice.

The bit-byte structure of CPU, in which task assignment is predefined, are often met in real solutions. The tasks operating on discrete input/outputs are executed by bit-processor. Such processor may be implemented in programmable structures as PLDs or FPGA [3,4]. It makes the positive effects on user programme execution time (fast operating processor). On the other hand a byte-processor (word-processor) is built on the base of standard microprocessors or embedded microcontrollers. The byte processors are used for control of analogue objects, for numeric data processing and for execution of the operations indirectly connected to user (control) program but connected to the operating system of the programmable controller of a

CPU. Set of such operations consists of timer servicing, reading-out of the input states, setting of the outputs, LAN servicing, communication to the personal computer and so on.

Very interesting problem and difficult for realisation in programmable controller is timer module [5]. A time interval is counted, asynchronously to the program loop execution. It causes difficulties with testing of an end of a counted interval. At long time of program loop execution and short counted time intervals serious errors may occur. The accuracy of time intervals counting may be increased by special program tricks but achieving good results is typically connected to prolongation of control program loop. In some programmable controllers the end of time interval counting interrupts the control program and the service procedure of this interrupt is called however the number of interrupts is typically limited and only a few timers can act in this way. That is why it would be worth to reflect on the way of improving of an accuracy of time counting in the programmable controllers. Another problem is related with this matter. As it was mentioned above the scan time is one of the most important parameters of programmable controllers. However it seems that throughput time more precisely describe the dynamic features of a programmable controller. Naturally it may be said that throughput time is closely linked to the scan time unless a programmable controller does not execute a control program in serial cyclic way. Let us imagine a programmable controller, which operates on the rule based on processing of the segments (tasks) of the control program. These segments are triggered only by the changes of the input signals (input conditions). In this situation one can talk about throughput time (response time) but it would be difficult to talk about the time of program loop execution. It would be only possible to define the mean time of program loop execution for given application. For the application where the signals change sparsely the mean time of program loop execution will be much less than the maximum time evaluated for execution of a whole program. In particular applications the certain group of signals may do changes more often the other signals. The segments of the control program triggered by these signals will be executed often than the other segments. To avoid situation where two or more segments are triggered at the same moment it would be necessary to assign the priorities to the control program segments. The described method of programmable controller operation changes the approach to the preparing of control program but it seems to the authors that in such programmable controller the problems with for example timers will be easier. It is not necessary to observe the moment when time interval will be completed. At the end of the time interval counting the suitable segment may be called and executed. It means that the currently executed program segment should be interrupted. It depends on the priorities assigned to the particular program segments.

Such type of programmable controllers CPUs will be the subject of the future work while in this paper the few proposals of programmable controller bit-byte CPU structures are presented. These are CPUs with serial-cyclic program execution but they are structurally prepared to event-triggered operation.

2. THE REQUIREMENTS FOR PROGRAMMABLE CONTROLLER CPUs

The aim of the work which results are described in the paper was design and implementation of programmable controller CPU based on bit-byte structure. The main design condition was maximum speed of control program execution. This condition should be met rather by elaborating of suitable structure than application the fastest microprocessors. Additionally it was assumed that bit processor will be implemented using catalogue logic devices or programmable structures while as the byte processor will be used the microcontroller 80C320 from Dallas Semiconductor. The CPU should be capable of carrying-out of logic and

arithmetic operations, conditional and unconditional jumps, test states of the inputs, set or reset outputs, timers, counters, and so on.

In the simplest case each programmable control circuit might be realised as microprocessor device. We have to remember about application in which we are going to use constructed logic controller. Those applications force special requirements and constraints. Controlled objects heave a great number of binary inputs and outputs while standard microprocessor (or microcontroller) operate mainly on bytes. Instruction list of those devices is optimised for operation on bytes or words (some of them can carry out complicated arithmetical calculation) variables that are not required in industrial applications. Each task is connected with reading external data, computation and writing computed data to the outputs. Logical instructions like AND or OR on individual bits take the same amount of time. When we take under consideration number of binary inputs and outputs, those in greater units reach number of thousands. In such cases parallel computation of all inputs and outputs is impossible. In this situation all inputs and outputs must be scanned and update sequentially as fast as it is possible. If we would like to achieve good control parameters bits operation should be done very quickly.

Creation of specialised bit processor, which fast can carry out bit operations is fully excused. If there is a need of computation of byte data for example from analogue to digital converters or external timers, it is required to use additional 8, 16 or even 32 bits processor or microcontroller. General structure of that device was presented in [1].

Presented solution consists of two processors. Each of them has its own instruction set. Instruction decoder recognises for which processor instruction was fetched and sends activation signal to it.

Basic parameter that was taken under consideration was program execution speed. Following assumption were made in order to support two processors operation

- Separate address buses for bit and byte processors;
- Two data buses: 1 bit wide for bit processor and 8 bit wide for microcontroller;
- Two control buses with signals RD and WR of microcontroller, IORD and IOWR of bit processor, REFRESH (latches state of all inputs and outputs at once) and ERROR (immediate switch off of all external modules of controller).

3. SELECTED STRUCTURES OF BIT-BYTE CENTRAL PROCESSING UNIT

In this paragraph different conception rules of co-operation bit and byte processor are presented, that allow achieving maximal execution speed by logic controller.

The most typical solution is a circuit with separate program and data memories for both processors. There is also common area of memory through which processors exchange information between them in order to:

- exchange data;
- set and clear flags that request execution of specific tasks instead of exchanging whole instruction.

Other conception is presented in [2]. It based on similar idea as previously presented. This solution assumes common program memory for both processors. Each of them has unique operation codes. One of the processors fetches operation code and recognise it. If fetched

instruction is assigned to it, it is immediately executed; in other cases it is send to the second processor for execution.

The unit is equipped with 3 memory banks for control program:

- main memory;
- standard procedures memory;
- program memory for byte processor.

Such CPU has three states of operation:

- both processors execute control program;
- one processor operates;
- bit processor executes control program while byte processor actualises the timers.

The modification of the above solution, referring to the first conception is the unit where bit processor generates pulses activating the sequential tasks in byte processor. These tasks are stored in suitable areas of byte processor memory.

Finally the CPU structure presented in work [2] was accepted. This structure was additionally equipped with the system of fast data exchange keeping easy way of PLC programming. This system – in simple words – causes that the processors do not wait for finishing their operations but they execute next commands up to the moment when command of waiting for result of operation carried-out by the second processor. The important thing is the suitable program compiler and the way of control program by the user.

Bit processor delivers commands to the byte processor through the command buffer informing about it by means of NEXT=0- signal. On the other hand byte processor after accepting of a command sends to the bit processor EMPTYBUF signal (Fig.1). The processors may transfer one to other the result of recently executed operations through F1B and F2B flip-flop.

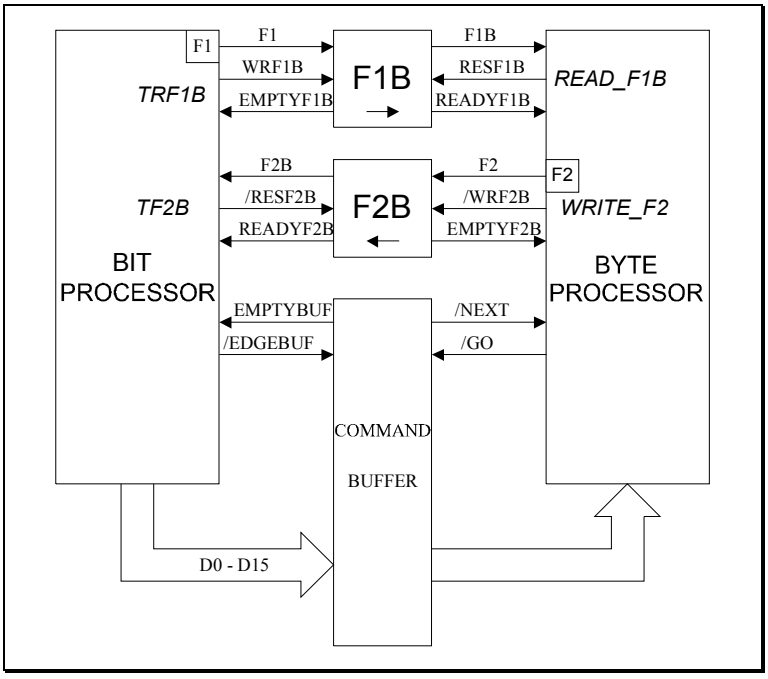


Fig. 1. The block diagram of the CPU based on exchanging of the flags and commands

Two situations cause that the speed of data exchange goes down:

- one processor has not yet execute operation expected by the second processor and this one have to wait for the result (READYF2B=0 or READYF1B=0);
- second processor has not yet received the previous result and the first one can not write the next result (EMPTYF1B=0 or EMPTYF2B=0).

To exclude waiting states the programs has to be written and compiled in such a way to get these two processor working possibly parallel. However in the second case one can take into account the solution basing on increased number of the accessible data exchange flip-flops or on assignment of common memory area for the data exchange purpose.

At that time appears the need for assignment of the marker to every task. One can try to solve the marker problem in the following ways:

- the fixed marker can be assigned to every type (kind) of operation, e.g. comparison instruction (of the byte processor) have to set the marker at the given (particular) address, and, say, the counter increment instruction will use the marker of other address (this solution is not flexible, both processors need for frequent access to the common memory area, or many flip-flops have to be used);
- the successive tasks will use successive markers and this process will repeat periodically after the number of markers is run out. The assignment process can be led automatically by the compiler. However this solution can be applied for the instruction sequences not disturbed by jumps (except the jump to the beginning of program loop).
- the third way is to charge the programmer with the duty of marker assignment. Marker has to contain the operation result, or condition has to be read from that marker. In similar way the Modicon PLCs are programmed where instruction blocks outputs carrying results can be assigned to the marker by programmer himself.

4. SYNCHRONISATION OF PROCESSORS

The designed CPU can work in one of two modes:

- dependent work – the parallel – serial work of processors with transferring of all the necessary data, co-ordinated by the bit processor which is faster. It is basic work mode of the designed CPU. All the mechanisms described for the solution of Fig.1 are made use of;
- independent mode – fully parallel. Both units work fully independent, each one has its own program so there is no waiting for the instruction transfers. There are no data transfers between the two processors. Unfortunately such a mode is applicable only some particular control programs.

5. CONCLUSION

Studies on the information exchange optimisation between the processors of the bit-byte CPU of the PLC have shown the great capabilities and many possible applications of this architecture.

When considering many ways of optimal application of this architecture it seems that quite serious problem is lying in some kind of accepted standard, which describes the way the CPU of PLC is executing the control program should be looked at.

One should go farther in such considerations and try to solve the program execution method taking more task - oriented (problem - oriented) rather than serial – periodical approach.

It seems that probably better results can be obtained when PLC is assumed an event – dependent block (module), which executes particular precisely determined tasks in response to particular constraints i.e. particular elements change of state.

In CPU of that type many problems will be connected with change scanning, because not only inputs and outputs but also markers, timers and counters should be scanned. Other problems will be related to continuous signals.

It seems however that the described architecture "enriched" with an event – dependent control program execution is quite interesting solution of the CPU for PLC.

REFERENCES

[1] Z. Getko, „Programmable systems of binary control", *Elektronizacja*, Zeszyt 18, WKiŁ, Warszawa 1983, (in Polish)

[2] M. Chmiel, E. Hrynkiewicz, „Programmable controllers", *Pomiary Automatyka Kontrola*, No.5, 1994 (in Polish)

[3] M. Chmiel, L. Drewniok, E. Hrynkiewicz, „Single board PLC Based on PLDs", *International Conference on Programmable Devices and Systems*, Gliwice, Poland. November 1995

[4] E. Hrynkiewicz, „Based on PLDs Programmable Logic Controller with Remote I/O Groups", *Third Workshop on Electronic Control and Measuring Systems*, Toulouse, France. June 1997

[5] M. Chmiel, W. Ciążyński, A. Nowara, „Timers and Counters Applied in PLCs", *International Conference on Programmable Devices and Systems*, Gliwice, Poland. November 1995

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

SYSTEM OF DIGITAL DEVICE TEST GENERATION FOR ACTIVE HDL

Vladimir I. HAHANOV, Anna V. BABICH, Masud M.D. MEHEDI

Computer-aided Design Department, Kharkov State Technical University of Radioelectronics, 14. Lenin ave., Kharkov, UKRAINE,
hahanov@kture.kharkov.ua

Abstract. *The models and methods for digital design analysis and test generation, where problems of digital design testing are formalized as linear equations, are offered. The method of test generation for stuck-at faults, which uses fault list cubic coverings (FLCC) for single activation path building, is developed. The ATPG for digital designs created in Active-HDL is offered as well. The obtained tests are used for digital design verification by means of simulation in Active-HDL.*

Key Words. *ATPG, Active HDL, FSM model.*

1. MATHEMATICAL APPARATUS FOR FSM MODEL ANALYSIS

Sequential primitive automatic model [1,2] is represented as follows:

$$M=<X, Y, Z, f, g>, \quad (1)$$

where $X=(X_1,X_2,...,X_i,...,X_m)$, $Y=(Y_1,Y_2,...,Y_i,...,X_h)$, $Z=(Z_1,Z_2,...,Z_i,...,Z_k)$ – are sets of input, internal and output State (FSM) variables, the interconnection of which is described by the following characteristic equations:

$$Y(t)=f[X(t-1), X(t), Y(t-1), Z(t-1)];$$
$$Z(t)=g[X(t-1), X(t), Y(t-1), Y(t), Z(t-1)]. \quad (2)$$

Variables Z(t) are external and, therefore, they are observed on output lines. Variables Y(t) are internal and, therefore, they are non-observed. FSM variables format for cubic covering, corresponding to (1), is as follows:

X(t−1)	Y(t−1)	Z(t−1)
X(t)	Y(t)	Z(t)

.

FSM model from Fig.1 corresponds to the mentioned format.

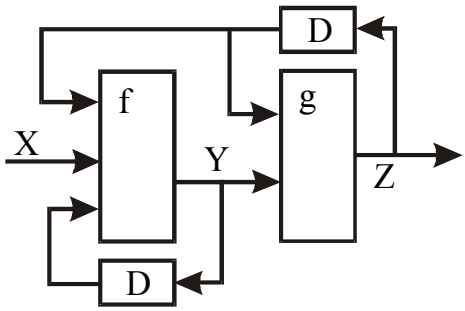


Fig.1. The primitive's state model

Functional sequential primitive is specified by components:

$$F^2 = <(t\text{-}1,t),(X,Z,Y),\{A^2\}>, \tag{3}$$

where (t-1,t) – are two state consecutive frames in function description; (X,Z,Y)– are vectors of input, internal and output variables; $\{A^2\}$ – is a two-frame alphabet of FSM state (transition) description:

A^2={Q=00,E=01,H=10,J=11,O={Q,H},I={E,J},A={Q,E},
B={H,J},S={Q,J},P={E,H},C={E,H,J},F={Q,H,J},L={Q,E,J}, V={Q,E,H},Y={Q,E,H,J},
A^1={0,1,X={0,1}}, (U)}.

Execution of the concatenation operation (#):

$$C_{t-1}\#C_t = \begin{array}{|c|c|c|c|}\hline \# & 0 & 1 & X \\ \hline 0 & Q & E & A \\ \hline 1 & H & J & B \\ \hline X & O & I & Y \\ \hline \end{array}. \tag{4}$$

Problem 1. FLCC L for the vector T and the primitive covering C is computed by a linear equation

$$T \oplus C = L, \tag{5}$$

where $\oplus$ – is a binary coordinate operation XOR which determines interaction of components T, C, L in the three-valued alphabet:

$$T_j \oplus C_{ij} = \begin{array}{|c|c|c|c|}\hline \oplus & 0 & 1 & X \\ \hline 0 & 0 & 1 & X \\ \hline 1 & 1 & 0 & X \\ \hline X & X & X & X \\ \hline \end{array}. \tag{6}$$

The universal formula of FLCC analysis obtained as a result of application of (3) to the test-vector T and to covering of the multioutput primitive C for definition of detectable faults Lr by output r, is as follows:

$$L = \bigcup_{\forall i(T_r \oplus C_{ir}=1)} \bigcap_{j=1}^{k} L_j^{T_j \oplus C_{ij}}, \qquad L_j^{T_j \oplus C_{ij}} = \begin{cases} L_j \leftarrow T_j \oplus C_{ij} = 1; \\ \bar{L}_j \leftarrow T_j \oplus C_{ij} = 0, \end{cases} \tag{7}$$

where $\bar{L}_j$ – is considered as a line j fault list that should be subtracted from faults which are detected at non-output primitive lines; L_j – are faults that are required to be intersected with non-output lists.

Problem 2. Test for FSM faults, which are set by L_i^1-cube from FLCC $L = (L_i^0, L_i^1)$, is defined by equation

$$L_i^1 \oplus C = T^k \tag{8}$$

where L_i^0, L_i^1 - are cubes having "0" or "1" values on output coordinate r. At the same time vectors-candidates to test $T_t^k \in T^k$ are defined. From set T^k the test T is formed, where the set $T_t \in T$ is regarded as an element. Each $T_t \in T$ must fulfil conditions:

$$T_t = C_i \cap T_t^k \Leftarrow \exists i(C_i \cap T_t^k \neq \varnothing) \tag{9}$$

2. ATPG SYSTEM

Purpose of automatic test pattern generator (ATPG) creation is verification of digital designs in Active-HDL system, which later will be implemented in Field Programmable Gate Array (FPGA), Complex Programmable Logic Device (CPLD), which are on one level with general matrix chips and signal processors now.

ATPG system solves the following tasks:

1. Test generation for FSM represented by transition graph. Representative languages are VHDL, Verilog.

2. Test generation for digital designs projects represented by Boolean equations. Representative language is VHDL.

3. Evaluation of test quality by single stuck-at- fault simulation.

2.1. The structure of TESTBUILDER program

The program is intended for automatic test generation with respect to single stuck-at faults of digital designs, which are described in language of Boolean equations.

Program operations:

1. Pseudo-random test generation in term of built-in binary code generators and decimal code generators.

2. Deterministic binary test-vector generation, where the mentioned test-vectors activate single logical paths in circuit.

3. Single stuck-at fault simulation [1] with purposes of evaluation of fault coverage of obtained test.

4. Test formatting in standard of VHDL - Testbench.

As an example of test building for circuit (Fig. 2) the following result is given:

Test =

No	012345	%
1 – Vector	01X010	
Faults =	10.101	41.67
Total =	10.101	41.67
2 – Vector	100000	
Faults =	1X1101	58.33
Total =	1X1101	58.33
3 – Vector	11X111	
Faults =	00.000	41.67
Total =	XX1X0X	83.33
4 – Vector	X01001	
Faults =	0000X0	58.33
Total =	XXXXXX	100.00

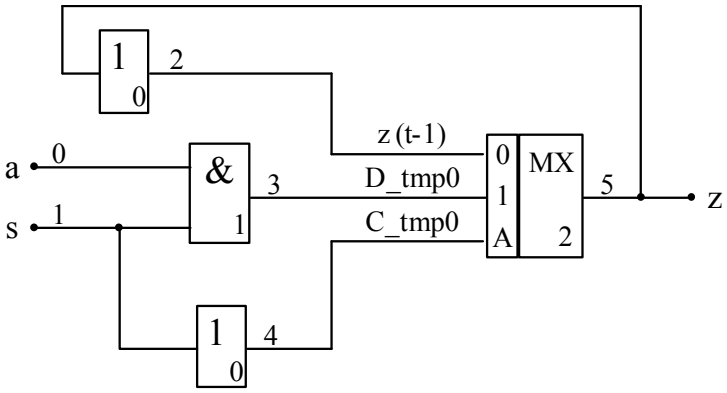


Fig. 2. Example of digital circuit

The program has processed:

- 10 combinational circuits from list ISCAS'85;
- 140 combinational and sequential circuits from PRUS; 45 sequential circuits with large complexity from PRUS;
- 22 sequential circuits from list ITC'99; average time of deterministic test generation is 2 hours.

Average complexity of design is 1000 lines. Average time of pseudo-random test generation is 5 minutes. Test coverage is more than 90 %.

CONCLUSIONS

The proposed models and methods are realised in the form of program applications. The last ones are used for test generation of digital designs based on FPGA and CPLD. The class of evaluated structures is the FSM in the form of transition graph and Boolean equations with flip-flop circuit. Digital circuit description language is VHDL. Program applications are directed toward their use in CAD systems: Aldec, Xilinx.

REFERENCES

[1] Hahanov V.I., *Cubic fault simulation and test generation for digital circuits*, Annual report of KTURE, 1999-2000, pp.139 - 146

[2] Hahanov V.I., KoKovalyov EE.V., Hanko V.V., Mehedi M.M., "Test generation system for FSM designing in ACTIVE-HDL", *ACS and automaton devices*, KTURE, Kharkov, №.111, pp.5-22.

[3] Bondarenko M.F., Krivoulya G.F., Riabtcev V.G., Fradkov S.A., Hahanov V.I., *Designing and diagnosis of computer systems and networks*, K.: SMC HE. 2000, 306 p.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

A SYSTEMATIC DEVELOPMENT OF VIRTUAL COMPONENTS COMPATIBLE TO STANDARD ICs
(AN INDUSTRIAL EXPERIENCE)

Mirosław BANDZEREWICZ[*)], Wojciech SAKOWSKI[**)],
Włodzimierz WRONA[***)]

*) Evatronix S.A., ul. Dubois 16, 44-100 Gliwice, POLAND
tel./fax +48 32 231 11 71, e-mail: *mirek@gliwice.evatronix.com.pl*
http://www.evatronix.com.pl

**) Institute of Electronics, Silesian University of Technology
ul. Akademicka 16, 44-100 Gliwice, POLAND
tel. +48 32 237 17 47, fax +48 32 237 22 25, e-mail: *sak@boss.iele.polsl.gliwice.pl*

***) Technical University of Łódz, Bielsko-Biała branch
Willowa 2, 43-308 Bielsko-Biała, POLAND

Abstract *Paper presents a proven methodology of development and productization of virtual electronic components. Methodology consists of rigorous approach to development of component specification, reverse engineering of behavior of reference circuits, application of industry-standard rules to coding of RTL model in a hardware description language and extensive testing and verification activities leading to (measured) high quality of hdl model and to FPGA prototype. In the final stage called productization a series of deliverables are produced to ensure effective reuse of the component in different (both FPGA and ASIC) target technologies.*

Key Words. *Virtual Components, IP Cores, Hardware Description Languages, high level design, quality assurance*

1. OBJECTIVE AND MOTIVATION

Objective of the effort described in this paper was to define a quality assurance policy for the development of the virtual components based on existing integrated circuits. At present our company specializes in cores compatible to 8-bit and 16-bit microcontrollers and microprocessors. Our methodology is based on the methodology recommended in [1], but it reflects to some extent peculiarities of our current profile as well as the fact that we have no access to certain EDA tools recommended in [1]. We found a lot of inspiration in the paper presented by SICAN company (now SCI-WORX) at the FDL'99 in Lyon [2].

The main motivation for the definition of a formalized methodology was to assure a high quality of the cores that we develop. Our first (bad) experiences in the development of a

microcontroller core that was compatible to Intel 8051 chip [3] showed that lack of consistent and rigorous methodology results in a buggy core. Moreover, lack of a clear and complete specification turn the debugging of our first core into nightmare.

2. OVERVIEW OF THE METHODOLOGY

2.1. Design flow

Basic development steps in the creation of a virtual component include:

- Development of the macro specification,
- Partitioning the macro into subblocks,
- Development of a testing environment & test suite,
- Design of subblocks,
- Macro integration and final verification,
- Prototyping the macro in FPGA,
- Productization.

We will discuss these stages one by one below focusing on details related to our experiences.

2.2. Project management issues

At the beginning of a new project all the steps enumerated above are refined into subtasks and scheduled. Human and material resources are allocated to the project. Usually several projects are being realized in parallel. Therefore people, equipment and software have to be shared among these projects. We use MS Project software to manage scheduling of tasks and allocation of resources.

3. DEVELOPMENT OF THE MACRO SPECIFICATION

We use the documentation of an original device as a basis for specification of the core modelled after it. However, the documentation provided by the chip manufacturer is oriented towards chip users and it usually does not contain all details of chip behavior that are necessary to recreate its full functionality. Therefore analysis of the original documentation results in a list of ambiguities that have to be resolved by testing the original chip. The overall testing program is usually very complex, but the first tests to be written and run on a hardware modeler (see point 5) are those that provide resolve ambiguities in documentation.

At a later stage of specification we use an Excel spreadsheet to document all operations and data transfers that take place inside the chip. Spreadsheet columns represent time slots and rows represent communication channels. Such approach enables gradual refinement of scheduling of data transfers and operations up to the moment when clock cycle accuracy is reached. It reveals potential bottlenecks of the circuit architecture and makes easy to remove them at this early design stage.

4. PARTITIONING INTO SUBBLOCKS

Dataflow spreadsheet makes easier to define proper partitioning of the macro into subblocks. This first level of design hierarchy is needed to handle the complexity and to easier to divide design tasks between several designers. The crucial issue in this process is distribution of functions between the subblocks, definition of the structural interfaces and specification of timing dependencies between them.

5. TESTING ENVIRONMENT AND PROCEDURES

5.1. Testing the reference chip

As a reference for our virtual components we use hardware models that run on a (second hand) CATS hardware modeler (*Fig 1*). The hardware modeler interfaces over network to the CADAT simulator. The environment of the chip is modeled in C. Test vectors supplied from a file may be used for providing stimuli necessary to model interaction of the modeled chip with external circuits (e.g. interrupt signals).

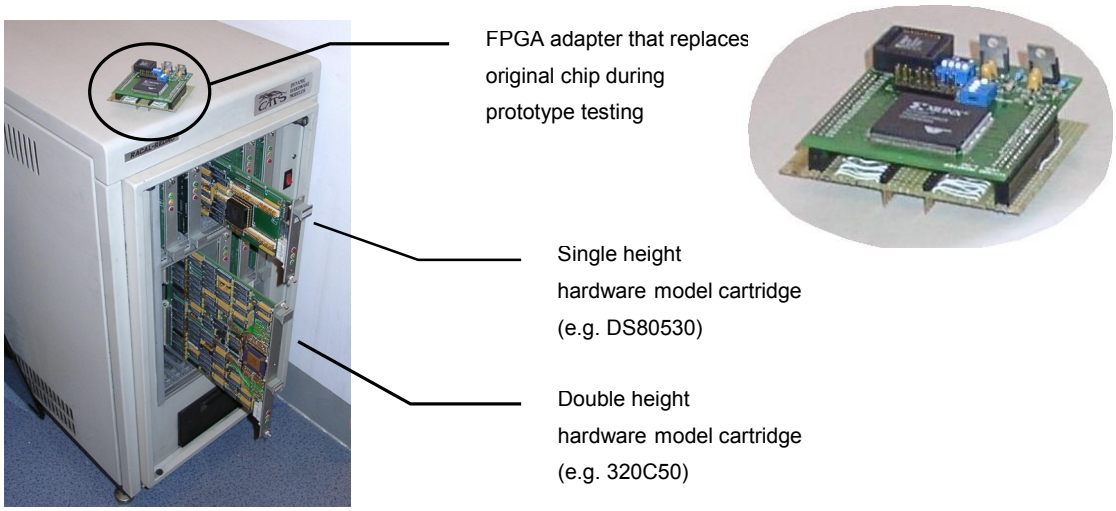


Fig. 1 CATS hardware modeler

An equivalent testing environment is developed in parallel as a VHDL testbench to be run on VHDL simulator. We use Aldec's ActiveHDL simulator that proved to be very effective in model development and debugging phase. It enables us to import the testing results obtained with a hardware model into its waveform viewer in order to compare them against simulated behavior of the core under development.

5.2. Test suite development

Test suite development is based on the specification. Specification is analyzed and all the functional features of the core that should be tested against the original device are enumerated. Test development team (engineer) starts with development of tests that are needed to resolve ambiguities in available documentation of the chip to which a core has to be compliant.

Most of the functional tests are actually short programs written in assembly language of the processor that is modeled. Each test exercises one or several instructions of the processor. For instructions supporting several addressing modes tests are developed to check all of them. After compiling a test routine the resulting object code is translated to formats that may be used to initialize models of program memory in the testbenches (both in CADAT and VHDL environments). We have developed a set of utility procedures that automate this process.

In order to test processor interaction with its environment (i.e. I/O operations, handling of interrupts, counting of external events, response to reset signal) a testbench is equipped with stimuli generator.

5.3. Code coverage analysis

The completeness of the test suite is checked with code coverage tool (VN-Cover from TransEDA). The tool introduces monitors into the simulation environment and gathers data during a simulation run. Then the user can check what percentage of code statements was actually executed. More sophisticated measures like branch or path coverage may be also determined.

Incompleteness of the test suite may be a reason for leaving bugs in untested part of code [4]. Therefore we set a requirement to achieve 100% statement coverage during RTL simulation (i.e. each statement must be executed at least once during simulation of the complete test suite). Code coverage also helps to reveal redundancy of the test suite and sometimes the redundancy in the hardware design under test.

TransEDA State Navigator tool complements VN-Cover with special tools for verifying finite state machines. It may extract fsm from the VHDL source and present it graphically as state diagram. It also analyzes the results of simulation and shows what edges of the state diagram were taken or whether particular sequence of edges was exercised.

5.4. Automated testbench

Our cores are functionally equivalent to the processors they are compliant to, but they are not always cycle accurate. Therefore a strategy for automated comparison of results obtained with hardware modeler to those obtained by simulating RTL model was developed.

Scripts that control simulators may load the program memory with subsequent tests and save the simulation data into files. These files may serve as reference for post-synthesis and post-layout simulation. The testbench that is used for these simulation runs contains a comparator that automatically compares simulator outputs to the reference values.

6. SUBBLOCK DEVELOPMENT

The main part of the macro development effort is the actual design of subblocks defined during specification phase. For the moment we have no access to tools that check the compliance of the code to a given set of rules and guidelines. We follow the design and coding rules defined in [1]. We check the code with VN-Check tool from TransEDA to ensure that the rules are followed. Violations are documented.

For certain subblocks we develop separate testbenches and tests. However, the degree to which module is tested separately depends on its interaction with surrounding subblocks. As we specialize in microprocessor core development it is generally easier to interpret the results of simulation of the complete core than to interpret the behavior of its control unit separated from other parts of the chip. The important aspect here is that we have access to the results of the test run on the hardware model that serve as reference.

On the other hand certain subblocks like arithmetic-logic unit or peripherals (i.e. uarts and timers) are easy to test separately and are tested exhaustively before integration of the macro starts.

Synthesis is realized with tools for FPGA design. We use Synplify, FPGA Express and Leonardo. We realize synthesis with each tool looking for the best possible results in area-oriented and performance-oriented optimizations.

7. MACRO INTEGRATION

Once the subblocks are tested and synthesized they may be integrated. Then all the tests are run on the RTL model and the results are compared against the hardware model. As soon as the compliance is confirmed (which may require a few iterations back to subblock coding and running tests on integrated macro again) a macro is synthesized towards Xilinx and Altera chips and the tests are run again on the structural model.

8. PROTOTYPING

The next step in the core development process is building of a real prototype that could be used for testing and evaluation of the core.

For the moment we target two technologies: Altera and Xilinx. Our cores are available to users of Altera and Xilinx FPGAs through AMPP and AllianceCORE programs. In the near future we are going to implement our cores in Actel technologies, too. Placing and routing of a core in a given FPGA technology is realized with vendor specific software. The tests are run again on the SDF-annotated structural model. We developed a series of adapter boards that interface FPGA prototype to a system in which a core may be tested or evaluated.

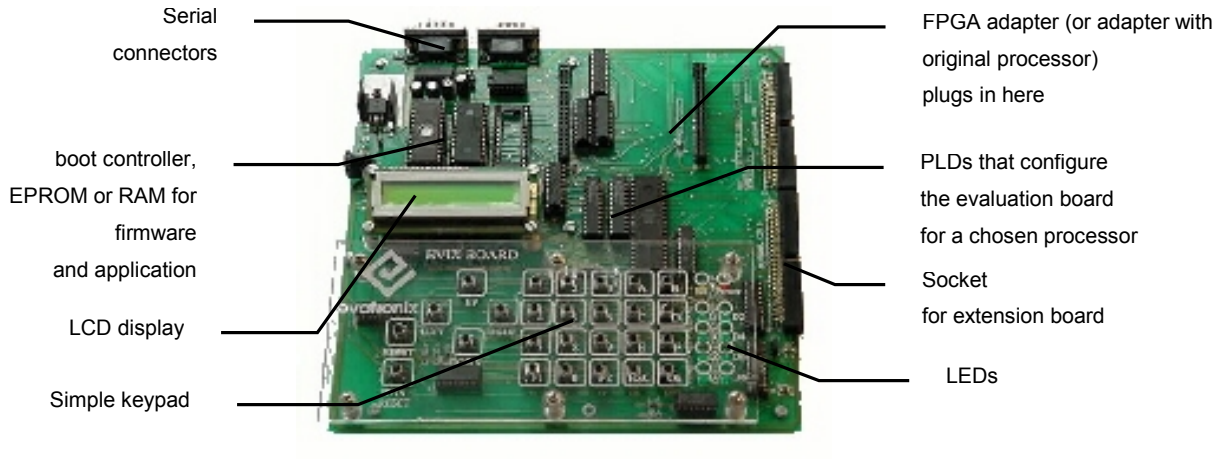


Fig. 2 Processor core evaluation board

The simplest way to test the FPGA prototype is to replace an original reference chip used in the hardware modeller with it. This makes possible to compare behavior of the prototype against the original chip. However for some types of tests even hardware modeller does not provide necessary speed. These tests can only be executed in prototype hardware system at full speed. Such approach is a must when one need to test a serial link with a vast amount of data transfers or to perform floating point computations for thousands of arguments. Our experience shows that even after an exhaustive testing program, some minor problems with the core remains undetected until it runs real-life application software.

For this reason we have developed a universal evaluation board (*Fig.2*). It can be adapted to different processor cores by replacement of on-board programmable devices and EPROMs. An FPGA adapter board (*see Fig. 1*) containing the core plugs into this evaluation board. An application program may be uploaded to the on-board RAM memory over a serial link from PC. Development of this application program is done by a separate design team. This team plays actually a role of an internal beta site, that reveals problems in using the core before it is released to the first customer.

The FPGA adapter board may also be used to test the core in the application environment provided that a prototype system exists. Such system should contain a microcontroller or microprocessor that is to be replaced with our core in the integrated version of the system. The adapter board is designed in such a way that it may be plugged it into the microprocessor socket of the target system. Using this technique we made prototypes of our cores run into ZX Spectrum microcomputer (CZ80cpu core) and SEGA Video Game (C68000 core), in which they replaced original Zilog® and Motorola® processors.

9. PRODUCTIZATION

The main goal of productization phase is to define all deliverables that are necessary to make the use of the virtual component in the larger design easy. We develop simulation scripts for Modelsim simulator and we run all the tests with this simulator to make sure that the RTL model simulates correctly with it. As we target FPGA market an important issue in productization phase is to develop all the deliverables for firm cores required by Altera and Xilinx from their partners participating in AMPP and AllianceCore programs.

Our foreign partners help us in developing synthesis scripts for Synopsys Design Compiler which we do not have access to. This deliverable is a must for customers targeting ASIC technologies. Synthesis scenarios for high performance and for minimal cost are developed.

We use VHDL during core development we but we translate our cores into Verilog, to make them available to customers that only work with Verilog HDL. The RTL model is translated automatically while the testbench have to be developed in Verilog manually (the translation tool is not able to translate all VHDL constructs into Verilog).

User documentation is also completed at this stage (an exhaustive, complete and updated specification is very helpful).

10. EXPERIENCES

The methodology described in this paper was defined over last few years during design of several versions of 8051-compatible microcontroller core [3].

It was then successfully applied to development of several virtual components compatible to Microchip PIC® 1657 microcontroller, to Motorola MC68000 16-bit microprocessor, to Zilog Z80 8-bit microprocessor and its peripherals, to TI® 32C025 digital signal processor as well as to VCs that implement controllers of standard serial links (I2C, SDLC and USB).

We continue to improve this methodology in order to turn it into a set of formal quality assurance procedures compliant to ISO 9000 requirements.

REFERENCES

[1] M. Keating, P. Bricaud, *Reuse Methodology Manual 2nd ed.*, Kluwer Academic Publishers, 1999

[2] J.Haase, *Virtual Components - from Research to Business*, Proceedings of the FDL'99 Conference, Lyon, 1999

[3] M.Bandzerewicz, W.Sakowski, *Development of the configurable microcontroller core*, Proceedings of the FDL'99 Conference, Lyon, 1999

[4] M.Stuart, D. Dempster, *Verification Methodology Manual*, Teamwork International, Hampshire UK, 2000

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

A POSITIONAL FILTER SYNTHESIS FOR FPGA IMPLEMENTATION

Dariusz CABAN

Institute of Engineering Cybernetics, Wrocław University of Technology,
Janiszewskiego 11-17, 50-370 Wrocław, POLAND, *darek@ict.pwr.wroc.pl*

Abstract. *The paper reports on some experiments with implementing positional digital image filters using field programmable devices. It demonstrates that a single field programmable device may be used to build such a filter. By using extensive pipelining in the design, the filter can achieve performance of 50 million pixels per second (using Xilinx XC4000E devices) and almost 90 MHz (in case of Virtex-2 devices. These results were obtained using automatic synthesis from VHDL descriptions, avoiding any direct manipulation in the design.*

Key words: *Positional Filter, Median Filter, Synthesis, FPGA Implementation*

1. INTRODUCTION

The paper reports on the implementation of a class of filters used in image processing. The filtering is realised on a running, fixed size window of pixel values. Positional filtering is obtained by arranging the values in an ordered sequence (according to their magnitude) and choosing one that is at a certain position (first, middle, last or any other). Thus, the class of filters encompasses median, max and min filtering, depending on the choice of this position.

There are various algorithms used in positional filtering [7,10]. These are roughly classified to three groups: compare-and-multiplex [9], threshold decomposition [6] and bit-wise elimination [1,8,11]. All can be used with the currently available, powerful FPGA devices. Still, the bit-wise elimination methods seem most appropriate for the cell array organisation.

Some specific positional filters have commercial VLSI implementations. There is no device that can be configured to realise any position filtering, though. Even if only median, min or max filtering is required it may be advantageous to use FPGA devices, as they offer greater versatility and ease of reengineering. Of course, FPGA implementations are particularly well suited for application in experimental image processing systems.

First attempts to use FPGAs as reconfigurable image filters were reported almost as soon as the devices became available [2,3,5]. The devices proved to be too inefficient for full-fledged use forcing the designers to limit the window size, pixel rates or the width of their bit

representations. This is no longer the case since the XC4000E family of devices became available [4].

Filter reconfiguration can be fully utilised only if there is an easy route to obtain new configuration variants. In case of FPGA implementation this is offered by auto-synthesis: new algorithms are described in terms of a hardware description language and the rest is done by the design tools with no human interaction. The results in the paper were obtained using the Xilinx Foundation tools with FPGA Express.

2. BITWISE ELIMINATION METHOD

Positional filtering is based on reordering of the pixel values according to their magnitude. Let's denote the *k*-th value in the reordered sequence by P_n^k (where *n* is the length of the sequence). After reordering, only a single value at the specific position is of interest. In bit wise elimination, values that are certain not to be at this position are removed from the sequence.

Values are compared bit-wise starting from the highest order bits. Lets assume that the *r*-1 highest order bits have already been analysed by this method. Then, all the values that are left for consideration must have the high order *r*-1 bits equal to each other (and to the result under evaluation). Values that had these bits different were eliminated leaving only *n'* values in the sequence. The position also has to be adjusted from initial *k* to *k'* after eliminating values that were greater. The *r*-th bit of result is determined as $P_{n'}^{k'}(r)$ by ordering the corresponding bits of the reduced sequence and considering *k'*-position. All the values that differ on the r-th bit from $P_{n'}^{k'}(r)$ are eliminated and *n'* is modified accordingly. If the eliminated values are greater than the quantile bit, then *k'* is also modified.

The algorithm ends when there is only one value left (or all the values left are equal to each other).

The approach, with changing *k* and *n* is not well suited for circuit implementations. Instead of eliminating the values, it is more convenient to modify them in a way that is guaranteed not to change the result [2,8,11]. If one knows that a value is larger than the *k*-quantile, all its lower bits are set to 1. If one knows that it is smaller, the lower bits are set to 0. Thus, single bit voting may still be used and the values of *k* and *n* are fixed. This is the method used for the presented FPGA implementations. It can be formally described by the following iterative equations, where iterations start with the highest order bits (*r*=*m*-1) and end with the lowest (*r*=0):

$$
\begin{aligned}
&M_i(r-1) = M_i(r) \vee \left(P_n^k(r) \oplus x_i(r)\right), \\
&S_i(r-1) = \left(M_i(r) \wedge S_i(r)\right) \vee \left(!M_i(r) \wedge x_i(r)\right), \\
&M_i(n) = S_i(n) = 0, i = 0..n-1, \\
&P_n^k(r) = \sum_{i=0..n-1} \left\{\left(!M_i(r) \wedge x_i(r)\right) \vee \left(M_i(r) \wedge S_i(r)\right)\right\} > k
\end{aligned}
\qquad (1)
$$

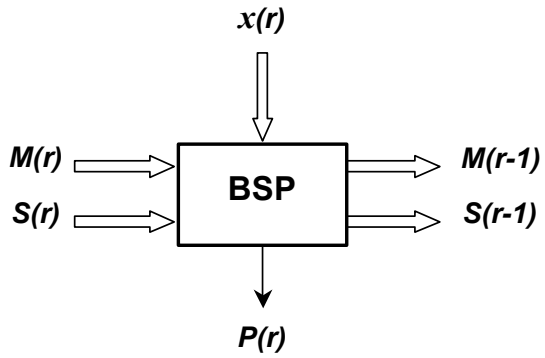


Fig. 1. Bit-slice processor

where $x_i(r)$ is the r-th bit of i-th pixel in the filtering window,
$P_n^k(r)$ is the r-th bit of the value at k-th position (k-quantile),
$M_i(r)$ and $S_i(r)$ are the modifying functions.

Using the presented equations (1) a bit-slice processor may be implemented (Fig. 1). This is a combinatorial circuit that processes the single bits of the input pixels to produce a single bit of the result. The most important part of this bit-slice is the thresholding function corresponding to the last of equations (1).

3. PIPELINED FILTER IMPLEMENTATIONS

The simplest hardware implementation of the filter can be obtained by using m bit-slice processors with connected modifying function inputs and outputs. This would be a fully combinatorial implementation with very long delays, as the modifying functions have to propagate from the highest to the lowest order bits.

Inserting pipelining registers between the bit-slice processors shortens the propagation paths [4]. The registers may be inserted either between every processor, as shown in Fig. 2, or only between some of them. Since this introduces latency between the bit evaluations, additional shift registers are needed on the inputs and outputs to ensure in-phase results.

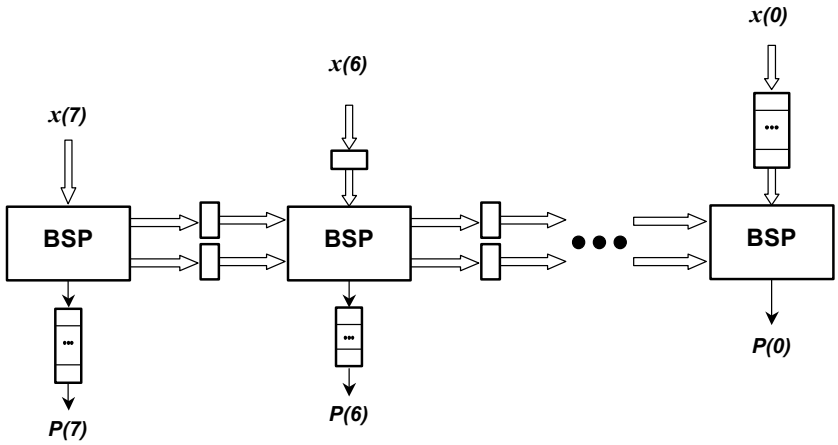


Fig. 2. Pipeline filter architecture

This architecture has very short propagation paths between registers and so it ensures highest pixel processing rates. There is latency between the input signals and the output equal to the number of bits in the pixel representations. Normally, in image processing applications this is not a problem. Just the image synchronisation signals need to be shifted correspondingly. It may be unacceptable, though, if image filtering is just a stage in a real-time control application.

4. FPGA IMPLEMENTATION RESULTS

The pipelined filter architecture was implemented for a filtering window of 3×3 pixels. The inputs of the filter were 3 pixel streams: one obtained by scanning the image and two delayed (by one and two horizontal scan periods). The consecutive horizontal window values were obtained by registering the input streams within the filter (to reduce the demand on input/output pads).

All the presented results were obtained by implementing the filter that computed the median. This has no significant effect on the device performance or complexity, except that the min and max filters have much simpler thresholding functions.

The filters were implemented using different size of pixel value representations (binary values of 4, 8 12 and 16 bits). In each case the smallest and fastest device that could contain the circuit was chosen for implementation. Table 1 contains the results of filter implementations using XC4000E family of devices, whereas Table 2 presents those for the Virtex-2 packages.

Table 1. Filter implementations using XC4000E devices.

Pixel representation	*Device*	*Used CLB's*	*Pixel rate*
4 bits	**4003EPC84-1**	**94**	**55.9 MHz**
8 bits	**4008EPC84-1**	**288**	**50.1 MHz**
12 bits	**4020EHQ208-1**	**645**	**50.6 MHz**
16 bits	**4025EPG223-2**	**993**	**37.5 MHz**

Table 2. Filter implementations using Virtex-2 devices.

Pixel representation	*Device*	*Used slices*	*Pixel rate*
4 bits	**2V40FG256-4**	**99**	**88.8 MHz**
8 bits	**2V40FG256-4**	**209**	**91.9 MHz**
12 bits	**2V80FG256-4**	**345**	**88.7 MHz**
16 bits	**2V80FG256-4**	**453**	**83.7 MHz**

The circuit complexity, expressed in terms of the number of cells used (CLB's or Virtex slices), results from the number and complexity of bit-slice processors (complexity of the combinatorial logic) and from the number of registers used in pipelining. The first increases linearly with the size of pixel representation. On the other hand the number of registers used in pipelining increases with the square of this representation. In case of the XC4000 architecture, the pixel representation of 8 bits is the limit, above which the complexity of circuit is determined solely by the pipelining registers (all the combinatorial logic fits in the lookup tables of cells used for pipelining).

The synthesis tools had problems in attaining optimal solutions for the synthesis of thresholding functions in case of the cells implemented in XC4000 devices (this was not an

issue in case of min and max positional filters). Most noticeably, the design obtained when the threshold function was described as a set of minterms required 314 CLB's in case of 8-bit pixel representation. By using a VHDL description that defined the function as a network of interconnected 4-input blocks, the circuit complexity was reduced to the reported 288 cells. The reengineered threshold function had slight effect on the complexity of 12-bit filter and none on the 16-bit one.

The most noticeable improvement in using the Virtex-2 devices for positional filter implementations was in the operation speed: approximately 50 MHz in case of the XC4000E devices and 80-90 MHz in case of Virtex-2. Some other architectural improvements are apparent, too. The increased functionality of Virtex slices led to much more effective implementations of pipelining registers: the FPGA Express synthesiser implemented them as shift registers instead of unbundled flip-flops, significantly reducing the slice usage. Improved lookup table functionality eliminated the problem of efficient decomposition of threshold function, too (at least in case of the 3×3 filtering window).

5. CONCLUSIONS

The presented implementation results show that FPGA devices have attained the speed grades that are more than adequate for implementing positional image filters of very high resolution. Furthermore, it is no longer necessary to interconnect multiple FPGA devices or limit the circuit complexity by reducing the pixel representations. In fact, the capabilities of Virtex-2 devices exceed these requirements both in terms of performance and cell count.

The proposed bit-wise elimination algorithm with pipelining is appropriate for the cell architecture of FPGA devices. The only problem is the latency, which may be too high in case of long pixel representations. By limiting the pipelining to groups of 2, 3 or more bit-slice processors it is possible to trade off latency against performance.

Positional filtering is just a stage in complex image processing. The analysed filter implementations leave a lot of device resources unused. This is so, even though the cell utilisation for representations of 8 bits or more is between 60 and 97%. The cells are mostly used for registering and the lookup tables are free. These may well be used to implement further stages of image processing.

It is very important that the considered implementations were directly obtained by synthesis from functional descriptions, expressed in VHDL language. This makes feasible the concept of reconfigurable filters, where the user describes the required filtering algorithms in a high-level language and these are programmed into the filter. Still, the design tools have not reached the desirable degree of sophistication and reliability. This is especially true of the obscure template matching rules, peculiar to specific synthesis tools. Also, the correctness by design paradigm is not always met – some errors of improperly matched templates were detected only by testing the synthesised device.

REFERENCES

[1] M.O.Ahmad, D.Sundararajan, "A Fast Algorithm for Two-Dimensional Median Filtering", *IEEE Trans. Circuits and Systems, 34(11)*, pp.1364-1374, 1987

[2] D.Caban & J.Jarnicki, “A Reconfigurable Filter for Digital Images Processing” (in Polish), *Informatyka, 6*, pp.15-19, 1992

[3] D.Caban, “Hardware implementations of a real time positional filter”, *Proc. 5th Microcomputer School Computer Vision and Graphics*, Zakopane, Poland, pp.195-200, 1994

[4] D.Caban & W.Zamojski, “Median filter implementations”, *Machine Graphics & Vision, 9(3)*, pp.719-728, 2000

[5] S.C.Chan, H.O.Ngai & K.L.Ho, “A programmable image processing system using FPGAs”, *Int. J. Electronics, 75(4)*, pp.725-730, 1993

[6] J.P.Fitch, E.J.Coyle & N.C.Gallagher, “Median Filtering by Threshold Decomposition”, *IEEE Trans. Acoustics, Speech and Signal Processing, 32(6)*, pp.553-559, 1984

[7] M.Juhola, J.Katajainen & T.Raita, “Comparison of Algorithms for Standard Median Filtering”, *IEEE Trans. Signal Processing, 39(1)*, pp.204-208, 1991

[8] C.L.Lee & C.W.Jen, “Binary Partition Algorithms and VLSI Architecture for Median and Rank Order Filtering”, *IEEE Trans. Signal Processing, 41(9)*, pp.2937-2942, 1993

[9] S.Ranka & S.Sahni, “Efficient Serial and Parallel Algorithms for Median Filtering”, *IEEE Trans. Signal Processing, 39(6)*, pp.1462-1466, 1991

[10] D.S.Richards, “VLSI Median Filters”, *IEEE Trans. Acoustics, Speech and Signal Processing, 38(1)*, pp.145-153, 1990

[11] C.-W. Wu: Bit-Level Pipelined 2-D Digital Filters for Real-Time Image Processing. *IEEE Trans. Circuits and Systems for Video Techn., 1(1)*, pp.22-34, 1991

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

IMPLEMENTATION OF PIPELINING MECHANISM IN RE-PROGRAMMABLE LOGIC STRUCTURES WITH VHDL LANGUAGE USAGE

Maciej MICHALCZAK, Zbigniew SKOWROŃSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND,
M.Michalczak@willow.iie.pz.zgora.pl, Z.Skowronski@iie.pz.zgora.pl

Abstract. *Using re-programmable logic components along with HDL languages encompasses wider and wider areas of practical applications, becoming a standard of complex digital system design. One of the basic tasks, which are to be carried out in the process of design, is obtaining the highest efficiency of the solution under design. Thereby designers are still looking for methods making it possible to speed up design processing time. Pipelining mechanism is one of these methods. It helps to speed up some dedicated operations. This paper contains an example of practical application of multiplier system of floating - point numbers described in VHDL language (model in the shape of a net - structure), while using operands described by the format compatible with the IEEE 754 standard of writing the floating - point numbers. In the early stage of design, a given unit described by high level language, is divided into some independent parts, which are synchronized with each other via intermediate registers and synchronization signal (pipelining mechanism). The main goal of this paper is to present practical aspects of designing an advanced and complex digital system while using pipelining mechanism in re-programmable logic structures with using VHDL language.*

Key Words. *Pipelining Mechanism, Re-programmable Logic Devices, FPGA, PLD, ASIC, Hardware Description Language, VHDL*

1. INTRODUCTION

Digital units can be found in almost every domain of the surrounding world. Modern inventions like cellular telephony or digital television use digital signal processing for communications and couldn't exist without such units. So it is very important to ensure effective and unfailing design methods. In the last years we can notice a rapid development of the design methods. For example: hardware description languages (HDL), logical synthesis or design implementation methods in, still improving, re-programmable devices such as FPGA, CPLD or ASIC. Now, the design path with HDL and with a high level logical synthesis is an

industry standard. Advanced re-programmable units are larger and larger (in respect of gate number) so it enables the implementation of complicated digital units like SoC (System on chip), processors or specialized controllers. One of the important aspects of the design is its speed.

The main, functional part of a computer and other electronic devices are arithmetic blocks. The operation of multiplication is one of the basic arithmetic operations possible to carry out by digital units. Practically, it is a partial product accumulation of operator A by the value of the next digit of the operator Xi with its weight [4]:

$$P_{i+1} = P_i + AX_i\beta^i = A\sum_{s=-m}^{i+1} X_s\beta^s, \qquad i = -m,...,0,1,...,k-1, \qquad P_{-m} = 0 \tag{1}$$

what in normalized result $p_i = \beta^{-i}P_i$ is:

$$p_{i+1} = \beta^{-1}(p_i + X_i A), \qquad i = -m,...,0,1,...,k-1, \qquad p_{-m} = 0 \tag{2}$$

The scaled end result $P_k = \beta^k p_k$ is equal to the value of multiplication A*X.

In binary system, partial product may be the operator A or the value 0, so the multiplication algorithm can be realized by the add-and-shift method (Figure 1).

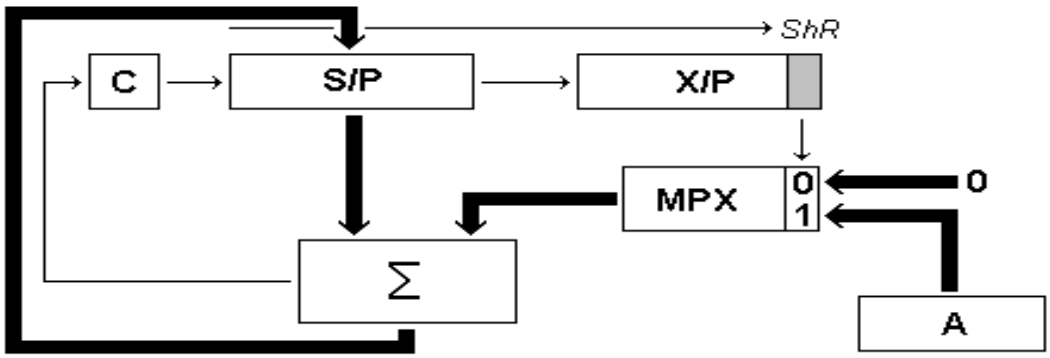


Figure 1. Algorithm of accumulation multiplication in the natural binary system

The main goal of this work is presenting a practical method of speeding up the operation of multiplication. The paper shows the example solution of a multiplication module. Operators used in the module are consistent with IEEE 754 floating-point number notation standard. The module includes a constant-number-multiplication module, which will be sped up.

2. RULES OF USE OF PIPELINING MECHANISM

Pipelining is a technique increasing the function speed of a design [6]. While pipeline implementation, the design is divided into individual pipelining levels, creating a line of matrix of elementary processing modules. Each matrix level may be modeled individually and independently. It allows choosing the optimal method of realization. In the case of sequential operation unit, when each stage is implemented in a different functional block of a design, other blocks remain idle. When individual parts of a unit are separated by latches (thus locking partial products), it becomes possible to process simultaneously each data stream. That is what pipelining is. In such a solution the results can be produced in each cycle of the synchronization signal, triggering given levels, not only once per several cycles. The frequency of producing results is limited by the delay time of the longest processing

matrix level. Compared to combinational units, it is often possible to create the situation where the delay time of the whole unit (propagation time) is significantly larger than the period between the synchronization signals in the unit with implemented pipelining. Assuming that results of a unit with pipelining are produced with each cycle of the synchronization signal [4], it becomes possible to increase significantly the frequency of results produced, in comparison with combinational units.

3. THE DESCRIPTION OF FUNCTIONAL ASSUMPTIONS

The presented unit is an attempt of implementation of floating-point number multiplier, based on IEEE 754 binary standard numbering notation. Interface has been restricted to support only one mode of that standard. It is the simple mode with single precision, called *real*. A number written in that notation consists of three components (Figure 2):

- sign: 1 bit;
- exponent: 8 bits;
- mantissa: 23 bits + one hidden bit used for normalization.

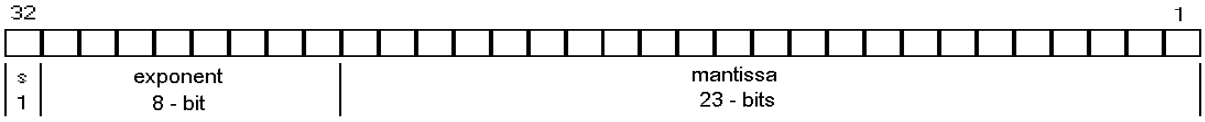


Figure 2. The representation of the floating-point number in IEEE 754 notation

The implementation was based on the floating-point–number-multiplication algorithm [7]. The first step is the operation of mantissa multiplication (23 bits + 1 hidden bit. For the next operation it is necessary to round the result to 24-bit number using only the top 24 bits of the 48-bit result. After that this number should be normalized if overflow occurs on the hidden bit (the most important bit) by shifting the mantissa to the right along with the exponent value increment. That is the outcome value of the mantissa. The exponent in IEEE 754 standard is represented in biased-127 code. The next step is the addition of exponents, which prior to that, are converted to the value in 2's complementary code through adding the value of 127. After the addition, the result should be corrected by subtracting the value of 127. The overflow occurs when the value of the addition result is bigger than 127. In that case the exponent is set to the value of 128, and the mantissa is set to the 0 value. Underflow occurs when the value of the exponent addition is smaller than minus 126. In that case the exponent is set to the value of minus 127. The last step is to determine a sign value through XOR operation on the signs of operands.

The suggested solution here is a structure of three modules [10]:

- adder of exponents;
- multiplier of mantissas;
- and the main module, realizing all the necessary corrections and setting the sign of the result value.

The main part of the entire unit is the multiplier module, which is used for realizing the operation of multiplication of two given operands. Different conceptions of realization of that module have been tested. After that it has been decided to present two solutions:

- behavioral description, with using the multiplication function from standard library;
- multiplication with using multiplying matrix, with implemented pipelining mechanism.

4. PRESENTED DESIGN IMPLEMENTATION

The design has been realized in the programming environment of Aldec's Active-HDL (compilation & simulation code), and by using the program of Foundation 2.1i from Xilinx [2] (logic synthesis & hardware implementation). The whole design has been created in VHDL language [3, 8, 9]. Functionality has been checked during functional simulations and during timing simulations using the pseudo-hardware time delays of the target device.

The realization has been divided into two modules. The realized adder module with the result correction in overflow case on the hidden bit position during mantissas multiplication and with the estimation of the outcome overflow of the exponent:

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
--
entity a_der is
port (
   INC_E      : in STD_LOGIC;
   EXP_A      : in STD_LOGIC_VECTOR(7 downto 0);
   EXP_B      : in STD_LOGIC_VECTOR(7 downto 0);
   OV         : out STD_LOGIC;
   EXP_Q      : out STD_LOGIC_VECTOR(7 downto 0)
     );
end a_der;
--
architecture ADDER_ARCH of adder is
-- -127 value for conversion
constant CONST : STD_LOGIC_VECTOR(7 downto 0) := "10000001";
-- temporary for overflow
signal TEMP_OV : STD_LOGIC;
-- temporary for underflow
signal TEMP_UN : STD_LOGIC;
-- result in 2's complement code
signal TEMP : STD_LOGIC_VECTOR(7 downto 0);
-- biased-127 result representation
signal TEMP_Q : STD_LOGIC_VECTOR(7 downto 0);
--
begin
--
TEMP <= EXP_A + EXP_B + INC_E;                      -- addition
TEMP_Q <= TEMP + CONST;                             -- conversion from
                                                    -- 2's to biased-127
TEMP_OV <= TEMP(7) and E_A(7) and E_B(7);           -- set if overflow occurs
TEMP_UN <= not (TEMP(7) or E_A(7) or E_B(7));       -- set if underflow occurs

E_Q <= (others => '1') when TEMP_OV='1'
       else (others => '0') when TEMP_UN='1'
       else TEMP_Q;
OV <= TEMP_OV;

end ADDER_ARCH;
```

The main part of the design is a constant-number multiplication module. It ensures the realization of the multiplication operation of two 23-bit operands. In the overflow case multiplication results are set to 0 value.

The main goal of that paper is to present the possibility of realization of fast multiplication module. Multiplication modules are usually the source of the longest delays in projected units. Two different solutions have been worked out:

- functional description (operation based on standard library);
- multiplication matrix with pipelining (description in the form of a structure) [5].

4.1. Functional description

The description of the module has been based on the following construction:

```
RESULTS <= MANTISSA_A * MANTISSA_B;
```

The '*' operator is defined in IEEE standard library. The style of realization of this module is the same as, the formerly presented, adder module.

4.2. Matrix multiplier with pipelining

The whole operation has been divided into stages. The matrix of elementary adding modules has been defined. Each of them is a full 1-bit adder (FA) [10]. The operation of multiplication is carried out using 24-bit operands, so it is necessary to use the matrix of 23x23 adders + one last level of 23 adders for counting the end results of 24 top bits of the result. Each matrix level is composed of 23 adders counting independently introducing the delay equaled to the delay of one elementary module. So in that solution, the whole delay consists of all stages delays plus the addition of 23 adders of the last stage. The unit has been modified by introducing some more memorizing elements called flip-flops, to separate each matrix stage [10]. This pipelining allows for synchronized data flow between each matrix levels. This solution makes it possible that independent data is processed at different stages simultaneously. Data exchange between stages is realized during the active edge of the synchronizing signal. At the output of the unit we obtain results with every synchronizing signal cycle. Also, the input data should be fed into the unit while observing the synchronizing cycle. Synchronization signal period cycle must be bigger than the largest delay of unit stages of the given matrix.

5. COMPARISON OF SPEED

The final stage of realizing each of presented solutions was the unit implementation while using the FPGA re-programmable structure. During work, device from Xilinx's families Virtex have been used. Table 1 presents the results of the implementation process and hardware time delay.

Table 1. Comparison of results for the implementation of example multiplying units

VIRTEX – V100PQ240		
	Behavioral	***Matrix (with pipelining)***
Maximal combinational delay [ns]	**34.419**	39.316
Maximal path delay [ns]	7.095	7.424
Maximal frequency [MHz]	-	**103.681**
Minimal clock period time [ns]	-	**9.45**
SLICE	320	1200

Combinational delay of the behavioral unit is 34 ns for Virtex. It is the time required for producing valid results by a module at the output on the basis of the input data.

For synchronized matrix unit with implemented pipelining mechanism the most important parameter is the synchronization signal frequency. In the Virtex case it is equaled to 103.681 MHz, so the cycle period time for synchronization signal is 9.45 ns.

Synchronization signal period time determines the time between producing consecutive results. The main difference between presented solutions is the fact that for a behavioral unit the whole operation time is 34.419 ns. Matrix unit takes 39.316 ns, but this unit produced results after 9.45 ns. When the input data is fed between that times, the time of produced results is much better than in a behavioral unit.

6. SUMMARY

This paper presents a method that allows decreasing the period of time required for producing consecutive multiplication results. A practical example of floating-point number multiplier has been presented. It consists of the constant-number multiplier module for which the attempt of pipelining mechanism implementation has been presented. Two solutions have been worked out and implemented. The obtained results have been compared paying special attention to the processing speed (time between generated results) (Table 1).

Combinational delay of behavioral unit is bigger than the minimal cycle period time of the synchronization signal in the matrix with a pipelining unit. Thanks to that it is possible to achieve the situation where the pipelining unit allows to produce results every 9.45 ns in comparison to 34.419 for a behavioral unit. That is a significant difference, in the aspect of obtaining consecutive unit results. The disadvantage of that solution is its size and more complicated description. A designer must ensure suitable synchronization of partial results and correct data flow between the unit stages.

The application of the presented proposal concerns the sequential processing structures, where the most important thing is data processing time and size of system is not the main concern.

REFERENCES

[1] http://www.aldec.com

[2] http://www.xilinx.com

[3] P. J. Ashenden: *The Designer's Guide to VHDL*, Morgan Kaufmann Publisher, Inc., 1996

[4] J. Biernat: *Architecture of Computers*, Oficyna Wydawnicza Politechniki Wrocławskiej, Wrocław, 1999 (in Polish)

[5] J. Biernat: *Arithmetics of Computers*, Wydawnictwo Naukowe PWN, Warszawa, 1996 (in Polish)

[6] G. De Micheli: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., Stanford, 1994

[7] M. Roth: Assembly Language, *http://www.cs.uaf.edu/~cs301/notes/Chapter6/node4.html*

[8] A. Rushton: *VHDL for Logic Synthesis - Second Edition*, John Wiley & Sons, 1998

[9] S. Sjoholm, L. Lindh: *VHDL for Designers*, Prentice Hall, 1997

[10] M. Michalczak, Z. Skowroński: „Practical Principles of Design of Digital Devices with Pipeline in the Programmable Logic Structures Using VHDL", *Proceedings of IV National Scientific Conference, RUC'2001,* Szczecin, May 7-8, 2001, ISBN 83-87362-35-2, pp.93-101 (in Polish)

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

PIPELINE PROCESSING FOR SERIAL REALIZATION OF BASICAL ARITHMETICAL OPERATIONS

Janusz JABŁOŃSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *J.Jablonski@iie.pz.zgora.pl*

Abstract. *In the paper a high-speed realization of converter from a residue-to- binary and binary-to-residue form for the three-modules residue number system (RNS) is proposed. The bases $\{B_1, B_2, B_3\}$ for separate modules are respectively $\{2^n-1, 2^n, 2^n+1\}$. In opposite to well-know converters, the proposed method is based on bits grouping algorithm, which is different from the Chinese Residue Theory (CRT). Using the method of grouping bits causes a regular structure of the converter. There is possible dynamical changing of RNS range in FPGA structures and realization of stream processing.*

Key Words. *Residue Number System, Residue-to-Binary, Binary-to-Residue, Stream-processing, Reconfigurable structures, regular structure.*

1. INTRODUCTION

Residue Number System (RNS) in arithmetical operation causes division of huge argument into a vector of smaller values [8]. Operations, such as addition, subtraction and multiplication, in RNS are executed independently from components of a residue vector. Cary-free nature of Residue Number System makes it attractive for implementation of a high-performance digital signal processing (DSP) systems [6]. RNS has bean found perfectly suiting for high-speed computation involving high precision, when arithmetic operations are predominant of which DSP hardware such as convolvers, digital filters and FFT processing are perfect examples[4][5]. The three-moduli RNS $\{2^n-1, 2^n, 2^n+1\}$ is of special interest because several operations in this system can be performed efficiently with limited amount or even without ROM. The periodicity properties exhibited by the three-moduli RNS result in superb performance of the binary-to-residue (B-to-R) converter and modulo addition even for large n. On the other hand, a residue-to-binary (R-to-B) converter should be realized in similar, short time [1,9].

2. PROCESSING USING RESIDUE NUMBER SYSTEM

RNS [5][2][8] is defined by a set of r positive integers $\{B_1, B_2, ..., B_r\}$, which are pair-wise, relatively prime, i.e. for any pair of module B_i, Bj, $i \neq j$. The dynamic range W of the RNS with r moduli, i.e. the number of different integers that can be uniquely represented in the RNS, is

given by $W = \Pi_i B_i$, $i=1,2...r$. In RNS a numerical value of a natural number X in the range $[0, W-1]$ is represented by an r-tuple $\{X_1, X_2, ..., Xr\}$, whose components are the residues of X with respect to an ordered set of module B_i, i.e.:

$$X_i = X \bmod B_i = |X|_{Bi}$$

In RNS the operations of addition, subtraction, and multiplication are performed simultaneously upon their residues:

$$(X_1, X_2, ..., X_r) \oplus (Y_1, Y_2, ..., Y_r) = (Z_1, Z_2, .., Z_r)$$

where $Z_i = |X_i \oplus Y_i|_{Bi}$, $1 \le i \le r$. Thus, the carry propagation delay becomes dependent on the size $b_i = \lceil \log_2 B_i \rceil$ of a single module B_i only. In most RNS-based applications, computation process is ended by R-to-B conversion, because other system elements use conventional positional binary system. Fig. 1 presents a general schema of the computation process in RNS. There are operation block and two converters blocks (marked by area 1).

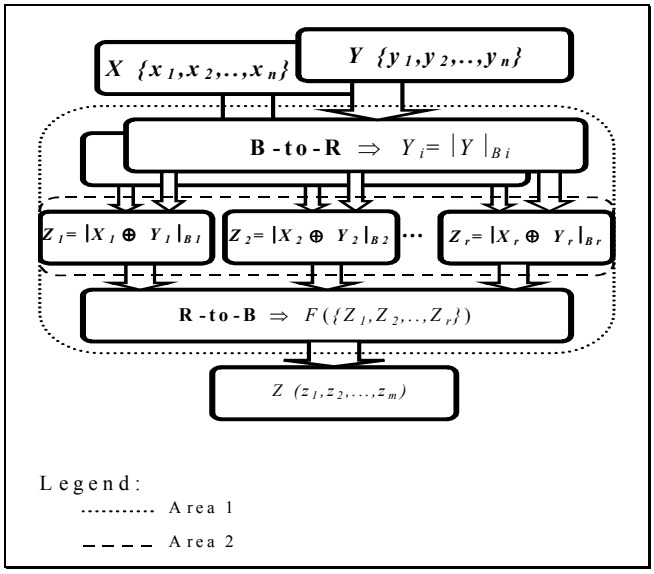


Fig. 1. General scheme of sequential computation in RNS

Using RNS there is exchanged n-bits value arguments into few smaller k_i-bit values, where $k_i < n$ and $1 \le i \le r$. Both numbers k and r (quantity of parallel channels) are depended on a method of construction RNS. Shorting of computation (area 2 in Fig. 1) time can be obtained by decreasing of bits number in separated channels. Increasing of number of moduli causes growing up complexity of R-to-B and B-to-R converters. Choice of moduli for building of RNS, in order to meet big parallelisms, faster conversion and unique representation in all range $W = \Pi_i B_i$ for $i=1,2...r$ is non-trivial problem [2,5].

2.1. "Bit grouping" method in realization of converters

There are many methods for choice of RNS moduli [8]. One of the most used is the method, where moduli are based on power *2*, in the form 2^k-1 and 2^k+1. Calculation of residues relies on addition or subtractions of k-bit groups obtained from original n-bit binary representation (independent at amount position $W\{w_{n-1}, ... w_1, w_0\}$). Interpreting of B-based k-bit groups of a number $W\{w_{n-1}, ..., w_1, w_0\}$, together with their weight, as digits of base-constant system B^k, there is easily converted this number into a vector $\{W_{s-1}, ..., W_1, W_0\}$ in the B^k-based number system.

Thus:
$$W=\sum_{i=0}^{n-1} w_i B^i = \sum_{i=0}^{s-1}\left(\sum_{l=0}^{k-1} w_{ik+1} B^l\right) B^{ki} = \sum_{i=0}^{s-1} W_i B^{ki} \qquad (1)$$

where $W_i = \sum_{l=0}^{k-1} w_{ik+l} B^l$ are values of a number W in the B^k-based system.

Basing on this "bit grouping" method, and properties of modulo operation presented in [1], residues for moduli B^k-*1* and B^k+*1* can be calculated without division, and can be reduced to addition and subtraction related bit groups, in accordance with the following rules:

$$|W|_{B^k-1} = \left|\sum_{i=0}^{s-1} W_i B^{ki}\right|_{B^k-1} = \left|\sum_{i=0}^{s-1} W_i\right|_{B^k-1} \qquad (2)$$

If the value W_i is equal to B^k-*1*, then W_i is omitted. Similar situation is in equation:

$$|W|_{B^k+1} = \left|\sum_{i=0}^{s-1} W_i B^{ki}\right|_{B^k+1} = \left|\sum_{i=0}^{s-1} (-1)^i W_i\right|_{B^k+1} \qquad (3)$$

where W_i is omitted, if W_i is equal to B^k+*1*.

Several authors implement effective *B-to-R* converters without ROM by using adders only. Very often used method of RNS building is based on three-moduli in the form:

$B_1=2^k-1, \qquad B_2=2^k, \qquad B_3=2^k+1$

$W=B_1*B_2*B_3=2^n-2^k$ is the RNS domain, and range is of *[0÷W-1]*. The method of a residue computation up to help pattern 3 for binary 12-bit representation of argument *A* is shown in Fig. 2.

$$W=\sum a_i 2^i$$
$$rB_1 = |W|_{B1}$$
$$rB_2 = |W|_{B2}$$
$$rB_3 = |W|_{B3}$$

$$for\ a=\{0,\ 1\}\ and\ i=0\div 11$$
$$\underbrace{a_{11},a_{10}}_{W_2},\underbrace{a_9,a_8,a_7,a_6}_{W_1},\underbrace{a_5,a_4,a_3,a_2}_{W_0},a_1,a_0 \qquad (1)$$
$$rB_1 = |W_0+W_1+W_2|_{B1} \qquad (2)$$
$$rB_2 = W_0$$
$$rB_3 = |W_0-W_1+W_2|_{B3} \qquad (3)$$

Fig. 2. Example of bit grouping for residue computation purpose

Basing on scheme shown in Fig. 2, its possible building efficiency less ROM circuits realization of residue-to-binary and binary-to-residue conversion.

2.2. Binary to residue conversion (B-2-R)

Based at Bit grouping method from Fig. 2 is possible realized $(W_0-W_1+W_2)$ in regular structure combinational block [3] near-full adder (Cary Save Adder, CSA). For this circuits

(CSA) the residue generation process for the most time consumption channel with modulus 2^n+1 is performed at similar addition of two arguments at time. Taking advantage of additional modulo operations is possible by constructing efficient residue generator. Follow-up results at order position and chose equivalently proper value is realized by residue-to-binary conversion. This method is shown at Fig. 3. Analogical but in shorter time it is possible to realize a residue generator for moduli 2^n-1. In order to evaluate residue for moduli 2^n it is sufficient to copy *k* bit from the original value.

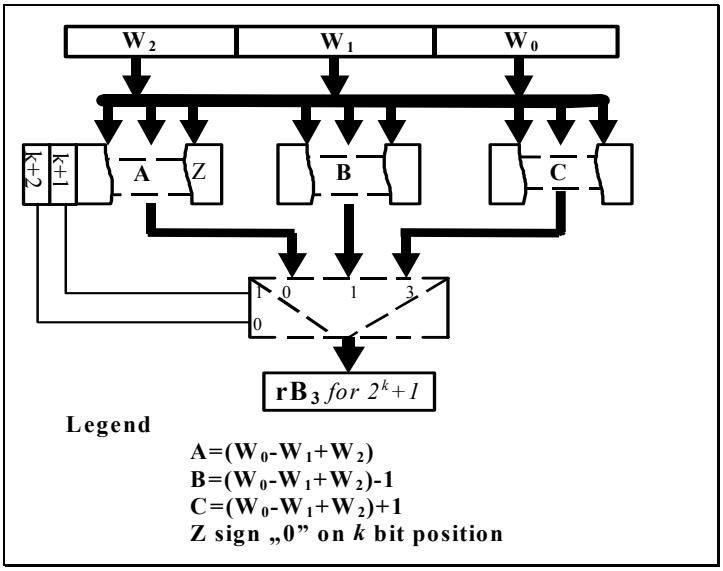


Fig. 3 Scheme determine residue for 2^n+1 moduli.

Increasing or decreasing range residue generator is performed by copy adequate amount of combinational blocks. Is obtaining a regular structure of combinational block is important in this B-*2-R* method. Also is important for stream processing is obtaining similar time conversion and addition in 2^n+1 moduli channel.

2.3. Residue-to-binary conversion (R-2-B)

Knowing values rB_1, rB_2, rB_3 show Fig. 2 it's possible to compose a system of equations for computation of W_0, W_1, W_2 Calculation and combining of $W_2 \& W_1 \& W_0$ is a binary representation of a finding number *W*.

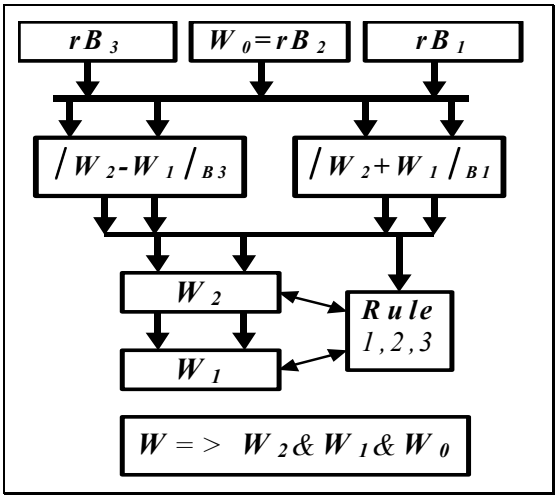


Fig. 4 Scheme sequential realization a residue-to-binary conversion.

Sequential model of *R-2-B* conversion is based at bit grouping method demonstrated in [9] with additional rule property modulo operations show at Fig. 4:

$$0 \le (W_1+W_2) \le 2(B-1)-1 \qquad \textit{(rule 1)}$$

$$W_1+W_2 \ge |W_1-W_2| \qquad \textit{(rule 2)}$$

$$[(W_1+W_2)+(W_1-W_2)] \text{ condition of parity} \qquad \textit{(rule 3)}$$

3. STREAM PROCESSING STRUCTURE

Using presented sequential conversion model, time of sequential stage is similar to the time of operation in the most time-consuming channel 2^n+1 moduli. This method makes possible implementation of pipeline for increasing productivity [10]. For stream processing important time specified stage and number steps to make calculation of a first value. Based at this method the first result is obtained after five steps, and is time step proportional to operation time at n/3 bit position. To estimate minimal value operations in series below is used the equation:

$$mT_n=T_{B\text{-}2\text{-}R}+(m+1)T_k+T_{R\text{-}2\text{-}B} \qquad (4)$$

where:

- m - value operations
- $T_{B\text{-}2\text{-}R}$ - time *B-2-R* conversion
- $T_{B\text{-}2\text{-}R}$ - time *R-2-B* conversion
- T_n - time askew *n* position carry propagation (look-ahead propagation)
- T_k - time askew *n/3* position carry propagation

substitution this value to equations (4) is

$$m=\frac{T_{B-2-R}+T_k+T_{R-2-B}}{T_n-T_k}=\frac{\frac{1}{3}+\frac{1}{3}+1}{1-\frac{1}{3}}=\frac{\frac{5}{3}}{\frac{2}{3}}=\frac{5}{3}*\frac{3}{2}=\frac{15}{6}=2\frac{1}{2}$$

4. CONCLUSIONS

In the paper a new method of serial realization of basically arithmetical operations using residue number system (RNS) is present. RNS consists of three values, and use moduli in form of $\{2^n-1, 2^n, 2^n+1\}$. This method, unlike others known [1][7], based directly on bit grouping algorithm. Because of the fact that realization time of arithmetic operation is proportional to number of bits in arguments, using of Residue Number Systems makes possible increasing of performance of serial arithmetic operations. It is obtained through parallel calculation on less-bits arguments that represent together the whole input data. In addition, the regular structures of the converter are easily implemented in FPGA circuits. Using many-context FPGAs, especially FPSLIC [11], there are possible changes of RNS range at run time. For such RNS representation, a time of *B-to-R* conversion and of simple arithmetical operation, are comparable, and are proportional to value *k*. The presented approach and the sequential method of RNS computation give opportunity to compute with a dynamic range in pipeline. In this approach time-profit is obtain for three operations train. Profit is independent at bit positions arguments and increasing with growth number of operations.

REFERENCES

[1] M.Bhardwaj, A.B.Premkumar and T.Srikanthan, "Breaking the 2n-Bit Carry Propagation Barrier in Residue to Binary Conversion for the [2n-1, 2n, 2n+1] Modula Set", *IEEE Transactions on Circuits and Systems - I*, Vol.45, No.9, September 1998, pp.998-1002

[2] M.Bhardwaj, T.Srikanthan and C.T.Clarke, "VLSI Cost of Arithmetic Parallelism: A Residue Reverse Conversion Perspective", *Proceedings of the 14th IEEE Symposium on Computer Arithmetic, ARITH 14*, Adelaide, Australia, 14-16 April 1999

[3] J.Biernat, *Computer Arithmetics*, Wydawnictwo Naukowe PWN, Warszawa, 1996, (in Polish)

[4] S.Kozielski and Z.Szczerbiński, *Paralel Computers*, WNT, Warszawa, 1993 (in Polish)

[5] A.Svoboda, "Rational numerical system of residual classes", *Stroje na Zapracowani Informaci, Sbornik V,* Praha, 1957

[6] N.S.Szabo and R.I.Tanaka, *Residue arithmetic and its applications to computer technology*, McGraw-Hill, New York, 1967

[7] J.Piestrak, "A High Speed Realization of a Residue to Binary Number System Converter", *IEEE Transactions on Circuits and Systems - II*, Vol.42, No.10, October 1995, pp.661-663

[8] Z.Ulman, " Residue Number Systems with Quasi-Base", *Thesis*, *Technical University of Gdańsk*, No.83, Gdańsk, 1998 (in Polish)

[9] J.Jabłoński, M.Węgrzyn, "FPGA realisation of a residue to binary number system converter", *Polish German Symposium SRE'2000,* Zielona Góra, 28-29.09.2000 (in Polish)

[10] J.Jabłoński, J.Biernat, "FPGA realisation of a residue number system-based circuit", *RUC'2001, Szczecin,* 7-8.05.2001, pp.215-220, (in Polish)

[11] Atmel, Inc., *http://www.atmel.com*

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

BLOCK SYNTHESIS OF COMBINATIONAL CIRCUITS IN THE BASIS OF PLA AND LIBRARY GATES

Pyotr BIBILO, Natalia KIRIENKO

Institute of Engineering Cybernetics of National Academy of Sciences of Belarus,
Surganov str. 6, 220012 Minsk, BELARUS, <*Bibilo, Kir*>*@newman.bas-net.by*

Abstract. *Circuit realization in one PLA may be unacceptable because of the large number of terms in SOP, therefore a problem of block synthesis is considered in this paper. This problem is to realize a multi-level form of Boolean function system by some blocks, where each block is PLA of smaller size. A problem of block synthesis in gate array library basis is discussed in this paper too. The results of experimental research of influence of previous partitioning of Boolean function systems on circuit complexity in PLA and gate array library basis are represented in this paper.*

Key Words. *Synthesis of Combinational Circuits, Partitioning, PLA, Gate Array Library.*

1. INTRODUCTION

There are different ways of implementation of the control logic of custom digital VLSI circuits. The most important ways are the realization of two-level AND/OR circuits in Programmable Logic Arrays (PLA) basis [9] and the realization of multi-level circuits in library gates basis [7]. Each of them has its advantage and disadvantage. The advantage of PLA-circuits is simplicity of layout design, testing and modification, because circuits are regular. There are effective methods and programs of the PLA-area minimization [9,4]. The disadvantage of two-level PLA-circuit is the large chip area compared with the area required for multi-level library gates circuit. But the synthesis of a multi-level circuit is a very difficult task [5], moreover, such circuits are harder for testing and topological design than PLA-circuits. The implementation of a circuit in one PLA may be unacceptable because of the large size of the PLA. Therefore a problem of block synthesis is considered in this paper. The problem is to realize a multi-level form of Boolean functions system by some blocks, every of which being a PLA of smaller size. The problem of block synthesis in the library gates (LG) basis is considered in this paper too. The results of experimental research of influence of preliminary partitioning of Boolean function systems on circuits complexity in the PLA and LG basis are given.

2. REPRESENTATION OF BOOLEAN FUNCTIONS AND THE BASIS OF SYNTHESIS

It is well known, that the formal (mathematical) model of functioning of multi-output combinational circuit is a system of completely defined Boolean functions [9]. Let a combinational circuit have n inputs and m outputs. One of the forms of Boolean function system representation is the sum-of-product (SOP) system. Let us denote by $D_f(n,k,m)$ the system of Boolean functions $\boldsymbol{f}(\boldsymbol{x})=(f^1(\boldsymbol{x}),\ldots,f^m(\boldsymbol{x}))$, $\boldsymbol{x}=(x_1,\ldots,x_n)$, specified on k common elementary products of Boolean variables $x_1,\ldots,x_n$. Let us represent $D_f(n,k,m)$ by a pair of matrices – the ternary $k\times n$ matrix $\boldsymbol{T}^{\boldsymbol{x}}$ and the Boolean $k\times m$ matrix $\boldsymbol{B}^{\boldsymbol{f}}$. A pair of the appropriate rows of $\boldsymbol{T}^{\boldsymbol{x}}$ and $\boldsymbol{B}^{\boldsymbol{f}}$ represents accordingly the product and the subset of the functions it belongs to their SOPs. The representation of a system of Boolean functions in the form of SOP system will be called two-level representation. We refer to specification of a system of Boolean functions in the form of a system of parenthesised algebraic expression in the basis of logic AND, OR, NOT operators as the multi-level representation.

A programmable logic array is a classical two-level structure for realization of SOP-system of Boolean functions. A system of SOP $D_f(n,k,m)$ can be realized on $PLA(n,m,k)$, which has not less than n input pins, m output pins and k intermediate lines. Elementary products are realized on intermediate lines of matrix AND of PLA, sum-of-products are realized in matrix OR of PLA. The PLA structure is adequate to the system of SOPs $D_f(n,k,m)$. The commutation points between input pins and intermediate lines in AND matrix correspond to fixed (0,1) elements of $\boldsymbol{T}^{\boldsymbol{x}}$, the commutation points between output pins and intermediate lines in OR matrix correspond to elements 1 of $\boldsymbol{B}^{\boldsymbol{f}}$.

The library gates used as basis elements for synthesis of combinational logic circuits are the elements from the logic gate library K1574 [3]. Each element of such a library is characterized by the number of basis gates needed for its location in the chip. There are various elements in the gate library K1574: inverters, multiplexers, buffer elements, gates AND, OR, NAND, NOR, XOR etc.

3. ALGORITHM FOR PARTITIONING MULTI-LEVEL REPRESENTATION OF A SYSTEM OF BOOLEAN FUNCTIONS

Let us consider the multi-level algebraic form of a Boolean function system in AND/OR/NOT basis. Each intermediate or output function is given by a separate SOP. Let non-overlapping SOP subsets $R_1,\ldots,R_h$ form a partition of SOP set $D=\{D^1,\ldots,D^m\}$. $R_1,\ldots,R_h$ are the blocks of the partition with the following parameters: n_i is the number of input variables, m_i is the number of output variables, k_i is the number of products in two-level form of SOP of functions of the block.

Let the restrictions (n^*, m^*, k^*) on block complexity be given: n^* is the maximal number of input variables, m^* is the maximal number of output variables, k^* is the maximal number of products in two-level representation of SOP of functions of the block.

Transformation of multi-level representation to two-level one and determination of the parameter k_i is connected with solving problems of intermediate variables elimination [6] and joint minimization of a system of Boolean functions in SOP form [4, 5, 9].

The problem of partitioning multi-level representation of a system of Boolean functions is to find a partition $R_1,\ldots,R_h$ which fulfils (n^*, m^*, k^*)-restriction and has a minimum number of blocks h.

The main idea of the technique for solving this problem consists in the following. The blocks (or SOP subsystems) are building up step-by-step. A SOP with the maximal number of external variables is chosen out as starting for formation of the next subsystem. Then, SOPs that are most closely connected to this subsystem (on common variables) are added to this subsystem, the elimination of intermediate variables and joint minimization of functions in SOP subsystem are performed. The resulting subsystem is checked for the fulfilment of (*n*, m*, k**)-restriction. If (*n*, m*, k**)-restriction isn't violated, then the next SOP is added to this subsystem, otherwise the constructing the next subsystem begins. As a result, each SOP is in one of the subsystems. This algorithm was described in detail in [2].

4. BLOCK METHOD FOR SYNTHESIS IN PLA BASIS

The block method for synthesis in PLA basis has two procedures. The first procedure is the partitioning multi-level representation of the system of Boolean functions into blocks with (*n*, m*, k**)-restricted parameters. The second procedure is realization of each block by one PLA.

Let the area of a circuit consisting of some (possibly interconnected) PLAs be equal to the sum of the areas of these PLAs. The conditional area of one PLA(*n*,*m*,*k*), evaluated in conditional units (bits), is determined by the formula

$$S_{PLA}^{log} = (2n + m)k \text{ (bit)}. \quad (1)$$

At the level of particular layout used in silicon compiler SCAS [1], the real PLA area is determined by the formula

$$S_{PLA}^{top} = S_{AND,OR} + S_{BND}, \quad (2)$$

where $S_{AND,OR}$ is the area of information matrixes *AND, OR* determined by the formula

$$S_{AND,OR} = 2\left\lceil\frac{k}{8}\right\rceil\left(9\left\lceil\frac{n}{2}\right\rceil + 10\left\lceil\frac{m}{4}\right\rceil\right), \quad (3)$$

S_{BND} is the area of PLA-boundary (load transistors, buffers etc.), determined by the formula

$$S_{BND} = 34{,}992\left\lceil\frac{k}{8}\right\rceil + 85\left\lceil\frac{n}{2}\right\rceil + 49{,}104\left\lceil\frac{m}{4}\right\rceil + \left(\left\lceil\frac{m}{4}\right\rceil - 1\right)\cdot 12{,}276 + 60{,}423. \quad (4)$$

The value of S_{PLA}^{top} determined by formulas (3), (4) is the number of real layout cells that the PLA layout is composed of [1].

5. BLOCK METHOD FOR SYNTHESIS IN THE GATE ARRAY LIBRARY BASIS

The block method for synthesis in gate array library basis has two procedures. The first procedure is the partitioning of multi-level representation of the system of Boolean functions into blocks with (*n*, m*, k**)-restricted parameters. The second procedure is realization of technology mapping each block into the logic gate library. This procedure includes synthesis in the gate array library using a basic method "Cover".

The basic method "Cover" is a process of covering Boolean expressions in AND/OR/NOT basis by elements from the gate library. Previously each multi-place AND(OR) operator of the system is replaced by superposition of two-place AND(OR) operators, respectively. Then the Boolean network is building for each expression, where each node corresponds to two-place operator AND(OR) or one-place operator NOT. The problem of covering Boolean network is to find subnetworks in it, which are functionally equivalent to library elements. The basic method "Cover" was described in detail in [3]. It is experimentally confirmed to be better than the method represented in [7].

6. EXPERIMENTAL INVESTIGATION OF BLOCK METHODS FOR SYNTHESIS IN PLA AND GATE ARRAY LIBRARY BASISES

The block methods for synthesis were realized in computer programs and investigated experimentally. The experiments were done on a series of combinational circuits from well-known MCNC benchmark set chosen from design practice. The programs run on PC Celeron 600, RAM 64 Mb.

Experiment 1. Two realizations of multi-level representation in PLAs were compared: the first one is realization in one PLA; the second one is realization in several PLAs, obtained by partition algorithm. Table 1 shows the results of Experiment 1.

Table 1. The comparison of two realizations of multi-level representation for PLAs: the first is realization in one PLA; the second is realization in several PLAs, obtained by partition algorithm.

Circuit name	n	m	k	One PLA		PLA net			
				S_{PLA}^{log}	S_{PLA}^{top}	n*, m*, k*	h	$\sum S_{PLA}^{log}$	$\sum S_{PLA}^{top}$
x1	51	35	274	37538	26715,29	40,20,500	3	***24521***	***21982,5***
Apex6	132	94	432	154656	99092,84	100,70,900	3	104864	73608,43
						105,80,900	2	113252	79032,04
						70,40,900	4	***75872***	***60442,43***
Apex7	49	35	213	28329	20680,35	30,30,500	3	17104	16603,77
						35,30,500	2	18666	16008,26
						25,20,500	4	***8825***	***10526,71***
example2	85	63	161	37513	28394,06	40,30,900	4	***19501***	20406,34
						60,40,900	2	23450	20429,21
						55,35,900	2	21227	***19194,59***
X4	94	71	371	95347	63474,61	60,35,900	4	***40638***	***35639,54***
						66,50,900	3	46910	46237,44
Frg2	143	139	3090	1313250	794310,4	50,30,500	11	***171581***	***166857,8***
Too_large	38	3	1021	***80659***	***52539,5***	50,2,900	4	102170	73014,45
Ttt2	24	21	222	15318	11824,2	24,10,900	3	10773	13958,77
						24,8,900	5	***8092***	***9907,832***
Cm150a	21	1	796	34228	26343,73	16,2,500	3	***6155***	***6422,373***
						18,3,500	2	11623	14899,51
Frg1	28	3	119	7021	***5904,407***	26,1,100	3	***5783***	6441,453
						28,2,100	2	6523	8530,519
Lal	26	19	117	8307	***6994,927***	30,16,100	5	***4574***	8248,276
Add8	17	9	2519	108317	81949,77	12,9,900	2	***9324***	***9184,462***
X3	135	99	915	337635	209646**,**7	80,50,900	5	***170755***	***122437,9***
Term1	34	10	818	63804	42979,46	30,8,500	2	49896	56050,37
						34,4,500	3	36750	27895,76
						20,8,500	8	***21327***	***23514,69***
Mux	21	1	425	18275	14706,1	16,10,100	2	1862	***3025,974***
						12,6,100	3	***1519***	3270,493
count	35	16	89	7654	7091,571	25,8,100	4	***2769***	***5072,029***
						18,6,100	6	4851	5606,067

Experiment 2. Three realizations of multi-level representation in the basis of logic gate library were compared: the basis method "Cover", combining method and block method of synthesis. The combining method is composed of two steps. The first step is transformation of multi-level representation into two-level representation. The second step is synthesis by basis method "Cover". Table 2 shows the results of Experiment 2.

Table 2. Comparison of basis method, combining method and block realization of multi-level representation

Circuit name	n	m	Basis method		Combining method			Block realization		
			L	S	k	L	S	h	L	S
cu	14	11	41	***204***	35	76	397	4	41	219
comp	32	3	110	555	3	12	***70***	2	23	123
cmb	16	4	27	***142***	56	116	630	3	38	212
cm162a	14	5	43	198	20	57	290	2	24	***117***
mux	21	1	61	***319***	425	1032	6094	5	61	327
count	35	16	111	506	89	188	1024	13	76	***356***
frg1	28	3	298	1683	119	298	1683	3	298	***1683***
cm138a	6	8	9	53	6	16	88	3	9	***53***
cm82a	5	3	20	90	11	14	74	3	13	***57***
9symml	9	1	154	738	30	70	***359***	5	141	727
lal	26	19	117	***576***	117	207	1141	3	180	849
unreg	36	16	96	432	49	80	384	2	80	***384***
z4ml	7	4	102	554	19	26	***140***	2	102	554
x3	135	99	933	***4475***	915	1948	10808	4	966	4679
pcle	19	9	39	181	16	40	214	5	36	***173***
term1	34	10	495	2407	818	2414	13403	7	465	***2360***
cm150a	21	1	61	***261***	796	2136	11886	4	65	309
too_larg	38	3	5217	30145	1027	5273	30473	4	4915	***28392***
ttt2	24	21	299	1609	222	561	2937	5	225	***1197***
sct	19	15	120	584	64	124	618	6	127	***583***
c8	28	18	159	798	70	92	464	4	92	***452***
frg2	143	139	1315	***6560***	3090	15385	85093	15	1361	6886
cm42a	4	10	13	***65***	4	20	94	3	14	72

The minimization of two-level representations of Boolean function system in partition algorithm was done by the program of joint minimization in SOP [8]. The basis method "Cover" uses the computer program from [3].

The notation in tables 1, 2 is as follows:

n – the number of arguments of realized Boolean function system (the number of input pins in the circuit);

m - the number of functions in the system (the number of output pins in the circuit);

k - the number of products in the SOP system (two-level representation);

S_{PLA}^{log} - the conditional area of one PLA(n,m,k), evaluated in conditional units (bits) by the formula (1);

$\sum S_{PLA}^{log}$ - the sum of conditional areas of PLAs, found by the block synthesis method;

S_{PLA}^{top} - the real area of one PLA(n,m,k), evaluated by the formula (2);

$\sum S_{PLA}^{top}$ - the sum of real areas of PLAs, found by the block synthesis method;

S – the circuit complexity in library gates basis (the total number of gates, required for logical elements, i.e. area for elements);

L – the number of logical elements in a circuit;

n^* – partition parameter (the number of arguments of a block);

m^* – partition parameter (the number of functions in a block);

k^* – partition parameter (the number of products in a block);

h – the number of blocks in the partition of multi-level representation of Boolean functions system.

According to the results of the experiment the following conclusions can be stated.

1. The block method for synthesis of multi-level representation by a PLA net is more preferable, than one PLA realization. The gain for area is obtained in 13 circuits from 16. The circuits with the smallest area are printed in bold type (see table 1). Only macroelement area was taken into account in this experiment, and the bound area was not. Thus the final conclusion about replacement of one PLA with PLA net can be done after layout design. Using formula (2) for area calculation is more preferable, than (1). For example, area calculation for Frg1, Lal with (1) gives advantage, but the real PLA net area is more then the one PLA area.
2. The block realization is more preferable for synthesis in logic gate library too. The better (minimum) valuations of circuit complexity are printed in bold type (see table 2). The ttransformation multi-level representation into two-level representation (combining method) is advisable in only three examples; it is not competitive with the basis method "Cover" and the block method.

REFERENCES

[1] P.N.Bibilo, "Symbolic Layout of VLSI Array Macros with an SCAS Silicon Compiler. I", *Russian Microelectronics,* Vol.27, No.2, pp.109-117, 1998

[2] P.N.Bibilo & N.A.Kirienko, "Partitioning a System of Logic Equations into Subsystem under Given Restrictions", *Proc. of the Third International Conference on Computer-Aided Design of Discrete Devices* (*CAD DD'99*), Minsk, Republic of Belarus, Vol.1, pp.122-126, 1999

[3] P.N.Bibilo & V.G.Litskevich, "Boolean Network Covering by Library Elements", *Control Systems and Computers,* Kiev, Ukraine, Vol.6, pp.16-24, 1999, (in Russian)

[4] K.R.Brayton, G.D.Hachtel, C.T.McMullen & A.L.Sangiovanni-Vincentelli, *Logic minimization algorithm for VLSI synthesis.* Kluwer Academic Publisher, Boston, e.a., 1984

[5] K.R.Brayton, R.Rudell, A.L.Sangiovanni-Vincentelli & A.R.Wang, "MIS: A Multiply-level Logic Optimisation Systems", *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems,* Vol.CAD-6, pp. 1062-1081, 1987

[6] L.D.Cheremisinova & V.G.Litskevich, "Realization of operations on systems of logic equations and SOPs", *Logical design,* Minsk: Institute of Engineering Cybernetics of National Academy of Sciences of Belarus, pp.139-145, 1998, (in Russian)

[7] F. Mailhot & G. De Micheli, "Algorithms for technology mapping based on binary decision diagrams and on Boolean operations", *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, Vol.12, No 5, pp.599-620, 1993

[8] N.R.Toropov, "Minimization of systems of Boolean functions in DNF", *Logical Design,* Minsk: Institute of Engineering Cybernetics of National Academy of Sciences of Belarus, pp.4-19, 1999, (in Russian)

[9] A.D.Zakrevskij, *Logic Synthesis of the Cascade Circuits,* Nauka, Moscow, 1981, (in Russian)

SESSION IV:
HARDWARE MODELLING

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

DESIGN OF EMBEDDED CONTROL SYSTEMS USING HYBRID PETRI NETS

Thorsten HUMMEL, Wolfgang FENGLER

Ilmenau Technical University, Department of Computer Architectures,
P.O. Box 10 05 65, 98684 Ilmenau, GERMANY,
<*thummel|wfengler*>*@theoinf.tu-ilmenau.de*

Abstract. *The paper describes the challenges of modeling embedded hybrid control systems at a higher abstraction level. It discusses the problems of modeling such systems and suggests the use of hybrid Petri nets. Modeling an exemplary embedded control system with a special hybrid Petri net class using an object-oriented modeling and simulation tool shows the potential of hybrid Petri nets.*

Key Words. *Embedded Control Systems, Hybrid Petri Nets*

1. INTRODUCTION

The design of complex embedded systems makes high demands on the design process due to the strong combination of hardware and software components. These demands rise rapidly, if the system includes components of different time and signal concepts. That means there are systems including both event parts and continuous parts. Such systems are called heterogeneous or hybrid systems.

The behavior of such hybrid systems cannot be covered homogeneously by the well-known specification formalisms of the different hardware or software parts because of the special adaptation of these methods to their respective field of application and the different time and signal concepts the several components are described with. A continuous time model usually describes continuous components, whereas digital components are described by discrete events.

For describing both kinds of behavior in its interaction, there are different approaches to describe such systems. On the one hand, the different components can be described by their special formalisms. On the other hand, homogeneous description formalism can be used to model the complete system with its different time and signal concepts, and that is what we are in favor of.

Therefore, we have investigated modeling methods that can describe the behavior of such systems homogeneously at a high abstraction level independently from their physical or technical details. Apart from considering the heterogeneity, the modeling method must cope with the high complexity of the modeled system. This demand requires support for modularization and partitioning, and capabilities for hierarchical structuring.

In the following, a graph based formal modeling approach is presented. It is based on a special Petri net class, which has extended capabilities for modeling of hybrid systems. To model the hybrid systems, we have used an object-oriented modeling and simulation tool based on this Petri net class. This tool can be used for modeling hybrid systems from an object-oriented point of view. It can be used for modeling and simulating components or subsystems and offers capabilities for hierarchical structuring.

2. HYBRID PETRI NETS

The theory of Petri nets has its origin in C.A. Petri's dissertation "Communication with Automata" [1], submitted in 1962. Petri nets are used as describing formalism in a wide range of application fields. They offer formal graphical description possibilities for modeling systems consisting of concurrent processes. Petri nets extend the automata theory by aspects like concurrency and synchronization.

A method to describe embedded hybrid systems homogeneously is the use of hybrid Petri nets [2]. They originate from continuous Petri nets introduced by David and Alla [3]. A basic difference between continuous and ordinary Petri nets is the interpretation of the token value. A token is not an individual anymore, but a real quantity of token fragments. The transition moves the token fragments from the pre-place to the post-place with a certain velocity of flow. The essence of hybrid Petri nets is the combination of continuous and discrete net elements in order to model hybrid systems.

In the past, there were applications of hybrid Petri nets described in many cases but essentially, they were concentrated on the fields of process control and automation. In the following we demonstrate the possibilities of using hybrid Petri nets to model embedded hybrid systems. The used Petri net class of Hybrid Dynamic Nets (HDN) and its object-oriented extension is described in [4] and [5]. This class is derived from the above-mentioned approach of David and Alla and defines the firing speed as function of the marking from the continuous net places.

Components or subsystems are modeled separately and abstracted into classes. Classes are templates, which describe the general properties of objects. They are grouped into class libraries. Classes can be used to create objects, which are called instances of these classes. If an object is created by a class, it inherits all attributes and operations defined in this class.

One of the important advantages in this concept is the ability to describe a larger system by decomposition into interacting objects. Because of the properties of objects, the modification of the system model could by achieved easier. The object-oriented concept unites the advantages of the modules and hierarchies and adds useful concepts like reuse and encapsulation.

3. MODELING AN EMBEDDED CONTROL SYSTEM

The application example we have chosen to discover the possibilities of using hybrid Petri nets for modeling of embedded hybrid control systems is an integrated multi-coordinate drive [6]. This is a complex mechatronic system including a so-called multi-coordinate measuring system.

Fig. 1 shows this incremental, incident light measuring system consisting of three scanning units fixed in the stator and a cross-grid measure integrated into the stage. The two y-systems allow determining the angle of rotation ν. The current x, y1 and y2 position is determined by the cycle detection of its corresponding sine and cosine signals. The full cycle counter keeps track of completed periods of the incremental measuring system. This is a precondition for the

following deep interpolation. The cycle counter of these signals is a function of the grid constant and the shift between the scanning grids and the measure. The cycle counter provides a discrete position, and in many cases, this precision is sufficient for the motive control algorithm. To support a very precise position control with :m or nm resolution, it must be decided, which possibility of increasing the measure precision is the most cost-efficient. There is a limit of improving the optic and mechanical properties because of the minimum distances in the grid.

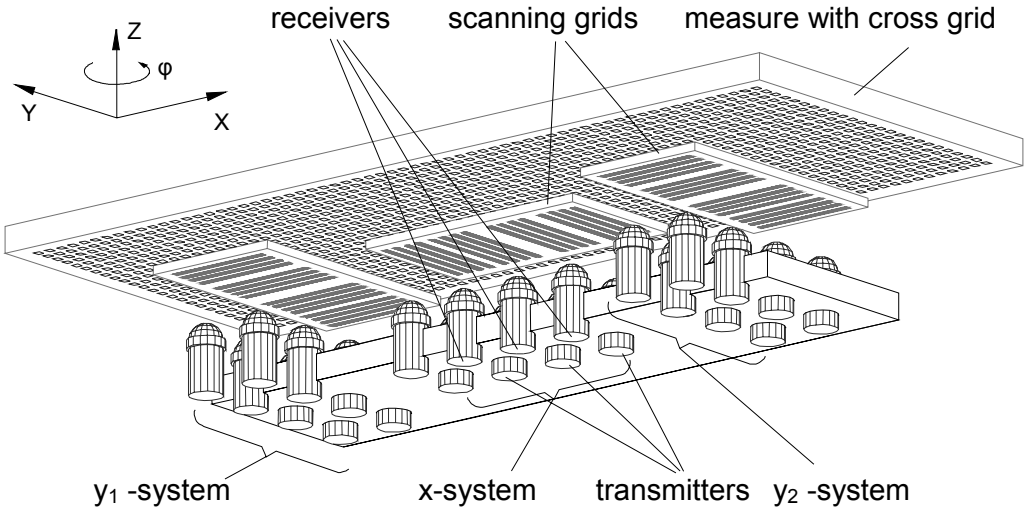
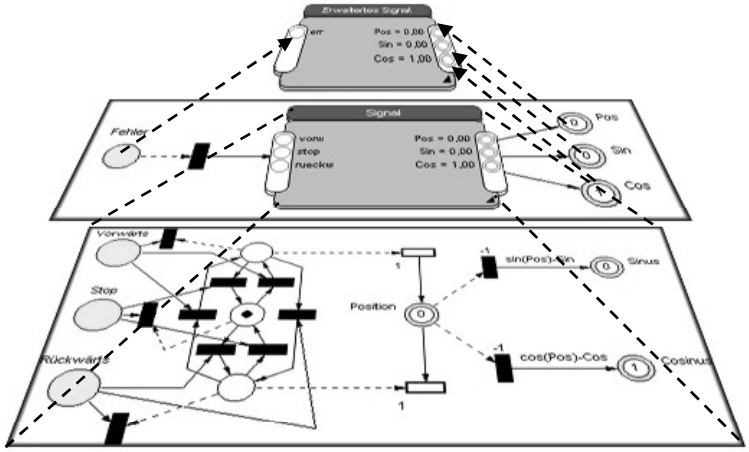


Fig. 1 Multi-coordinate measuring system

Alternatively, an interpolation within a signal period can be used, whereby the sampling rate of the A/D-Converter is increased, which would allow a more detailed evaluation of the continuous signals of the receiver. The problem to be solved in this application example leads to in modeling and simulating the measure system together with the evaluation algorithm for the position detection.



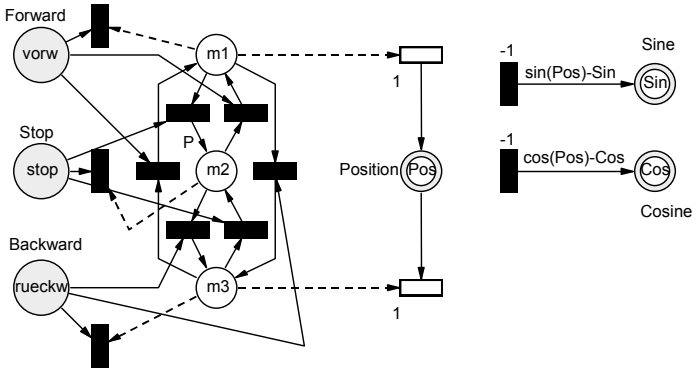
The measuring system is hierarchically modeled using components (Fig. 2).

Components with the same functionalities are abstracted into classes, put into a class library, and instantiated while modeling. The modeling of a multi-hierarchical system is possible as well.

Fig. 2 The principle of hierarchical modeling

3.1. System environment

The component “Signal generation“ (Fig. 3) simulates the sensor data and provides the sine and cosine signals as well as a position value. For clearness reasons this net is saved as a component into a subnet and gets the input places “Forward”, “Stop”, and “Backward”. It provides a sine and a cosine signal and additionally a position signal as a comparative value for a later error control function.



To simulate a potential misbehavior of the measuring system, external disturbances are modeled in the subnet "Scrambler", which is included in the component "Disturbance" of the complete system.

Fig. 3 Component "Signal generation"

3.2. Measuring system components

The position detection of one axis is modeled with the component "Axismess" (Fig. 4).

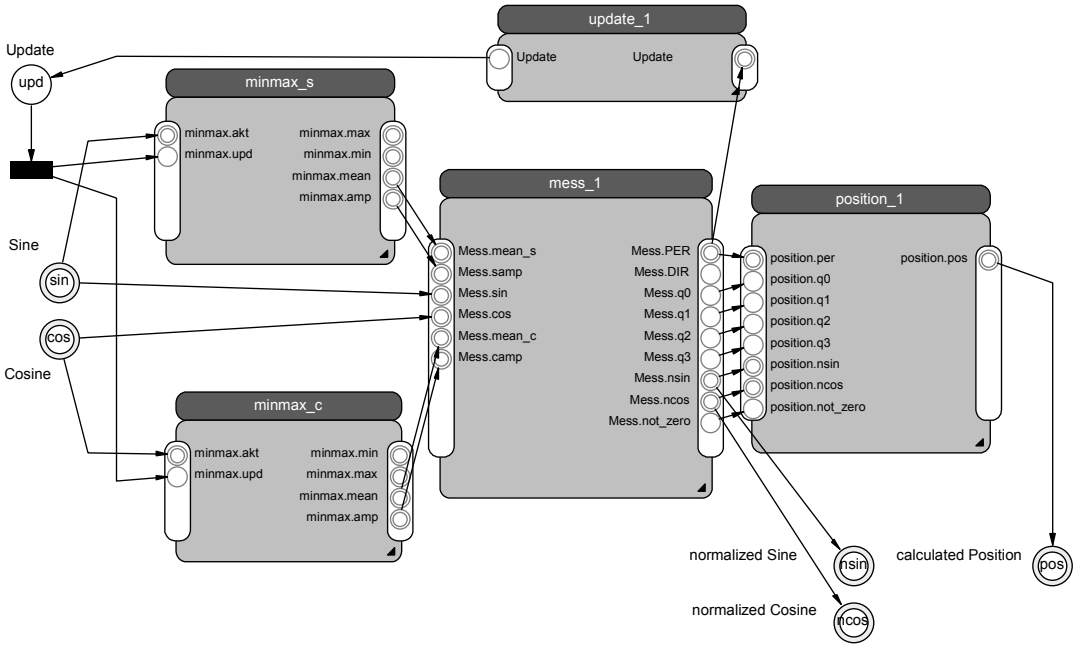


Fig. 4 Subnet "Axismess"

At first, the input signals "Sine" and "Cosine" are normalized in the subnets "Minmax_s" and "Minmax_c". These subnets are identical in its functions and were instantiated during the modeling process from the same class "Minmax" (Fig. 5).

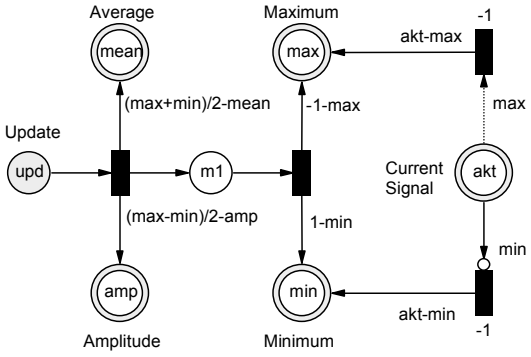


Fig. 5 Subnet "Minmax"

To find out the exact position of the carrier, the cycle number has to be determined in "Mess_1". To determine this correctly, the measuring system has to detect the moving direction of the carrier and with it the increasing or decreasing of the cycle number. The original measuring system used a look-up table, but this was very hard to model with Petri nets. Therefore, we changed this into logic rules and used this to model the subnet "Position_1".

3.3. Model of the entire system

In Fig. 6, the model of the entire system is shown. Besides the measuring system, it includes the components for signal generation and external disturbance simulation. The components for signal generation "x/y1/y2-direction" are instances of the class "Signal" and model the signals of an ideal environment.

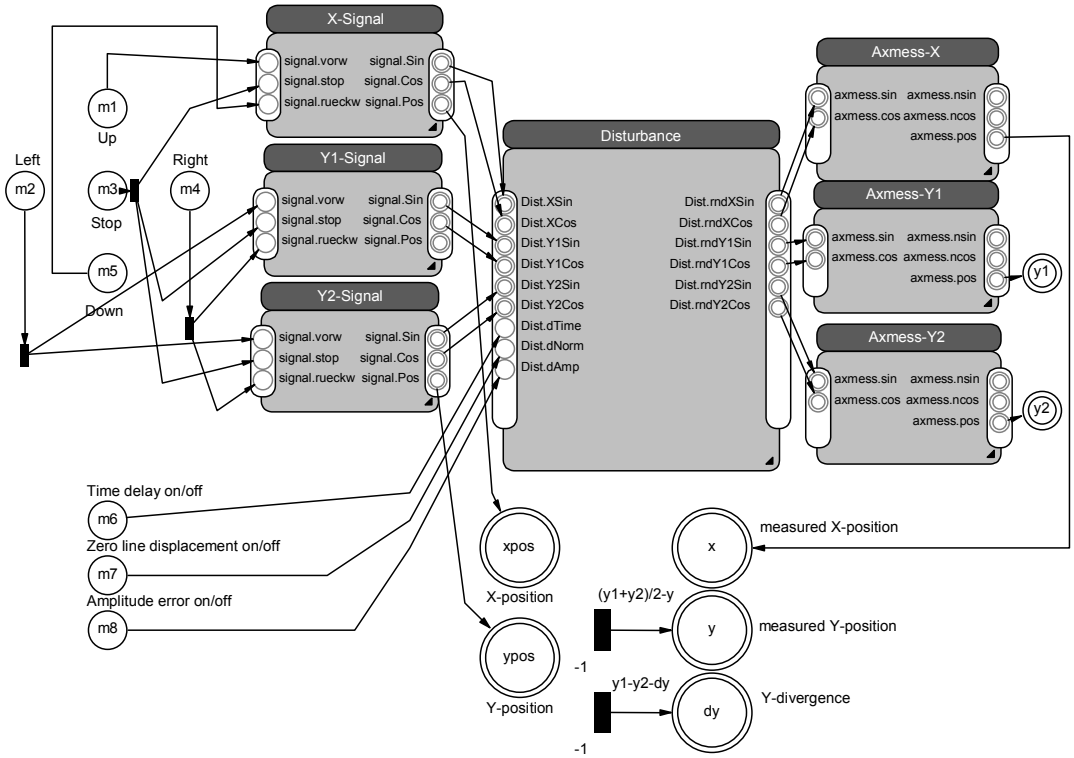


Fig. 6 Model of the entire system

The component "Disturbance" includes the simulation of various kinds of signal disturbances (displacement of the zero line, amplitude errors, time delay etc.). The signal disturbances can be turned on and off at any time during the simulation.

The objects "Axismess_x/y1/y2" are based on the class "Axismess" and include the evaluation algorithm for the three directions. The motion of any desired direction can be controlled by feeding marks into the places m1 to m8.

The x-position, the average y-position and the divergence of the y-position arose as result of the net calculation.

3.4. System simulation

The tool "Visual Object Net++" [5] allows not only the modeling but also the simulation of systems described with Hybrid Dynamic Nets. During the simulation the firing of the transitions and the transport of the tokens are animated. The changes of the place values can be visualized by signal diagrams (Fig. 7).

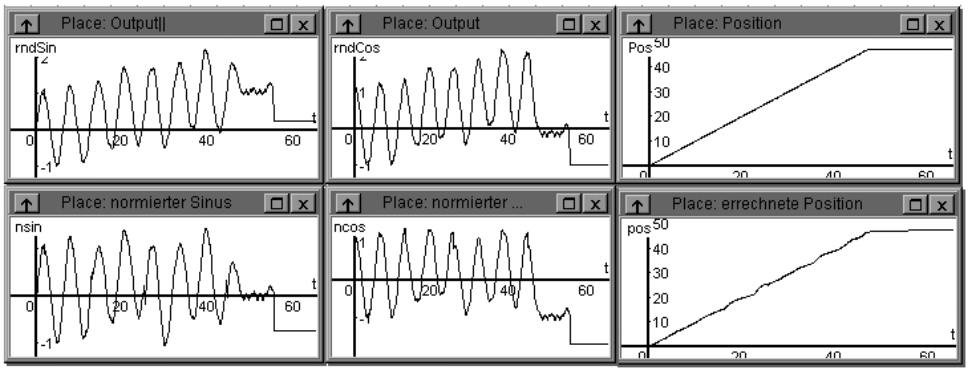


Fig. 7 System behavior with different disturbances

E.g., the middle top diagram in Fig. 7 shows an extreme example of a simulation with disturbances. It shows a clear exceeding of the zero line of the cosine signal. Nevertheless, the normal values are correctly calculated and the position of the machine is correctly displayed.

4. CONCLUSION

Our investigation has shown the advantages of using hybrid Petri nets for homogeneous modeling of an embedded hybrid system. The object-oriented approach of the hybrid Petri net class used makes possible a clear modeling of complex hybrid systems.

Future things that have to be done are the extension and completion of the system model, and the integration of the modeling process in a complete design flow. Here we focus our future work on connecting our approach to other approaches related to hardware/software partitioning.

5. ACKNOWLEDGEMENT

This research work is supported by the DFG (Deutsche Forschungsgemeinschaft, German Research Association) as part of the investigation project “Design and Design Methodology of Embedded Systems” with the subject “Design of Embedded Parallel Control Systems for Integrated Multi-Axial Motive Systems” under grant FE373/13-1.

REFERENCES

[1] Petri, C.A.: *Communication with Automata*. Schriften des IIM Nr. 2, Institut für Instrumentelle Mathematik, Bonn, 1962. (In German)

[2] Alla, H., David, R., Le Bail, J.: Hybrid Petri nets. *Proceedings of the European Control Conference*, Grenoble, 1991.

[3] Alla, H., David, R.: Continuous Petri nets. *Proceedings of the 8th European Workshop on Application and Theory of Petri nets*, Saragossa, 1987.

[4] Drath, R.: *Modeling Hybrid Systems Based on Modified Petri Nets*. PhD Thesis, TU Ilmenau, 1999. (In German)

[5] Drath, R.: Hybrid Object Nets: An Object-oriented Concept for Modeling Complex Hybrid Systems. In: *Hybrid Dynamical Systems*. Third International Conference on Automation of Mixed Processes, ADPM'98, Reims, 1998.

[6] Saffert, E., Schäffel, C., Kallenbach, E.: Control of an Integrated Multi-coordinate Drive. *Mechatronics’96*, 18.-20.09.1996, Guimaraes, Portugal, Proceedings Vol. 1, S. 151-156.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Góra, Poland

PETRI NET MODELS OF VHDL CONTROL STATEMENTS

Ewa IDZIKOWSKA

Dpt. of Control, Robotics and Computer Science, Technical University of Poznań,
pl. M. Skłodowskiej-Curie 5, 60-965 Poznań, POLAND,
Idzikowska@sk-kari.put.poznan.pl

Abstract. *Hardware description languages are used to model logical systems on the behavioural level. The model describes the system as a set of interconnected processes, which can be executed in parallel and represents two aspects of the process – computation and control. Control can be modelled by Petri nets. Such a model is very useful to analyse, which state can be reached starting from a given system state. Petri net models of all VHDL control statements are shown in this paper. These models are used to generate automatically control flow model.*

Key Words. *Hardware Description Language, Petri nets, control flow*

1. INTRODUCTION

In the VLSI area a structured design process is required. In response to this need hardware description languages (HDL) are developed. VHDL is a language for describing digital electronic systems. It is designed to fill a number of needs in the design process. Firstly, it allows description of the structure of a design, that is how it is decompose into sub-designs and how these sub-designs are interconnected. Secondly, it allows the specification of the function of designs using familiar programming language forms. Thirdly, as a result, it allows a design to be simulated before being manufactured, so that designers can quickly compare alternatives and test for correctness without the delay and expense of hardware prototyping. VHDL models are also used in the process of generating test and silicon compilation.

2. CONCURENCY

The hardware description languages must have a mechanism for modelling signals flow through the circuit. In VHDL (VHSIC Hardware Description Language) this requirement is handled by the process construct. Each process represents a block of logic and all processes execute in parallel. The process construct represents the method by which concurrent activities (parallel signal flow) in digital circuits are modelled.

The VHDL model represents two separate aspects of the process - computation and control, but the model does not explicitly distinguish between control and data. We have to find out ourselves this different semantic. The control part of circuit usually has much less states than

the data part thus it is possible and feasible to derive the control structure This structure can be represented by using Petri nets. It is interesting to analyse, which state can be reached starting from a given system state.

3. CONTROL FLOW

As mentioned above, the VHDL control information can be represented using Petri nets. The sequence of instructions, the flow of information and the order of computation performance can be modelled by means of Petri nets.

The control flow model is derived from the VHDL source code in the following way:

- places represent VHDL code statements,
- transitions represent actions - execution of code statements from its pre-places.

Control flow can be modelled with using Conditional Petri Nets with Time [3].

Def. Control flow model

The Petri-net representation of the control flow in VHDL model, CFPN, is a digraph derived from the VHDL source code, with mapping of VHDL code statements to the places. Transitions represent execution of these statements.

CFPN = (P, T, F, M_o, D, C), where:

P = { p_1, p_2,..., p_n} - a finite set of places,
T = { t_1, t_2,..., t_m} - a finite set of transitions,
M_o = { m_1, m_2,..., m_n} - an initial marking,
D = { d_1, d_2,..., d_m} - a finite set of time intervals associated with transitions,
C = { c_1, c_2,...., c_m} - a finite set of conditions associated with transitions,
F - a control flow relation.

Petri net model of the control flow is automatically derived from the VHDL source program. To do it, it's necessary to find all VHDL control statements and control variables. In order to do it, an index *S* is assigned to each statement. The Petri net place, which represents statement S, has the same index. The next chapters describe Petri net models of VHDL control statements.

4. PROCESSES AND THE WAIT STATEMENT

The primary unit of behavioural description in VHDL is a process. It is a sequential body of code which can be activated in response to changes in state. When more than one process is activated at the same time, they execute concurrently. The process is specified as follows [2]:

PROCESS*(sensitivity_list)*
process declarative part
begin
sequence of statements (1)
END PROCESS (2)

The process is activated initially during the initialisation phase of simulation. It executes all of the sequential, and then repeats, starting again with the first statement. The process may suspend itself by executing a *wait* statement. This is of the form:

WAIT ON *sensitivity_list* UNTIL *condition* FOR *time _expression*; (1)

The *sensitivity_list* of the *wait* statement and the list in the header of the process statement specify a set of signals to which the process is sensitive while it is suspended. When an event

occurs on any of these signals, it means the value of the signal changes, the process resumes and evaluates the condition. If it is true or if the condition is omitted, execution proceeds with the next statement. Otherwise the process resuspends. The *time_expression* indicates the maximum time for which the process will wait. If it is omitted, the process may wait indefinitely.

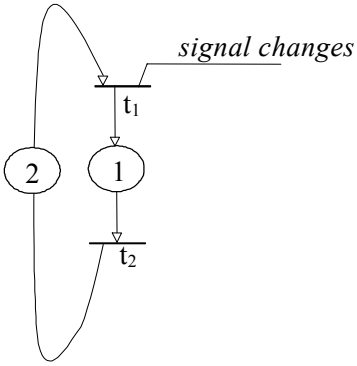


Fig.1. Petri net model of a process.

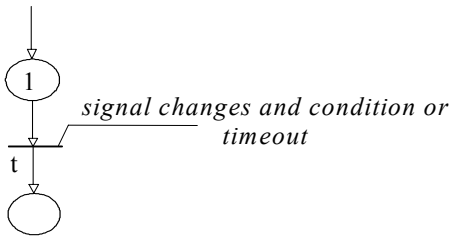


Fig.2. Petri net model of a Wait statement.

A Petri nets model of a process is shown at the Fig. 1. Transition t_1 is the conditional transition and represents an input to the process. It can be fired, when a token is in the place *2*, and the value at least one of the signal from *sensitivity_list* changes. A token in the place 2 indicates, that the process is suspended. After firing t_1 process is resumed (a token in the place *1*) and is active until transition t_2 is fired. Then the process will be suspended again.

Fig. 2 shows Petri net model of *Wait* statement. This statement suspends process. The process is resumed after firing transition *t*.

5. PETRI NET MODEL OF CONTROL STATEMENTS

In this section Petri net models of all control statements are shown. These models are used in the process of automatic control flow model derivation.

There are some control statements in VHDL:

- *IF-THEN-ELSEIF-ELSE*
- *CASE*
- *LOOP*
- *EXIT*
- *NEXT*

5.1. Conditional statement

The full form of the conditional IF statement is as follows [1]:

IF *condition_1* THEN	(1
sequence of statements	(2)
ELSEIF *condition_2* THEN	(3)
sequence of statements	(4)
ELSE	
sequence of statements	(5)
END IF;	(6)

The *elseif* and *else* clauses are optional. Petri net model of IF statement is shown at the Fig.3.

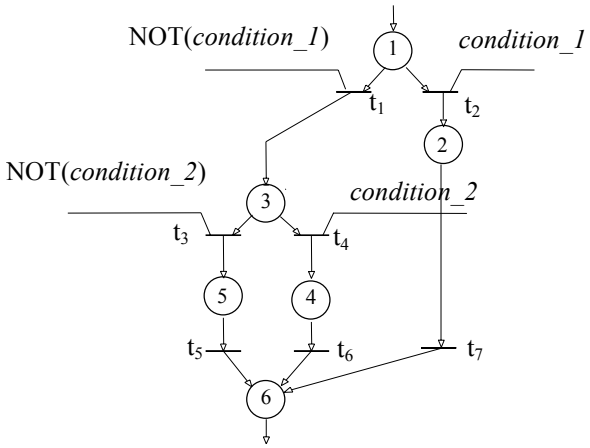


Fig.3. Petri net model of the IF statement.

Transitions t_1, t_2, t_3 and t_4 are conditional. It means, that the each of this transition may be fired if it is enabled and if its condition is *TRUE*.

Transitions t_5, t_6 and t_7 are the time transitions. Each time attributed to these transitions represents delay associated with realization of statements *5, 4* and 7 respectively.

5.2. CASE statement

The *case* statement performs decoding based on the value of a control expression and then executes a selected statement or group of statements. The full form of this statement is as follows:

CASE *n* IS (1)

WHEN *expression_1*⇒*statement_1*; (2)

WHEN *expression_2*⇒ *statement_2*; (3)

…

…

WHEN *expression_n*⇒ *statement_n*; (n+1)

END CASE; (n+2)

The Petri net model of this statement is shown at Fig.4. Transitions t_1, t_2, t_n are conditional transitions and can be fired, if the value of control expression is equal *n*.

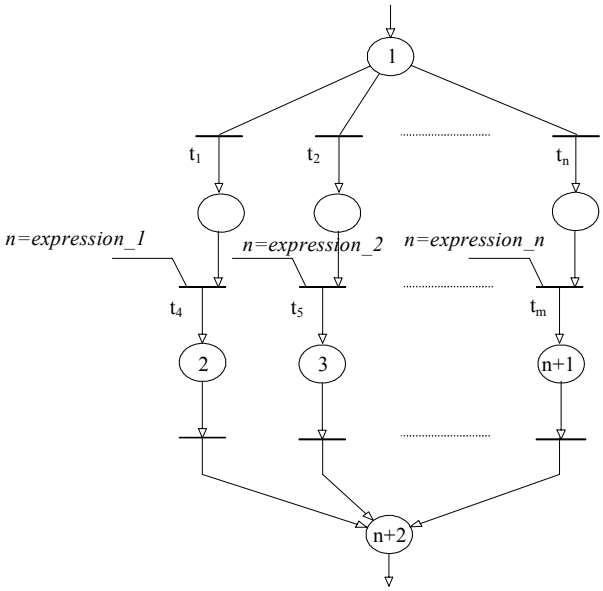


Fig.4. Petri net model of the CASE statement.

5.3. Loop statements

VHDL has a basic *loop* statement, which can be augmented to the usual forms *while* and *for* loops seen in other programming languages. The *for* iteration scheme allows a specified number of iterations. The loop parameter *l* declares an object, which takes on successive values from the range [*wp, wk*] for each iteration of the loop [1].

FOR *l* IN *wp* TO *wk* LOOP (1)

sequence of statements (2)

END LOOP; (3)

The *while* iteration scheme allows a test condition to be evaluated before each iteration. The iteration only proceeds if the test (*l*<= *wk)* evaluates to *TRUE*.

loop label: WHILE *condition* LOOP (1)

sequence of statements (2)

END LOOP *loop label*; (3)

Petri net models of these two loop statements are shown at Fig. 5 and 6 respectively.

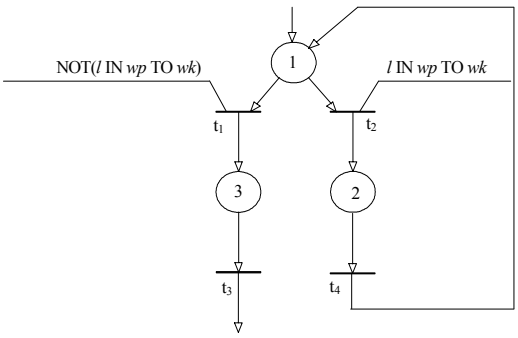


Fig.5. Petri net model of the FOR…LOOP statement.

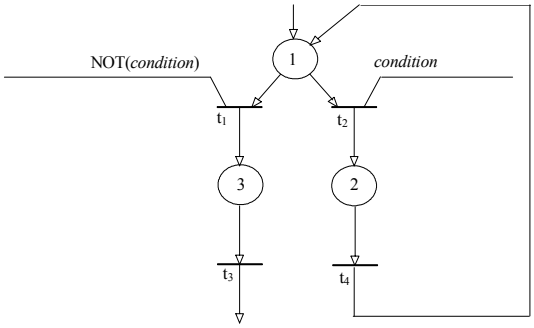


Fig.6. Petri net model of the WHILE…LOOP statement.

There are two additional statements, which can be used inside a loop to modify the basic pattern of iteration. The *next* statement terminates execution of the current iteration and starts the subsequent iteration. The *exit* statement terminates execution of the current iteration and terminates the loop.

loop label: WHILE *condition_1* LOOP (1)

sequence of statements; (2)

NEXT *loop label* WHEN *condition_2* (3)

sequence of statements; (4)

END LOOP *loop statement*; (5)

The Petri net model of this statement is shown at Fig.7.

The loop statement with exit is presented below and its Petri net model is shown at Fig. 8.

LOOP (1)
sequence of statements; (2)

EXIT WHEN *condition*; (3)

sequence of statements; (4)

END LOOP; (5)

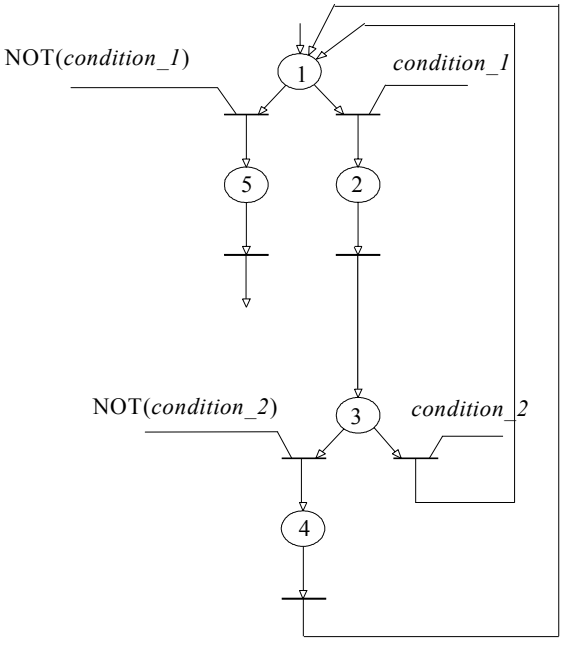


Fig.7. Petri net model of the WHILE…LOOP with NEXT statement.

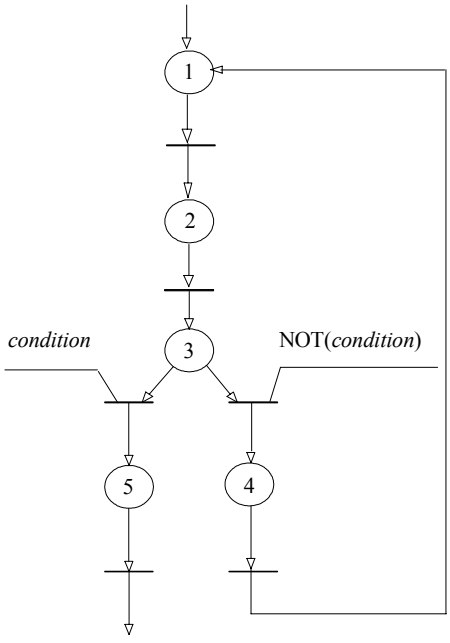


Fig.8. Petri net model of the LOOP with EXIT statement.

6. SUMMARY

In this paper Petri net models of VHDL control statements are presented. These models are used to generate automatically of control flow model. The control flow dictates the partial ordering of data flow in VHDL model and represents the conditions under which the processes are activated. Petri net model of VHDL control flow is very useful in the process of generating tests for design verification.

REFERENCES

[1] Armstrong J.R., *Chip-level modeling with VHDL*. Prentice Hall, Englewood Cliffs, New Jersey,1989

[2] Ashenden P.J., *The VHDL Cookbook*, Dept. Computer Science University of Adelaide, South Australia 1990

[3] E. Idzikowska, Generation Of Validation Tests From Behavioural Description In Vhdl *Proc. of Sixth Annual Advanced Technology Workshop ATW98, Ajaccio, France 1998*

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

CHDL - AN APPROACH FOR HARDWARE DESIGN AT THE SYSTEM LEVEL

Miroslaw FORCZEK

Aldec-ADT, Compilers Division,
ul. Lutycka 6, 44-100 Gliwice, POLAND, *mirekf@aldec.katowice.pl*

Abstract. *Currently, designers turn to C/C++ instead of using HDL languages at the initial stage of their projects. Manual translation from C/C++ into an HDL is extremely time-intensive. Even with latest approaches such as C++ library of HDL classes, a designer still has invested a lot of time on re-writing the project code at the RTL level. Aldec's CHDL approach, a C subset, addresses precisely the problem of C to HDL conversion automation. The possibilities of an algorithm parallelism reconstruction from its software version on example of the DCT routine were shown.*

Key Words. *System-Level Hardware Design, Behavioral Synthesis, HDL Languages, C/C++ Language*

1. INTRODUCTION

As devices become more complex, their design processes take more time and become more expensive than before. One of the most important improvements, which was introduced over the past years, is an RTL synthesis tools, which automates the design transformation from RTL into gate-level process [10]. Since its introduction, most of a designer's efforts stop at the RTL stage of the design specification. Automated tools perform the rest of work (fig. 1). The synthesis tools have been improved since their first appearance, and it now makes no sense in terms of an economical aspect to try to make a better design manually than a synthesized one. A few decades ago the algorithm's distinction into either software or hardware was introduced. The possibilities of today's highly integrated chips cause, such a distinction is not so obvious now [8,11]. The algorithms also become more complex and change frequently. Making updates in the present hardware implementations is a costly and time-intensive process. The algorithms are now prototyped and verified in a software implementation version. Often they exist in software form for a longer time, acquiring stability before hardware implementation is required. This is a way hardware designers turn to classical programming languages such as C or C++ at the first stage of the project instead of using HDL languages. This causes new problems. One of them is making the transition from C/C++ implementation into HDL implementation. These two implementations are totally different:

- C/C++ implementation is a sequential process (in most cases) while HDL implementation is a parallel, multi-process design,
- - C/C++ implementation runs without any clock signals while HDL implementation must take into consideration system clock signal and must address the signals timing issues.

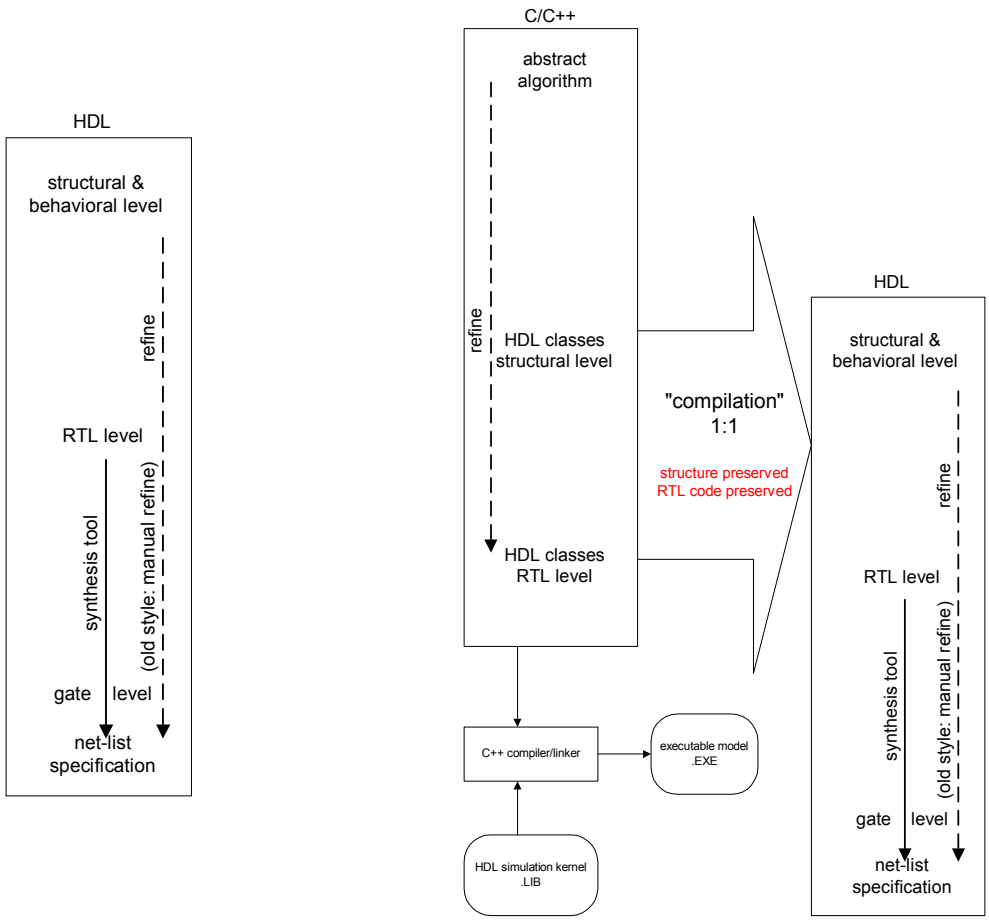


Fig. 1. Hardware design paths: with use of an HDL language (left) and with use of HDL classes in C++ (right)

The designer has to wait until the entire project is converted into HDL before he or she can validate the design again. This causes the conversion bugs to accumulate, making running the entire project much more difficult than in the case of having regular regression tests. A different approach has been proposed that address these problems. Currently, the most popular design solution is C++ library of HDL classes [2,3,9]. HDL classes enables a normal C/C++ environment with HDL constructs like modules, processes and signals. The designer performs a full conversion process in the same environment. As a result, the designer is able to make regular regression test at any stage of the conversion. Unfortunately, HDL classes library is not a solution that fully automates the hardware design path. The designer still has spent a lot of time on re-writing (refining) the project code from software form into HDL form (fig. 1).

Aldec's CHDL approach addresses exactly the problem of C to HDL conversion automation. Instead of enabling C/C++ environment with HDL features and pushing the user to go throught the refine steps until reaching RTL model, CHDL enables users to synthesize the HDL code directly from the C algorithm in its natural form.

2. CHDL NOTATION

CHDL is a subset of the C language [6]. The C constructs that made it impossible to perform full static data and analyze control flow were removed. The table 1 summarizes the C constructs included in or rejected from CHDL notation.

Table 1. Summary of C constructs included and rejected in CHDL notation

Constructs class	Included C constructs	Rejected C constructs	Comments
Built-in types	char, int, float, double	void, pointer types	
Complex types	structure, union		translated by fields expansion into HDL
Operators	most of C operators	* & (indirection and address of)	
Statements	if, switch, while, do while, for, function call		non-recursive functions are allowed only
Local scopes and visibility rules		cross-references over function boundary	

The only limitation for a C programmer when using CHDL is the reduced spectrum of available language features. There is no requirement to re-write the C algorithm in terms of modules, processes or registers like it is in case of HDL classes. The designer only needs to eliminate the forbidden constructs, but the algorithm structure remains untouched at the same level of abstraction. As a result, CHDL will offer true system level design capabilities (un-timed design).

While manual C code refine into HDL classes the designer explicitly specifies the algorithm's inherent parallelism (by decomposing the algorithm into processes). Also the hardware architecture and available resources are explicitly denoted. Consequently, the compilation from HDL classes model to HDL is very simple process, preserving all semantics of constructs used in design specification.

All of this additional information (regarding an algorithm's parallelism and its preferred implementation in hardware) is not present in the CHDL description of an algorithm. Instead, it utilizes a CHDL compiler task to take all required decisions while compiling the algorithm into HDL (fig. 2).

3. BEHAVIORAL SYNTHESIS FROM CHDL DESCRIPTION

Having a C algorithm written with the use of behavioral constructs from a small subset (CHDL notation) is very simple to make its transformation into HDL version. Each CHDL construct has a directly corresponding construct in HDL language. Of course, such a naive conversion would result in a single-process design. After synthesizing it into gate level, the designer will create a working design, but with very poor throughput to area size rate. The tool for real use must perform CHDL to HDL conversion in an intelligent way, which means that the behavioral synthesis approach must be implemented. The tool will take on the majority of decisions on its own. The user should only specify guidelines for preferred implementation architecture. The synthesized implementation consists of a two kinds of logical blocks, which are:

- the processing units,
- the control circuits.

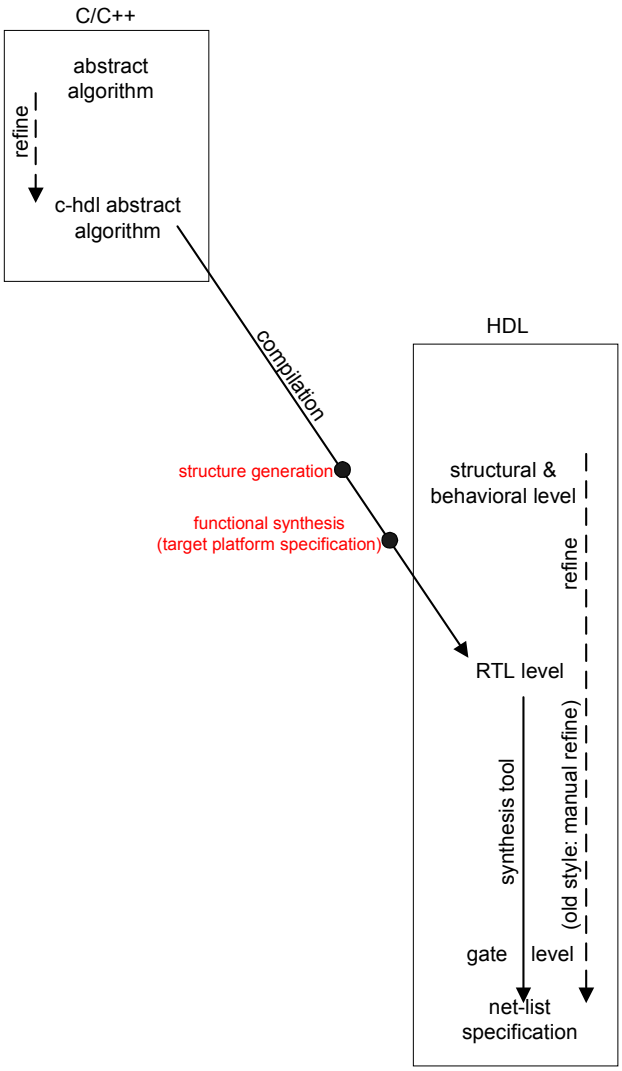


Fig. 2. The hardware design path with use of CHDL notation

The size of the control overhead depends on the proportion between the number of the algorithm's elementary sub-tasks units and the number of allocated resources for them. In the event there are less resources then sub-tasks to be processed, the control circuits must provide sharing the processing units in time, which translates into additional cycles for input data fetching, retrieving results from outputs and storage of intermediate results. As in any case, there is a trade-off between implementation size and its efficiency (power dissipation, throughput) [12]. In a normal design path, the designer has to make such decisions early in the design cycle, and the initial decision will make a large impact on the final result. With an automated path, designers can explore more than one architecture with relatively low costs or risks.

As an example, refer to a 2-D Discrete Consine Transform algorithm [4,5,7]. Assuming that the software implementation is already in place, the formula would appear as illustrated below [1]:

```
void fct2d(double f[], int nrows, int ncols) {
  int u,v;
  // ...
  for (u=0; u<=nrows-1; u++) {
    for (v=0; v<=ncols-1; v++) {
       g[v] = f[u*ncols+v];
```

```
    }
    fct(g,ncols);
  }
  for (v=0; v<=ncols-1; v++) {
    for (u=0; u<=nrows-1; u++) {
       g[u] = f[u*ncols+v];
    }
    fct(g,nrows);
    for (u=0; u<=nrows-1; u++) {
      f[u*ncols+v] = g[u]*two_over_sqrtncolsnrows;
    }
  }
}
```

This algorithm works as follows:

- 1-D DCT (**fct()** function) is performed for each row of the matrix **f**,
- the 1-D DCT is performed for each column from the result matrix after rows processing,
- the whole result matrix is scaled with a constant coefficient.

From the data and control flow analysis, it is possible to find and extract elementary sub-tasks that are independent of each other and can be processed in parallel. In this example, there are few groups of elementary tasks (fig. 3):

- **fct()** on each row of **f**,
- **fct()** on each column of result from previous processing,
- scaling each element of result from previous processing.

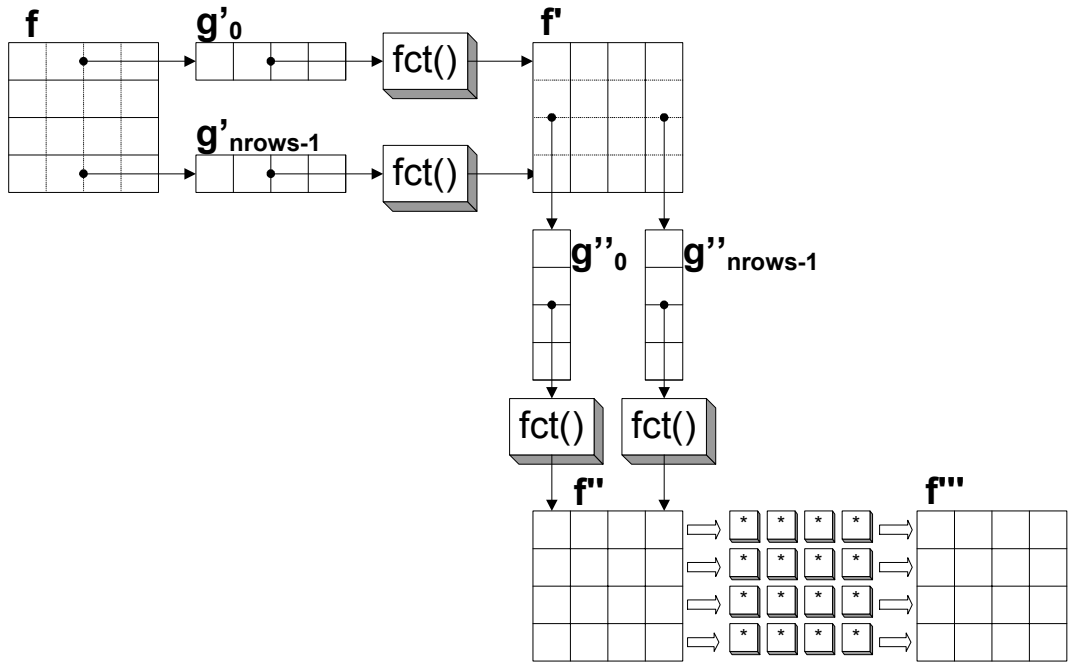


Fig. 3. The data and control flow diagram extracted from source code analysis

The designer now needs to decide what the synthesis mode to use for hardware implementation of this algorithm will be. The **blank array** mode can be used if the **f** matrix sizes are fixed. In this case, the fastest implementation as well as the larger one will allocate

its own processing unit. The **fixed resources** mode may be preferred, especially when the **f** matrix size varies in run-time. In this case, one will get an implementation that contains the limited number of processing units and the control block. It could be that the resulting size of the implementation will still not satisfy the requirements for the first time. If this is the case, the user has to try other tradeoffs by specifying various synthesis constraints.

4. CONCLUSION

The presented example clearly shows how the behavioral synthesis can be performed from the system level C algorithm. There is no need to manually re-write the algorithm in HDL manner to precise parallelism of the algorithm. The compromise made in the aforementioned approach is to reduce the flexibility of a source language (C in this case) in favor of a predictable construct for algorithm notation that allows its static analysis. From the very coarse version of the synthesis tool, new compilation techniques are applied incrementally for improvement results. There are still several problems to be researched and solved as a practical implementation in this synthesis tool. The most important issues are as follows:

- processing units synthesis or re-use of library units,
- automatic processing unit functionality selection based on the elementary sub-tasks code analysis.

REFERENCES

[1] *A Fast Discrete Cosine Transform*, Signal and Image Processing Group, The University of Bath, 1998

[2] "An Introduction to System-Level Modeling in SystemC 2.0", January 2001, *http://www.systemc.org*

[3] "Functional Specification for SystemC 2.0", January 2001, Synopsys Inc., CoWare Inc., Frontier Design Inc., 2000, *http://www.systemc.org*

[4] *Intel Image Processing Library. Reference Manual*, Intel Corporation, USA, 1999

[5] *Intel Signal Processing Library. Reference Manual*, Intel Corporation, USA, 1999

[6] *International Standard ISO/IEC 9899: 1999(E), Programming Languages – C,* ISO/IEC, 1999

[7] *Image Processing Toolbox. Users Guide. Version 2*, The MathWorks Inc., Natick, MA, 1999

[8] "QuickSilver: Technology Backgrounder", QuickSilver Technology, 2000, *http://www.quicksilvertech.com*

[9] "SystemC. Version 1.1 User's Guide", Synopsys Inc., CoWare Inc., Frontier Design Inc., 2000, *http://www.systemc.org*

[10] M.D.Ciletti, *Modeling, Synthesis and Rapid Prototyping with the Verilog HDL*, Prentice Hall, New Jersey, 1999

[11] P.Master, K.Lane "Powering up 3G Handsets for MPEG-4 Video", Communication Systems Design, January 2001, *http://www.cdsmag.com/main/2001/01/0101feat2.htm*

[12] J.Phil, *Tradeoffs Between Parallel and Serial Architectures in High Performance Digital Signal Processing*, Norwegian University of Science and Technology, Faculty of Electrical and Computer Engineering, Trondheim, Norway, 1997

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

SYMBOLIC STATE EXPLORATION OF CONTROLLERS SPECIFIED BY MEANS OF STATECHARTS

Grzegorz ŁABIAK

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND, *G.Labiak@iie.pz.zgora.pl*

Abstract. *The FSM and Petri nets theories have elaborated many techniques and algorithms, which enable the employment of formal method in the fields of synthesis, testing and the verification. Many of them are based on symbolic state exploration. This paper focuses on the algorithm of the symbolic state exploration of controllers specified by means of statecharts. Statecharts are new technique for specifying behaviour of controllers, which, in comparison with FSM and Petri nets is enriched with notions of hierarchy, history and exception transitions. The paper presents the mathematical model of the diagram, its physical interpretation as a digital circuit and the characteristic function, which is the key notion in state exploration.*

Key Words. *Statechart, Logic Control, Symbolic Analysis, BDD*

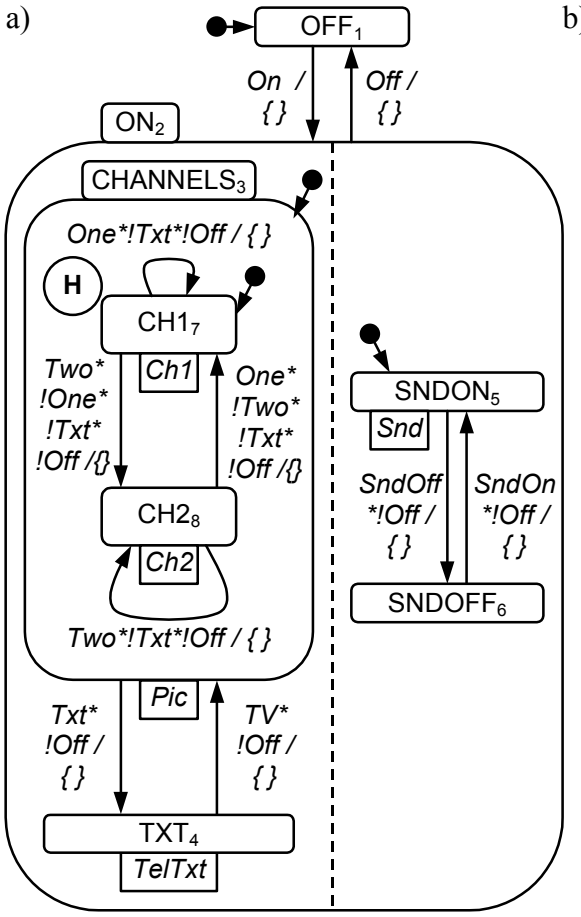
1. INTRODUCTION

Statecharts are a visual formalism for the specification of reactive systems, which is based on the idea of enriching state-transition diagrams with notions of hierarchy, concurrency and broadcast communication [6,7,8,10]. It was invented as a visual formalism for complex systems by David Harel [7]. Today, as a part of UML technology, it is widely used in many fields of modern engineering [11]. The presented approach features such characteristics as Moore's and Mealy's automata, history and terminal states. There are many algorithms based on a State Transition Graph traversal for finite state machines, which have applications in the area of synthesis, test and verification [2,3,4,5,10]. It seems to be very promising to use well developed techniques from FSM and Petri net theory in the field of synthesis [1], testing and the verification of controllers specified by means of statechart diagrams. These considerations caused the elaboration of the new algorithms of symbolic state space exploration.

2. SYNTAX AND DEFINITIONS

Based on the formalism contained in [8], the following definition of syntax can be given. Let S be the infinite set of states, T the infinite set of transition, E the infinite set of events. Symbols $s, s', s_1, s_2, s_3, \ldots$ are used to range over S, $t, t', t_1, t_2, t_3, \ldots$ to range over T, $e, e', e_1, e_2, e_3, \ldots$ to range over E.

a)



b)

$$[M_0\rangle = \{10000000,\ 10000010,\ 10000001,\ 01101010,\ 01100110,\ 01101001,\ 01100101,\ 01011010,\ 01010110,\ 01011001,\ 01010101\}$$

c)

$$[C_0\rangle = \{10000000,\ 01101010,\ 01100110,\ 01101001,\ 01100101,\ 01011000,\ 01010100\}$$

d)

$$\begin{aligned} X_{[C_0\rangle} = {} & s_1\bar{s}_2\bar{s}_3\bar{s}_4\bar{s}_5\bar{s}_6\bar{s}_7\bar{s}_8 \\ & + \bar{s}_1 s_2 s_3 \bar{s}_4 s_5 \bar{s}_6 s_7 \bar{s}_8 \\ & + \bar{s}_1 s_2 s_3 \bar{s}_4 \bar{s}_5 s_6 s_7 \bar{s}_8 \\ & + \bar{s}_1 s_2 s_3 \bar{s}_4 s_5 \bar{s}_6 \bar{s}_7 s_8 \\ & + \bar{s}_1 s_2 s_3 \bar{s}_4 \bar{s}_5 s_6 \bar{s}_7 s_8 \\ & + \bar{s}_1 s_2 \bar{s}_3 s_4 s_5 \bar{s}_6 \bar{s}_7 \bar{s}_8 \\ & + \bar{s}_1 s_2 \bar{s}_3 s_4 \bar{s}_5 s_6 \bar{s}_7 \bar{s}_8 \end{aligned}$$

Fig. 1. TV remote controller: a) statechart diagram, b) set of all global states, c) set of all reachable configurations, d) characteristic function $X_{[C_0\rangle}$

Definition 1 *Statechart*

A *Statechart Z* is a tuple consisting of the following elements:

$$(S_z, hrc_z, type_z, default_z, history_z, E_z, T_z, out_z, in_z, tlabel_z, saction_z)$$

where:

1. $S_z \subseteq \boldsymbol{S}$ is the finite non-empty set of states.
2. $hrc_z : S_z \to 2^{S_z}$ is the *hierarchy function*, which for every state $s \in S_z$ assigns the set of immediate sub-states of *s*.
3. $type_z : S_z \to \{AND, OR\}$ is the *state-type function*.
4. $default_z : S_z \to S_z$ is the *default function*.
5. $history_z : S_z \to \{true, false\}$ is the Boolean *history function*.
6. $E_z \subseteq \boldsymbol{E}$ is the finite set of *events*.
7. $T_z \subseteq \boldsymbol{T}$ is the finite set of *transition*.
8. $out_z : T_z \to S_z \setminus \{root_z\}$ is a total function, called *source function*, such that $out_z(t) = s$ if transition *t* originates from state *s*.
9. $in_z : T_z \to S_z \setminus \{root_z\}$ is a total function, called *target function*, such that $in_z(t) = s$ if transition *t* ends in *s* state.
10. For every transition $t \in T_z$, the following predicate holds:
    $parent(out_z(t)) = parent_z(in_z(t)) = s$ with $type_z(s) = OR$.

11. $tlabel_z : T_z \to 2^{E_z} \times 2^{E_z}$ is the *transition labelling function*. The first component of $tlabel_z$ is called $trigger_z(t)$, the second is called transition action and is denoted $taction_z(t)$.
12. $saction_z : S_z \to 2^{E_z}$ is the *state labelling* function, which gives the set of events associated to state *s*.

To use statecharts as a model for the specification of the digital controller it is necessary to give a real world interpretation of such notions as event, set of events or label. The following definition introduces the interpreted statecharts model. Based on this definition it is possible to use the statechart diagram as a mean of the specification of the digital controller or reactive systems.

Definition 2 *Interpreted Statechart*
An I*nterpreted Statechart* is a statechart as in Definition 1 where:

1. $X \subseteq E_z$ is a set of events coming from the environment, $Y \subseteq E_z$ is a set of events visible to the outside world
2. An event is a named signal that is either *present* or *absent*. *I* is a set of all signals in the system, both input, output and internal ones.
3. $input : X \to I_X$ where $I_X \subseteq I$ – is a function assigning the event coming from the environment to a signal. $output : Y \to I_Y$ where $I_Y \subseteq I$ – is a function assigning the event visible to the environment to a signal and $I_X \cap I_Y = \varnothing$. Signals related to events coming from the outside world and visible to the outside world are, respectively, the input and the output of the system. The sets of input and output events are disjoint.
4. Component $trigger_z(t)$ of the transition labelling function $tlabel_z$ called guard is a Boolean expression generated by the following grammar:
   `g ::= true | false | i | !g | g + g | g * g | (g)`
   where $i \in I$ is a signal associated to event $e \in E_z$. The evaluation of an event is either **true** or **false** when the event is either present or absent. The operators !, + and * correspond to the Boolean operators **not**, **or** and **and**, respectively.
5. Functions $taction_z(t)$ and $saction_z(s)$ lists a set of events $a \subseteq E_z$ associated with transitions and states respectively, according to the following rule:
   ```
   a ::= nil | b
   b ::= i | b, b
   ```
   where $i \in I$ is a signal associated to an event and "," distinguishes two events in an action.

It is essential from a symbolic technique point of view to express the concept of the set of states. The notion of characteristic function, well known in algebra theory, can be applied [2].

Definition 3 C*haracteristic function*
A characteristic function X_A of a set of elements $A \subseteq U$ is a Boolean function $X_A : U \to \{0,1\}$ defined as follows:

$$X_A(x) = \begin{cases} 1 \Leftrightarrow x \in A, \\ 0 \ otherwise. \end{cases} \quad (1)$$

The characteristic function is calculated as a disjunction of all elements of *A*. Operations on sets are in direct correspondence with operations on their characteristic functions. Thus:

$$X_{(A \cup B)} = X_A + X_B;\ X_{(A \cap B)} = X_A * X_B;\ X_{(\overline{A})} = \overline{X_A} \quad (2)$$

The characteristic function allows sets to be represented by BDDs. Fig. 1d presents the characteristic function of all possible configurations [2].

3. MODELLING SYNCHRONOUS INTERPRETED STATECHART BY BOOLEAN EQUATIONS

The modelling of statecharts is based on the assumption that for every state $s_i \in S_z$ one flip-flop is assigned, and then for every such flip-flop excitation function, as a Boolean expression, is produced, Fig. 2. The excitation function δ evaluates to 1 when the flip-flop associated with s_i will be active in the next iteration or remembers past activity, otherwise it equals 0. A state is said to be active when every state belonging to the path, carried from it to the root state, is active. Global state G of the system, called marking, is represented by the set of all states of flip-flops. A configuration C is a set of all active states. The excitation function $\delta_i(S_z, I)$ is defined on signals and current states of flip-flops. A detailed description of the creation of the functions is beyond the scope of this paper and the method developed by the author will be published soon.

Let Z be a synchronous interpreted statechart and Ω the set of all possible markings of Z. Each marking of Z can be coded as a vector $M_{1\times n} = (\mu_1, \mu_2, \ldots, \mu_n)$ where $\mu_i \in \{0, 1\}$ represents the activity of flip-flop representing a state $s_i \in S_z$ and n is a number of all states in S_z. The set of all reachable markings from default marking M_0 is denoted $[M_0\rangle$. Firing of a transition t_k transforms a marking M_i into marking M_j. This fact is denoted by $M_i[t_k\rangle M_j$. It is possible to fire a set of enabled transitions in a given moment of discrete time. Any set of markings can be represented using its characteristic function.

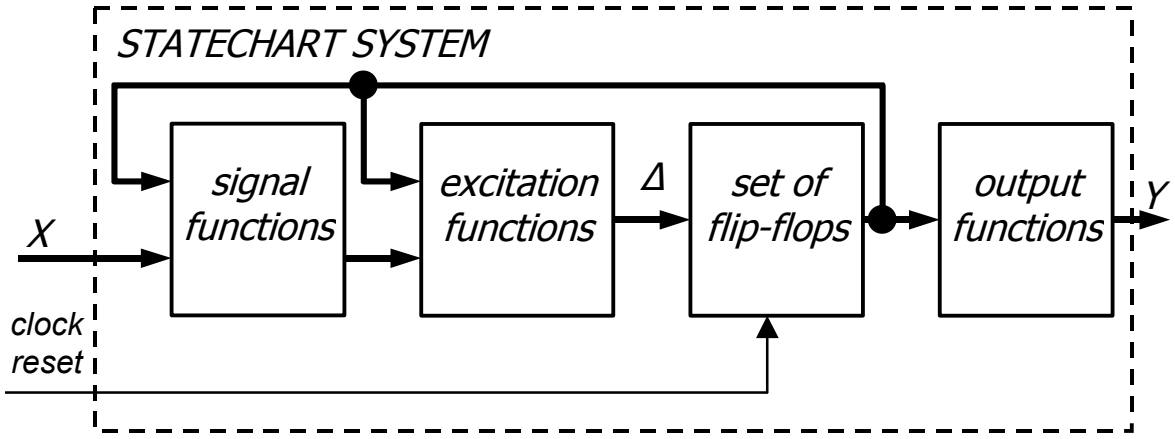


Fig. 2. Statechart system model

By the association of the excitation function with a state, a direct application of FSM and Petri Nets traversal algorithm can be used. The transition function in Fig. 2 $\Delta : \Omega \to \Omega$, is defined as a functional vector of a Boolean function: $\Delta : [\delta_1(S_z, I), \delta_2(S_z, I), \ldots \delta_n(S_z, I)]$, where $\delta_i(S_z, I)$ is an excitation function of the state s_i flip-flop and I is the set of signals in the system represented by their functions. In Fig. 1d symbol s_i denotes both a state in the diagram and a variable of characteristic function. Boolean expressions related to transition functions can be implemented by using topological information from the diagram.

4. SYMBOLIC STATES SPACE EXPLORATION OF STATECHARTS

Symbolic state space exploration techniques are widely used in the area of synthesis, testing, and the verification of finite state systems. Coudert *et al* were the first to realise that Binary Decision Diagram (BDDs) could be used to represent sets of states [4]. This led to the

formulation of an algorithm that traversed the State Transition Graph in breadth-first manner, moving from a set of a set of states to the set of its fan-out states. In this approach a set of states is represented by means of characteristic functions. The key operation required for traversal is the computation of the range of a function, given a subset of its domain [2]. The computational cost of these symbolic techniques depends on the cost of the operation performed on the BDDs and does not depend on the number of states and transitions. For example, from Fig. 1a BDD characteristic function for the set of all global states (Fig. 1b) consists of 21 nodes, and characteristic function for the set of all configurations (Fig. 1c) counts 20 nodes. The symbolic state exploration of statecharts relies on:

- association transition functions to states,
- association logic functions to signals,
- representation of Boolean function as BDDs,
- representation of sets of states using their characteristic functions,
- computation of a set of next states as an image of the state transition function on the current state set for all input signals.

Starting from the default configuration and the set of signals, symbolic state exploration methods enable the computation of the entire set of next states in one formal step. Burch *et al* and Coudert *et al* were the first to independently propose the approach to the image computation [4,5]. Two main methods are transition relation and transition function. The latter is the method implemented by the author. The symbolic state space algorithm of statechart *Z* is as follows:

```
symbolic_traversal_of_Statechart(Z, initial_marking) {
   X[M0⟩ = current_marking = initial_marking;
   while (current_marking != ∅) {
      next_marking = image_computation(Z, current_marking);
      current_marking = next_marking * ¬X[M0⟩;
      X[M0⟩ = current_marking + X[M0⟩;
   }
}
```

Fig. 3. The symbolic traversal of Statecharts

The variables in italics represent characteristic functions of corresponding sets of configurations. All logical variables are represented by BDDs. Several subsequent configurations are simultaneously calculated using the characteristic function of current configurations and transition functions. This computation is realised by the *image_computation* function. The set of subsequent configurations is calculated from the following equations:

$$next\_marking = \exists_s \exists_x \left(current\_marking * \prod_{i=1}^{n} \left[s_i' \odot \left(current\_marking * \delta_i(s, x) \right) \right] \right) \quad (3)$$

$$next\_marking = next\_marking \langle s' \leftarrow s \rangle \quad (4)$$

where s, s', x denote the present state, next state and input signal respectively; $\exists_s$ and $\exists_x$ represent the existential quantification of the present state and signal variables; symbols $\odot$ and * represent logic operators XNOR and AND respectively; equation (4) means swapping variables in expression.

Given the characteristic function of all reachable global states of a system, it is possible to calculate the set of all configurations. As mentioned earlier in section 3, a state is said to be active when every state belonging to the path, carried from it to the root state, is active. This

leads to the formulation of a state activating function. Let α_i be a Boolean function $\alpha_i : S_z \to \{0,1\}$ which evaluates to 1 when state s_i is active. The generation of a set of all configurations relies on the image computation of a characteristic function in transformation by activating functions:

$$X_{[C_0\rangle} = \exists_s \exists_x \left(X_{[M_0\rangle} * \prod_{i=1}^{n} \left[s_i' \odot \left(X_{[M_0\rangle} * \alpha_i(s,x) \right) \right] \right) \tag{5}$$

$$X_{[C_0\rangle} = X_{[C_0\rangle} \langle s' \leftarrow s \rangle \tag{6}$$

The example in Fig. 1a describes the behaviour of a remote controller. The remote can be in 11 global states (Fig. 1b) which correspond to the set of 7 possible configurations (Fig. 1c).

5. CONCLUSION

A visual formalism proposed by David Harel can be effectively used to specify the behaviour of digital controllers. Controllers specified in this way can subsequently be synthesised in FPGA circuits. In this paper it has been shown that state space traversal techniques from FSM and Petri nets theory can be efficiently used in the fields of statechart controllers design. The presented issues are a matter of the author's investigations. Within the framework of the research, a software system called *HiCoS* has been developed, where presented algorithms have been successfully implemented.

REFERENCES

[1] Adamski M., "SFC, Petri Nets and Application Specific Logic Controllers, *Proc. of The IEEE Int. Conf. on Systems, Man and Cybernetics* San Diego, USA Nov. '98, ss 728-733

[2] K. Biliński, *Application of Petri Nets in parallel controllers design*, PhD. Thesis, University of Bristol, Bristol, 1996

[3] J. R. Burch, E. M. Clarke, K. L. McMillan, and D. Dill. "Sequential Circuit Verification Using Symbolic Model Checking", *Proceedings of the 27th Design Automation Conference,* pp. 46-51, June 1990

[4] O. Coudert, C. Berthet, and J. C. Madre, "Verification of Sequential Machines Using Boolean Functional Vectors", *IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pp. 111-128, November 1989

[5] A. Ghosh, S. Devadas, A. R. Newton, *Sequential logic testing and verification*, Kluwer Academic Publisher, Boston 1992

[6] D. Harel, Statecharts, A Visual Formalism for Complex Systems, *Science of Computer Programming*, No 8, North-Holland, 1987, pp. 231-274

[7] G. Łabiak, Implementacja sieci Statechart w reprogramowalnej strukturze FPGA. *Mat. I Krajowej Konf. Nauk. Reprogramowalne Układy Cyfrowe*, Szczecin, pp.169-177 '98

[8] A. Magiollo-Schettini, M. Merro, *Priorities in Statecharts, Diparamiento di Informatica*, Universita di Pisa, Corso Italia

[9] E. Pastor, O. Roig, J. Cortadella, and R. M. Badia, "Petri Net Analysis Using Boolean Manipulation", *Proc. of 15th Int. Conference: Application and Theory of Petri Nets*, Vol. 815 of *Lecture Notes in Computer Science* pp. 416-435, Springer Verlag June 1994

[10] M. Rausch B. H. Krogh, *"Symbolic Verification of Stateflow Logic"*, *Proceedings of the 4th Workshop on Discrete Event System*, Cagliari, Italy, pp. 489-494 1998

[11] UML 1.3 Documentation, Rational Software Corp. '99, *http:// www.rational.com/uml*

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

XML APPLICATION FOR MODELLING AND SIMULATION OF CONCURRENT CONTROLLERS

Agnieszka WĘGRZYN, Piotr BUBACZ

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul.Podgórna 50, 65-246 Zielona Góra, POLAND, <*A.Wegrzyn, P.Bubacz*>*@iie.pz.zgora.pl*

Abstract. *The Extensible Markup Language (XML) is a subset of SGML that is completely described in this document. Such language can be used for modelling Petri nets. XML is used, because it can be exchanged between different systems. On the other hand, XML format is platform-independent, well supported, and license-free. XML is a set of rules, guidelines, and conventions, whatever you want to call them, for designing text formats for such data, in a way that produces files that are easy to generate and read. In this paper, a possibility of XML modelling and simulation of Petri net is presented.*

Key Words. *Concurrent Controllers, Petri Nets, XML, Modelling, Simulation*

1. INTRODUCTION

Petri net is a graphical and mathematical modelling tool. It can describe processing systems, which are characterized as being concurrent, asynchronous, distributed, nondeterministic [6]. In the presented method, Petri net is used for modelling of digital circuits, especially concurrent controllers.

In the paper a new way of describing of Petri net is presented. The proposed method is based on the Extensible Markup Language (XML).

XML describes a class of data objects called XML documents and partially describes the behavior of computer programs which process them. XML is an application profile or restricted form of SGML, the Standard Generalized Markup Language [ISO 8879]. By construction, XML documents are conforming SGML documents [8].

The next step (after modelling) of design of digital circuit is analysis. There are different methods of verification of modelled system. One of the simplest methods is simulation. It is similar to the debugging of program execution. During simulation it is possible to check whether a circuit modelled by a Petri net behaves correctly. Therefore it is possible to recognise and remove errors in the controllers, even at early stage of design.

Several specification and design techniques based on Petri nets have been proposed [1,4,7]. They are based on software tools that help designers to develop and simulate (animate) the

system at a conceptual and abstract level. They are usually very formal and oriented towards the verification (simulation) aspects of design.

2. BACKGROUND

In this chapter basic information about Petri net and XML application is presented.

2.1. Petri nets

Petri nets are mathematical object that exists independently of any physical representation. The nets can effectively describe parallelism. The graphical form of a Petri net is used as a tool for the modelling and analysis of digital circuits, especially concurrent controllers.

A Petri net is a well-known mathematical formalism. There are different classes of Petri nets [6]. However, for the modelling of digital systems only selected classes are applicable. In this paper, coloured interpreted Petri nets are considered. Firstly, only a basic information and definitions are presented.

Petri net is an oriented bipartite graph with two subsets of nodes called the places and the transitions and the arcs joining places to transitions or transition to places. Petri net *PN* is a 4-tuple [2,3]:

$PN = (P, T; F, M_0)$,

where P – a finite set of places;
T – a finite set of transitions;
F – flow function (i.e. a finite set of arcs);
M_0 – initial marking.

On Fig. 1 an example of interpreted Petri net is presented.

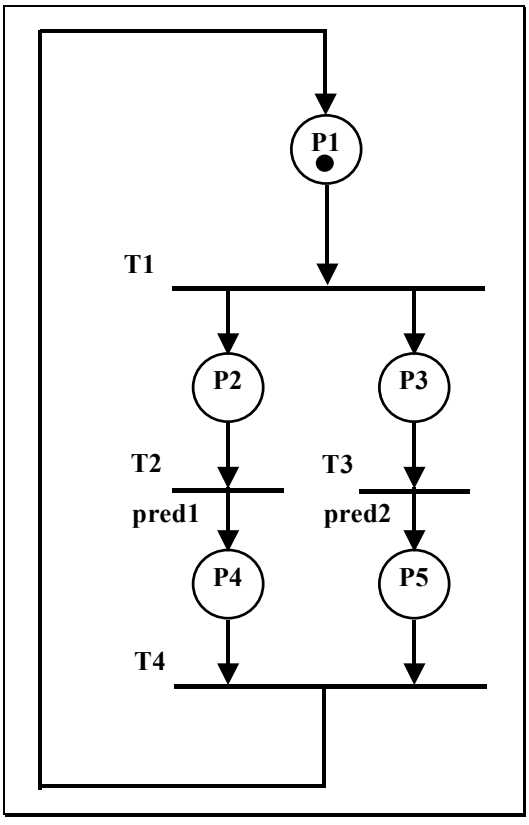


Fig. 1. An example of Petri net

2.2. XML application

The Extensible Markup Language is a set of rules for defining semantic tags that break a document into parts and identify the different parts of the document. It is a meta-markup language that defines syntax used to define other domain-specific, semantic, structured markup languages [8].

XML is a meta-markup language for designing domain-specific markup languages. Each XML-based markup language is called an XML application. This is not an application that uses XML like the Mozilla Web browser, the Gnumeric spreadsheet, or the XML Pro editor, but rather an application of XML to a specific domain such as Chemical Markup Language (CML) for chemistry or GedML for genealogy.

Each XML application has its own syntax and vocabulary. This syntax and vocabulary adheres to the fundamental rules of XML. This is much like human languages, which each have their own vocabulary and grammar, while at the same time adhering to certain fundamental rules imposed by human anatomy and the structure of the brain.

XML is an extremely flexible format for text-based data. The reason XML was chosen as the foundation for the wildly different applications discussed in this chapter is that XML provides a sensible, well-documented format that's easy to read and write. By using this format for its data, a program can offload a great quantity of detailed processing to a few standard free tools and libraries. Furthermore, it's easy for such a program to layer additional levels of syntax and semantics on top of the basic structure XML provides.

2.3. Petri Net Specification Format 3

Petri net can be present in textual form as a set of rules. Petri Net Specification Format 3 (PNSF3) is one of such textual formats. The format is based on Extensible Markup Language. PNSF3 is developed at Technical University of Zielona Gora and it based on PNSF2. By using PNSF3 an interpreted, hierarchical and coloured Petri net can be modelled.

The PNSF3 format specifies the structure of Petri net. PNSF3 does not keep information about placement of places, transitions, arcs, etc. It keeps information about names of places, transitions, number of markers, connection between places and transitions. That's why this format is simple to create and modify. PNSF3 format is easy to parse and converse to various representations. Such a way of describing creates new possibilities for example of simulation of Petri net using scalable vector graphics.

Most Petri-net research groups have their own software packages and tools to assist the drawing, analysis and simulation of various applications. They have their own Petri net format, too [5]. Now, XML based format helps to exchange data between various tools. Such approach gives possibility to verification new methods of analysis using well know and tested systems. The old formats of describing Petri net are not quite enough to apply for such verification. The goals for new XML based format are [8]:

- flexible (extendable),
- complex description,
- platform independent,
- human readable,
- easy to parse and transform.

Presented format (Fig. 2) is different from XML based format using in another Petri net tools. It differs in stored information about Petri net. In other well know systems for modelling and

analysis, XML based format keeps information about placement of elements of Petri net [5]. Preparing such format without graphical editor is very difficult. Because system in which PNSF3 is used, have no graphical editor; there cannot be store information about placement.

```
<?xml version="1.0"
encoding="ISO-8859-2" standalone="yes"?>

<pnsf3>

<clock>CLOCK</clock>

<input id="x1"> </input>
<output id="S1"> </output>
<output id="R1"> </output>

<place id="P1">
      <initmark id="m1">
            <noofmark>1</noofmark>
      </initmark>
</place>
<place id="P2" />
<place id="P3" />
<place id="P4" />
<place id="P5" />

<predicate id="pred1">x1</predicate>
<predicate id="pred2">!x1</predicate>

<transition id="T1" />

<transition id="T2">
      <condition id="c1">pred1
      </condition>
</transition>

<transition id="T3">
      <condition id="c2">pred2
      </condition>
</transition>

<transition id="T4" />
```

```
<net>
      <arc>
            <inplace>P1</inplace>
            <outplace>P2</outplace>
            <outplace>P3</outplace>
            <trans>T1</trans>
      </arc>

      <arc>
            <inplace>P2</inplace>
            <outplace>P4</outplace>
            <inpred>pred1</inpred>
            <trans>T2</trans>
      </arc>

      <arc>
            <inplace>P3</inplace>
            <outplace>P5</outplace>
            <inpred>pred2</inpred>
            <trans>T3</trans>
      </arc>

      <arc>
            <inplace>P4</inplace>
            <inplace>P5</inplace>
            <outplace>P1</outplace>
            <trans>T4</trans>
      </arc>
</net>

<MooreOutputs>
      <placeMoore>P2</placeMoore>
      <postcon>S1</postcon>
</MooreOutputs>

<MooreOutputs>
      <placeMoore>P4</placeMoore>
      <postcon>R1</postcon>
</MooreOutputs>
</pnsf3>
```

Fig. 2. PNSF3 for Petri net (Fig. 1)

3. SCALABLE VECTOR GRAPHICS

Scalable Vector Graphics (SVG) was created by the World Wide Web Consortium (W3C), the non-profit, industry-wide, open-standards consortium that created HTML and XML, among other important standards and vocabularies. Over twenty organizations, including Sun Microsystems, Adobe, Apple, IBM, and Kodak, have been involved in defining SVG [9].

SVG is a language for describing two-dimensional graphics in XML. SVG allows for three types of graphic objects: vector graphic shapes (e.g., paths consisting of straight lines and curves), images and text. Graphical objects can be grouped, styled, transformed and composited into previously rendered objects. A text in any XML namespace can be suitable to the application, which enhances searchability and accessibility of the SVG graphics. The feature set includes nested transformations, clipping paths, alpha masks, filter effects, template objects and extensibility [9].

SVG drawings can be dynamic and interactive. The Document Object Model (DOM) for SVG, which includes the full XML DOM, allows for straightforward and efficient vector graphics animation via scripting. A rich set of event handlers such as onmouseover and onclick can be assigned to any SVG graphical object. Because of its compatibility and leveraging of other Web standards, features like scripting can be done on SVG elements and other XML elements from different namespaces simultaneously within the same Web page [9].

3.1. SVG Features

SVG has many advantages over other image formats, and particularly over JPEG and GIF, the most common graphic formats used on the Web today. Specifically [9]:

- Plain text format - SVG files can be read and modified by a range of tools, and are usually much smaller and more compressible than comparable JPEG or GIF images.
- Scalable - Unlike bitmapped GIF and JPEG formats, SVG is a vector format, which means SVG images can be printed with high quality at any resolution, without the "staircase" effects you see when printing bitmapped images.
- Zoomable - You can zoom in on any portion of an SVG image and not see any degradation.
- Searchable and selectable text - Unlike in bitmapped images, text in SVG text is selectable and searchable. For example, you can search for specific text strings, like city names in a map.
- Scripting and animation - SVG enables dynamic and interactive graphics far more sophisticated than bitmapped or even Flash images.
- Works with Java technology - SVG complements Java technologies' high end graphics engine, the Java 2D API.
- Open standard - SVG is an open recommendation developed by a cross-industry consortium. Unlike some other graphics formats, SVG is not proprietary.
    - True XML - As an XML grammar, SVG offers all the advantages of XML:
    - Interoperability;
    - Internationalization (Unicode support);
    - Wide tool support;
    - Easy manipulation through standard APIs, such as the Document Object Model (DOM) API;
    - Easy transformation through XML Stylesheet Language Transformation (XSLT).

4. FROM PNSF3 TO SVG

The main advantage of XML is that it is very easy to transform into another format (e.g. SVG). In this chapter a way of transformation from XML based format (PNSF3) into SVG is presented.

A large majority of generally available XML based format use XSL to transformation into another format. XSL is a stylesheet language for XML. XSL specifies the styling of an XML document by using XSLT, to describe how the document is transformed into another.

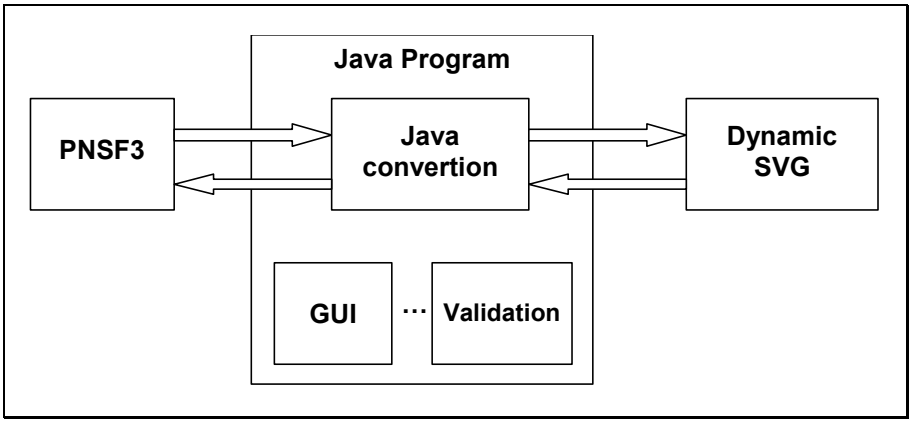


Fig. 3.Diagram of method of transformation

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20001102//EN"
      "http://www.w3.org/TR/2000/CR-SVG-20001102/DTD/svg-20001102.dtd">

<svg width="500" height="500">
<g>
      <ellipse cx="231.424" cy="76.3614" rx="20.5" ry="20.5"
            style="stroke-width:1;stroke-opacity:1;stroke:rgb(0,0,0);
            fill-opacity:0;fill:rgb(0,0,0);opacity:1"/>

            <text x="219.032" y="84.4564" style="font-family:Arial;font-size:24;
                  stroke-width:1;stroke-opacity:1;stroke:rgb(0,0,0);
                  fill-opacity:1;fill:rgb(0,0,0);opacity:1">P1</text>

            <path d="M232.125 23.0995 L232.125 56.0114"
                  style="stroke-miterlimit:4;stroke-linejoin:miter;
                  stroke-linecap:round;stroke-width:1;stroke-opacity:1;
                  troke:rgb(0,0,0);fill-opacity:1;fill:rgb(0,0,0);
                  opacity:1"/>

            <path d="M230.315 130.331 L225.457 122.902"
                   style="stroke-miterlimit:4;stroke-linejoin:miter;
                   stroke-linecap:round;stroke-width:1;stroke-opacity:1;
                   stroke:rgb(0,0,0);fill-opacity:1;fill:rgb(0,0,0);
                  opacity:1"/>

            <path d="M231.086 130.293 L235.944 122.864"
                  style="stroke-miterlimit:4;stroke-linejoin:miter;
                  stroke-linecap:round;stroke-width:1;stroke-opacity:1;
                  stroke:rgb(0,0,0);fill-opacity:1;fill:rgb(0,0,0);
                  opacity:1"/>

      . . .
</g>
      <ellipse cx="230.5" cy="87.5" rx="4.5" ry="4.5" style="stroke-width:0;
            stroke-opacity:1;stroke:rgb(255,255,255);
            fill-opacity:1;fill:rgb(255,255,255);;opacity:1">

            <animateColor attributeName="fill" attributeType="CSS"
                        values="black;white;white;white;white;white"
                        dur="6s" repeatCount="indefinite"/>
      </ellipse>
      . . .
```

Fig. 4. A part of SVG file

In section 2.3 it is mentioned, that PNSF3 do not keep information about placement. Because of this fact, it is difficult to directly make XSL file for transformation into SVG. For such

case, a special program should be created to prepare XSL file. SVG file can be made in the other way, too. On Fig. 3 diagram of method of transformation from PNSF3 into SVG file, without preparing a XSL file, is presented. For the conversion a Java application is used. The program generates a dynamic SVG file. The presented part of file (Fig. 4) is prepared for PNSF3 (Fig. 2). SVG file keeps information about places, transitions, predicates, markings and placement of these elements. Using option of animation of dynamic SVG, it is possible to simulate a Petri net. At present stage, simulation of Petri net without interpretation is made. The next step of researches will be studying possibilities of simulation of interpreted Petri net.

5. CONCLUSIONS

In the paper a new format (PNSF3) based on XML for describing Petri net, is presented. The benefit of this format is that it allows easy changes and transformations to other formats. PNSF3 is described in XML syntax because XML parsers are easily available and XML is becoming an international standard. In the presented format, placement and set of rules are separated. It is both, advantage and disadvantage, because the format is very easy for preparing and change, but a bit difficult for converting into a graphical format. This problem is solved by application, which generate graphical format. Using of XML for describing Petri net makes it possible to simulate Petri net, by using dynamic SVG.

REFERENCES

[1] M.Adamski, M.Wegrzyn, "Hierarchically Structured Coloured Petri Net Specification and Validation of Concurrent Controllers", *39th International Scientific Colloquium IWK'94*, 1994, Ilmenau, Germany, Band 1, pp.517-522

[2] E.Best, C.Fernandez, "Notations and terminology on Petri net theory", *Petri Net Newsletter*, 1986

[3] R.David, H.Alla, *Petri Nets & Grafcet. Tools for modelling discrete event systems*, Prentice Hall, New York, 1992

[4] L.Ferrarini, "An Incremental Approach to Logic Controller Design with Petri Nets", *IEEE Transactions on System, Man, and Cybernetics*, Vol.22, No.3, May/June 1992, pp.461-474

[5] R.B.Lyngsø, T.Mailund, "Textual Interchange Format for High-Level Petri Nets", *Workshop on Practical Use of Coloured Petri Nets and Design*, June 1998, Aarhus University, pp.47-64

[6] T.Murata, "Petri Nets: Properties, Analysis and Applications", *Proceedings of the IEEE*, Vol.77, No.4, April 1989, pp.541-580

[7] J.L.Peterson, *Petri Net Theory and The Modeling of Systems*, Prentice-Hall, Inc., Englewood Cliffs, N.J., 1981

[8] Extensible Markup Language (XML) 1.0. W3C Recommendation, *http://www.w3.org*

[9] Scalable Vector Graphics (SVG) 1.0 W3C Candidate Recommendation, *http://www.w3.org*

The International Workshop
on Discrete-Event System Design, DESDes 01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

FAIL-SAFE VHDL DESCRIPTIONS OF PETRI NET SPECIFICATIONS

Miguel Pereira[1], Enrique Soto[2]

1 Intelsis Sistemas Inteligentes S.A. - R&D Digital Systems Department, Vía Edison 16
Polígono del Tambre 15890, Santiago de Compostela (La Coruæa), mpereira@intelsis.es;
2 Dept. Tecnología Electrónica, Universidad de Vigo, Apdo. Oficial, 36200 Vigo,
Espaæa, esoto@uvigo.es, http://www.dte.uvigo.es;

Abstract. This work presents a method for obtaining fail-safe systems based in parity alternation from Petri net specifications. A fail-safe system is a system with adequate redundancy for detecting failures and preventing them. This method generates a VHDL description of a system from a Petri net or state diagram description. These results have relevance in the integration of access technologies to high speed telecommunication networks, where fail safe mechanisms are becoming an important concern.

Key Words. Fail-safe Systems, Hardware Description Languages, Petri Net specifications

1. Introduction

This work uses the parity alternation method to provide fail-safe [1] [2] [3] characteristics to graphical specifications of the type state diagram or Petri net [4]. The starting point is the unsafe state diagram or Petri net specification from which a VHDL description of the equivalent fail-safe system is generated. To do that a program providing a graphical interface reads the specification and internally generates a state array from a binary structure characterising the specification. The array is transformed appropriately to obtain a fail-safe system and turn it into a VHDL description. A block diagram of the processes involved is shown in figure 1.

2. Parity alternation method

This work uses the parity alternation method to provide the fail-safe mechanism, as described in [1]. The method and the proof of equivalence between the safe and unsafe specifications, which are available to the reader in the references, is not the objective of this paper. However, it can be summarised in the following paragraph.

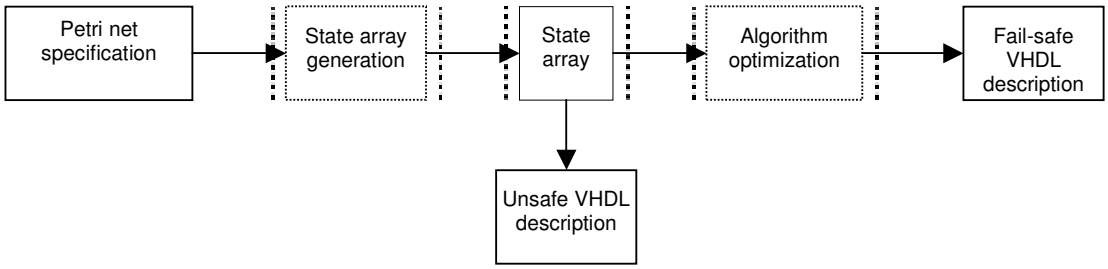


Fig. 1.Block diagram of the proposed methods

The method consists of coding every state in a state diagram with a different parity respect to those contiguous states in the diagram. Every transition in the state diagram goes from a source state to a destination state with a different parity. Fail-safe means that if an error occurs and a transition between two states with the same parity has occurred, then the error can be detected and the system driven to a known safe state where no harm can be done. If the system could correct the error and keep working, then it would be called fault-tolerant (which is not the case). The new diagram is equivalent in the sense that its behaviour is identical from an external point of view.

3. Algorithm process

The initial structure from which all the rest derive is the starting state diagram. Figure 3a shows an example of a state diagram data structure for a Petri net. That information is stored in a structure formed by three object lists. The first is the list of places or states, the second is the list of transitions and the third is the list of the connections between places and transitions. The handling of the information is best achieved using an object oriented programming language. The hierarchy of the objects involved in the treatment of the specification is shown in figure 2, where objects in double boxes form the data structure, while the simple boxes are objects defined to inherit the properties provided by the programming language.

Tlugar defines the places and Ttransicion the transitions in the Petri net. TVar_ES is dedicated to input/output signals. TPuntero and inherited objects are pointers that relate all the structure. TPetriNet contains, with the help of all the other definitions, the information of the whole Petri net.

In the case of a state diagram, the information to store and the process to follow is simplified. The structure is first processed to obtain a state array, searching in the structure for every state its predecessor. This array is transformed as per [1] to get a new state array (see figure 1). This new state array is stored in a new class of objects from which the VHDL is generated. Since the process multiplies the number of states, it may be necessary to compress the information in memory.

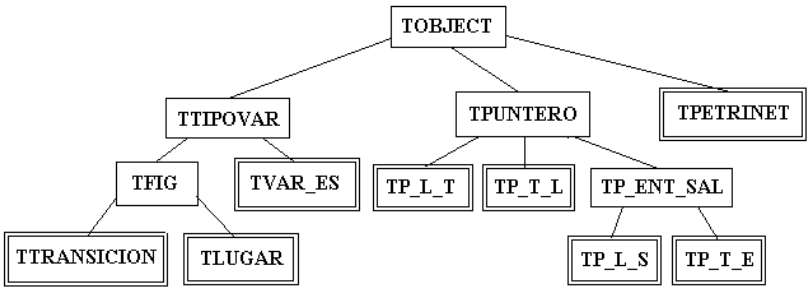


Fig. 2. Data structure

In the case of a Petri net, the process starts by transforming it in an equivalent state diagram, trying all the transitions between the places that can form a state, even if most will never happen. For a Petri net with 10 places, the equivalent state diagram can have at most 2 to the power of 10 states. That will make a 1024x1024-state array that if necessary to duplicate in the fail-safe version will make 2048x2048, forcing a compression of the information. The compression algorithm is facilitated by the fact that most of the values of the array will be 0 s.

The application program identifies whether it is working with a Petri net or a pure state diagram, but uses the same data structure. If there is a transition associated to several places, coming from or leading to the transition, the structure is processed as a Petri net. Petri nets are encoded using one-hot encoding. State diagrams are encoded using binary codification. The number of bits used to code each state in this last case follows:

$$number\_of\_bits = exc(\log2(number\_of\_states)) \quad (1)$$

where exc (x) means rounded up.

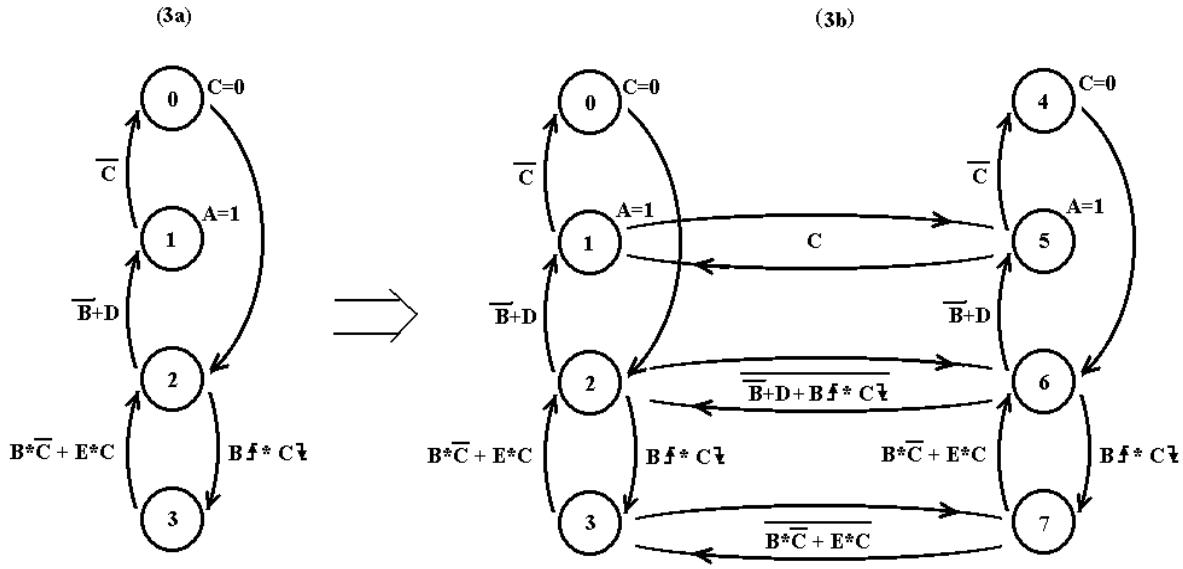


Fig. 3. Example

4. Examples

Figure 3a shows a state diagram, list 1 its VHDL entity description, list 2 the unsafe version and list 3 the results obtained after been transformed by the algorithm into a fail-safe VHDL description (see figure 3b). Observe that it was necessary to duplicate the number of states to obtain the required parity alternation.

REFERENCES

[1] J.J.Rodríguez Andina, J.` lvarez y E.Mandado, Design of safety systems using Field Programmable Gate Arrays , FPL 94 4th International Workshop on Field Programmable Logic and Applications, Springer-Verlag, ISBN 3-540-58419-6, Praga 94.

[2] J.J.Rodríguez Andina, S.FernÆndez y E.Mandado, Implementation of logic controllers with concurrent fault detection capabilities in PLDs , , IOLTW 96 2nd IEEE International On-Line Testing Workshop, St. Jean-de-Luz 96.

[3] J.J. Rodriguez Andina, Santiago Fernandez and Enrique Mandado, "Design and Validation of Fail-Safe FSMs Using Regular Structures", Proceedings of the XII Design of Circuits and Integrated Systems Conference - DCIS'97, pp. 131-136. Sevilla, November 18-21, 1997.

[4] Zurawski, R., M.C. Zhou. "Petri Nets and Industrial Applications: a Tutorial". IEEE Trans. on Industrial Electronics, Dec. 1994.

ACKNOWLEDGEMENTS

This work was financed by the European Commission and the Comisión Interministerial de Ciencia y Tecnología (Spain) through research grant TIC 1FD97-2248-C02-02 in collaboration with the company Versaware S.L. (Vigo, Spain).

List 1

```
-- Entity description of Petri net example
Entity Example is
Port (
clk: In std_logic;
INIT: In Std_logic;
B, C, D, E: In Std_logic;
X,Y: Out Std_logic  );
end Example;
```

List 2

```
-- Unsafe architecture
architecture unsafe of Example is
--Biestables for detecting transition activated inputs
Signal  ant_B, ant_C:  Std_logic;
-- Current state register
Signal  current_state: Integer range 0 to 3;

begin
-- Descriptions of the transitions
transic: process
begin -- Only with a positive transition in clk
wait until (clk'event and clk='1');
if INIT = '0' then current_state <= 2 ;  --Init. signal
else
case current_state is
-- state 0
when 0 =>  current_state <= 2 ;

-- state 1
when 1 => if C='0' then current_state <= 0 ; end if;

-- state 2
when 2 =>
if ( (B='1') and (ant_B='0') ) or ( (C='0') and (ant_C='1') )
then  current_state <= 3 ;  end if;
if ( B='0' ) or ( D='1' ) then current_state <= 1;  end if;

-- state 3
when 3 =>  if ( (B='1') and (C='0') ) or ( (E='1') and (C='1')
)
then  current_state <= 2 ;   end if;

when others => null;

end case;
end if;
end process;

-- Description of the outputs
salidas: process
begin
wait until clk'event and clk='1';
case current_state is
when 0 =>  Y <= '0';
when 1 =>  X <= '1';
when others => null;
end case;
end process;

-- Description of transition activated inputs
flancos: process
begin -- Only with a positive transition in clk
wait until clk'event and clk='1';
ant_B <= B;  ant_C <= C;
end process;
end unsafe;
```

List 3

```
architecture safe of Example is
--Biestables for detecting transition activated inputs
Signal  ant_B, ant_C:  Std_logic;
-- Current state register
Signal  current_state: Integer range 0 to 7;

begin
 -- Description of the transitions
transic: process
begin -- Only with a positive transition in clk
wait until clk'event and clk='1';

if INIT = '0' then current_state <= 2;  -- Init. signal
else
case current_state is

-- state 0
when 0 =>  current_state <= 2 ;

-- state 1
when 1 => if C='0'then current_state <= 0 ;
else current_state <= 5 ;   end if;

-- state 2
when 2 =>
if ((B='1') and (ant_B='0')) or ((C='0') and (ant_C='1'))
then current_state <= 3 ;
elsif (B='0') or (D='1')then current_state <= 5 ;
else  current_state <= 6 ;  end if;

-- state 3
when 3 => if((B='1') and(C='0')) or((E='1') and(C='1'))
then current_state <= 2 ;
else current_state <= 7 ;    end if;

-- state 4
when 4 =>  current_state <= 6 ;

-- state 5
when 5 => if C='0' then current_state <= 4 ;
else current_state <= 1 ;  end if;

-- state 6
when 6 =>   if(B='0') or(D='1') then current_state <= 1 ;
elsif ( (B='1') and (ant_B='0') ) or ((C='0') and (ant_C='1')
) then current_state <= 7 ;
else current_state <= 2 ;  end if;

-- state 7
when 7 =>  if ( (B='1') and(C='0') ) or( (E='1') and(C='1') )
then current_state <= 6 ;
else current_state <= 3 ;   end if;

when others => null;
end case;
end if;
end process;

 -- Descriptions of the outputs
 salidas : process
begin  --Only with a positive transition in clk
 wait until clk'event and clk='1';
case current_state is
when 0 =>  Y <= '0';
when 1 =>  X <= '1';
when 4 =>  Y <= '0';
when 5 =>  X <= '1';
when others => null;
end case;
end process;

-- Description of transition activated inputs
flancos: process
begin
-- Only with a positive transition in clk
wait until clk'event and clk='1';
 ant_B <= B;  ant_C <= C;
end process;

end safe;
```

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

BENEFITS OF HARDWARE ACCELERATED SIMULATION

Remigiusz WIŚNIEWSKI, Arkadiusz BUKOWIEC, Marek WĘGRZYN

Computer Engineering and Electronics Institute, Technical University of Zielona Góra,
ul. Podgórna 50, 65-246 Zielona Góra, POLAND,
R.Wisniewski@cookie.iie.pz.zgora.pl, A.Bukowiec@cookie.iie.pz.zgora.pl,
M.Wegrzyn@iie.pz.zgora.pl

Abstract. *Today's sophisticated digital designs are rapidly evolving. Software-only simulation of such designs takes weeks or even months. Therefore a new generation of chip design requires a hardware-assisted solution. Hardware acceleration delivers a 10-1000 times performance boost. In the paper technologies of hardware simulation on selected examples are described.*

Key Words: *HDL, design verification, simulation, hardware acceleration, PLI*

1. INTRODUCTION

The designs are rapidly evolving, doubling in size with each generation and heading to tens million gates by 2002. This causes dramatic increase of the simulation run time. It is harder and harder to simulate designs because the simulation time has increased from minutes and hours to days and weeks. Therefore, the days of being able to verify ASICs and system-on-a-chip (SoC) designs through software-only simulation are over. Simulation assisted by special hardware is the best solution for speeding up the simulation of large design sections that have been tested and accepted by RTL simulations.

In the paper the three selected examples of hardware simulators are discussed. In addition, an analysis of software-only and hardware accelerated simulation of example complex design is described.

2. DESIGN VERIFICATION

A verification of SoC was time consuming because there were no fast RTL simulators and easy to use hardware emulators. The existing hardware accelerators for gate level design verification require extensive set-ups, good design understanding, lengthy compilations and tedious design partitioning into multiple FPGA devices. Companies involved in large SoC designs with critical time-to-market demands use this technology. It is not for the entry-level designers working with a limited budget.

The new RTL code hardware accelerators hold some promise but it is alleged that they involve lengthy compilations. They also do handle neither mixed VHDL/Verilog, nor very

large designs. Also, some IP cores come in EDIF or other proprietary formats and they need to be simulated outside those RTL accelerators. Until now there hasn't been a simple and universal tool or method that would accelerate verification of both RTL code and netlist-based (gate level) IP cores [10].

Figure 1 shows current design prototyping process [12]. At the beginning a designer prepares module *A* that is verified in simulator. Then he can create module *B*. When module *B* is ready, it is verified as a separate module and then with the rest of the design, and so on. This process is called also incremental prototyping. However the described process takes hours and even days. Current designs consists of several millions gates, so the simulation takes too much time. Therefore CAD companies introduced new technology – hardware accelerated simulation.

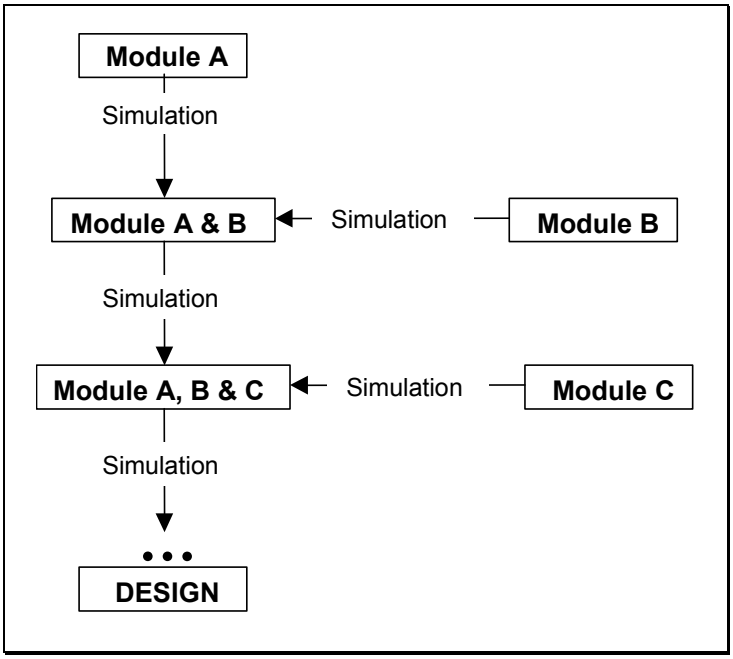


Fig. 1. Current design verification process

3. HARDWARE ACCELERATED SIMULATION TECHNOLOGY

Hardware simulation is a technology that allows speed up simulation time, turning weeks or months of simulation into days or even hours. Designer can "push" whole or a part of the design into hardware. Because it is rather a new technology every solution is different and has different features. Some vendors produce only hardware simulators, others manufacture hardware and also software simulators.

3.1. Xcite-2000 (Axis Systems, Inc.)

Xcite-2000 [3] is the simulation acceleration technology. Xcite-2000 preserves the native simulation-debugging environment. With its Re-Configurable Computing (RCC) technology, Xcite-2000 offers simulation performance up to 100K cycles/second on a design capacity up to 10 million ASIC gates.

Transparent Simulation access to RCC Architecture for Xcite-2000 integrates software and hardware into one unified package. The hardware contains a Re-Configurable Computer (RCC) composed of the Altera FPGA Flex [2]. Xcite-2000 RCC directly connects with the workstation via one set of PCI extender. Whether the system design is described at the

behavioural, RTL or gate level, the Xcite-2000 compiler custom configures its computing elements to maximize parallel processing. Thus, design simulation with Xcite-2000 is identical to software simulation at hardware speed.

Design description can be separated into three components: behavioural, RTL and gates. The Xcite compiler automatically maps sections, which can be RCC accelerated (RTL and gate level components) and builds a native compiled simulation image for behavioural sections, which need to stay within the Axis software simulator, Xsim. Using "Hierarchy Extracted" mapping technique, the Xcite compiler automatically maps the design onto arrays of FPGAs.

One of the unique capabilities of Xcite-2000 is its ability to swap software and RCC state in real time. Thus during simulation, the user may choose to swap all RCC state into Xsim in order to debug the design and continue software simulation. Once the circuit is fully diagnosed, simulation state value can be swapped back into RCC for maximum performance acceleration.

Within Xcite RCC simulation, simulation history for all nodes is compressed within RCC and stored onto the workstation. Either during or after simulation, the Xcite VCD-on-Demand capability can extract all node history values without re-simulation. Thus design debugging has become highly efficient without the high cost of disk storage or simulation slowdown.

3.2. Viking CSM (IKOS Systems, Inc.)

The Viking CSM [5] co-simulator offers hardware accelerated simulation performance for verification of VHDL designs on the ModelSim software simulator from Model Technology, (MTI) [7]. The Viking CSM system delivers mixed-level acceleration through a tight integration of the ModelSim software with the NSIM and ARES hardware accelerators from IKOS.

Users work with their existing verification environment, workflow and language, using the same simulation test-benches to accelerate their designs. Viking CSM extends the command set and GUI capabilities of ModelSim's to support IKOS' accelerators. The product also allows the Foreign Language Interface (FLI) to ensure that Viking CSM will integrate properly with the user's systems and all peripheral tools.

Viking CSM offers also designers access to the debug capabilities of ModelSim without having to switch between different software and hardware environments.

The NSIM is an event-based hardware accelerator. This proven technology is based on a massively parallel, hyper-cube architecture optimised for event-driven simulation. Large designs are automatically partitioned across parallel custom simulation processors or "clusters" to achieve the accelerated speeds. ARES is a very affordable desktop accelerator based on the proven NSIM architecture, but is optimised specifically for RTL acceleration and comes bundled with software.

3.3. Hardware Embedded Simulation (Aldec, Inc.)

Hardware Embedded Simulation (HES) [1] is the technology that facilitates the incremental design verification of FPGA and ASIC devices while speeding up design verification. HES technology allows you to download selected modules of your design into an FPGA and perform hardware-software co-simulation. After a design block has been verified at the behavioral level, it is synthesized, implemented and downloaded into an FPGA residing on an accelerator board. HES technology supports up to four acceleration boards residing in one computer. The boards are the PCI cards inserted into the slots of the computer.

The entire design is simulated in the HES environment, which consists of an HDL software simulator and PCI boards. This environment assures correct communication between modules located in silicon and modules simulated in software.

Using the HES technology, verified modules of the design can be put into silicon after the synthesis of even a small part of the design. User needs to synthesize the modules that should be pushed into silicon, and the HES Design Verification Manager (DVM) will help to configure HES environment.

Aldec's simulator is based on the Incremental Prototyping. Figure 2 shows the idea of Incremental Prototyping. When module *A* is finished, it is synthesized and implemented and finally downloaded to the HES board. Since module *A* resides in the hardware simulator, the designer can prototype module *B* in software. When module *B* is verified successfully at the software level, it goes thru incremental synthesis and incremental place and route processes. Note that since module *A* now resides in the hardware, it is not synthesized and implemented again.

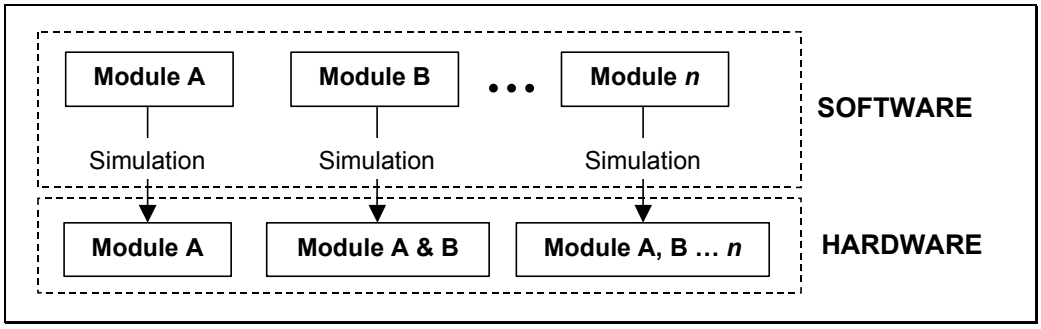


Fig. 2. Hardware acceerated design simulation process

HES boards have been built with Xilinx FPGA Virtex [11] or Altera CPLD Apex [2] devices. They are placed on the PCI board that can be put into the slot of the PC or Sun computer.

Aldec's hardware simulator works on the following platforms and configurations:

- Solaris (Sun) with MTI, Cadence [4], or Riviera simulators [1];
- Linux/Unix (PC) with Cadence, or Riviera simulators;
- Windows NT/2000 (PC) with MTI, or Active-HDL simulators [1].

3.4. Summary

Table 1 shows the summary of the selected hardware accelerators. Information about other solutions is described, for example, on [5,11].

Table 1. Summary of hardware accelerator

Feature	***HES***	***Xcite-2000***	***Viking CMS***
Real simulation	YES	YES	YES
Maximum Capacity	2,5 mln gates	1 mln gates	?
Used Chip(s)	Xilinx – Virtex Altera - Apex	Altera - Flex	Xilinx – Virtex
Supported Languages	Verilog, VHDL, EDIF netlist	Verilog	VHDL
Supported Simulators	Active-HDL, Riviera, ModelSim, Verilog XL/ NC-Sim	Xsim	ModelSim
Platforms	PC, SUN	SUN	SUN
Supported Operating Systems	Windows, Linux, Solaris	Solaris	Solaris

4. DESIGN EXAMPLE

This section shows the benefits of hardware simulation. The following results are based on a comparison between software and hardware simulation. The following example is based on using Aldec's software simulator (Active-HDL) and hardware accelerator board (Hardware Embedded Simulation - HES).

As an example a system consisted of two processors *MASTER* and *SLAVE* (Fig. 3). The design is a part of a bigger, hypothetical digital system, where data are processed in various ways. The purpose of these blocks is to process input data, and subsequently to send the result of processing to another module. The way that data are processed is not important for considered analysis. Since data can be transmitted from one module to another in an encrypted format, the design must be able to decrypt encrypted data. The first part (*MASTER* block) is responsible for data processing and communication with the rest of the system. The second part (*SLAVE* block) is responsible only for data encryption and decryption. Both blocks are modelled in Verilog-HDL [6]. Programming Language Interface (PLI) [8], a part of Verilog 1364-1995 standard (or a new, updated version – Verilog 1364-2001) is used for implementation of data exchange between the hardware and the software.

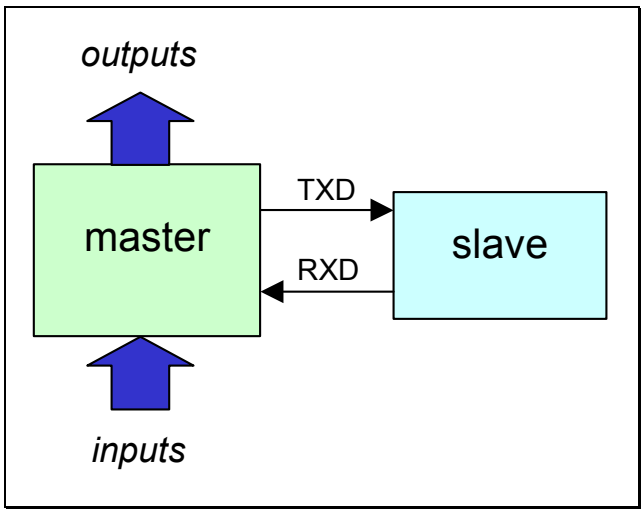


Fig. 3. Block diagram of an example system

When the *MASTER* block of the design is ready it is verified in software. The software verification of this Master block takes approximately 45 minutes. Next the *SLAVE* block is developed in software and also verified. The verification of the *SLAVE* block takes approximately 30 minutes. Verification of entire design is completed in approximately one hour and 15 minutes. Now if the design is not working properly it should be corrected and re-verified, repeating the entire cycle over again until the design is correctly verified in software. Conversely, this reiterative process can take weeks or even months.

Using HES hardware, the *MASTER* block is "pushed" into hardware after correct software verification. When the verification process is run again, the *MASTER* block will now simulate and verify in approximately 15 seconds versus 45 minutes. Next the *SLAVE* block is developed in software. Simulation and verification of this block takes 30 minutes in software. But when simulation and verification of both blocks is run again, verification of the entire design is completed in 30 minutes and 15 seconds. There is a noticeable decrease in time due to the *MASTER* block residing in hardware. Once the *SLAVE* block is verified correctly in software, it is "pushed" into the hardware, as well. The two blocks now reside in hardware.

The next simulation of the two blocks takes place in hardware, and the whole design is completed in 25 seconds (15 seconds for the *MASTER* and 10 seconds for the *SLAVE* block).

Table 2 shows times of hardware and software simulation for the considered design.

Table 2. Hardware vs. software simulation

Part of system	*Software simulation [min]*	*Hardware accelerated simulation [sec]*	*Software vs. Hardware [times]*
1. Master Block	45	17	159
2. Slave Block	30	10	180
3. The whole system	75	27	167

5. SUMMARY

Present digital designs are bigger and bigger. Very often simulation of such designs takes not only minutes and hours, but whole days or even weeks. The best solution for this problem seems to be hardware accelerated simulation. After finishing the design at the RTL level, a user can push it into a real device and do simulation or emulation of the design.

Hardware acceleration delivers a 10-1000 times performance boost. In the paper a comparison between only three representative examples of hardware accelerations has been presented. Advantage of such approach for system verification has been described on an example of a system that consists of master and slave processors.

REFERENCES

[1] Aldec, Inc., *http://www.aldec.com*

[2] Altera, Inc., *http://www.altera.com*

[3] Axis Systems, Inc., *http://www.axiscorp.com*

[4] Cadence Design Systems, Inc., *http://www.cadence.com*

[5] IKOS Systems, Inc., *http://www.ikos.com*

[6] Institute of Electrical and Electronics Engineers, *IEEE standard 1364-1995 „IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language"*, New York, 1996

[7] Model Technology, Inc., *http://www.modelsim.com*

[8] S.Mittra, *Principles of Verilog PLI*, Kluwer Academic Publishers, Norwell, 1999

[9] Quickturn, *http://www.quickturn.com*

[10] M.J.S.Smith, *Application-Specific Integrated Circuits*, Addison-Wesley, Boston, 1997

[11] Tharas Systems, *http://www.tharas.com*

[12] Xilinx, Inc., *http://www.xilinx.com*

[13] M.Zwoliński, *Digital System Design with VHDL*, Prentice Hall, Harlow (England), 2000

SESSION V:
IMAGE RECOGNITION

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

AUTOMATIC SYSTEM FOR TV RASTER PARAMETERS TUNING

Raouf SADYKHOV[*], Aliaksei KLIMOVICH[*], Leonid PODENOK[**]

[*] Belarussian State University of Informatics and Radioelectronics, , 6 P.Brovka st., Minsk., BELARUS, phone 239-80-39, e-mail: *nil36@bsuir.edu.by*

[**] Institute of Engineering Cybernetics, Belarussian State University of Informatics and Radioelectronics, 6 Surganov st., Minsk, BELARUS, ph/fax +375 17 2310982, e-mail: *podenok@lsi.bas-net.by*

Abstract. *The automatic system of TV raster parameters tuning for television receivers has been worked out. For achieving a required precision of raster parameters and compensation of nonlinearity and geometric distortions the original algorithms of image processing are used.*

Key Words: *nonlinear system, neural network, Volterra series, approximation.*

INTRODUCTION

In modern TV receivers the problem of digital control such parameters of a raster as a vertical and horizontal shift, raster size horizontal and vertical, and also number of other parameters digitally is very actual. All these parameters as a rule depend on parameters of the deflector system and electronic circuits. Unfortunately scatter of parameters makes hard influence on characteristics of a displayed raster, that stimulates a problem of raster parameters tuning on the base of computer. In the industrial conditions this problem is executed by specially trained expert, who manually reduces distortions of a raster to a minimum. The development of automatic system for TV raster parameters tuning without human participation allows to reach required minimum distorsions of the raster and essentially reduce time of tuning. The created closed system satisfies the requirements for the computer vision systems and includes original algorithms of image processing, approximation of nonlinearities on the base of Volterra series and neural network approach.

1. METHODS AND ALGORITHMS OF TV RASTER IMAGE PROCESSING

For TV raster image capture the CCD colour video camera "Sanyo" has been used and the resolution of an obtained image is equal to 640x480. Unfortunately using of the video camera calls a number of problems connected with decrease image quality for TV raster. Since the video camera has an automatic system of image brightness adjustment, it results in the certain delay for obtaining an image. Furthermore brightness of separate areas of an image is received

unequal because of the nonlinear distortions of the raster are present, and obtained image also can be hardly noised.
We have considered and tested various methods of image preprocessing such as a median and Gaussian filtration. As experiments have shown fast and qualitative outcomes Gaussian filtration with a size of the window 3x3 has shown. It allows to smooth some noise added into an image by an equipment and the external factors of illumination.
Further the histogram of image intensity is created, which has been used in future for determination of an average level of light exposure for image and calculation of black and white levels.

2. COMPENSATION OF DISTORTIONS

The image of the same grid field to be captured with CCD-camera and frame grabber depends on a lot of factors, such as curvature of the TV-screen surface, variable distance from the screen up to the CCD-camera, orientation angles of view point, geometrical and metric distortions, entered by the optics-electronic section. Each of the factors makes the specific distortions to the image of a grid.
All significant distortions from the point of view of the precision of parameter measurement are divided on two classes. The first one includes distortions, caused by an relative location and orientation of the CCD-camera and the TV-receiver. The second one includes distortions caused by a CCD-camera lens construction and lens mounting system and distortions caused by optics-electronic channel "CCD matrix image - memory image".
The model of second class distortions describes real geometrical distortions, which take place in a specific opto-mechanical state of real lens. The model uses two representations of distortions field. The first one is based on spline approximation and second one is based on polynomial approximation that is usually used by some optics manufacturers and its parameters are listed in the product manual.
The construction of spline model of lens distortion field is carried out on the basis of experimentally received data for every lens, particular opto-electronic path of the CCD-camera and coder/decoder circuits of capture board. Planar grid was used as the standard object to form the curved and distorted image in computer memory. The specially developed technique of the image processing of this planar grid is used. As a result the set of approximation data corresponding to distinct states of the lens was received.
To determine the relative location of the CCD-camera and TV-set the vectorization of light screen boundaries is carried out. The curve spline representation of that boundary is used to find the coordinate system transformation that minimizes variation of source object boundary. That transformation is used to compensate distortions related with non-focal view point location. To determine the current distance between camera lens and surface of the TV-set we have used angular dimensions of the screen.
The distortions are compensated in consecutive order, at first distortions caused by objective and opto-electronic section, then distortions caused by relative arrangement. Further after all distortions have been compensated the grid detection is carried out and signal frame center is found. After vectorization of the grid image distortions entered by TV-set deflector system are evaluated.

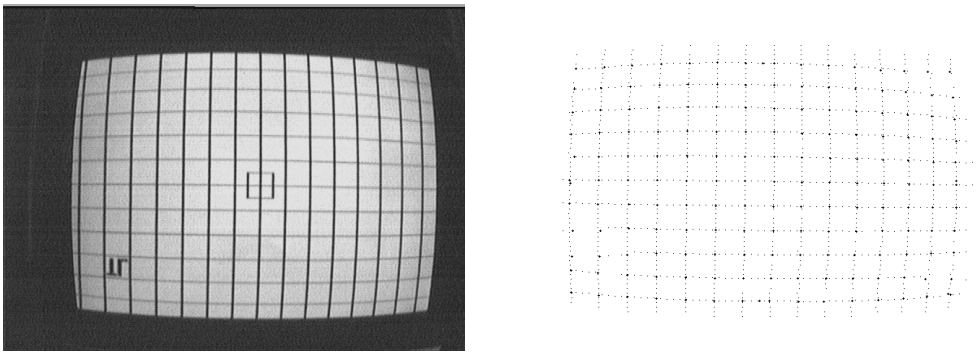


Fig. 1. The compensated image of the TV set raster and point of knots of vectorial representation.

The Fig. 1 shows one from stages of a vectorization of the compensated image.

3. IMAGE PROCESSING FOR SELECTION OF KEY DETAILSES

A primary problem of image analysis is determination of knots of an image grid. For determination of lines on an image the original filtration algorithm based on multilayer perceptron is used. The source vector that contains 25 values was generated as a residual of intensities between adjacent pixels of an image. Neural network was used to increase visibility of lines. The application of the given method has increased performance if previously trained neural network was applied. Neural network recognizes in the processed image a fragment of a standard line of a grid and produces coordinates of that fragment in the vector. This coordinate was used for the recognition of the line fragment in the source vector. For neural network training back-propagation algorithm with an adaptive learning step has been used. [1, 4]

4. APROXIMATION OF NONLINEARITIES

Further an obtained knots allow the calculation of geometric and nonlinear distortions of the raster. It should be noted that evaluation of nonlinear distortions is nontrivial problem. There is a set of methods and approaches to decide this problem. For calculation of vertical nonlinearity, the set of knots on the central vertical line was used. We have used the approximation of obtained points via polynomial of the *n* degree, where the *n* is maximum order of existing nonlinearity.

$$y(x) = \sum_{i=0}^{n} a_i x^i \qquad (1)$$

In practice it is sufficient to calculate nonlinearities up to the third order (n=3). The coefficient of the second order (a_2) corresponds to nonsymmetry, and third order (a_3) - to nonlinearity.

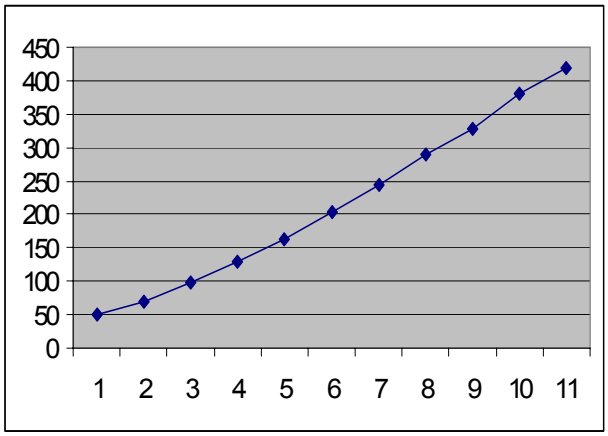


Fig. 2 Experimental data of the grid knots coordinates on the raster for an evaluation of nonlinear distortions.

During experiments (fig.2) we have clarified that nonlinearity and nonsymmetry are correlated with each other and are non-correlated with the next tuning parameters (horizontal position and vertical size of a raster).

Another approach to calculate the nonlinearity and nonsymmetry values is using a Volterra kernels. The Volterra series is a well-known method of nonlinear system describing. The Volterra approach characterises a system as mapping betwen two function spaces of that system. The Volterra series is an extension of Taylor series representation to cover dynamic systems and has the general form

$$y(t) = h_0 + \int_{-\infty}^{+\infty} h_1(\tau_1)u(t-\tau_1)d\tau_1 + \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} h_2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2)d\tau_1 d\tau_2 + \ldots \qquad (2)$$

where $y(t)$ is the output of the system at time t, $u(t)$ is the input at time t, and $h_n(\tau_1, \ldots, \tau_n)$ is the n-th order Volterra kernel.

For a practical systems with finite memory the equation becomes

$$\begin{aligned} y(t) = h_0 &+ \sum_{\tau_1=0}^{T} h_1(\tau_1)u(t-\tau_1) + \sum_{\tau_1=0}^{T}\sum_{\tau_2=0}^{T} h_2(\tau_1,\tau_2)u(t-\tau_1)u(t-\tau_2) + \ldots \\ &+ \sum_{\tau_1=0}^{T}\ldots\sum_{\tau_n=0}^{T} h_n(\tau_1,\ldots,\tau_n)u(t-\tau_1)\ldots u(t-\tau_n) + \ldots \end{aligned} \qquad (3)$$

where T is the memory of the system (i.e. the number of time sampled values to be needed to describe the dynamics of the system).

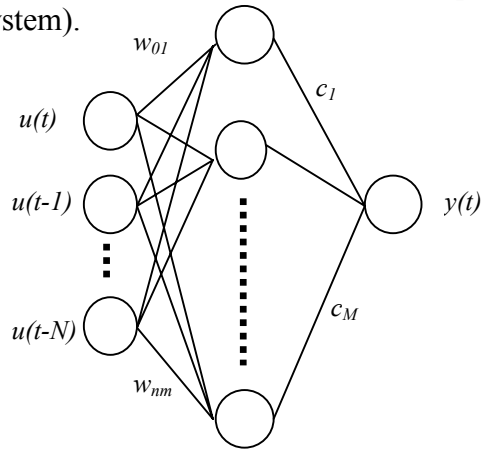


Fig.3 A neural network architecture for time series prediction.

This network (fig.3) is trained to perform a certain mapping between its input layer (on the left) and its output by altering the weights associated with each internal connection. These weights are altered by a training algorithm which takes pairs of ideal input/output data and changes the weights to make the network reproduce the mapping described by the data pairs.
A typical node processing function is

$$op_i = \sigma_i\left(b_i + \sum_{j=0}^{N} w_{ji} u(t-j) \right) \tag{4}$$

where op_i is the output from the hidden unit i, σ_i is the activation function of hidden unit i, w_{ji} is the weight connecting input unit j to hidden unit i, $u(t-j)$ is the input u at delay j, b_i is the bias input to unit i, and $N+1$ is the number of input units. A typical output function, σ, is of sigmoidal shape such as hyperbolic tangent (*tanh x*).
The Volterra kernels are given by

$$h_0 = \sum_{i=1}^{m} c_i a_{0i} \tag{5}$$

$$h_1(j) = \sum_{i=1}^{M} c_i a_{1i} w_{ji} \tag{6}$$

$$h_2(j,k) = \sum_{i=1}^{M} c_i a_{2i} w_{ji} w_{ki} \tag{7}$$

and so the general n-th order kernel is given by

$$h_n(v_1, v_2, ..., v_n) = \sum_{i=1}^{M} c_i a_{ni} w_{v_1 i} w_{v_2 i} ... w_{v_n i} \tag{8}$$

When network has been trained, then the Volterra kernels of all dimensions of this system can be extracted.
In the results the network was trained with the training data set from the TV raster grid nodes, using with the back-propagation algorithm.
So, as we have found the Volterra kernels of 2-nd and 3-rd order are corresponds to such control parameters of TV raster as nonsymmetry and nonlinearity.

5. EXPERIMENTAL INSTALLATION

As controlled and tuned object some serial TV-sets supplied with spherical or toroidal tube were have been used. All these TV-sets had digitally controlled deflector system and were equipped with external data link.
Hardware equipment of the experimental installation consists of the personal computer under OS/2 Warp v.4, CCD-camera and PCI video-capture board. To form the grid field with a TV-frame center mark on the surface of TV tube precise test pattern generator was used. Some the CCD cameras, capture boards and lens systems have been investigated to determine minimal resolution that allow to evaluate the distortions of raster with accuracy that is necessary the raster to be tuned. To reach large non-linearity and non-symmetry of the vertical and horizontal scanning with the purpose of checking the limits of the control algorithm some special changes were brought in to the electronic circuits of the deflection system. The experimental installation is shown on fig.5.

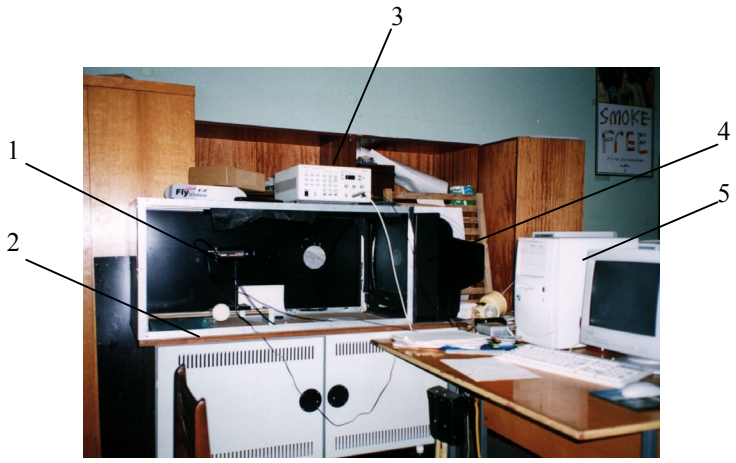


Fig. 5. Structure of an experimental system of automatic TV raster parameters tuning.
1 - CCD the camera; 2- chassis; 3 - generator of a standard signal; 4 - TV set; 5 - control computer.

CONCLUSION

The developed system satisfies to required requests, on quality of set-up for parameters of the raster. The system has shown a high accuracy and performance of tuning, on a comparison with tuning with the help of expert by a manual method. As a rule system defines all the parameters of a raster less than for 25-30 seconds, while the person for entering in control mode, makes tuning at the best 1-2 minutes. And in case of tuning by the person could be quality defects.

REFERENCES

[1] Kohonen, T. "Self-Organization Maps" Springer, 1995.
[2] E. Tsao "Fuzzy Kohonen Clustering Networks" Pattern Recognition, v.27, n.5, 1994, pp. 757-764.
[3] Powell, M.J.D. "Radial basis function for multivariable interpolation: a review" In J.C. Mason and M.G. Cox (Eds), Algorithms for Approximation, 1987, pp. 143-167. Oxford: Clarendon Press.
[4] Y. Hwang "Recognition of Unconstrained Handwritten Numarals by a Radial Basis Function Neural Network Classifier" Pattern Recognition Letters, v.18, n.7, 1997, pp. 657-664.
[5] Moody, J. and C.J. Darken "Fast learning in networks of locally tuned processing units" Neural Computation 1 (2) 1989, 281-294.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

FLEXIBLE RESOURCE ARBITER FOR HETEROGENOUS IMAGE PROCESSING SYSTEM

Jaromir PRZYBYLO, Marek GORGON

Biocybernetic Laboratory, Institute of Automatics, AGH University, al. Mickiewicza 30,
30-059 Krakow, Poland, <*phoenix@biocyb.ia.agh.edu.pl; mago@biocyb.ia.agh.edu.pl*>

Abstract. *In the present paper realisation of resource arbiter for RETINA image processing module has been described. The 32-bit RETINA module is used for image acquisition, processing and analysis. The module's resources include A/C converter, Virtex FPGA device, Motorola 96002 floating-point DSP device and PCI Master interface and they allow real-time realisation of those operations. Resource arbiter is an important control block of the module. It is responsible for the resource allocation for the main control elements of the module, it arbitrates the allocation of internal and external buses, and keeps the information concerning the system state.*

Key Words. *arbiter, real-time, FPGA, re-configurable computing, image processing*

1. INTRODUCTION

The technological progress in the field of production of FPGA devices in the last two years, oriented towards tailoring of the resources of FPGA devices for the needs of digital image processing, makes possible realisations of hardware or software-hardware vision systems. The mostly widespread way of realisation of digital image processing algorithms is the software method [1][2][3][4]. The software's basic advantage is the possibility of convenient and flexible realisation of the algorithm. Because of the high calculational complexity of the image processing algorithms their software realisation is not always efficient enough to allow the algorithm's work in real time. Therefore search continues for multiprocessor architectures [5], hardware methods [6][7] and software-hardware methods [8][9] speeding up the execution of calculations. Great popularity has been achieved by the realisation of image processing and analysis algorithms in DSP [10][11] and specialised or dedicated hardware processors [12][13]. Many attempts have been made of realisation in complex, multiprocessor architecture [5][7][8]. Solution earning a steadily growing popularity is the implementation of processing algorithms, and recently also image analysis algorithms, in reprogrammable devices [12][13][14][15].
Essential advantages of the FPGA-based architectures for image processing are their flexibility, efficiency and structural adaptation to tasks consisting of multiple and parallel execution of algorithms for relatively simple data like image pixels.
In the present paper heterogeneous architecture has been shown, in which a single FPGA chip of high densities has been used both for realisation of the image processing and controlling the resources of the device itself. The paper contains the discussion of architecture and

working modes of that part of the implemented FPGA chip, which realises the function of the resource arbiter for the constructed image processing system.

2. THE ARCHITECTURE OF THE RETINA HETEROGENEOUS IMAGE PROCESSING SYSTEM.

The 32-bit architecture of heterogeneous image processing system is based on Virtex device working in co-operation with 96002 floating point DSP. The board is equipped with high-speed analogue to digital converter, several memory blocks, real-time clock and 32-bit PCI Master interface (AMCC). The Virtex chip combines two functions. It contains all the necessary control logic (FPGA Controller) and is used for performing the image processing operations (Video Processor) - see Fig.1.

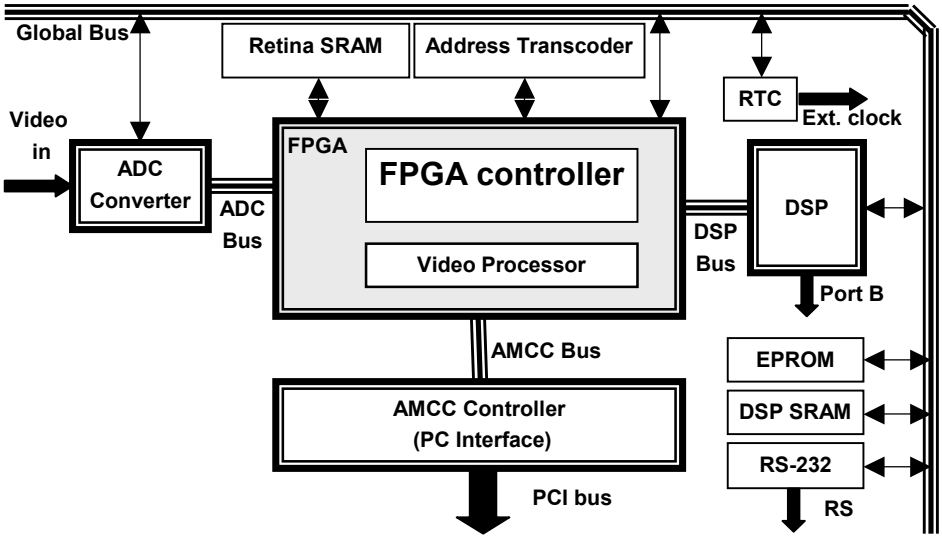


Figure 1. The architecture of heterogeneous image processing system.

The board contains three that can be treated as master devices - DSP96002, Video Processor and PC (through AMCC chip). Optimising the data transfer between the master devices and memory blocks was an essential goal of implementation of the FPGA Controller.
The FPGA Controller (Fig.2) consists of four modules - three dedicated local sub-controllers (Video Processor Controller, DSP Controller and AMCC Controller) and the resource arbiter module. The local sub-controllers are responsible for local arbitration and matching signals between devices. Therefore parallel work is enabled. Resource arbiter controls the data transfers between master devices.

3. RESOURCE ARBITER MODULE

The proposed resource arbiter module enables data exchange between master devices and synchronises their access to memory resources of the module. It enables various arbitration schemes (e.g. token ring), and due to that the possibility of conflict occurrence between devices with "master" privileges. It also enables flexible management of the data transfer by application of two types of device priorities - global and local. If necessary the arbiter's configuration can be changed by using the possibility of reconfiguration of the FPGA Virtex device.
The system bus arbiter module (see Fig.2 - module D) consists of the following elements:

- the resource arbiter (RA) itself
- configuration registers (FPGA Registers)

- buses: FPGA Bus1,2,3 and Internal Bus
- additional arbitration signals

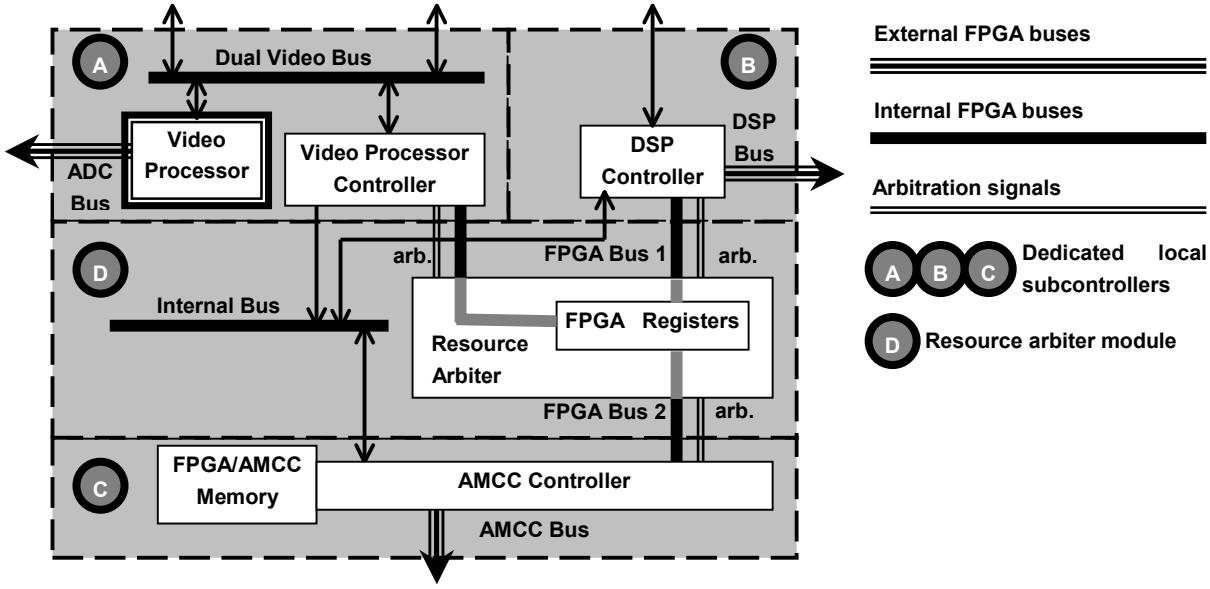


Figure 2. General layout of the controller.

The resource arbiter makes use of the configuration registers, containing the information on the priorities controlling the interrupts and control semaphores governing the arbitration, attributed to particular modules. Their state is mapped to local address spaces, what allows an independent access in any moment - with an essential restriction for the DSP block, when the external global bus is being used by another device. The configuration registers are directly connected with local controllers via FPGA Buses 1,2,3. The resource arbiter also obtains, due to dedicated controllers, various control signals (e.g. interrupts) and the remaining arbitration signals - allowing supervision over the functioning of every "master" device (e.g. reclaiming some resources).

4. THE ARCHITECTURE OF RESOURCE ARBITER.

The architecture of the Resource Arbiter provides the possibility of taking full advantage of module's resources. The introduction of co-operation of three devices provided with potential possibility of work in "master" mode leads to the necessity of ensuring sufficient resources, allowing the device's work in various configurations. The application of the FPGA device allows changes of the control module's infrastructure to be realised fully in hardware, in order to provide the possibility of the device's work in various modes and configurations.
Resource Arbiter consists of the following blocks (see Fig.3):

- Arbitration Unit (AU) - the main control unit of the arbiter, responsible for conflict arbitration and management of the modules resources, containing the Grant Register storing information about intermodular transfers currently taking place;
- Global Arbitration Logic (GAL) - the block co-operating with local arbitration systems;
- Interrupt Controller (IC);
- Control Unit (CU) - the unit generating the clock signals and RESET signals;
- Watch Dog (WD) - the block containing timers, used for supervision of the correctness of particular module blocks functioning;
- Additional Logic - additional auxiliary chips, not included in the block diagram.

Local Arbitration Logic systems have been related with every master devices. The LAL's take over the local arbitration tasks from RA, while the Resource Arbiter synchronises the data transfer between the modules. The arbitration, because of the specific features of the DSP, has been realised in software-hardware manner and it is based on solutions applied in PCI and DSP96002 processor.

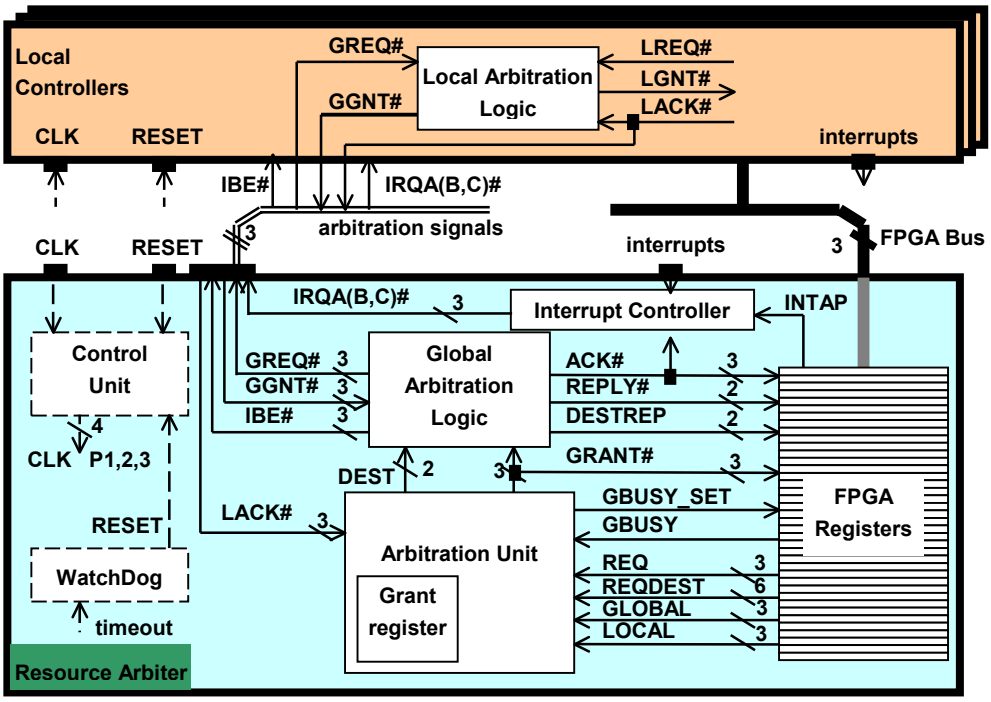


Figure 3. Layout of the Resource Arbiter.

For configuration and setting of the arbiter's working modes control flags semaphores have been used, located in configuration registers (FPGA Registers):

- Interrupt Register (8-bit) - the respective bits of the register are responsible for masking the interrupts, which inform the local "master" devices about allocation of the required resources;
- Priority Register (8-bit) - store the information about local and global priorities;
- Request/Acknowledge Register (3x3-bity + 1 bit) - contains the respective semaphores controlling the arbitration;
- Auxiliary Registers;

4. FUNCTIONING OF THE RESOURCE ARBITER

The operation of the Resource Arbiter should be analysed taking into account the working algorithms of the Local Arbitration Logic systems (LAL's), which take over the local arbitration task from the RA.

The work cycle of the arbitration systems can be divided into several stages, which are supervised by the Control Unit (CU), responsible also for initialisation (RESET) of the module and generation of the clock signals. Division of the work cycle into phases allows the elimination of changes of the FPGA Registers contents during the Resource Arbiter's work, what might lead to irregularities of its functioning. During the arbitration cycle the following phases can be distinguished (see Fig.4):

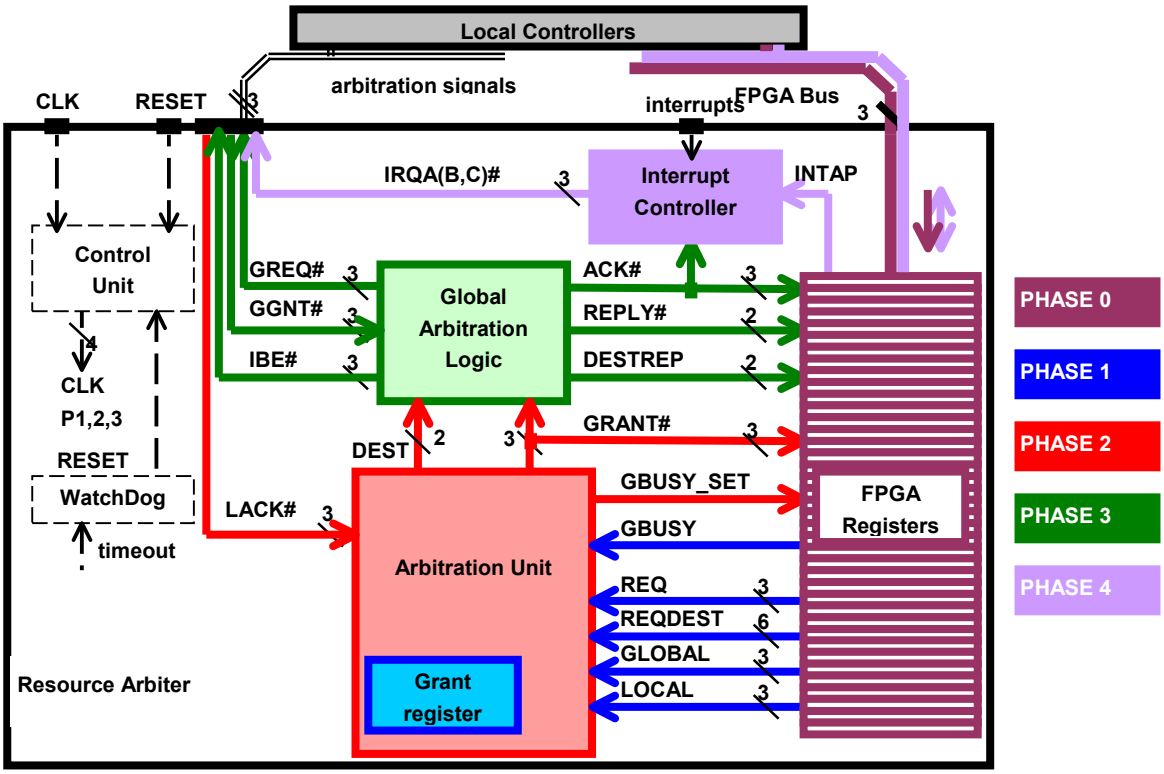


Figure 4. Phases of the arbitration cycle.

- Phase 0 - Writing to FPGA Register by the "master" devices (setting the semaphores) - requests for intermodular transfers (optional phase).
- Phase 1 - Start of the arbitration in the Arbiter system - collection of the information concerning requests, priorities from the FPGA Registers. In absence of Phase 0 initialisation of the Grant Register
- Phase 2 - Resolving the conflicts in the AU unit. Setting of appropriate resource allocation signals for the GAL system and reservation of the internal bus.
- Phase 3 - Negotiation of the resources reclaiming from the current user (its GAL block and local LAL system). After regaining control over the resources - setting the confirmation signal for the initiator. Possible storage of local resources reclaim requests, generated by the previous user
- Phase 4 - Realisation of the intermodular transfer (optional phase) - reception of information concerns the allocation of Internal Bus. Bus reclaiming after finishing the transfer.

5. Conclusions.

The only possibility of fulfilling all the requirements that should be met by the Resource Arbiter of the RETINA image processing module, was its implementation in the reprogrammable FPGA device. Such a solution allows the realisation of flexible and changeable arbiter structure fully in hardware. It provides a possibility of module adaptation for realisation of various arbitration procedures. The important thing is the possibility of configuration of the FPGA's internal memory resources as the module's configuration registers. The hardware implementation ensures great operation speed and high integration level of the arbiter's structure. There is also a possibility to adapt the device's structure to specific algorithms implemented in the system during its usage by the end-user.

The possibility of gradual development of the arbiter's structure during the prototype's testing should be also highly appreciated. The resources of the FPGA device allow a construction of additional modules, supporting the developing process of the arbiter and the FPGA module as a whole. Flexible FPGA structure opens a series of possibilities of RETINA module testing and monitoring of its co-operation with external systems via the PCI bus and serial port.
The implementation has been done in Xilinx Virtex XCV 300BG432-6 device. A prototype of the RETINA card is supplied with BGA432 socket, so XCV800E is considered as a final implementation platform.

ACKNOWLEDGEMENTS

We wish to thank the Xilinx University Program for software donation. The research has been performed under University of Mining and Metallurgy Research Grant no.: 10.10.120.39.

REFERENCES.

1. PRATT W.K., Digital Image Processing, John Wiley & Sons, Inc., 1991.
2. TADEUSIEWICZ, R., Komputerowa analiza i przetwarzanie obrazów, Wydawnictwo Fundacji Postępu Telekomunikacji, Kraków 1997.
3. ULLMAN, S., High-level Vision, The MIT Press 1997
4. UMBAUGH, S. E., *Computer Vision and Image Processing*, Prentice Hall International, Inc., 1998.
5. TADEUSIEIWICZ, R. The multiprocessors architectures for image processing, *Lecture Notes on Computer Vision and Artificial Intelligence*, Warszawa, pp. 226-269, 1989.
6. KUNG, S.Y., VLSI Array Processors, Prentice Hall, New Jersey 1989.
7. JONKER, P.P., Morphological Image Processing: Architecture and VLSI design, Delft University of Technology, 1992.
8. LANGE, A.A.J., *Design and Implementation of Highly Parallel Pipelined VLSI System*, Delft University of Technology, 1991.
9. DAGLESS, E. at al., Image Processing Hardware, *Proc. of the 5-th School Computer Vision and Graphics Zakopane*, Wyd. Format Wrocław 1994.
10. SOREL, Y., Real - Time Embedded Image Processing Applications using the A^3 Methodology, Proc. of ICIP – 96, IEEE 1996.
11. BRUDŁO, P., System przetwarzania i analizy sekwencji obrazów video w oparciu o sieć procesorów sygnałowych serii ADSP-21060, *Konferencja Systemy Czasu Rzeczywistego 2000*, Katedra Automatyki Akademii Górniczo-Hutniczej 2000.
12. WIATR, K. *Architektura potokowa specjalizowanych procesorów sprzętowych do wstępnego przetwarzania obrazów*, AGH Uczelniane wydawnictwa Naukowo-Techniczne, Kraków 1999.
13. GORGOŃ, M. 1995. Evaluation of reliability of dedicated image processors in low level image processing, *Ph.D. Thesis: AGH Kraków, 25 September 1995.*
14. GORGOŃ, M. 1997. Universal Reprogrammable Architecture for Implementation Dedicated Image Processors Based on FPGA. *Proc. of Sixth International Conference on Image Processing and its Applications IPA 97, Trinity College, Dublin, Ireland 14-17 July 1997*, IEE Conference Publication no.: 443, vol.2, pp. 556-560.
15. XUE, J, at al., Approach to constructing reconfigurable computer vision system, Proc. Of SPIE Vol. 4212, SPIE Photonics East Conference, Boston 2000.
16. XILINX 1999. AppLINX Rev.9 First Quarter 1999.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

ALGORITHM FOR IMAGE PROCESSING OF INTEGRATED CIRCUITS ON THE BASIS OF THE "NEOCOGNITRON" NEURAL NETWORK

*Raouf Kh. SADYKHOV, **Maksim E. VATKIN

* Belorussian State University of Informatics and Radioelectronics, 6. P.Brovka Str.,
Minsk, Belarus, 220600, Tel.: (017) 2310982, Fax: (017) 2310982,
rsadykhov@gw.bsuir.unibel.by
** Institute of Engineering Cybernetics, 6. Surganov Str., Minsk, Belarus, 220012,
vatkin@tut.by

Abstract. *The architecture of "neocognitron" neural network in the task of search of structural units on a gray scale image of an integrated circuit is considered. The updated rule for activation of the network neurons invariant to distortions of brightness is represented. The comparative outcomes of recognition have shown an advantage of neural network approach.*

Key Words. *Neural Network, Image Recognition*

1. INTRODUCTION

At present time different kinds of neural networks are applied in the tasks of recognition and image analysis [1, 2, 7, 8]. In the report the search technology on a gray-scale photosnapshot of a chip of an integrated circuit (IC) and its structural units on the basis of the multilayer "neocognitron" neural network [3-6] has been represented. The search algorithm of a separate unit on the image IC is realized by a scanning method of this image by the sliding window, where for each position of the window the fitness measure of the image in the window with the image of a required unit IC is defined. The multilayer neural network of the simplified "neocognitron" architecture with the modified algorithm of training has been applied for calculation of fitness measure.

2. ARCHITECTURE OF THE NEURAL NETWORK.

The common scheme of the neural network architecture is represented in a fig.1. In structure "neocognitron" let us select the following units: *R*-layers, *S*-layers, *C*-layers, $\overline{S}$ -sublayers, $\overline{C}$ -sublayers, $\widetilde{S}$ -neurons, $\widetilde{C}$ -neurons, $\breve{S}$ -links, $\breve{C}$ -links. The *R*-layer is receptor layer, its neurons do not carry any functional load and answer only for transmission of the entry image to the neural network. *S* and *C*-layer are outlined in a figure by a thin line, they consist of $\widetilde{S}$

and $\widetilde{C}$-neurons accordingly, which fulfil the function of feature detection on the image, such as lines, corners, intersections and etc. Everyone *S* and the *C*-layer is divided on $\overline{S}$ and $\overline{C}$-sublayer, which are outlined by a thick line. $\overline{S}$-sublayer consists of $\widetilde{S}$-neurons, which detect the same feature of the image, for example, a line. Thus, $\overline{S}$-sublayer forms some kind of a map of this feature in the previous layer.

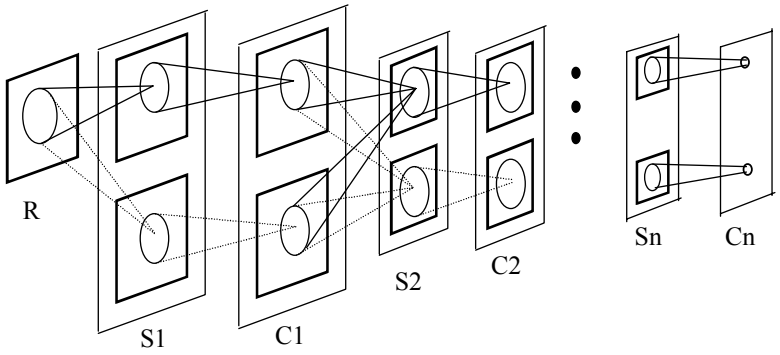


Fig. 1. The common network architecture

$\widetilde{S}$-neurons have $\breve{S}$-links with modified weight coefficients, which accept the values during sublayer training. $\breve{S}$-links to be directed from one $\widetilde{S}$-neuron form in the previous layer a receptor fields, which can be divided on *P* subgroups according to an amount $\overline{C}$-sublayers in the previous layer. Each receptor field subgroup is characterized by size and position in previous sublayer. The size of a receptor field subgroup corresponds to a size of detected feature. The position of a receptor field is defined by a position of a $\widetilde{S}$-neuron in $\overline{S}$-sublayer, i.e. the position of receptor fields for neurons from one $\overline{S}$-sublayer differs only by parallel shift rather each other. As the neurons from one $\overline{S}$-sublayer detect the same feature, then it is possible to train only one neuron from this sublayer and use its weight coefficients for all remaining neurons. With the purpose of time performance optimization in terms of an amount of elementary mathematical operations new procedure of network training and activation algorithms were worked-out. In the given paper the approach based on rejection of inhibitory neurons using that is connected with the problem of complementary memory and mathematical operation, is represented. Actually for training such S-neuron the methods on the base of Kohonen network and radial basis functions are proposed. As a result of the offered approach it is possible to reduce volume of used memory for weights and also amount of multiplication operations in 2 times. $\widetilde{S}$-neuron is trained on the base of function of weight coefficients change

$$w(t+1) = w(t) + \frac{1}{t+1} \cdot \left[u(t+1) - w(t)\right], \tag{1}$$

where *t* is number of training iteration, *w* is value of weight coefficient, *u* is value of activity for the neuron on an input of link to be trained.

The activation function of $\widetilde{S}$-neuron corresponds to the radial basic function.

$$U_{Sl}(n,k) = \exp\left(-\frac{\sqrt{\sum_{p=1}^{P_{Cl-1}} \sum_{v \in Al} \left(U_{Cl}(n+v,p) - \min_{\forall v}(U_{Cl}(n+v,p)) - w(v,k)\right)^2}}{N}\right), \tag{2}$$

where l is the serial number of a layer; k is the number of a trained plane; n - two-dimensional index of a neuron in k-th plane; w - weight coefficient of link; v - two-dimensional shift of entry link in a subgroup of links Al; Al - two-dimensional value describing a size of a receptor subgroup in previous $\overline{C}$-sublayer, N, p, - total amount of entry links and serial number of sublayer to be joined with a trained neuron correspondingly. The parameter $\min_{\forall v}(U_{Cl}(n+v,p))$ is entered into a relation with the purpose of exception in receptor a subgroup of a constant component influence. This is necessary because the neural network mast be invariant to change of brightness for the recognized image.

$\overline{C}$-sublayer consists from $\widetilde{C}$-neurons, that generalize one feature from previous sublayer. That is $\widetilde{C}$-neuron realizes "or" fuzzy logic function on all neuron receptor field

$$U_{Cl}(n,k) = \max(U_{Sl-1}(n+v,p)), \quad \forall v \in Dl, \forall p \in P, \tag{3}$$

where l is serial number of a layer; p, P - serial number of a plane and set of planes from the previous S-layer accordingly; Dl - two-dimensional value describing sizes of $\widetilde{C}$-neuron for a receptor subgroups; v - two-dimensional index of link inside these subgroups.

Usually $\widetilde{C}$-neurons form receptor field in one sublayer from the previous layer, or in several, when it is necessary to combine features to be detected by these sublayers. For example, when in $\overline{C}$-sublayer the feature of brightness overfall is detected, then the links from it $\overline{C}$-sublayer are necessary for installing with two $\overline{S}$-sublayers from the previous layers, first of which detects feature of brightness overfall on dark, and second - with dark on bright.

The position of receptor fields subgroups for $\widetilde{C}$-neuron is defined similarly as for $\widetilde{S}$-neuron.

3. TRAINING

The applied architecture of the neural network is outlined in fig.2,

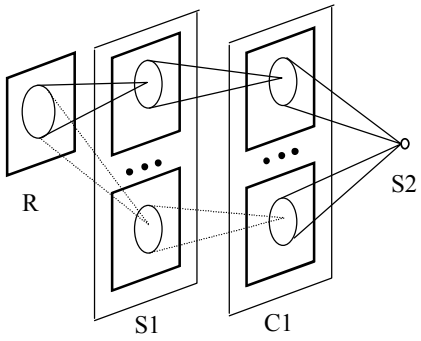


Fig. 2. The used network architecture

where a R-layer is receptor layer. The size of a receptor layer is equal to a size of an image of the unit of IC.

The $S1$-layer is intended for detection of common features for all units of IC, such as linear boundaries of different orientation brightnesses overfalls. All sublayers of this layer consist of neurons with identical sizes of subgroups for receptor field including 4x4 neurons. The learning images for these sublayers are shown in a fig. 3.

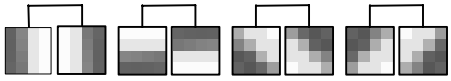


Fig. 3. S1-*layer training images*

The *C1*-layer is intended for generalization of features to be detected in a *S1*-layer, and receptor subgroups of its neurons are organized to combine such pair features, as vertical brightness overfalls with dark on light and with light on dark. Such features are joined in fig.3 by rectangular bracket. As result 4 sublayers in a *C1*-layer are obtained. The size of receptor field subgroups is such, that the activity of neurons from this layer was invarianted in relation to small shifts of features to be detected in the previous layer and makes a 2x2 field.
The *S2*-layer is intended for detection of feature amount to be belonged one concrete unit of IC. As the *S2*-layer forms output value of the network, it consists of one $\widetilde{S}$ -neuron, and sizes of receptor field subgroups of this neuron coincides with a size $\overline{C}$ -sublayer from the previous layer. As the weight coefficients of this neuron keep the unique information on the required image, the creation of the database of IC units is possible.
The structure of the program system is represented on fig.4

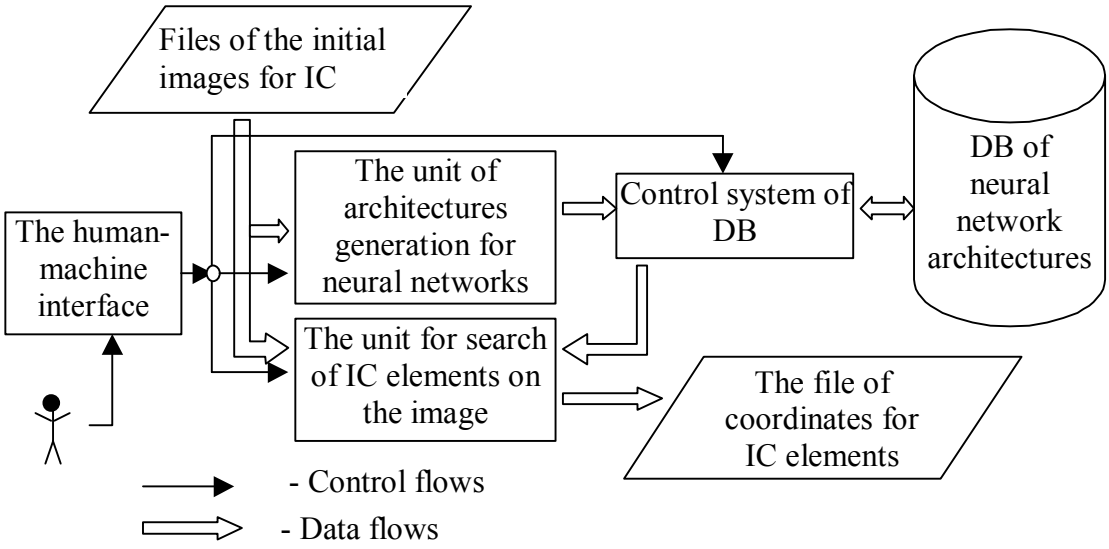


Fig. 4. The program system structure

Taking into account the circumscribed above approach to creation of the architecture of the neural network the program system is realized that fulfils the following operations:

1. Requests of the images samples for required units or loads the indicated architecture from a data base (DB) are fulfilled.
2. The architecture for search of these units on the IC image is formed
3. The information about this architecture of the neural network in a DB, including: a sizes for receptor of a layer and matrix of weight coefficients for output $\widetilde{S}$ -neuron is saved.
4. The search of units on the IC image, with forming of the file containing coordinates of the locations of an indicated unit is made.

4. TESTING

As input data for testing the gray-scale fragments of the photo-image of IC polysilicon layer (fig.5) were used.

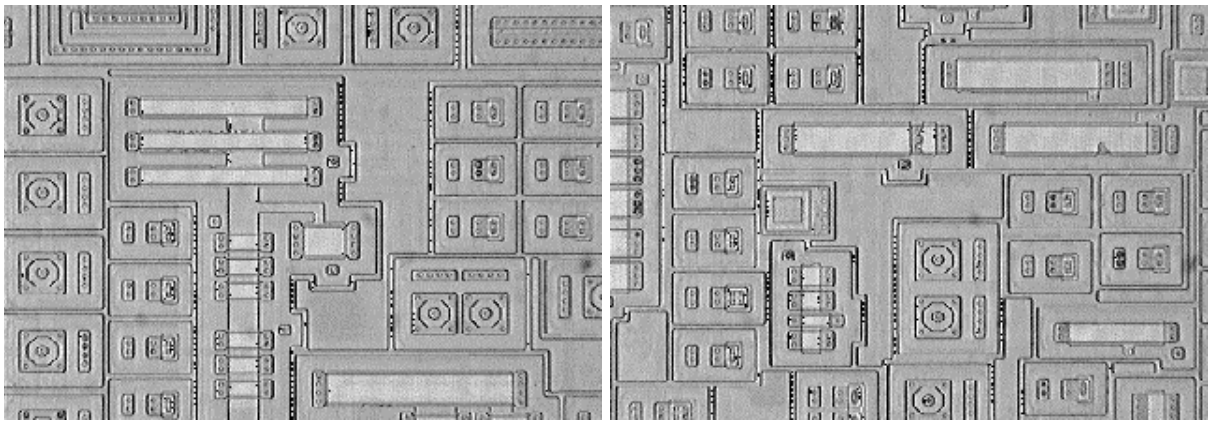


Fig. 5. The IC polysilicon layer images

Two variants of the neural network training have been produced. The learning samplings for these variants are represented in a fig.6

Variant 1 Variant 2

Fig. 6. S2-*layer training images*

To determine the efficiency neural network method for creation of fitness measure the matching with a correlation method was produced. The correlation method consists in an average of the image of IC unit on all learning sampling and calculation of correlation coefficient between the averaged image and image obtained in the moving window. The coefficient of correlation is evaluated in accordance with the relation:

$$\mu = \frac{\sum_{x,y} a_{x,y} \cdot b_{x,y}}{\sqrt{\sum_{x,y} a_{x,y}^2 \cdot \sum_{x,y} b_{x,y}^2}}, \qquad (3)$$

where *x, y* are the point coordinates for the images to be compared, *a, b* - color value of a point for the averaged image and image in the moving window accordingly.

The results of testing can be observed in the table 1. The testing has shown, that the method of criterion creation of similarity for two images on the base of circumscribed neural network is best in comparison with a correlation method these two images.

This outcome was obtained because the neural network is invariant to distortions of the form and brightness of the recognized object image, and also requires a smaller amount of learning samples to reach necessary accuracy.

Table 1. Testing results

Size of learning sampling	Accuracy (percent of recognition)			
	Correlation method		Neural network	
	Variant 1	Variant 2	Variant 1	Variant 2
1	**63,1**	**67,4**	**87,0**	**89,0**
3	**72,3**	**73,6**	**96,3**	**97,0**
6	**81,4**	**84,2**	**98,5**	**99,0**

5. CONCLUSION

The new algorithm for processing of gray scale IC images on the basis of the "neocognitron" neural network is represented. The developed algorithms were realized in computer-aided system for LSI circuit video image processing that will be used for reverse engineering (re-design) of VLSI circuits as tools of layout restoring. The new neuron activation algorithm of the network has property of invariance to the image brightness oscillations for the recognized object. The advantage of neural network approach to classification of the images in matching with a correlation method within the framework of learning sampling size decrease and increase of recognition accuracy is shown. The average recognition accuracy is equal to 98,7 %.

REFERENCES

[1] Carpenter G. A., Ross W. D.ART-EIMAGE: A neural network architecture for object recognition by evidence accumulation, *IEEE Transaction on Neural Networks*, Vol. 6, pp. 805-818, 1995

[2] Carpenter G. A., Grossberg S., Lesher G. W. The What-and-Were Filter. A Spatial Imageping Neural Network for Object Recognition and Image Understanding, *Computer Vision and Image Understanding*, Vol. 69, N.1, pp. 1-22, 1998

[3] Fukushima K., Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, *Biol. Cybern.*, Vol. 36, pp. 193-202, 1980

[4] Fukushima K. and Miyake S., Neocognitron: A new algorithm for pattern recognition tolerant of deformations and shifts in position, *Pattern Recognition*, V. 15, pp. 455-469, 1982

[5] Fukushima K. Miyake S. and Ito T., Neocognitron: A neural network model for a mechanism of visual pattern recognition, *IEEE Trans. Syst., Man, Cybern.*, Vol. SMC-13, pp. 826-834, 1983

[6] Fukushima K. and Wake N., Handwritten alphanumeric character recognition by the Neocognitron, *IEEE Trans. on Neural Networks*, Vol. 2, N.3, p. 355-365, 1991

[7] Grossberg S. Mingolla E., Williamson J. Synthetic aperture radar processing by multiple scale neural system for boundary and surface representation, *Neural Networks*, Vol. 8, pp. 1005-1028, 1995

[8] Hans G.H. Traven A Neural Network Approach to Statistical Pattern Classification by "Semiparametric" Estimation of Probability Density Function, *IEEE Transactions on Neural Networks*, Vol. 2, N.3, pp.366-377, 1991.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

SELECTION OF CLOSE CLASSES OBJECTS USING BRIGHTNESS HISTOGRAM

Valery A.PRYTKOV[*], Raouf Kh. SADYKHOV[**]

[*] Belarussian State University of Informatics and Radioelectronics, 6 P.Brovka st., Minsk., BELARUS, phone +375 17 239 80 39, e-mail: *nil36@bsuir.edu.by*
[**] Institute of Engineering Cybernetics, Belarussian State University of Informatics and Radioelectronics, 6 Surganov st., Minsk, BELARUS, ph/fax +375 17 2310982, e-mail: *rsadykhov@gw.bsuir.unibel.by*

Abstract. *The method for selection of close classes objects on a earth surface images is offered. A distinctive feature of a method is usage the histogram analysis with automatic selection of a threshold with a further filtration of the intermediate binary image via distance transforms.*

Key Words. *Earth Surface Images, Histogram Analysis, Distance Transforms, Classification.*

INTRODUCTION

In technology of updating digital maps on earth surface images the large attention is given to automatic and semiautomatic methods for separation of objects with the purpose of their subsequent description in the vector format [1]. The process of manual outlines of a contour takes a lot of time therefore his full or partial automation is rather vital problem.

The simplest method is usage of a histogram of brightness [2] for allocated objects. A main problem on this path is the overlapping of the range of histograms values for close classes objects. The method with automatic selection of a threshold with subsequent proceeding based on the distance transforms is offered.

1. SELECTION OF OBJECTS WITH USAGE OF A BRIGHTNESS HISTOGRAM

The histogram of object brightness has definite range of values. However pixels of a background can be distributed on all range of possible values and enough overlap values range of object (fig. 1, a). The offered method eliminates such uncertainty and limits range of values of the background in particular limits (fig. 1, b). The processing is realized in a semiautomatic mode and the operator selects rectangular object area at first time and then rectangular background area. The histograms of brightness are constructed on these areas.

The problem of recognition thus is transformed in a problem with two classes. If the pixel belongs to range of values for object and does not belong to range of values for background, it will be identified as an object pixel. If the pixel does not belong to range of values for object, it will be identified as a background pixel. The task of belonging take places only when the pixel is in the overlapping range. In the elementary case overlapped range does not include into object (fig. 1, c).

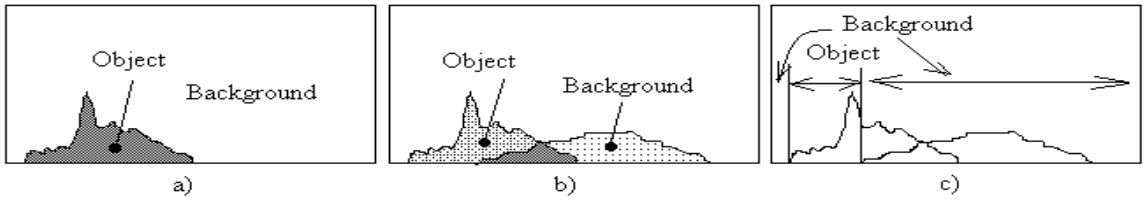


Fig. 1. A method to limit the range of a brightness of background: a) one class, b) two classes, c) recognition.

Let I is the set of pixels for initial image, $B \subseteq I$ – set of pixels of object,, $G \subseteq I$ – set of pixels of a background, and $B^h \subseteq [0,1,\ldots, N]$ – range, which one is received by pixels of object, and $G^h \subseteq [0,1,\ldots, N]$ – range, which one is received by pixels of background, N – the maximum brightness to be restricted by color depth. Then pixels i_{xy} with brightness $\varphi (i_{xy}) \in (B^h/G^h)$ will be belong to object.

Such approach justifies itself if in the range of overlapping there is a great set of background pixels and minor quantity of object pixels. For example on fig.2, a, the input image with rectangles for constructed histogram is presented. In this case the overlapping range there was 14.47 % pixels of object and 96.79 % pixels of a background.

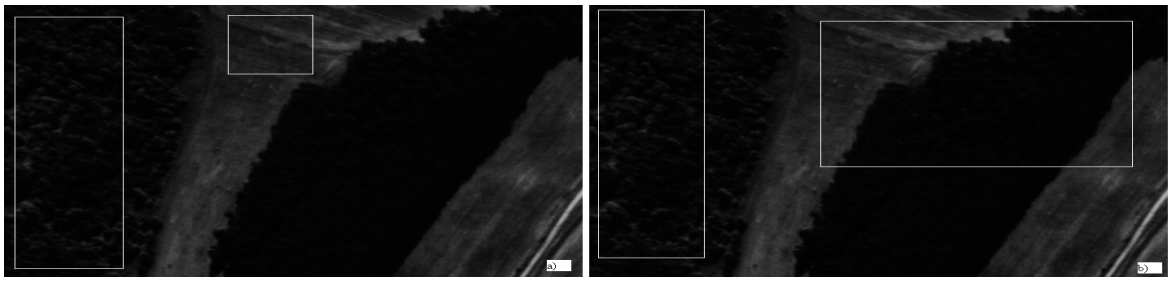


Fig. 2. The initial map with selected rectangles for constructed histograms of object and background :a) forest as object and field as background, b) deciduous forest as object and field with coniferous forest as background

On fig. 3, a, the result of such selection is shown. On the image there are false objects and false background being a forest but not recognized are represented. The linear sizes of false areas are much less than the sizes of object. To calculate linear dimensions distance transforms have been used [1,3,4]. As result the white and black areas with width lower then threshold are inverted. Result of such filtration is presented on fig.3, b.

For a number of cartography problems it is required to recognize objects of close classes, for example, coniferous and deciduous forest (fig.2, b). On a fig. 4 the histograms of object (deciduous forest) and background (coniferous forest and field) are shown. Thus in the over-

lapped range there is 100 % pixels for object and 97.53 % pixels for background. For this reason it is offered to eliminate not all pixels of overlapping range.

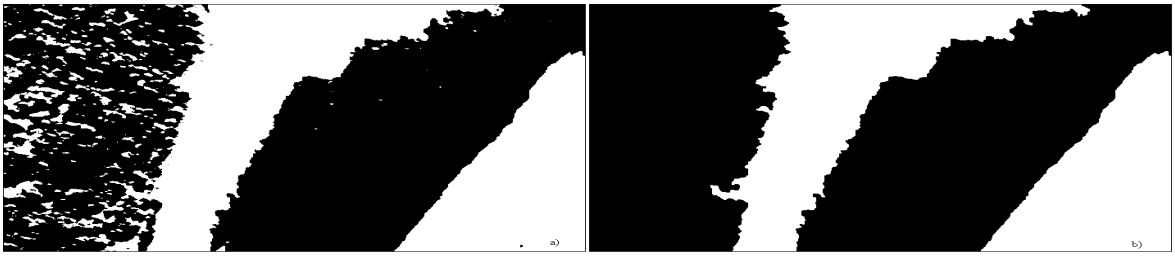


Fig. 3. Automatic selection of a threshold for selection of a forest: a) selection of object, b) filtration by distance transforms.

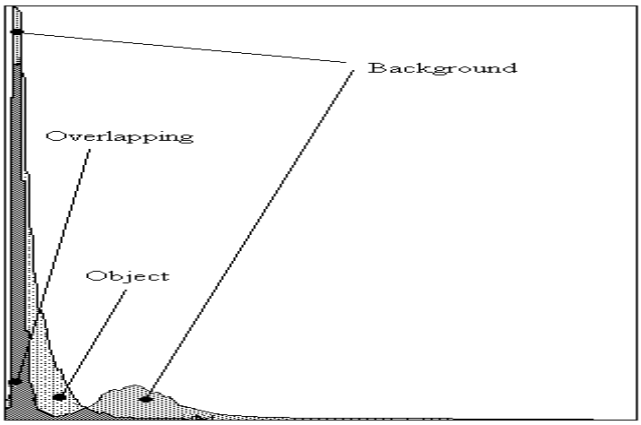


Fig.4. The histograms for object and background

2. SELECTION OF CLOSE CLASS OBJECTS

In the overlapping range the altitude of histograms of object and background can essentially differ. Threshold of this ratio is used for separation object and background. Let b_k is the relative quantity of the pixels of object with brightness k in total set of the pixels of object, and g_k - relative quantity of the pixels of background with brightness k in total number of the pixels of background. Pixels with brightness φ (i_{xy}) will belong to object if $b_{\varphi(i)} / g_{\varphi(i)} > s$, where s – given threshold.

Such approach extends cases with full exception of a background and without exception. So for s=0 all pixels of object will be selected without excluding of a background. Really, $b_{\varphi(i)} / g_{\varphi(i)} > 0 \Leftrightarrow b_{\varphi(i)} > 0$ or $i_{xy} \in B$. For $s \to \infty$ the excluding of a background take place: as $b_{\varphi(i)} / g_{\varphi(i)} \to \infty \Leftrightarrow g_{\varphi(i)} = 0$, $b_{\varphi(i)} > 0$ or $i_{xy} \in B$, $i_{xy} \notin G$. S=1 corresponds to a case when quantity of pixels of object of the given brightness exceeds number of pixels of a background for given brightness: $b_{\varphi(i)} / g_{\varphi(i)} > 1 \Leftrightarrow b_{\varphi(i)} > g_{\varphi(i)}$.

The method is critical to selection of a threshold and requires high proficiency of the operator. The given problem was decided by automatic selection of a threshold. The experiments have shown that the quality selection takes place at usage of quadratic relation:

$$s = \left(\sum_k b_k \Big/ \sum_k g_k \right)^2 \qquad (1)$$

The calculation makes only for pixels of overlapping range. The threshold to be computed on the base of quadratic function is equal to 0.95. The results of selection are shown on fig. 5, a, where two forest areas were dedicated. However linear characteristics of these areas are various. Using filtration algorithm based on distance transforms with specially fitted different thresholds for black and white areas give results shown on a fig. 5, b. Results of the selection of a coniferous forest are shown in a fig. 5, c, d. In overlapping range there is 99.99 % pixels for object and 73.01 % pixels for background and the threshold of separation is equal to 0.389.

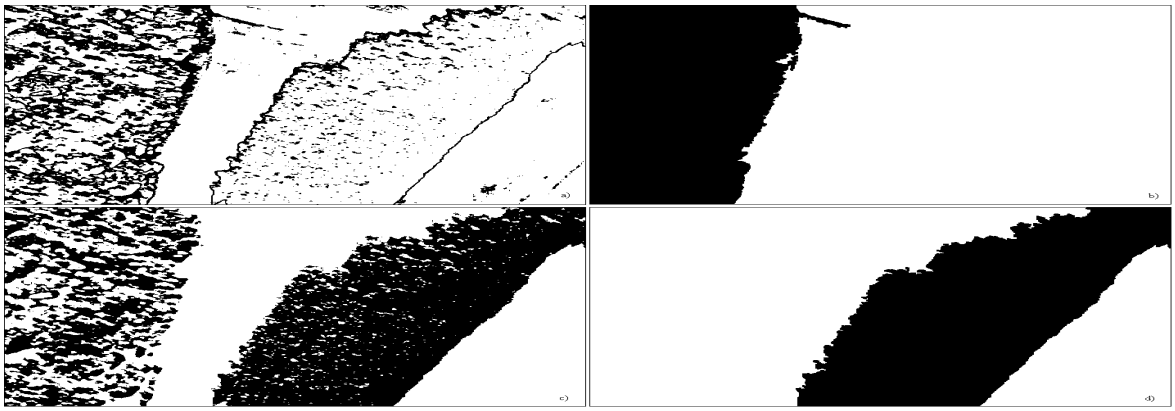


Fig. 5. Selection of the close classes objects: a) selection of a deciduous forest; b) a filtration of a deciduous forest; c) selection of a coniferous forest; d) a filtration of a coniferous forest.

CONCLUSION

The new method of semiautomatic selection of objects on the earth surface images is worked-out. Distinctive features of a method is the selection of a threshold for classification and application distance transforms for a filtration of false areas that gives opportunity for selection of the close class objects.

REFERENCE

[1] Ablameyko S.V., Lagunovsky D.M., “Image processing: technology, methods, applying”, Amalfeja, Minsk, 2000.

[2] Pavlidis T., “Algorithms of machine graphic and image processing”, Moskow, 1986.

[3] Sadykov S.S., Kadyrova G.H., Azimov S.R., “Systems of digital image processing”, Fan, Tashkent, 1988.

[4] Starovoitov V.V. “Local geometrical methods of digital processing and image analysis”, Minsk, 1997.

The International Workshop
on Discrete-Event System Design, DESDes'01,
June 27÷29, 2001; Przytok near Zielona Gora, Poland

THE SYSTEM OF HANDWRITTEN CHARACTERS RECOGNITION ON THE BASIS OF LEGENDRE MOMENTS AND NEURAL NETWORK

Maksim VATKIN[1], Mikhail SELINGER[2]

[1]Institute of Engineering Cybernetics, National Academy of Sciences of Belarus,
6, Surganov str., Minsk, BELARUS, *vatkin@tut.by*
[2]Belarusian State University of Informatics and Radioelectronics,
6, P.Brovki str., Minsk, BELARUS, *selinger@tut.by*

Abstract. *This article analyses the image feature extraction task on the basis of Legendre moments for image recognition problem. Computation algorithm of Legendre moments is presented. A new method for training RBF-neural network is introduced. Classification results for binary images (handwritten Arabic numerals) are presented. On the base of classification results the recommendations for choice of maximal order Legendre moments and various classifiers are given.*

Key Words. *Moment invariants, Legendre moments, Neural networks, Radial basis function*

1. INTRODUCTION

Succession of operations in most of digital image recognition systems can be divided into three stages. First stage is a preprocessing, including thresholding, improving image quality, segmentation and so on. Second – features extraction for avoiding data abundance and reducing its dimension. Third stage is a classification. During this stage class name is joint with unknown image by extracted features analyzes and matching its with representatives of the class, which the classifier has learned at a stage of training. In this article two last stages of digital image recognition are presented.

2. LEGENDRE MOMENTS

In feature extraction task considerable attention for methods that use moment functions is given. Moment invariant properties are investigated since sixties [4]. There are invariant on shifts, scaling and rotating of source object. During research time various types of moment functions were introduced, and fast computation algorithms for different types of moments were created. This part of article describes Legendre moments using for handwritten character (Arabic numerals) informative feature extraction.

Two-dimensional Legendre moments for image intensive function *f(x,y)* defines as [6, 8]:

$$L_{kl} = \frac{(2k+1)(2l+1)}{4} \int_{-1}^{1}\int_{-1}^{1} P_k(x)P_l(y)f(x,y)dxdy, \quad (1)$$

where *f(x,y)* – picture element with coordinates *(x,y)*;

$$P_0(x)=1; \quad P_1(x)=x; \quad P_k(x)=[(2k-1)xP_{k-1}(x)-(k-1)P_{k-2}(x)]/k - \quad (2)$$

Legendre polynomial by power *k*, *k>1*;

l≥0 и *k≥0* defines the order of moments.

Since definition area of Legendre polynomials is *-1≤x≤1*, then definition area of two-dimensional Legendre moments is unit square, so a rectangle image of *N×M* pixels with intensity function *f(i,j), 1≤i≤N, 1≤j≤M* will have to be scaled the region *1≤x,y≤1*, and image center of gravity must be located in the coordinate system origin. For this:

– source image center of gravity (i_c,j_c) is computed;

– distance *D* from the center of gravity to the farthest from it point of image is determined according to equation

$$\forall(i,j): D = \max\{|i_c - i|,|j_c - j|\}; \quad (3)$$

– scaling of image is performed according to

$$(x,y) = \left(\frac{j-j_c}{D}, \frac{i_c-i}{D}\right). \quad (4)$$

Legendre moments to maximum order *MAX_ORDER* can be computed by pseudo-code:

```
for k:=0 to MAX_ORDER
  for l:=0 to k
    L(k-l,l):=0
    for i:=1 to N
      for j:=1 to M
        x:=(j - jc)/D
        y:=(ic - i)/D
        L(k-l,l) := L(k-l,l) + Pk-l(x)*Pl(y)*f(i,j)
      end
    end
    L(k-l,l) := L(k-l,l)*(2k-2l+1)*(2l+1)/(N-1)/(M-1)
  end
end
```

Legendre polynomials $P_k(x)$ forms full orthogonal basis inside unit circle, so source image may be reconstructed from the finite number of Legendre moments as follows:

$$f(x,y) \cong \sum_k \sum_l L_{kl}P_k(x)P_l(y). \quad (5)$$

Fig.1 shows source binary and halftone images and reconstructed images using various number of Legendre moments.

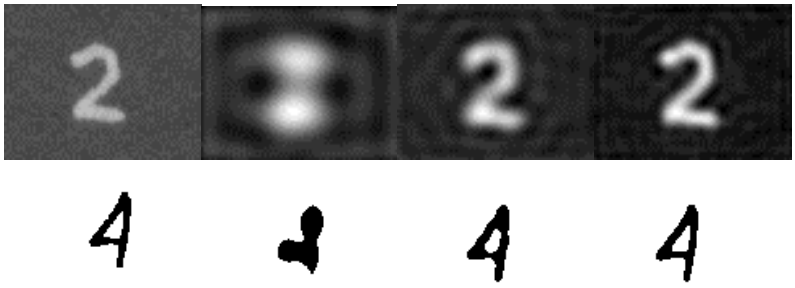


Fig.1. Source and reconstructed images using Legendre moments with maximum order 20, 40 and 60

3. CLASSIFIER ON THE BASIS OF THE RBF-NEURAL NETWORK.

In the tasks of classification the large attention is given to construction of classifiers on the basis of neural networks. Radial basis function (RBF) neural network is two-layer neural network offered by Moody and Darken in 1989 [5] (fig. 2).

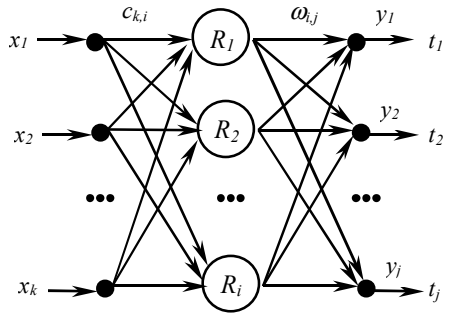


Fig. 2. The RBF- neural network architecture

The RBF-networks represent multilayer neural networks with RBF-neuron layer, which activation function correspond to the radial basic function

$$R(D_i) = \exp\left(-\frac{D_i^2}{2 \cdot \sigma_i^2}\right), \tag{6}$$

were $D_i = |\vec{x} - \vec{c}_i|$ is distance between an entry pattern $\vec{x}$ and *i*–th cluster of the radial basic function $\vec{c}_i$, σ - width of a cluster.

The RBF-neuron weight coefficients are associated with cluster of the radial basic function. Thus, the output data of RBF-layer represents a vector of closeness measures of entry pattern to all RBF-clusters.

The subsequent layers of such networks usually evaluate a linear combination of these functions.

$$y_j = \sum_i w_{i,j} R_i \ . \tag{7}$$

Key aspect for the RBF-network training is the possibility of layer-by-layer training, that result in two-phase training algorithm: a RBF-layer training and perceptron layer training.

3.1. RBF-layer training

The RBF-layer training consists of two tasks: definition of necessary amount of RBF-neurons, and their weight coefficients setting. The training is produced by the following rules:

If in RBF-layer there are no such neurons that $D_i < \sigma$ or amount of neurons is equal to 0, than it is necessary to add a new neuron initializing its weights by training pattern vector value.

Else modification *i*-th neuron weights for which $\min_{\forall i}(D_i)$ is performed

$$\vec{c}_i(t+1) = \vec{c}_i(t) + \frac{1}{t+1} \cdot (\vec{x}(t+1) - \vec{c}_i(t)), \tag{8}$$

were $\vec{x}, \vec{c}_i$ are training vector and cluster vector accordingly, *t*-amount of additions.

3.2. Perceptron layer training

The perceptron layer training is made by a gradient descent method with the purpose of the error function minimization in weight coefficient space $w_{i,j}$:

$$Err = \frac{1}{2} \cdot \sum_n \sum_j \left(\sum_i w_{i,j} \cdot R_i^n - t_j^n \right)^2 \to 0, \tag{9}$$

were *n=[1..N]* is amount of learning images, t_j^n - target value of an *j-th* output for a training pattern *n*.

4. CLASSIFICATION RESULTS

A part of binary image database used in experiments is presented in Fig. 3. Database consists of 10 classes of images (10 Arabic numerals). Each class consists of 225 objects. 175 objects from each class are used for training and 50 – for recognition.

For classification the RBF neural network classifier and minimal distance classifier [2] were used. Classification results for different maximum order of Legendre moments is presented in Table 1.

Table 1. Binary images classification results

MAX_ORDER	***Recognition percent, %***	
	Minimal distance classifier	***Neural network***
5	76,4	92
10	90,6	98,2
15	91,6	98,7
20	88,4	98,7
25	85,2	98,5

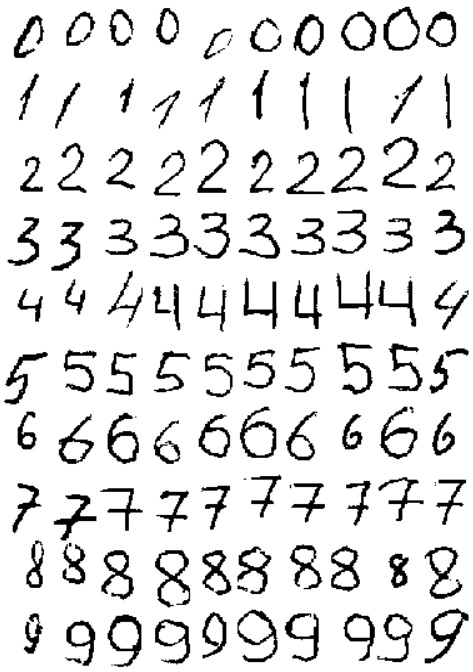


Fig3. Part of binary images database

5. CONCLUSION

In this paper the problem of handwritten characters (Arabic numerals) recognition was examined. Legendre moments properties as a classifiers features were investigated. Also characteristics of new classifier based on neural network in comparison with minimal distance classifier were tested. On classification results the next conclusions can be done:

- during classification performing it is no purpose to use moment functions with maximum order above 20.

- classification must be performed using neural network classifiers.

REFERENCES

[1] Christopher M. Bishop., “Neural networks for pattern recognition”, *Oxford: Clarendon press*, pp.165-191, 1997

[2] Richard O. Duda, Peter E. Hart, *Pattern classification and scene analysis*, Mir, Moscow, 1976

[3] Foor, W., “Adaptive optical radial basis function neural network for handwritten digit recognition”, *Applied optics*, v.34, p.7545-7555, 1995

[4] Hu M. K., "Visual Pattern Recognition by Moments Invariants", *IRE Trans. Inform. Theory*, vol. 8, pp. 179-187, 1962

[5] Moody, J. and C.J. Darken, “Fast learning in networks of locally tuned processing units”, *Neural Computations*, vol. 1, No. 2, pp. 281-294, 1989

[6] Mukundan R., Ramakrishnan K. R., "Fast Computation of Legendre and Zernike Moments", *Pattern Recognition*, vol. 28, No. 9, pp. 1433-1442, 1995

[7] Powell, M.J.D., “Radial basis function for multivariable interpolation: a review”, *In J.C. Mason and M.G. Cox (Eds), Algorithms for Approximation*, pp. 143-167, 1987

[8] Teh C. H., Chin R. T., "On Image Analysis by the Method of Moments", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, pp. 496-513, 1988

Author index

Design Exploration Tools

Software Simulation

Hardware Accelerations

Design Optimizers and Debuggers

IP Core & Design Services

To become a champion, you only need to make this one move…go to

www.aldec.com

UNIX-Linux-NT Tools for ASIC and FPGA Applications

Aldec, Inc.
2230 Corporate Circle
Henderson, NV 89014
Tel. (702) 990-4400

Aldec-ADT Sp. z o.o.
40-118 Katowice
ul. Widok 23
tel. (032) 201-0880

Aldec-Polska Sp. z o.o.
30-252 Krakow
ul. A. Szyszko-Bohusza 19
tel. (012) 429-7036

Computer Center

VADIM

the biggest IT company in Ziemia Lubuska

(western province in Poland)

www.vadim.com.pl

VADIM Center
65-426 Zielona Góra
ul. Kupiecka 28
tel. (+48 68) 327 0705

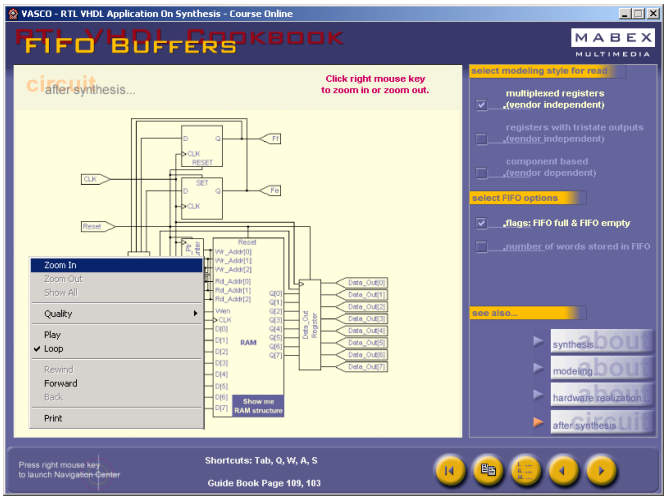
VADIM Sp. z o.o.
02-544 Warszawa
ul. Madalińskiego 57
tel. (+48 22) 849 2412

VADIM- Hurt
65-066 Zielona Góra
ul. Żeromskiego 27
tel. (+48 68) 324 6368

VASCO™
RTL HDL Solutions

for engineers by engineers...

MABEX-Multimedia proudly presents: VASCO HDL - multimedia interactive courses for VHDL and Verilog users.



MABEX services:

- Sale of highly successful VASCO HDL training software (over 1.000 copies sold in the year 2000)
- Production of interactive Computer-Based and Web-Based Training applications,
- Multimedia presentations,
- Internet portals,
- Applications with data exchange with the internet database systems, and more!

MABEX
MULTIMEDIA

UL. MORELOWA 57/17
65-434 ZIELONA GÓRA
POLAND

TEL. +48 68 454 88 91
+48 68 320 86 18
FAX +48 68 454 88 90

WWW.MABEX.COM INFO@MABEX.COM

ZIELONA GÓRA