n and:

POLISH ACADEMY OF SCIENCE
COMMITTEE FOR ELECTRONICS AND TELECOMMUNICATIONS

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY

KWARTALNIK ELEKTRONIKI I TELEKOMUNIKACJI

VOLUME 55 – No 2

WARSAW 2009

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY
Quarterly of Polish Academy of Sciences

Address of Editorial Office
Nowowiejska Street 15/19, 00-665 Warsaw, Poland
Institute of Telecommunications, room 484

Email: etq@tele.pw.edu.pl

Editor-on-duty
Mondays and Wednesdays from 2pm to 4pm, tel. +48 22 234 77 37

Ark. Wyd. 22,00 Ark. druk. 17, 62	Podpisano do druku w sierpniu 2009 r.
Papier offset, kl. III 80 g. B-1	Druk ukończono we wrześniu 2009 r.

Publishing
Warszawska Drukarnia Naukowa PAN
00-656 Warszawa, ul. Śniadeckich 8
Tel./fax 628-87-77

IMPORTANT MESSAGE FOR THE AUTHORS

The Editorial Board during their meeting on the 18th of January 2006 authorized the Editorial Office to introduce the following changes:

1. PUBLISHING THE ARTICLES IN ENGLISH LANGUAGE ONLY

Starting from No 1'2007 of E&T Quarterly, all the articles will be published in English only.

Each article prepared in English must be supplemented with a thorough summary in Polish (e.g. 2 pages), including the essential formulas, tables, diagrams etc. The Polish summary must be written on a separate page. The articles will be reviewed and their English correctness will be verified.

2. COVERING THE PUBLISHING EXPENSES BY AUTHORS

Starting from No'2007 of E&T Quarterly, a principle of publishing articles against payment is introduced, assuming non-profit making editorial office. According to the principle the authors or institutions employing them, will have to cover the expenses in amount of 760 PLN for each publishing sheet. The above amount will be used to supplement the limited financial means received from PAS for publishing; particularly to increase the capacity of next E&T Quaterly volumes and verify the English correctness of articles. It is neccessary to increase the capacity of E&T Quarterly volumes due to growing number of received articles, which delays their publishing.

In case of authors written request to accelerate the publishing of an article, the fee will amount to 1500 PLN for each publishing sheet.

In justifiable cases presented in writing, the editorial staff may decide to relieve authors from basic payment, either partially or fully. The payment must be made by bank transfer into account of Warsaw Science Publishers The account number: Bank Zachodni WBK S.A. Warszawa Nr 94 1090 1883 0000 0001 0588 2816 with additional note: "For Electronics and Telecommunications Quarterly".

Editors

E
Elekt
T
mitte
of PA
origi
dely
optoc
T
as yo
T
critic
bran
them
and
A
ciali
The
Scie
T
com
Mor
E
distr
the a
more
T
auth
publ
in ec
T
edito

Dear Authors,

Electronics and Telecommunications Quarterly continues tradition of the "Rozprawy Elektrotechniczne" quarterly established 55 years ago.

The E&T Quarterly is a periodical of Electronics and Telecommunications Committee of Polish Academy of Science. It is published by Warsaw Science Publishers of PAS. The Quarterly is a scientific periodical where articles presenting the results of original, theoretical, experimental and reviewed works are published. They consider widely recognised aspects of modern electronics, telecommunications, microelectronics, optoelectronics, radioelectronics and medical electronics.

The authors are outstanding scientists, well-known experienced specialists as well as young researchers — mainly candidates for a doctor's degree.

The articles present original approaches to problems, interesting research results, critical estimation of theories and methods, discuss current state or progress in a given branch of technology and describe development prospects. The manner of writing mathematical parts of articles complies with IEC (International Electronics Commision) and ISO (International Organization of Standardization) standards.

All the articles published in E&T Quarterly are reviewed by known, domestic specialists which ensures that the publications are recognized as author's scientific output. The publishing of research work results completed within the framework *of Ministry of Science and Higher Education* GRANTs meets one of the requirements for those works.

The periodical is distributed among all those who deal with electronics and telecommunications in national scientific centres, as well as in numeral foreign institutions. Moreover it is subscribed by many specialists and libraries.

Each author is entitled to free of charge 20 copies of article, which allows for easier distribution to persons and institutions domestic and abroad, individually chosen by the author. The fact that the articles are published in English makes the quarterly even more accessible.

The articles received are published within half a year if the cooperation between author and the editorial staff is efficient. Instructions for authors concerning the form of publications are included in every volume of the quarterly; they may also be obtained in editorial office.

The articles may be submitted to the editorial office personally or by post; the editorial office address is shown on editorial page in each volume.

Editors

G

fir
ga
do
in
sc
th

th
Co
St
th
A
in
fo
fir
cu
H
Fl
fir

al
Sy
ca
fo
10
H
Pr
bo

sc
-
-
-

Guest Editors Preface

Most papers in this issue are devoted to research in the area of design methods for finite state machines and microprogram control units targeted at field-programmable gate arrays and complex programmable logic devices. The selection of papers was done during the scientific seminar "Digital Control Units Design" which took place in March, 6, 2009, at the University of Zielona Góra. The seminar was confined to scientific visit of Professor Samary Baranov – one of the most experienced experts in the field of control circuits design.

Professor Samary Baranov received M.Sc. degree in Computer Engineering from the Electrotechnical University in St. Petersburg, Ph.D. degree and D.Sc. degree in Computer Engineering from the Institute of Precision Mechanics and Optics, also in St. Petersburg. In 1994, he became a professor in the Computer System Department and the Head of the Center for VLSI Design at the Holon Academic Institute of Technology. At the same time, he worked as a consultant for several high-tech companies in Israel, including National Semiconductors, Fortress, and M-Systems. In February 2001, he founded the North American Institute of Computer Systems (NAICS) in Toronto – the first training center in Canada to offer an advanced and intensive 250 hour post-graduate curriculum "Electronic Hardware Design" devoted to teaching Design Methodology, Hardware Description Languages (VHDL and Verilog), EDA Tools and ASIC and FPGA Design. More than 250 students (bachelors, masters and PhDs) successfully finished this course in 2001 – 2005.

Professor Samary Baranov is a member of program committees of many international conferences, including: "New Frontiers of Information Technology"; "Engineering Systems and Software for the next decade"; "Field Programmable Logic and Applications"; and the EUROMICRO Conferences on Digital System Design; a reviewer for Design Automation Conference (DAC 2001 – 2008). He is an author of 10 books, 10 textbooks for students and more than 70 papers in Russian, French and English. His book "Logic Synthesis for Control Automata" was published by Kluwer Academic Publishers in 1994. His latest book "Logic and System design of Digital Systems" has been published by Tallinn University of Technology in 2008.

Following extended articles have been prepared based on presentations from the scientific seminar "Digital Control Units Design":

- The paper by S. Baranov discusses some procedures of high level synthesis, implemented in the experimental EDA tool. These tools are based on ASM transformations and special algorithms for data paths and control units design.
- The paper by M. Adamski and M. Węgrzyn is concentrated on behavioral specification of Reconfigurable Logic Controller programs given initially as Petri nets and rewritten later in Hardware Description Languages.
- T. Łuba, G. Borowik and A. Kraśniewski focus on finite state machine synthesis including logic optimization techniques, the technology mapping techniques, and

the techniques that provide the resulting circuits with concurrent error detection capability.

- In the paper by L. Titarenko and J. Bieganowski two methods oriented on implementation of compositional microprogram control unit with PAL macrocells and embedded memory blocks of CPLD are presented. The first method is based on introduction of additional microinstructions, whereas the second one is based on expansion of the format of microinstructions.
- R. Wiśniewski and A. Barkalov focus on the structural decomposition of control units. Eight methods of compositional microprogram control units are described and compared. The aim of all the proposed solutions is to reduce the number of logic blocks of the targeted programmable device.
- New methods on synthesis and implementation of Mealy finite state machines in FPGAs are the subject of the paper by A. Bukowiec and A. Barkalov. The Synthesis methods presented are based on architectural decomposition of a logic circuit of an FSM and multiple encoding of its some parameters (states or collections of microoperations).
- The paper by M. Chmiel, E. Hrynkiewicz and A. Milik presents a modified idea of program execution in PLCs, where the event-driven execution is proposed instead of serial cyclic execution of a control program. The proposed method can be implemented either as software modification or as hardware accelerated solution.
- In the paper by D. Kania, A. Milik, J. Kulisz, A. Opara and R. Czerwiński the original synthesis strategies oriented towards PAL-based devices are presented. These synthesis methods are aimed at minimization of required chip area or propagation delay (by reducing number of levels).
- A. Barkalov, L. Titarenko and S. Chmielewski deal with the method of combined state assignment which targets on decreasing the amount of hardware exploitation by combinational part of Moore finite-state-machine. It is based on the existence of pseudo-equivalent states and a wide fan-in of PAL macrocells.
- The last paper by P. Szotkowski and M. Rawski proposes a heuristic algorithm for input selection and a new, clique-based algorithm for the construction of the crucial decomposition blankets. This method yields better results than the currently widespread, two-step approaches based on state encoding and mapping of the resulting binary function.

As Guest Editors of these ten papers, we would like to thank all the authors who submitted papers for this special issue.

Marian Adamski
Alexander Barkalov
University of Zielona Góra

CONTENTS

H
Algo
tion,
optin
on th
allov
an o
for d
High
1. C
2. D
3. C

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 123–156

High level synthesis in EDA tool "Abelite"

SAMARY BARANOV

Holon Institute of Technology, Dept. of Computer Sc., Holon, Israel
samary@012.net.il

Received 2009.01.21
Authorized 2009.03.18

The paper presents the first description of methods and algorithms realized in experimental EDA tool Abelite. High level synthesis, implemented in this tool is based on Algorithmic State machine (ASM) transformations (composition, minimization, extraction, etc.), special algorithms for Data Path and Control Unit design and a very fast optimizing synthesis of FSM and combinational circuits with hardly any constraints on the number of inputs, outputs and states. Design tools supporting this methodology allow very fast to implement, check and estimate many possible design versions, to find an optimized decision of the design problem and to simplify the verification problem for digital systems.

Keywords: EDA tools, High level synthesis, Logic synthesis, VHDL

1. INTRODUCTION

High level synthesis, implemented in the experimental EDA tool Abelite is based on Algorithmic State machine (ASM) transformations (composition, minimization, extraction, etc.), special algorithms for Data Path and Control Unit design and a very fast optimizing synthesis of FSM and combinational circuits with hardly any constraints on the number of inputs, outputs and states. Design tools supporting this methodology allow very fast to implement, check and estimate many possible design versions, to find an optimized decision of the design problem and to simplify the verification problem for digital systems.
High level synthesis contains three stages:
1. Construction of Combined Functional ASM and FSM;
2. Data Path Design;
3. Control Unit Design.

Since whole paper is devoted only to EDA tool Abelite we didn't include references to other works including works of the author of this paper.

2. CONSTRUCTION OF COMBINED FUNCTIONAL ASM AND FSM

Let us suppose that the task is to design a digital system. Problem orientation regarding this system is nonessential – it can be a processor, a robot, a controller, etc. If the system is rather complicated, it is possible to pick out some subbehaviors (modes) in its behavior. For a processor it can be an instruction or a set of instructions that can be described together; for a mobile robot – its different modes (cruise, follow, avoid, escape etc.). We also suppose that any digital system is usually regarded as a composition of a *Control unit* and an *Operational unit* (*Data path*). In a processor, for example, a data path contains such regular blocks as memory, registers, ALU, counters, coders, encoders, multiplexers, demultiplexers, etc. A control unit produces a sequence of control signals that force an implementation of microoperations in a data path.

Now we will discuss the main steps of the first stage (Fig. 1) in more details. As an example, we will consider the design of a simple processor with four instructions – *ao* (arithmetic operation), *lod* (operation load), *bun* (branch unconditional) and operation *out*.

Step1. Drawing Functional ASMs in ASM Creator (see box 1 in Fig.1, ***ASM Creator***). At this step a designer draws separate ASMs for each subbehavior (for each operation or a group of operations) in *ASM Creator*. It is really important that an ASM may contain any number of *generalized operators*. Each of such operators is an ASM itself and it will be automatically inserted in the combined ASM at the fourth stage. Moreover, there are no restrictions on the number of such generalized operators in an ASM and on the number of included levels – each of such operators can contain any number of generalized operators itself. Our four operations are presented in Fig. 2. The generalized operators for our processor are drawn in Fig. 3.

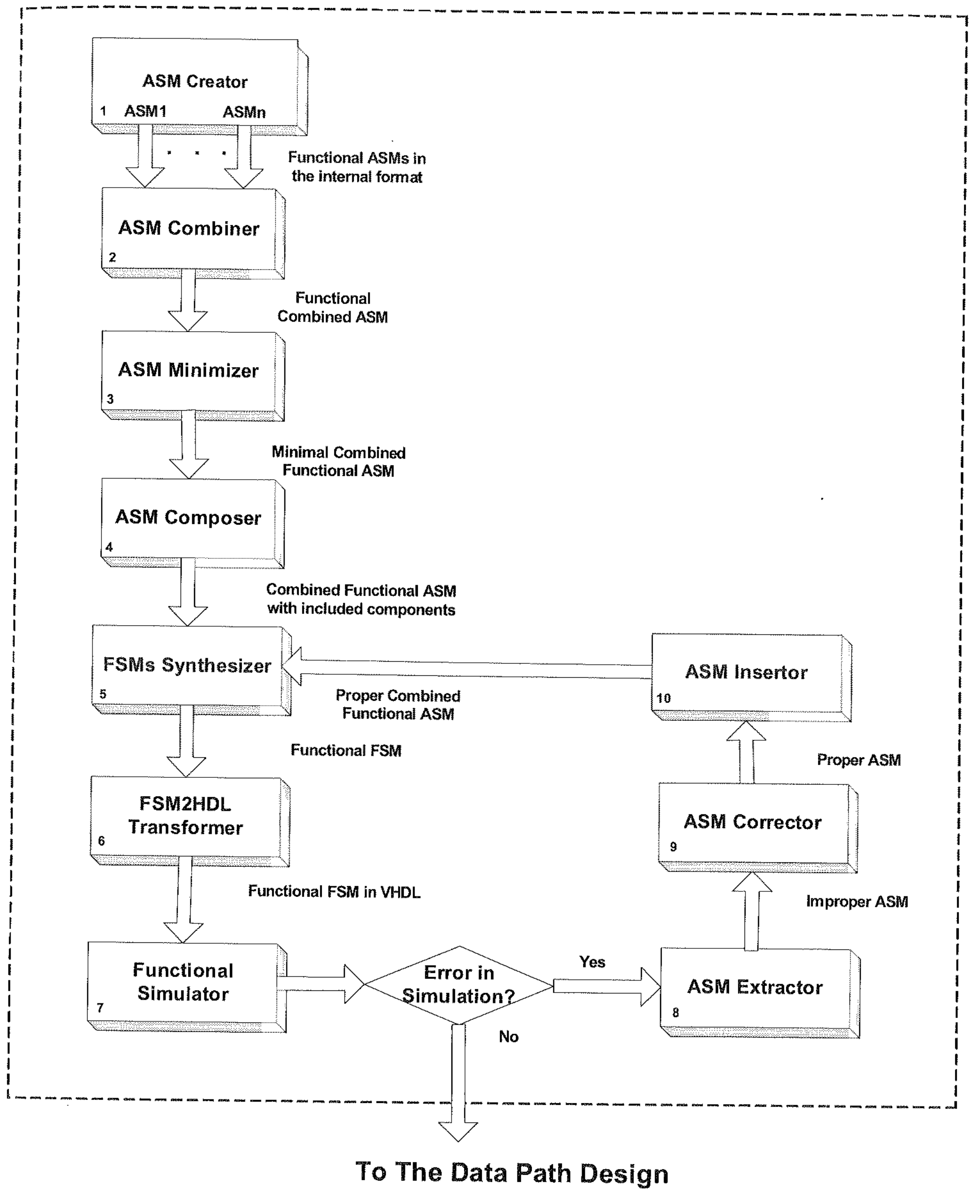


Fig. 1. Construction of Combined Functional ASM and FSM

Step2. Combining of several Functional ASMs into one Combined Functional ASM (box 2 ***ASM Combiner***). After constructing separate ASMs we combine them into one combined functional ASM still containing generalized operators. During ASM combining we minimize the number of operator vertices in the combined ASM.

If several ASMs contain the same operator vertex, there will be only one such operator vertex in the combined ASM.

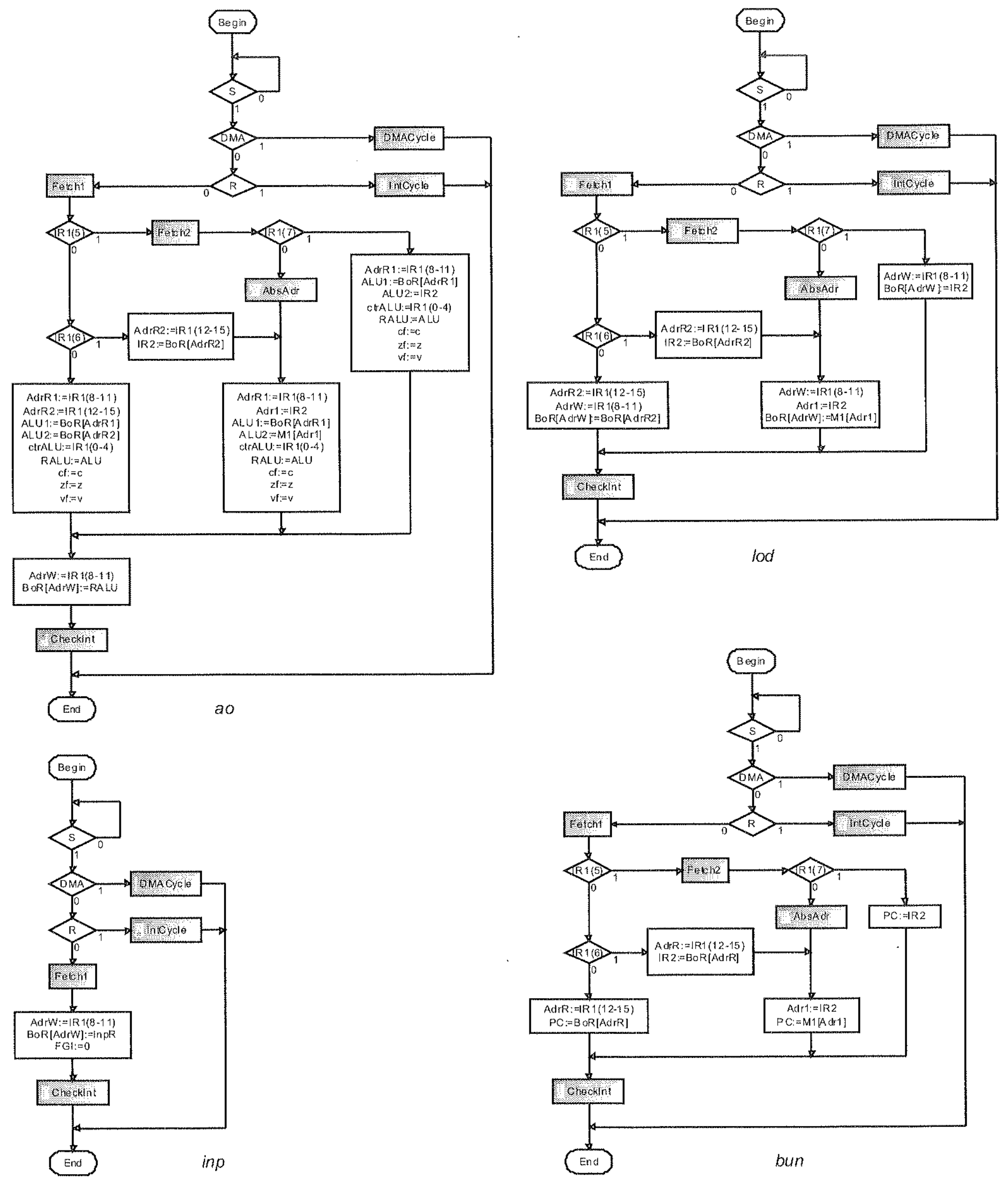


Fig. 2. ASMs *ao*, *lod*, *bun* and *out*

Step3. Minimization of combined Functional ASM (box 3 ***ASM Minimizer***). This procedure minimizes the number of conditional vertices in the combined functional

ASM. Such minimization allows us to reduce dramatically the number of vertices in the ASM (sometimes for two or three times) and to reduce the complexity of logic circuits at the stage of logic design. The combined and minimized functional ASM Funcm is presented in Fig. 4. It contains only 35 vertices, whereas there are 65 vertices in the four initial ASMs.

We can use *ASM Minimizer* after the second step as well. It means that a designer can concentrate on the behavior description and does not think about minimization of the number of vertices in ASMs in his drawings or in VHDL files. After compiling, *ASM Minimizer* will minimize each separate ASM.

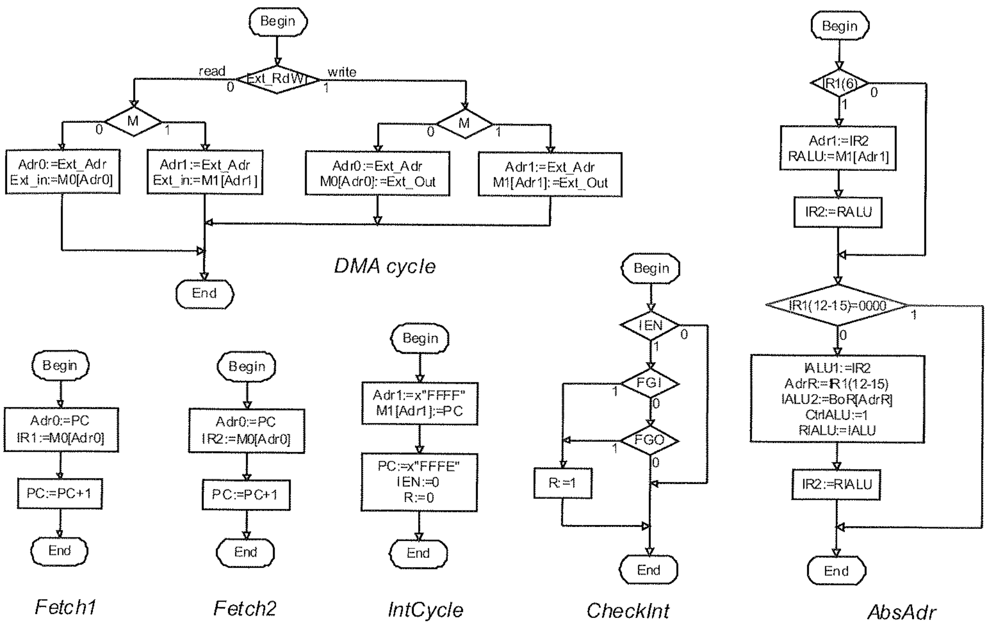


Fig. 3. Generalized operator in our example

Step 4. Including of generalized operators (box 4 ***ASM Composer***). At this stage, generalized operators constructed at the first stage are included into the minimized ASM constructed at the previous stage. It is the last stage of the functional ASM design. This ASM can be presented as two-connected list (file Funcmi.gsa):

```
0      begin  20    0          35     Y11    17    0
1      Y1     10    0          36     Y13    17    0
2      Y12    35    0          37     Y14    17    0
3      x10    40    41         38     Y15    17    0
4      Y2     1     0          39     Y16    17    0
5      Y3     26    0          40     x9     37    36
6      Y4     25    0          41     x9     38    39
7      Y5     6     0          42     Y17    45    0
8      x1     42    46         43     Y18    44    0
9      Y22    47    0          44     Y19    23    0
10     x14    50    17         45     Y20    46    0
11     Y24    51    0          46     x11    23    43
12     Y6     10    0          47     Y21    29    0
13     Y7     10    0          48     Y23    17    0
14     Y8     10    0          49     x12    48    17
15     Y9     10    0          50     x13    48    49
16     Y10    10    0          51     Y21    34    0
    .    .    .
```

The second file produced by Abelite after inserting generalized operators (file Funcmi.txt) contains microinstructions, microoperations and logical conditions which a designer wrote in ASMs in Fig. 2 and 3:

```
Micro Instructions:
Y1 = y1 y2
Y2 = y3 y4 y5 y6 y7 y8 y9 y10
Y3 = y11 y12
Y4 = y13
Y5 = y14 y15
Y6 = y16
Y7 = y14 y17
Y8 = y11 y18
Y9 = y1 y19
Y10 = y3 y20 y21
Y11 = y22 y23 y24
Y12 = y25 y26
Y13 = y27 y28
Y14 = y29 y30
Y15 = y29 y31
Y16 = y27 y32
Y17 = y14 y33
Y18 = y34 y11 y35 y36 y37
Y19 = y38
Y20 = y39
Y21 = y40
Y22 = y41 y42
Y23 = y43
Y24 = y41 y44

Micro Operations :
y1   : AdrW:=IR1(8-11)
y2   : BoR[AdrW]:=RALU
y3   : AdrR:=IR1(8-11)
y4   : ALU1:=BoR[AdrR]
y5   : ALU2:=IR2
y6   : ctrALU:=IR1(0-4)
y7   : RALU:=ALU
y8   : cf:=c
y9   : vf:=v
y10   : zf:=z
y11   : AdrR:=IR1(12-15)
y12   : IR2:=BoR[AdrR]
y13   : IR2:=BR
y14   : Adr1:=IR2
y15   : BR:=M1[Adr1]
y16   : PC:=IR2
y17   : PC:=M1[Adr1]
y18   : PC:=BoR[AdrR]
y19   : BoR[AdrW]:=IR2
y20   : OutR:=BoR[AdrR]
y21   : FGO:=0
y22   : PC:=x"FFFE"
y23   : IEN:=0
y24   : R:=0
y25   : Adr1:=x"FFFF"
y26   : M1[Adr1]:=PC
y27   : Adr0:=Ext_Adr
y28   : M0[Adr0]:=Ext_Out
y29   : Adr1:=Ext_Adr
y30   : M1[Adr1]:=Ext_Out
y31   : Ext_in:=M1[Adr1]
y32   : Ext_in:=M0[Adr0]
y33   : RALU:=M1[Adr1]
y34   : IALU1:=IR2
y35   : IALU2:=BoR[AdrR]
y36   : CtrIALU:=1
y37   : RIALU:=IALU
y38   : IR2:=RIALU
y39   : IR2:=RALU
y40   : PC:=PC+1
y41   : Adr0:=PC
y42   : IR2:=M0[Adr0]
y43   : R:=1
y44   : IR1:=M0[Adr0]

Logical Conditions :
File     Draw
x1   : IR1(6)
x2   : IR1(5)
x3   : IR1(7)
x4   : R
x5   : DMA
x6   : S
x7   : IR1(0)
x8   : IR1(1)
x9   : M
x10   : Ext_RdWr
x11   : IR1(12-15)=0000
x12   : FGO
x13   : FGI
x14   : IEN
```

Only for illustration we give this ASM as a graph in Fig. 5. Designer shouldn't draw any ASMs, except in Fig. 2 and 3, any more - Abelite continues to work only with files Funcmi.gsa and Funcmi.txt.

We would like to underline here that in the description at the level of a functional ASM we don't have Data Path and each unit in such a functional ASM is presented as a variable. For example, in microoperation *IR2 := M0[Adr0]* the word of memory M0 with address Adr0 should be sent to instruction register IR2, but we don not know yet, and wouldn not like to know, how these units are connected and what signals must come from the control unit to Data Path to implement this transfer. Really, a Data Path does not exist yet and our goal at the second stage (Data Path Construction, see Fig. 8) is to construct a Data Path formally using only the combined functional ASM.

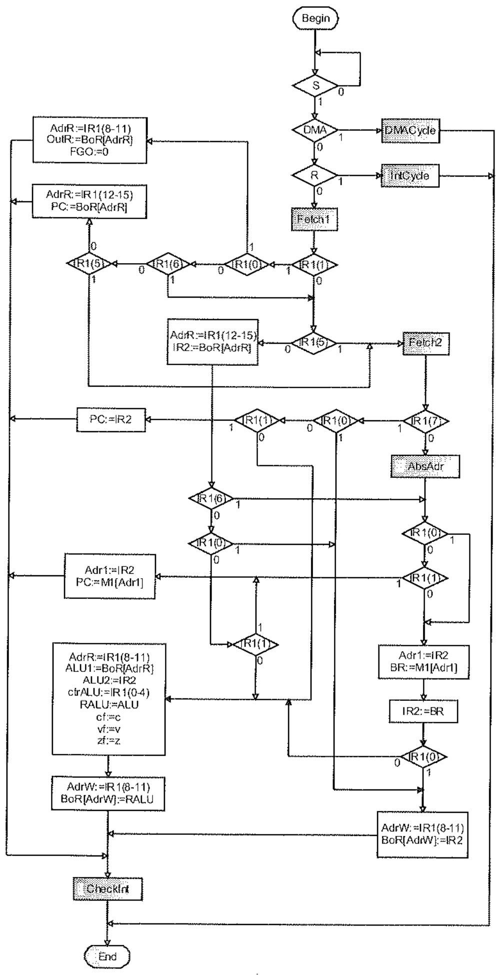


Fig. 4. Combined and minimized functional ASM

Step5. Synthesis FSM from combined Functional ASM (box 5 ***FSM Synthesizer***). This procedure constructs 24 various types of minimized Finite State Machine tables for Mealy, Moore and their combined model with or without state assignment (*log* or *one-hot*). FSM Mealy implementing ASM in Fig. 5 is presented in Fig. 6. In this FSM, $x_1, \ldots, x_{14}$ correspond to logical variables written in conditional vertices, while $y_1, \ldots, y_{44}$ – to microoperations written in operator vertices of ASM in Fig. 5.

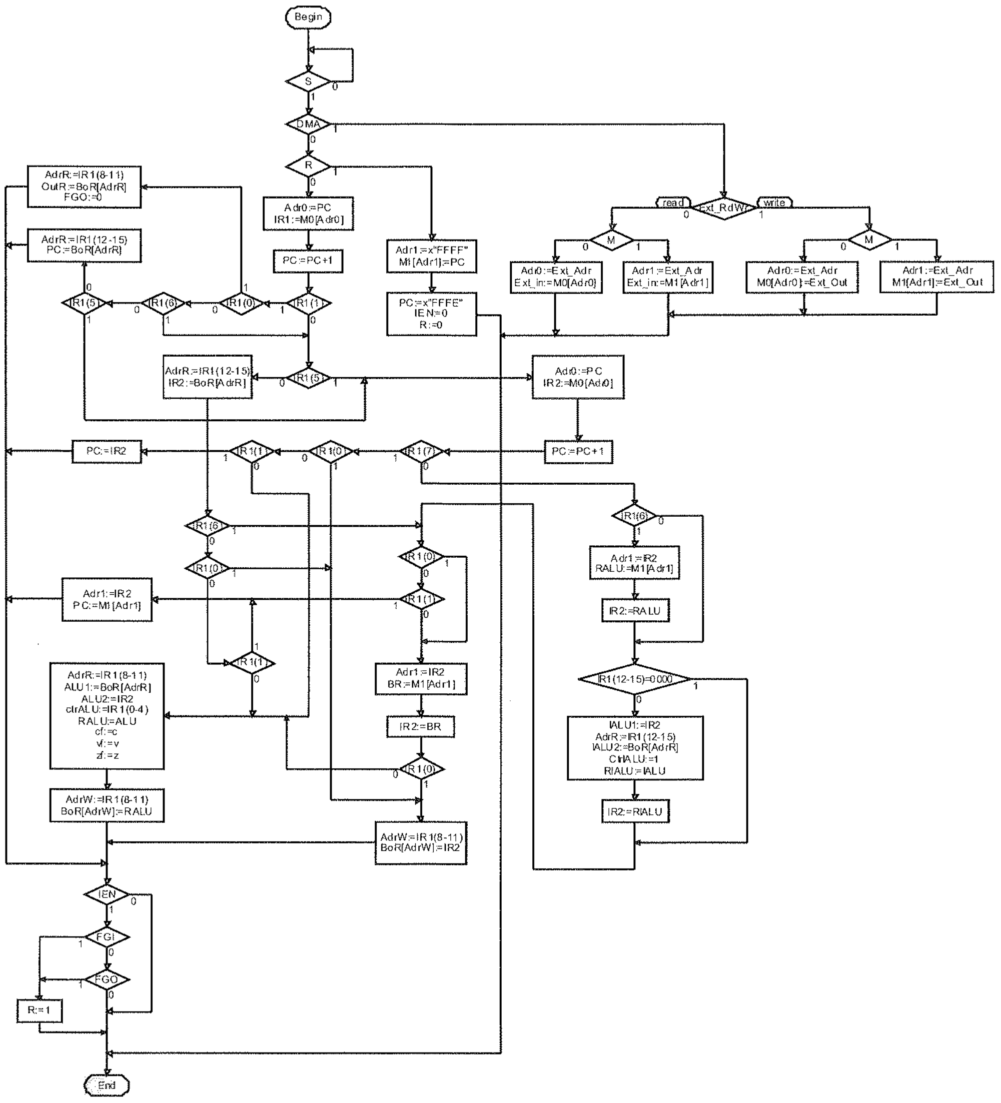


Fig. 5. Combined functional ASM with inserted generalized operators

a1	a1	x6*x5*x10*x9	y29y30	1
a1	a1	x6*x5*x10*~x9	y27y28	2
a1	a1	x6*x5*~x10*x9	y29y31	3
a1	a1	x6*x5*~x10*~x9	y27y32	4
a1	a3	x6*~x5*x4	y25y26	5
a1	a9	x6*~x5*~x4	y41y44	6
a1	a1	~x6	--	7
a2	a1	x14*x13	y43	8
a2	a1	x14*~x13*x12	y43	9
a2	a1	x14*~x13*~x12	--	10
a2	a1	~x14	--	11
a3	a1	1	y22y23y24	12
a4	a2	1	y1y2	13
a5	a7	x1*x7	y14y15	14
a5	a2	x1*~x7*x8	y14y17	15
a5	a7	x1*~x7*~x8	y14y15	16
a5	a2	~x1*x7	y1y19	17
a5	a2	~x1*~x7*x8	y14y17	18
a5	a4	~x1*~x7*~x8	y3y4y5y6y7y8y9y10	19
a6	a2	x7	y1y19	20
a6	a4	~x7	y3y4y5y6y7y8y9y10	21
a7	a6	1	y13	22
a8	a14	1	y40	23
a9	a15	1	y40	24
a10	a13	1	y39	25
a11	a12	1	y38	26
a12	a7	x7	y14y15	27
a12	a2	~x7*x8	y14y17	28
a12	a7	~x7*~x8	y14y15	29
a13	a7	x11*x7	y14y15	30
a13	a2	x11*~x7*x8	y14y17	31
a13	a7	x11*~x7*~x8	y14y15	32
a13	a11	~x11	y34y11y35y36y37	33
a14	a2	x3*x7	y1y19	34
a14	a2	x3*~x7*x8	y16	35
a14	a4	x3*~x7*~x8	y3y4y5y6y7y8y9y10	36
a14	a10	~x3*x1	y14y33	37
a14	a7	~x3*~x1*x11*x7	y14y15	38
a14	a2	~x3*~x1*x11*~x7*x8	y14y17	39
a14	a7	~x3*~x1*x11*~x7*~x8	y14y15	40
a14	a11	~x3*~x1*~x11	y34y11y35y36y37	41
a15	a2	x8*x7	y3y20y21	42
a15	a8	x8*~x7*x1*x2	y41y42	43
a15	a5	x8*~x7*x1*~x2	y11y12	44
a15	a8	x8*~x7*~x1*x2	y41y42	45
a15	a2	x8*~x7*~x1*~x2	y11y18	46
a15	a8	~x8*x2	y41y42	47
a15	a5	~x8*~x2	y11y12	48

Fig. 6. Virtual Functional FSM

Step6. Construction VHDL (Verilog) code for the Functional FSM (box 6 ***FSM2HDL Transformers***). A VHDL code for FSM in Fig. 6 is presented in Fig. 7. This FSM is not a Control Unit, it is a virtual FSM presenting the processor behavior at the functional level. We will use this FSM for simulation of our processor.

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.my_package.all;

entity FUNCMI is
       generic (
              AdrMem     : integer := 16;     -- the length of memory address
              WordMem0   : integer := 16;     -- the length of M0 word
              WordMem1   : integer := 16;     -- the length of M1 word
              AdrBoR     : integer :=  4;     -- the length of BoR address
              WordBoR    : integer := 16      -- the length of BoR word
       );
       port (
              clk        : in std_logic;
              rst        : in std_logic;
              S          : in std_logic;
              Ext_RdWr   : in std_logic;
              DMA        : in std_logic;
              Ext_Adr    : in std_logic_vector (0 to 15);
              Ext_out    : in std_logic_vector (0 to 15);
              FGI_Set    : in std_logic;
              InpR       : in std_logic_vector (0 to 15);
              FGO_Set    : in std_logic;
              M          : in std_logic;

              Ext_in     : out std_logic_vector (0 to 15);
              OutR       : out std_logic_vector (0 to 15);
              Idle       : out std_logic
       );
end FUNCMI;

architecture ARC_FUNCMI of FUNCMI is

type states_FUNCMI is (
       s1, s2, s3, s4, s5, s6, s7, s8, s9,
       s10, s11, s12, s13, s14, s15
       );
signal current_FUNCMI : states_FUNCMI;

type ram0_type is array (0 to 2**AdrMem - 1) of std_logic_vector (0 to WordMem0 - 1);
signal M0 : ram0_type;

type ram1_type is array (0 to 2**AdrMem - 1) of std_logic_vector (0 to WordMem1 - 1);
signal M1 : ram1_type;

type bor_type is array (0 to 2**AdrBoR - 1) of std_logic_vector (0 to WordBoR - 1);
signal BoR : bor_type;

       signal PC            : std_logic_vector (0 to 15);
       signal IR1           : std_logic_vector (0 to 15);
       signal IR2           : std_logic_vector (0 to 15);
       signal RALU          : std_logic_vector (0 to 15);
       signal cf,zf,vf      : std_logic;
       signal R             : std_logic;
       signal IEN           : std_logic;
       signal FGI           : std_logic;
       signal FGO           : std_logic;
       signal RIALU         : std_logic_vector (0 to 15);
       signal BR            : std_logic_vector (0 to 15);
       signal BAC           : std_logic_vector (0 to 3);
       signal MAC           : std_logic_vector (0 to 15);

begin
```

```
process (clk , rst)

          variable Adr0, Adr1     : std_logic_vector (0 to AdrMem - 1);
          variable ALU            : std_logic_vector (0 to 15);
          variable ALU1, ALU2     : std_logic_vector (0 to 15);
          variable ctrALU         : std_logic_vector (0 to 4);
          variable c, z, v        : std_logic;
          variable IALU           : std_logic_vector (0 to 15);
          variable IALU1, IALU2   : std_logic_vector (0 to 15);
          variable ctrIALU        : std_logic;
          variable AdrR, AdrW : std_logic_vector (0 to AdrBoR - 1);

     procedure proc_FUNCMI is
     begin
     case current_FUNCMI is
        when s1 =>
           if ( S and DMA and Ext_RdWr and M ) = '1' then
              Adr1 := Ext_Adr;
              M1(to_integer(unsigned(Adr1))) <= Ext_Out;
              current_FUNCMI <= s1;
              Idle <= '1';

           elsif ( S and DMA and Ext_RdWr and not M ) = '1' then
              Adr0 := Ext_Adr;
              M0(to_integer(unsigned(Adr0))) <= Ext_Out;
              current_FUNCMI <= s1;
              Idle <= '1';

           elsif ( S and DMA and not Ext_RdWr and M ) = '1' then
              Adr1 := Ext_Adr;
              Ext_in <= M1(to_integer(unsigned(Adr1)));
              current_FUNCMI <= s1;
              Idle <= '1';

           elsif ( S and DMA and not Ext_RdWr and not M ) = '1' then
              Adr0 := Ext_Adr;
              Ext_in <= M0(to_integer(unsigned(Adr0)));
              current_FUNCMI <= s1;
              Idle <= '1';

           elsif ( S and not DMA and R ) = '1' then
              Adr1 := x"FFFF";
              M1(to_integer(unsigned(Adr1))) <= PC;
              current_FUNCMI <= s3;

           elsif ( S and not DMA and not R ) = '1' then
              Adr0 := PC;
              IR1 <= M0(to_integer(unsigned(Adr0)));
              current_FUNCMI <= s9;

           elsif ( not S ) = '1' then
              current_FUNCMI <= s1;
              Idle <= '1';

           else
              current_FUNCMI <= s1;
              Idle <= '1';
           end if;

        when s2 =>
           if ( IEN and FGI ) = '1' then
              R <= '1';
              current_FUNCMI <= s1;
              Idle <= '1';
        elsif ( IEN and not FGI and FGO ) = '1' then
           R <= '1';
           current_FUNCMI <= s1;
           Idle <= '1';

        elsif ( IEN and not FGI and not FGO ) = '1' then
           current_FUNCMI <= s1;
           Idle <= '1';

        elsif ( not IEN ) = '1' then
           current_FUNCMI <= s1;
           Idle <= '1';

        else
           current_FUNCMI <= s2;
        end if;
                 .   .   .

     when s15 =>
        if ( IR1(1) and IR1(0) ) = '1' then
           AdrR := IR1(8 to 11);
           OutR <= BoR(to_integer(unsigned(AdrR)));
           FGO <= '0';
           current_FUNCMI <= s2;

        elsif ( IR1(1) and not IR1(0) and IR1(6) and IR1(5) ) = '1' then
           Adr0 := PC;
           IR2 <= M0(to_integer(unsigned(Adr0)));
           current_FUNCMI <= s8;

        elsif ( IR1(1) and not IR1(0) and IR1(6) and not IR1(5) ) = '1' then
           AdrR := IR1(12 to 15);
           IR2 <= BoR(to_integer(unsigned(AdrR)));
           current_FUNCMI <= s5;

        elsif ( IR1(1) and not IR1(0) and not IR1(6) and IR1(5) ) = '1' then
           Adr0 := PC;
           IR2 <= M0(to_integer(unsigned(Adr0)));
           current_FUNCMI <= s8;

        elsif ( IR1(1) and not IR1(0) and not IR1(6) and not IR1(5) ) = '1' then
           AdrR := IR1(12 to 15);
           PC <= BoR(to_integer(unsigned(AdrR)));
           current_FUNCMI <= s2;

        elsif ( not IR1(1) and IR1(5) ) = '1' then
           Adr0 := PC;
           IR2 <= M0(to_integer(unsigned(Adr0)));
           current_FUNCMI <= s8;

        elsif ( not IR1(1) and not IR1(5) ) = '1' then
           AdrR := IR1(12 to 15);
           IR2 <= BoR(to_integer(unsigned(AdrR)));
           current_FUNCMI <= s5;

        else
           current_FUNCMI <= s15;
        end if;

  end case;
  end proc_FUNCMI;
  begin
```

```
                if (rst = '1') then
                        PC              <= x"0000";
                        cf              <= '0';
                        zf              <= '0';
                        vf              <= '0';
                        R               <= '0';
                        FGI             <= '0';
                        FGO             <= '0';
                        IEN             <= '0';
                        c               := '0';
                        z               := '0';
                        v               := '0';
                        Ext_in  <= x"0000";
                        OutR    <= x"0000";
                        BR              <= (others => '0');
                        BAC             <= (others => '0');
                        MAC             <= (others => '0');

                        for i in BoR'range loop
                                BoR(i) <= (others => '0');
                        end loop;

                        current_FUNCMI <= s1;
                        Idle <= '1';
                elsif (clk'event and clk = '1') then
                        if FGI_Set = '1' then
                                FGI <= '1';
                        end if;
                        if FGO_Set = '1' then
                                FGO <= '1';
                        end if;

                        Idle <= '0';
                        proc_FUNCMI;
                end if;
        end process;

end ARC_FUNCMI;
```

Fig. 7. Virtual Functional FSM

Step7. Functional Simulation (box 7 ***Functional Simulator***). We can use Functional FSM in VHDL from Fig. 7 for a ***functional simulation*** (not a register transfer level simulation – it is possible to use the same test bench later when we construct a Data Path and a Control unit corresponding to this Data Path).
To construct a test bench we should write the test as an assembly program. The test bench for functional simulation is generated by the special program – Functional Simulation Generator.

Step8. Extraction one or several ASMs from Combined ASM (box 9 ***ASM Extractor***). If we detected error in ASM during the simulation we can extract ASM with error (*Step 8*, see box 8 *ASM extractor*), repair it (*Step 9*, see box 9 ***ASM corrector***) and return corrected ASM into combined Functional ASM (*Step 10*, see box 10, ***ASM inserter***).
In Abelite, we have special program "Check ASM equivalence", that verifies the equivalence of two ASMs. That permits designer to check each step of ASM transformer.

3. DATA PATH DESIGN

At this stage (Fig. 8), we use a sequence of programs for automatic design of Data Path. Input for this stage – Combined Functional ASM (Fig. 5) constructed at the previous stage. In our design we use the common model in which any digital system is regarded as a composition of *Control unit* and *Operational unit* (*Data Path*) – see Fig. 9. Data path contains such regular blocks as memory, registers, ALU, counters, coders, encoders, multiplexers, demultiplexers etc. A control unit produces a sequence of control signals that force implementation of microoperations in data path.

Very often designer includes cloud (non-regular) circuits in data path as well. In Fig. 10 we have a fragment of data path with two registers *R1*, *R2* and a cloud circuit. Suppose that in the digital system there are transfers from *R1* to *R2* at different times with different conditions. Designer often constructs a cloud circuit to realize some Boolean function, and the output of this circuit is the signal for the transfer. So, this circuit defines when and under which logic conditions the transfer information from *R1* to *R2* takes place.

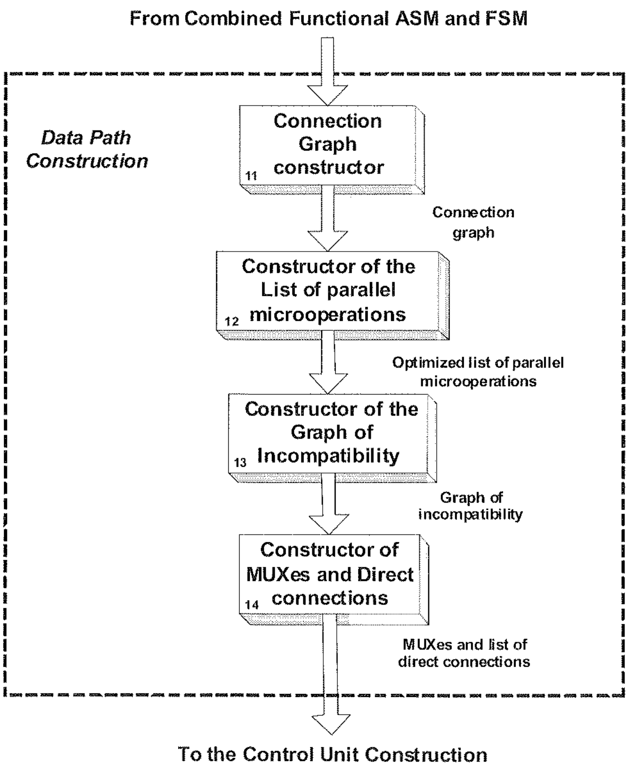

From Combined Functional ASM and FSM
Data Path Construction
Connection Graph constructor
11
Connection graph
Constructor of the List of parallel microoperations
12
Optimized list of parallel microoperations
Constructor of the Graph of Incompatibility
13
Graph of incompatibility
Constructor of MUXes and Direct connections
14
MUXes and list of direct connections
To the Control Unit Construction

Fig. 8. Stage 2 – Data Path design

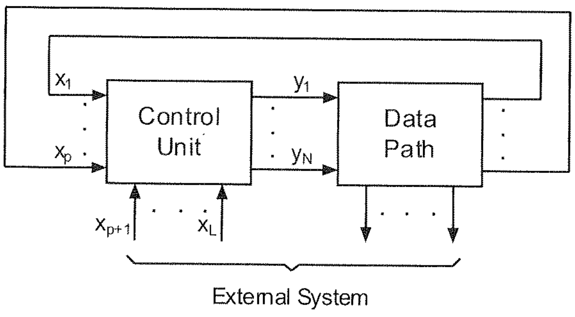


Fig. 9. Digital system as a composition of Control unit and Data path

One of the main concepts in our design methodology is the construction of "*naked data path*". Naked data path doesn't contain any cloud circuits, only standard regular units with their inputs and outputs. Such units can be predesigned or even taken from libraries. We leave all check-ups of conditions to control unit. We can afford this because we know how to design very complicated FSM with hardly any constraints on their size, that is, the number of inputs, outputs and states (see Section V). We will try to show that such design and its verification are very simple. Moreover, we will formalize a design of the digital system with naked data path.

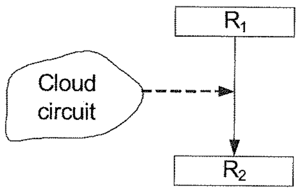


Fig. 10. Element of Data path with a cloud circuit

Data path and the following Control unit design are convenient to illustrate by Table 1. To fill this table, we copy each microinstruction from functional ASM (Fig. 5) into this table. If some microinstruction appears several times in this ASM we write it several times in Table 1. For example, microinstruction *PC* := *PC* + *1* is written in four vertices of ASM, so it is written four times in Table 1. The order of writing microinstructions in this table is unimportant.

Table 1

Microinstructions and microoperations for functional and structural levels

Functional ASM				Structural ASM		
Micro instr.	Functional microoperations			Structural microoperations	Minimized structural microoperations	
Y1	y1	AdrW:=IR1(8-11)	4	ctr_mux0:=100	ctr_mux0[0] <= 1	y1
	y2	BoR[AdrW]:=RALU	16	bor_en:=1	bor_en <= 1	y2
Y2	y3	AdrR:=IR1(8-11)	4	ctr_mux2:=1	ctr_mux1[2] <= 1	y3
	y4	ALU1:=BoR[AdrR]	16	ctr_mux1:=0011	ctr_mux1 [2] <= 1	y4
	y5	ALU2:=IR2	16	ralu_en:=1	ctr_mux1 [3] <= 1	y5
	y6	ctrALU:=IR1 (0-4)	5	cf_en:=1	ralu_en <= 1	y6
	y7	RALU:=ALU	16	vf_en:=1	cf_en <= 1	y7
	y8	cf:=c	1	zf_en:=1	vf_en <= 1	y8
	y9	vf:=v	1		zf_en <=1	y9
	y10	zf:=z	1			
y3	y11 y12	AdrR:=IR1(12-15) IR2:=BoR[AdrR]	4 16	ctr_mux2:=0 ctr_mux1:=0010 ir2_en:=1	ctr_mux1 [2] <= 1 ir2_en <= 1	y4 y10
Y4	y13	IR2:=BR	16	ctr_mux1:=0101 ir2_en:=1	ctr_mux1[1] <= 1 ctr_mux1[3] <= 1 ir2_en <= 1	y11 y5 y10
Y5	y14 y15	Adr1:=IR2 BR:=M1[Adr1]	16 16	ctr_mux0:=001 br_en:=1 m1_rdwr:=0	ctr_mux0[2] <= 1 br_en <= 1	y12 y13
Y6	y16	PC:=IR2	16	ctr_mux1:=1000 pc_en:=1	ctr_mux1[0] <= 1 pc_en <= 1	y14 y15
Y7	y14 y17	Adr1:=IR2 PC:=M1[Adr1]	16 16	ctr_mux0:=001 ctr_mux1:=0000 pc_en:=1 m1_rdwr:=0	ctr_mux0[2] <= 1 pc_en <=1	y12 y15
Y8	y11 y18	AdrR:=IR1(12-15) PC:=BoR[AdrR]	4 16	ctr_mux2:=0 ctr_mux1:=0010 pc_en:=1	ctr_mux1[2] <= 1 pc_en <= 1	y4 y15
Y9	y1 y19	AdrW:=IR1(8-11) BoR[AdrW]:=IR2	4 16	ctr_mux0:=001 bor_en:=1	ctr_mux0[2] <= 1 bor_en <= 1	y12 y2
Y10	y3 y20 y21	AdrR:=IR1(8-11) OutR:=BoR[AdrR] FGO:=0	4 16 0	ctr_mux2:=1 outr_en:=1 fgo_reset:=1	ctr_mux2 <= 1 outr_en <= 1 fgo_reset <= 1	y3 y16 y17

cd. Table 1

Y11	y22 y23 y24	PC:=x"FFFE" IEN:=0 R:=0	16 0 0	ctr_mux1:=1100 pc_en:=1 ien_reset:=1 r_reset:=1	ctr_mux1[0] <= 1 ctr_mux1[1] <= 1 pc_en <= 1 ien_reset <= 1 r_reset <= 1 y19	y14 y11 y15 y18
Y12	y25 y26	Adr1:=x"FFFF" M1[Adr1]:=PC	16 16	ctr_mux0:=011 ctr_mux1:=1001 m1_rdwr:=1	ctr_mux0[1] <= 1 ctr_mux0[2] <= 1 ctr_mux1[0] <= 1 ctr_mux1[3] <= 1 m1_rdwr <= 1	y20 y12 y14 y5 y21
Y13	y27 y28	Adr0:=Ext_Adr M0[Adr0]:=Ext_Out	16 16	ctr_mux0:=000 m0_rdwr:=1	m0_rdwr <= 1	y22
Y14	y29 y30	Adr1:=Ext_Adr M1[Adr1]:=Ext_Out	16 16	ctr_mux0:=000 ctr_mux1:=0100 m1_rdwr:=1	ctr_mux1[1] <= 1 m1_rdwr <= 1	y11 y21
Y15	y29 y31	Adr1:=Ext_Adr Ext_in:=M1[Adr1]	16 16	ctr_mux0:=000 ctr_mux1:=0000 m1_rdwr:=0		
Y16	y27 y32	Adr0:=Ext_Adr Ext_in:=M0[Adr0]	16 16	ctr_mux0:=000 ctr_mux1:=0001 m0_rdwr:=0	ctr_mux1[3] <= 1	y5
Y17	y14 y33	Adr1:=IR2 RALU:=M1[Adr1]	16 16	ctr_mux0:=001 ctr_mux1:=0000 ralu_en:=1 m1_rdwr:=0	ctr_mux0[2] <= 1 ralu_en <= 1	y12 y6
Y18	y34 y11 y35 y36 y37	IALU1:=IR2 AdrR:=IR1(12-15) IALU2:=BoR[AdrR] CtrIALU:=1 RIALU:=IALU	16 4 16 0 16	ctr_mux2:=0 ctrialu:=1 rialu_en:=1	ctrialu <= 1 rialu_en <= 1	y23 y24
Y19	y38	IR2:=RIALU	16	ctr_mux1:=1010 ir2_en:=1	ctr_mux1[0] <= 1 ctr_mux1[2] <= 1 ir2_en ¡= 1	y14 y4 y10
Y20	y39	IR2:=RALU	16	ctr_mux1:=0110 ir2_en:=1	ctr_mux1[1] <= 1 ctr_mux1[2] <= 1 ir2_en <= 1	y11 y4 y10
Y21	y40	PC:=PC+1	0	pc_count:=1	pc_count <= 1	y25
Y22	y41 y42	Adr0:=PC IR2:=M0[Adr0]	16 16	ctr_mux0:=010 ctr_mux1:=0001 ir2_en:=1 m0_rdwr:=0	ctr_mux0[1] <= 1 ctr_mux1[3] <= 1 ir2_en <= 1	y20 y5 y10
Y23	y43	R:=1	0	r_set:=1	r_set <= 1	y26
Y24	y41 y44	Adr0:=PC IR1:=M0[Adr0]	16 16	ctr_mux0:=010 ir1_en:=1 m0_rdwr:=0	ctr_mux0[1] <= 1 ir1_en <= 1	y20 y27

Step11. Construction of a Connection Graph from the Functional ASM designed at the step 4 (box 11 ***Connection Graph Constructor***). This graph and the following list of parallel microoperations should be constructed for each length of transfers (column

4 in Table 1). Such a graph contains the list of sources and targets for each component of an operational unit and some metrics that will be used in the optimization of Data Path. The connection graph as a text list for 16-bit transfers constructed by Abelite is presented in Fig. 11.

```
weight : sources      targets    : weight
     1 : ALU              RALU   : 1

     1 : BR                IR2   : 1

     2 : BoR[AdrR]        ALU1   : 0
                         IALU2   : 0
                           IR2   : 1
                          OutR   : 0
                            PC   : 1

     4 : Ext_Adr          Adr0   : 2
                          Adr1   : 2

     1 : Ext_Out      M0[Adr0]   : 0
                      M1[Adr1]   : 1

     0 : IALU            RIALU   : 0

     5 : IR2              ALU2   : 0
                          Adr1   : 3
                     BoR[AdrW]   : 1
                         IALU1   : 0
                            PC   : 1

     2 : M0[Adr0]       Ext_in   : 1
                           IR1   : 0
                           IR2   : 1

     3 : M1[Adr1]           BR   : 0
                        Ext_in   : 1
                            PC   : 1
                          RALU   : 1

     3 : PC               Adr0   : 2
                      M1[Adr1]   : 1

     2 : RALU        BoR[AdrW]   : 1
                           IR2   : 1

     1 : RIALU             IR2   : 1

     1 : x"FFFE"            PC   : 1

     1 : x"FFFF"          Adr1   : 1
```

Fig. 11. Connection Graph

Step12. Construction of the optimized list of parallel microoperations from the Functional ASM designed at the step 4 (see box 12). Such a list contains microoperations which should be implemented in parallel. It is important to increase the speed of the designed system (Fig. 12 for 16-bit transfers). In this list, we include microinstructions, containing two or more microoperations, marked by 16 in the fourth column of Table 2. We remind that if several microoperations are in one microinstruction, they are implemented concurrently (at the same clock).

We can compress the list of parallel microoperations (Fig. 13), if we remove microoperations corresponding to direct connections in the connection graph (microoperations with zero target weights (right column in Fig. 11). At the next steps of the design of Data path we will use only these two lists – the connection graph and the compressed list of parallel microoperations.

```
Y2      : y4     ALU1:=BoR[AdrR]
          y5     ALU2:=IR2
          y7     RALU:=ALU

Y5      : y14    Adr1:=IR2
          y15    BR:=M1[Adr1]

Y7      : y14    Adr1:=IR2
          y17    PC:=M1[Adr1]

Y12     : y25    Adr1:=x"FFFF"
          y26    M1[Adr1]:=PC

Y13     : y27    Adr0:=Ext_Adr
          y28    M0[Adr0]:=Ext_Out

Y14     : y29    Adr1:=Ext_Adr
          y30    M1[Adr1]:=Ext_Out

Y15     : y29    Adr1:=Ext_Adr
          y31    Ext_in:=M1[Adr1]

Y16     : y27    Adr0:=Ext_Adr
          y32    Ext_in:=M0[Adr0]

Y17     : y14    Adr1:=IR2
          y33    RALU:=M1[Adr1]

Y18     : y34    IALU1:=IR2
          y35    IALU2:=BoR[AdrR]
          y37    RIALU:=IALU

Y22     : y41    Adr0:=PC
          y42    IR2:=M0[Adr0]

Y24     : y41    Adr0:=PC
          y44    IR1:=M0[Adr0]
```

Fig. 12. Parallel Microoperations before considering direct connections

Step13. Construction of a Graph of Incompatibility from the Connection Graph and the optimized list of parallel microoperations designed at the step 11 and 12 (box 13). Vertices of this graph are all targets of the connection graph with nonzero weights written in the last column. We connect two vertices (targets) by edge (line) if these two targets are together in the same microinstruction in the set of parallel microoperations. For example, we connect *adr1* and *pc* by edge because *Adr1* and *PC* are the targets in microinstruction Y_7 in Fig. 13.

```
Y7          : y14     Adr1:=IR2
              y17     PC:=M1[Adr1]

Y12         : y25     Adr1:=x"FFFF"
              y26     M1[Adr1]:=PC

Y14         : y29     Adr1:=Ext_Adr
              y30     M1[Adr1]:=Ext_Out

Y15         : y29     Adr1:=Ext_Adr
              y31     Ext_in:=M1[Adr1]

Y16         : y27     Adr0:=Ext_Adr
              y32     Ext_in:=M0[Adr0]

Y17         : y14     Adr1:=IR2
              y33     RALU:=M1[Adr1]

Y22         : y41     Adr0:=PC
              y42     IR2:=M0[Adr0]
```

Fig. 13. Compressed list of parallel microoperations

If two vertices (targets) are connected by edge in this graph we cannot pass information to these targets through the same MUX because these targets are written together in some set of concurrent microoperations with *different* sources. For example, target *adr0* cannot be acquired from the same MUX with *ext_in* and *ir2* since *adr0* is connected with these vertices by arcs. However, *adr0* can be acquired from the same MUX with *adr1*, *m1*, *ralu* or *pc* – *adr0* is not connected with them in the graph of incompatibility.

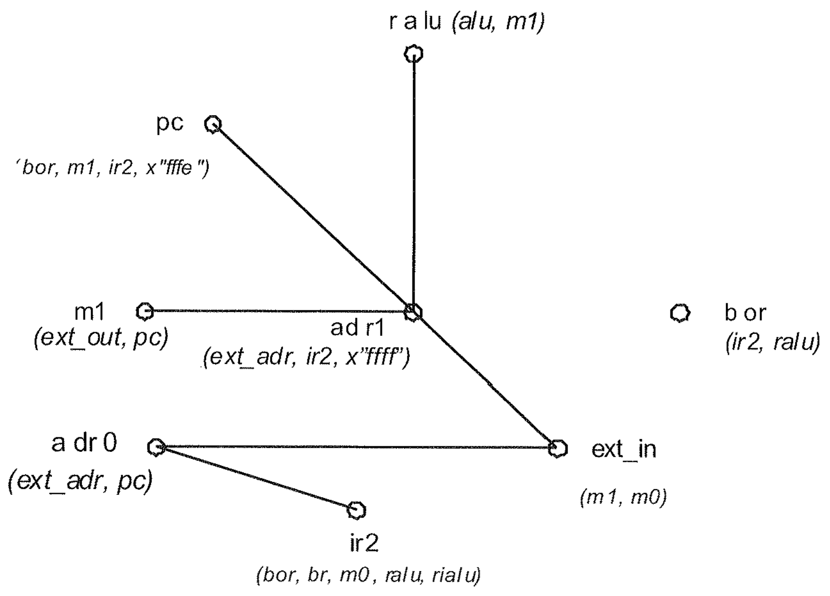


Fig. 14. Graph of incompatibility for 16-bit transfers

Step14. Construction MUXes by coloring the Graph of incompatibility and ***Construction the list of direct connections*** from the Connection graph (box14). To find the minimal number of MUXes in our design we must color this graph with a minimal number of colors. The targets (vertices) colored by the same colors will be received from the same MUXes and the number of MUXes will be equal to the number of colors.

Table 2

Coloring process

Vertices	Forbidden vertices	Colors
adr1	ext_in, ralu, pc, m1	mux0
ext_in	adr1, adr0	mux1
adr0	ext_in, ir2	mux0
ralu	adr1	mux1
pc	adr1	mux1
m1	adr1	mux1
ir2	adr0	mux1
bor	–	mux0

The coloring process is presented in Table 2. It is reasonable to order vertices in such table according to their ranks – to the decreasing number of edges connected with each vertex (four such edges for *adr1*, two edges for *ext_in* and *adr0*, one edge for *ralu*, *pc*, *m1* and *ir2* and zero edges for *bor*). We place these vertices in the column *Forbidden vertices*.

We color the first vertex *adr1* with color *mux0*. Since the second vertex *ext_in* is connected to *adr1* (*ext_in* has *adr1* in the column *Forbidden vertices*), we cannot color *ext_in* with the same color *mux0*, but we can use *mux0* for *adr0* not containing *adr1* as a forbidden vertex. We cannot color *ralu*, *pc*, *m1* and *ir2* with *mux0* either because these vertices are connected with vertex *adr1*. Continue until the end of the list with color *mux0* we use this color for *pc*.

In the next step, taking color *mux1* for *ext_in*, we go down the list and color *ralu*, *pc*, *m1* and *ir2* with *mux1*. Now all vertices are colored. The total number of MUXes (colors) is equal to two.

Thus, we got the outputs of MUXes by coloring process. To get inputs to these MUXes we should refer to the connection graph in Fig. 11. Let us discuss MUX0 with outputs *adr1*, *adr0* and *bor*. We go along the last but one column *targets* in this figure and search for target *Adr1*. The first time *Adr1* appears as a target with source *Ext_Adr* with the target weight equal *2* (last column). So we include *ext_adr* as an input with input weight equal to *2* (Fig. 15). Then we continue to descend and find *Adr1* with source *IR2*, its target weight is equal to *3*. We put the second input to MUX0. Going down with source *Adr1* we find the last input *x"ffff"* (weight = *1*).

Now we should repeat the same for target *adr0*. The first appearance of *Adr0* in column targets is with input *Ext_Adr*, target weight *2*. Since we already have such input in MUX0, we add the new weight *2* to the old weight *2* (the weight of input *ext_adr* became equal to *4*) and write *adr0* over the arrow for *ext_adr* near *adr1* to show that this source *Ext_Adr* sends information to *Adr1* and *Adr0* using MUX0 (Fig. 15). Coming down with target *Adr0*, we add new input pc with weight *2*. Then we repeat the same for output *bor*.

Continuing in the same way, we constructed MUX1for 16-bit transfers and MUX2 for for 4-bit transfers. Note, that the same input can appear in the different MUXes if such input has several targets distributed between several MUXes. For example, *ir2* sends information to *adr1* and *bor* through MUX0 and to *pc* – through MUX1.

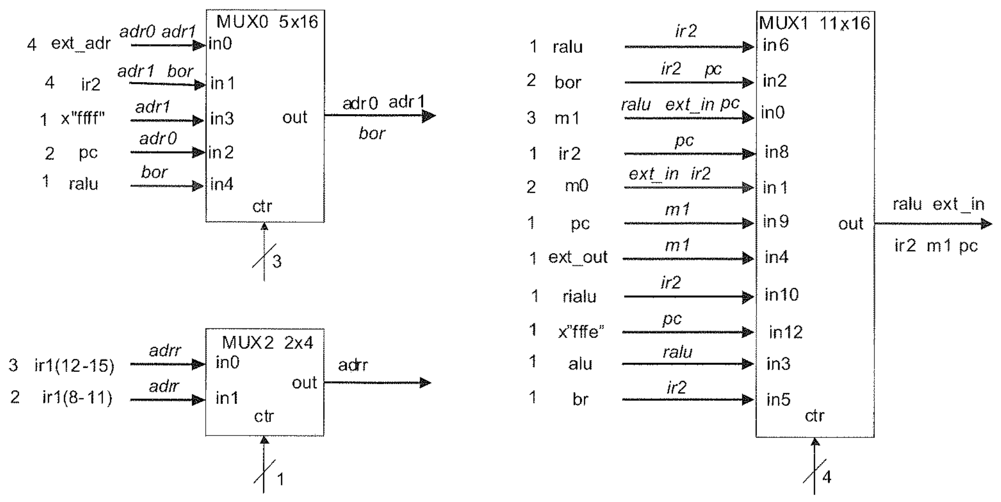


Fig. 15. Main MUXes

After constructing MUXes and the list of direct connections, it is very simple to draw data path (Fig. 18). To do that, first, we draw the units of data path and the main MUXes. Each input of each MUX is connected to the output of the corresponding unit in accordance with the name of the MUX input. Each output of each MUX is connected to the inputs of the corresponding units in accordance with the targets written at the output of MUX. Direct connections are drawn by dotted lines in accordance with the list of direct connections in Fig. 16 and Fig. 17. Of course, a designer shouldn't draw such picture, we give it here only as illustration.

```
MUX0
   OUTPUT:
       Adr0
       Adr1
       BoR[AdrW]
    INPUT:
       Ext_Adr                  4    000      in0    Adr0
                                                     Adr1
       IR2                      4    001      in1    Adr1
                                                     BoR[AdrW]
       PC                       2    010      in2    Adr0
       RALU                     1    100      in4    BoR[AdrW]
       x"FFFF"                  1    011      in3    Adr1

MUX1
   OUTPUT:
       Ext_in
       IR2
       M1[Adr1]
       PC
       RALU
    INPUT:
       M1[Adr1]                 3    0000     in0    Ext_in
                                                     PC
                                                     RALU
       M0[Adr0]                 2    0001     in1    Ext_in
                                                     IR2
       BoR[AdrR]                2    0010     in2    IR2
                                                     PC
       Ext_Out                  1    0100     in4    M1[Adr1]
       IR2                      1    1000     in8    PC
       ALU                      1    0011     in3    RALU
       BR                       1    0101     in5    IR2
       PC                       1    1001     in9    M1[Adr1]
       RALU                     1    0110     in6    IR2
       RIALU                    1    1010     in10   IR2
       x"FFFE"                  1    1100     in12   PC

DIRECT CONNECTIONS:

        SOURCES                      TARGETS

      BoR[AdrR]                        ALU1
                                      IALU2
                                       OutR

         Ext_Out                     M0[Adr0]
           IALU                       RIALU

           IR2                         ALU2
                                      IALU1

       M0[Adr0]                         IR1

       M1[Adr1]                         BR
```

Fig. 16. Muxes and direct connections for 16-bit transfers as text file from Abelite

MUXes and direct connections constructed for 4-bit transfers in the same manner are presented in Fig. 17.

```
MUX2
   OUTPUT:
       AdrR
    INPUT:
       IR1(12-15)          3    0        in0     AdrR
       IR1(8-11)           2    1        in1     AdrR

DIRECT CONNECTIONS:

        SOURCES                     TARGETS

      IR1(8-11)                        AdrW
```

Fig. 17. Muxes and direct connections for 4-bit transfers as text file from Abelite

The design of VHDL code for Data path is very simple because our naked Data path doesn not contain "cloud" (irregular) circuits. As it seen from Table 3, Data path contains 21 units and only 11 standard components which we can take from the library. Thus, to construct Data path in VHDL we should only instantiate all units as their components.

Table 3

Implementation of units by components

Units	Components
u0_mux0	*mux5x16.vhd*
u1_mux1	*mux11x16.vhd*
u2_mux2	*mux2x4.vhd*
u3_bor	*bor.vhd*
u4_alu	*alu.vhd*
u5_m0, u6_m1	*ram16.vhd*
u7_ir1, u8_ir2, u10_ralu, u11_outr, u13_rialu, u21_br	*reg_16bit.vhd*
u9_pc	*count16bit.vhd*
u12_ialu	*ialu.vhd*
u14_cf, u15_zf, u16_vf	*dff.vhd*
u17_ien, u18_r, u19_fgi, u20_fgo	*rsff.vhd*

her are

d Data
, Data
om the
nits as

Table 3

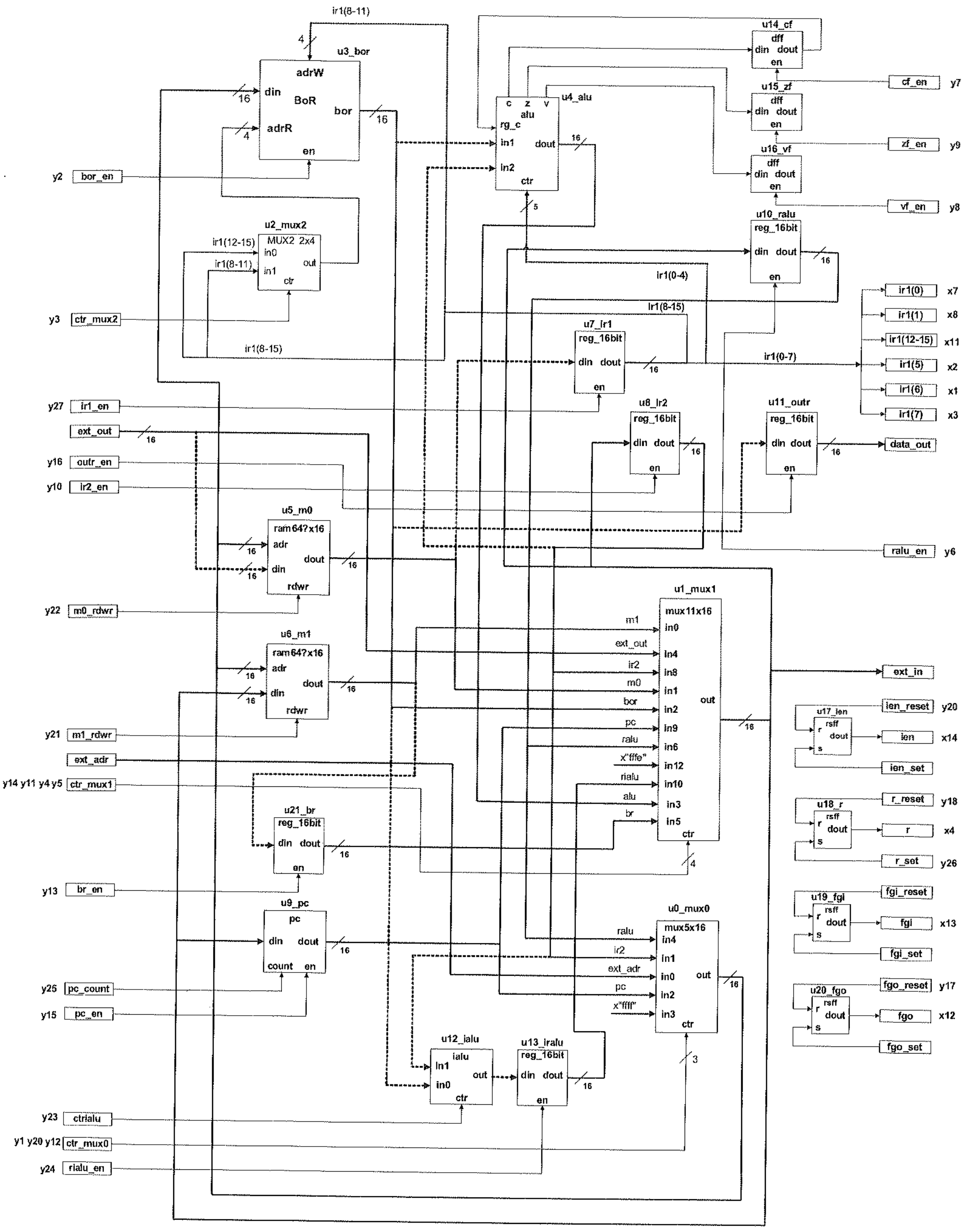


Fig. 18. Data path

Before we discuss the construction of a test bench for Data path, let us look at its figure again in Fig. 18. Even if a Processor is not very complicated, it isn't so easy to understand from what to start. Now we will show that it is possible to formalize and even to automatize design of the test bench for Data path using our design methodology.

First, let us return to the main MUXes presented in Fig. 15. From this figure for MUX0, it is evident that *ext_adr* (in0) should be transferred only to two inputs of our units – *adr0* and *adr1*, *ir2* (in1) – to *adr1* and *bor*, all other inputs of *MUX1* must be transferred only to one of the three possible units, reachable from MUX1. So, instead of checking 5 × 3 = 18 transfers through *MUX1* we must check only *7* (the number of units written over the inputs of MUX1). In the same way, instead of checking 11 × 5 = 55 transfers, it is sufficient to check only 15 transfers through *MUX2* during simulation of Data path.

All these transfers and some other microoperations which really should be implemented in our Processor, are in microinstructions $Y1, \ldots, Y24$ in Table 1. Thus, we must check only this microinstructions and it will be sufficient for checking our Data path. Of course, we assume that each unit of Data path was simulated and verified with its own test bench beforehand.

4. CONTROL UNIT DESIGN

After design of the Data Path, we can immediately pass on to the third stage of the design process – a construction of Control unit (Structural FSM) – see Fig. 19.

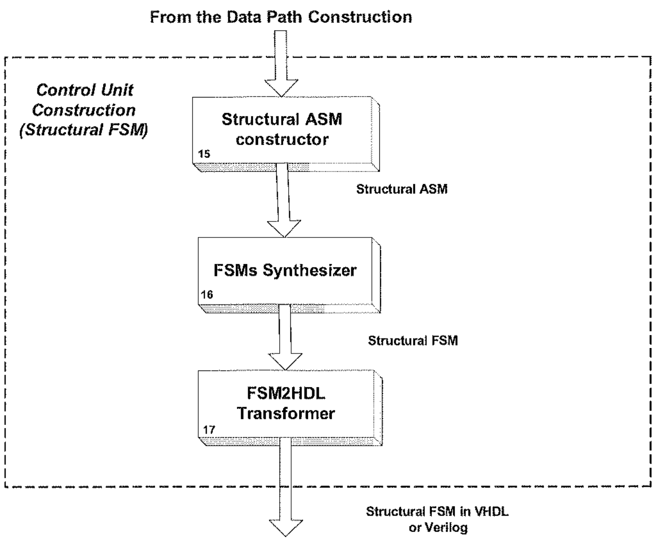

From the Data Path Construction
Control Unit Construction (Structural FSM)
Structural ASM constructor
15
Structural ASM
FSMs Synthesizer
16
Structural FSM
FSM2HDL Transformer
17
Structural FSM in VHDL or Verilog

Fig. 19. Construction of Control unit

Step15. Structural ASM Constructor (box15). To explain this next step let's apply to the process table (Table 1). Abelite implements each functional microoperation from the third column by structural microoperation (or by the set of structural microoperations) in the fifth column of this table. We will postpone consideration of input encoding for MUXes and will use codes from Fig. 15. We assume that a code of input is correspond to the number of input.

Let us discuss several examples of transformation of functional microinstructions into structural ones. Of course, Abelite makes it automatically.

1. *PC := IR2* (row Y_6, column *3*, Table 1).

   To pass information from *IR2* to the input of *PC* through *MUX1* (see Fig. 15) we must send signal *ctr_mux1 := 1000*, because the input *ir2* of MUX1 has code *1000 (in8 of MUX1)*. The signal *pc_en := 1* will write information from the output of *MUX1* into *PC*. Finally we use the following microoperations at the structure level:

   $$ctr\_mux1 := 1000; pc\_en := 1 (\text{row } Y_6, \text{column 5, Table 1}).$$

2. *AdrW := IR1(8-11); BoR[AdrW] := RALU* (row Y_1, column 3, Table 1).

   IR1(8-11) is connected directly with input *AdrW* of *BoR* (Fig. 17) so we do not need any signal for this transfer. To pass information from *RALU* to the input of *BoR* through *MUX0* we must supply signal *ctr_mux0 := 100*, because input *ralu* of *MUX0* has code *100* (Fig. 15 or Fig. 16). The signal *bor_en := 1* will write information from the output of *MUX0* into the register of the *BoR* with address *AdrW*. Finally we use the following microoperations at the structure level:

   $$ctr\_mux0 := 100; bor\_en := 1 (\text{row } Y_1, \text{ column 5, Table 1}).$$

3. *Adr1 := x"FFFF"; M1[Adr1] := PC* (row Y_{12}, column 3).

   The content of *PC* should be written to the word of memory *M1* with address *x"FFFF"*. To pass information from the constant *x"FFFF"* to the address bus *Adr1* of the *M1* through *MUX0* we must supply signal *ctr_mux0 := 011*, because input *x"ffff"* of *MUX1* has code *011* (Fig. 15 or Fig. 16). To pass information from *PC* to the input of memory *M1* through *MUX1* we must supply signal *ctr_mux1 := 1001*, because input *pc* of *MUX1* has code *1001*. The signal *m1_rdwr := 1* will write information from the output of *PC* into the cell of the *M1* with address *X"FFFF"*. Finally we use the following microoperations at the structure level:

   $$ctr\_mux0 := 011; ctr\_mux1 := 1001; m1\_rdwr := 1 := 1 (\text{row } Y_{12}, \text{ column 5}).$$

4. *AdrR :=IR*1(12-15)*; PC:=BoR[AdrR]* (row Y_8, column 3).

   To pass information from *IR1(12-15)* to *AdrR* through *MUX2* we must give signal *ctr_mux2 := 0*, because input *ir1(12-15)* of *MUX2* has code *0*. To pass information from *BoR* to *PC* through *MUX1* we must give signal *ctr_mux1 := 0010*, because input bor of *MUX1* has code *0010*. The signal *pc_en := 1* will write information from the output of *MUX1* into *PC*. Finally we use the following microoperations at the structure level:

$$ctr\_mux2 := 0; ctr\_mux1 := 0010; pc\_en := 1 (\text{row } Y_8, \text{ column } 5).$$

5. *Adr0 := PC; IR1 := M0[Adr0]* (row Y_{24}, column 3).

The content of the word in the memory *M0* with the address in *PC* should be written into *IR1*. To pass information from *PC* to the address bus *Adr0* of memory *M0* through *MUX0* we must supply signal *ctr_mux0 := 010*, because input pc of *MUX0* has code *010*. The signal *m0_rdwr := 0* will read information from the cell of *M0* with address *Adr0* equal to *PC*. Because memory *M0* is connected directly with the input of *IR1* (see Fig. 16), no *MUX* is used to pass information from *M0* to *IR1*. To write information into *IR1* it is sufficient to supply signal *ir1_en := 1*. Finally, we use the following microoperations at the structure level:

$$ctr\_mux0 := 010; m0\_rdwr := 0; ir1\_en := 1 (\text{row } Y_{24}, \text{ column } 5).$$

In such a manner, we have filled the whole fifth column "*Structural Microoperations*" of Table 1. To finish the filling of this table we remind that if some microinstruction, for example, $Y_5 = \{y_1, y_3\}$ is written in the operator vertex of ASM, it means that $y_1 = y_3 = 1$ and other microoperations are equal to zero. Our understanding of output signals in FSM is just like this. If y_1 and y_3 are written in the column for output signals at some transition, only these signals are equal to one at this transition but other output signals are equal to zero.

In consideration of this, let us continue to fill in Table 1. In the sixth column of this table, we write only assignments, which assign "*ones*" to the signals in the fifth column of Table 1. Doing this we present each vector signal (control signal of MUX) as a set of separate binary components and we write assignments only for components equal to one. Look, for example, at microinstruction *Y5* in Table 1:

$$Adr1 := IR2; BR := M1[Adr1].$$

In the fifth column, the following structural microoperations are written:

$$ctr\_mux0 := 001; br\_en := 1; m1\_rdwr := 0.$$

We write in the column 6 only

$$ctr\_mux0(2) := 1; bor_en := 1.$$

In this column, we do not write *ctr_mux0(0) := 0*, *ctr_mux0(1) := 0*, and m1*_rdwr := 0*, because zeroes are assigned in these microoperations.

The combined structural ASM is presented in Fig. 20. This ASM was constructed from functional ASM (Fig. 5) by replacing the functional microoperations in operator vertices, written in column 3 of Table 1 by structural microoperations from the column last but one in this table. As graphs, these two ASMs are absolutely identical. They have the same conditional and operator vertices and the same arcs (connections between these vertices), only the contents of operator vertices were changed in the structural ASM. Abelite constructs this ASM automatically.

Step 16. Synthesis FSM from Structural ASM (box 16). Abelite uses here the same Synthesizer which we presented at the step 5.

Step17. Construction VHDL (Verilog) code for the structural FSM (box 17). Abelite uses here the same ASM2HDL transformer which was used in box 6.

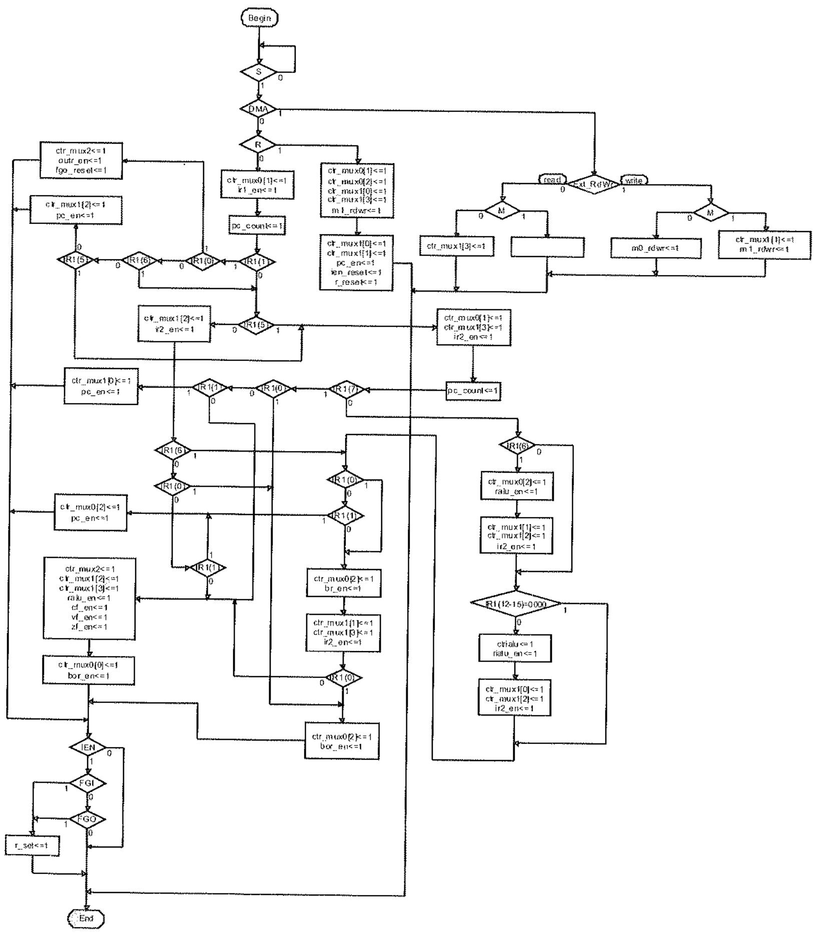


Fig. 20. Structural ASM

Thus, we constructed Data path and Control unit. Our next step is to combine two components – Control unit and Data path in one final block. The top level of our design is presented in Fig. 21.

To minimize the number of output signals at all transitions of FSM (Control unit) Abelite uses the special algorithm for MUX encoding minimizing the number of "ones" in the fifth column of Table 1. That reduces the total number of output signals written in the sixth column of this table.

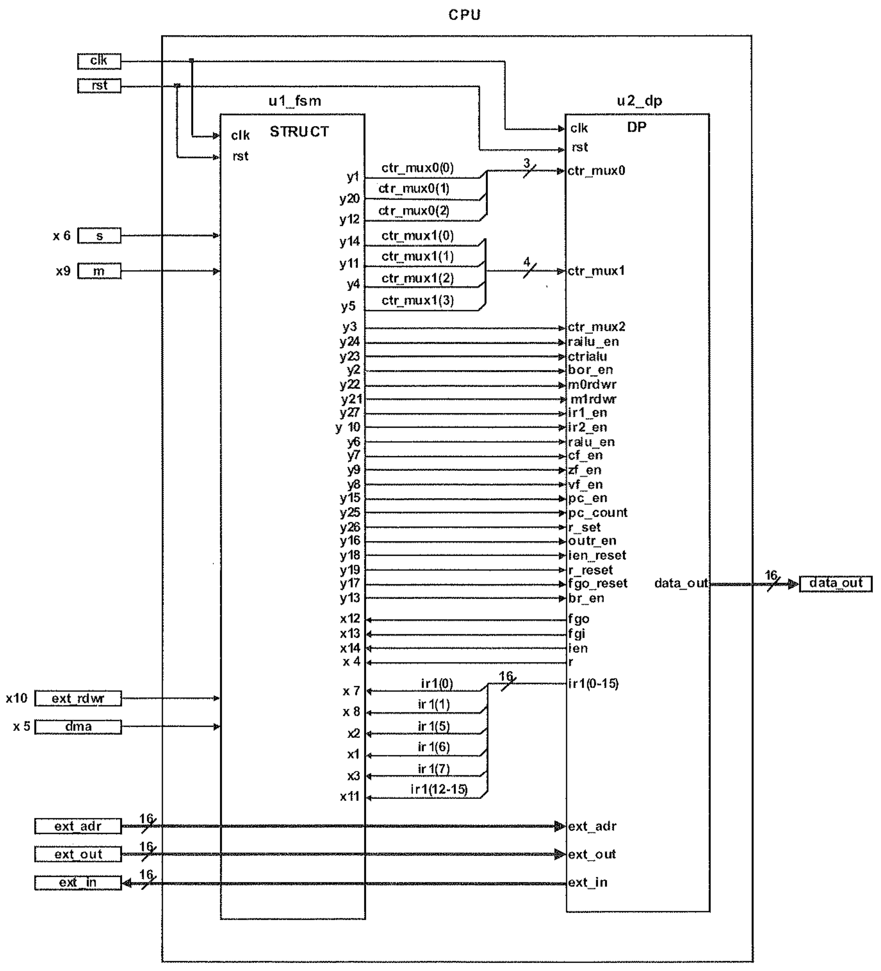


Fig. 21. Top level of Processor

5. LOGIC SYNTHESIS IN ABELITE

In this Section we will shortly present the results of experiments for synthesis of very complicated combinational circuits and final state machines, implemented by Abelite, and comparisons these results with tools from Synopsys, Mentor Graphics and ABC tool from Berkeley.

In experiments with combinational circuits we used circuits from Intel, their parameters are presented in Table 4. The last column of this table contains average number of inputs in one product.

Table 4

Parameters of combinational circuits

Examples	# functions	# inputs	# products	inputs/product
idxlat00n	78	30	3793	8.75
idxlat01n	80	30	4165	8.79
idxlat02n	77	25	2055	7.79
idxlat03n	83	28	6730	9.81
idxlat10n	80	20	4133	7.65

The parameters of very complicated FSMs used in experiments are presented in Table 5. To understand these parameters let's return to FSM in Fig. 6. This FSM contains 15 states, 48 lines in its table, 14 input variables, 44 output variables. Average length of a product in one row is equal to 2.35 and average number of input variables at the transition from one state is equal to 2.79.

Table 5

Parameters of Finite state machines

Examples	# states	# lines	# inputs	# outputs	inputs/product	inputs/state
Bigm2r	174	4899	63	54	6.37	22.35
Exx	79	1157	24	24	5.28	14.27
Group15	422	10185	39	39	7.58	15.55
Huge	199	84993	75	70	13.00	43.97
Other	1275	10980	67	96	5.86	6.69
Rex	1806	27897	70	97	7.19	11.89
Trym	199	24422	71	70	10.63	38.63
Zoom	319	5423	64	37	5.60	17.84

Table 6 and Table 7 contain the result of experiments with Abelite and synthesizer of Synopsys with combinational circuits and FSMs for ASIC with Library Class. Abelite constructed combinational circuits in 2.16 times better and 115 times faster than Synopsys. FSMs were constructed by Abelite in 1.75 times better and 750 times faster. These comparisons were made for five simpliest FSMs from this list because Synopsys could not synthesize the most complicated FSMs Huge, Rex and Trim.

Table 6

Comparison with Synopsys (combinational circuits, ASIC)

Examples	Chip area (gate equivalent)		Time of synthesis	
	Abelite	Synopsys	Abelite	Synopsys
idxlat00	1514	3274	1.162 sec	2 min 35.8 sec
idxlat01	1865	3843	2.130 sec	3 min 18.6 sec
idxlat02	1023	1772	0.519 sec	23.3 sec
idxlat03	2229	5472	4.446 sec	11 min 08.6 sec
idxlat10	1283	2769	1.712 sec	1 min 41.0 sec
Total	**7914**	**17130**	**9.969 sec**	**19 min 07.3 sec**

Table 7

Comparison with Synopsys (FSMs, ASIC)

Examples	Chip area (gate equivalent)		Time of synthesis	
	Abelite	Synopsys	Abelite	Synopsys
Bigm2r	10,964	16,076	16.0 sec	2h 13min 51 sec
Exx	2,713	3,938	1.9 sec	8min 46 sec
Group15	15,358	29,946	30.1 sec	6h 46min 49 sec
Huge	55,673	**	14min 12.4 sec	**
Other	19,874	34,645	43.6 sec	12h 15min 39 sec
Rex	49,513	**	6min 28.5 sec	**
Trym	35,158	**	3min 19.6 sec	**
Zoom	12,683	23,343	21.9 sec	2h 08min 53 sec
Total	**61,592**	**107,948**	**1min 53.5 sec**	**23h 33min 58 sec**

Table 8 and Table 9 contain the results of experiments with Abelite and EDA tool Leonardo from Mentor Graphics with combinational circuits and FSMs on FPGA. Abelite constructed combinational circuits in 1.92 times better and 450 times faster than Leonardo. FSMs were constructed by Abelite in 2.83 times better and 400 times faster. These comparisons were made for six simpliest FSMs from this list because Leonardo couldn't synthesize the most complicated FSMs Huge and Rex.

Table 10 and Table 11 contain the results of experiments with Abelite and EDA tool ABC from Berkeley with combinational circuits and FSMs on FPGA. ABC is a very fast synthesizer. Abelite constructed combinational circuits in 2.06 times better for the same time. However, for the best result, ABC should be consequently run 10

Table 8

Comparison with Leonardo (combinational circuits, FPGA)

Examples	Chip area ((Luts)/arrival time)*		Time of synthesis	
	Abelite	Leonardo	Abelite	Leonardo
idxlat00	739/14.04	1409/16.15	1.853 sec	16 min 59 sec
idxlat01	905/14.89	1572/15.31	2.413 sec	22 min 18 sec
idxlat02	492/11.50	832/14.46	1.052 sec	3 min 57 sec
idxlat03	1117/14.89	2288/17.45	4.326 sec	28 min 29 sec
idxlat 10	626/14.04	1347/15.31	1.602 sec	6 min 10 sec
Total	**3879**	**7448**	**11.246 sec**	**1h 17 min 53 sec**

* arrival time in nsec

Table 9

Comparison with Leonardo (FSMs, FPGA)

Examples	Chip area (Luts)/MHz		Time of synthesis	
	Abelite	Leonardo	Abelite	Leonardo
Bigm2r	3948/44.9	5643/39.9	7.531 sec	34min
Exx	990/46.7	1641/49.2	1.531 sec	2min
Group15	5933/31.7	14529/36.9	15.812 sec	1h 51min
Huge	22658/25.5	**	6min 35.984sec	**
Other	6772/36.6	19986/32.0	15.531 sec	1h 18min
Rex	18313/31.7	**	1min 44.829 sec	**
Trym	13970/30.1	50758/22.4	1min 17.109 sec	More than14hours
Zoom	4480/38.2	9681/43.1	9.046 sec	37min
Total	**36093**	**102238**	**2min 42.418 sec**	**More than18 hours**

times. Really, ABC improved its result and decrease ratio 2.06 to 1.75 but spent almost seven times more time. FSMs were constructed by Abelite in 1.37 times better (one run for ABC) and in 1.24 times better (several runs for ABC). Unfortunately, ABC couldn't synthesize the most complicated FSM Huge from this list.

Table 10

Comparison with ABC (combinational circuits, FPGA)

Examples	Chip area (Luts)			Time of synthesis		
	Abelite	Berkeley		Abelite	Berkeley	
		1 time	10 time		1 time	10 times
idxlat00	739	1519	1422	1.853 sec	2.264 sec	13.840 sec
idxlat01	905	1730	1492	2.413 sec	2.433 sec	17.355 sec
idxlat02	492	899	782	1.052 sec	1.622 sec	8.583 sec
idxlat03	1117	2482	2181	4.326 sec	3.395 sec	23.664 sec
idxlat 10	626	1370	1311	1.602 sec	2.173 sec	12.939 sec
Total	**3879**	**8000**	**7188**	**11.246 sec**	**11.887 sec**	**76.381 sec**

Table 11

Comparison with ABC (FSMs, FPGA)

Examples	Chip area (Luts)			Time of synthesis		
	Abelite	Berkeley		Abelite	Berkeley	
		1 time	10 time		1 time	10 times
Bigm2r	3948	5271	4502	16.884 sec	8.60 sec	56.53 sec
Exx	990	1147	1074	2.403 sec	1.96 sec	10.69 sec
Group15	5933	8061	6790	28.691 sec	18.54 sec	2 min 26.17 sec
Huge	22658	**	**	16 min 16.76 sec	**	**
Other	6772	8711	8006	27.59 sec	14.72 sec	1 min 53.37 sec
Rex	18313	24967	23448*	04 min 38.18 sec	1 min 09.84 sec	3 min 26.10 sec*
Trym	13969	20118	18761*	03 min 42.79 sec	1min 21.30sec	4 min 50.10 sec*
Zoom	4480	6007	5094	17.49 sec	10.18 sec	1 min 14.85 sec
Total	**54405**	**74282**	**67675**			

* Rexm and Trym were run 3 times, after that the result wasn't shanged

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 157–182

Petri Nets Mapping into Reconfigurable Logic Controllers

MARIAN ADAMSKI, MAREK WĘGRZYN

University of Zielona Góra, Institute of Computer Engineering and Electronics,
ul. Podgórna 50, 65-246 Zielona Góra,
M.Adamski@iie.uz.zgora.pl, M.Węgrzyn@iie.uz.zgora.pl

Received 2009.01.29
Authorized 2009.04.06

The paper concentrates on the behavioral specification of Reconfigurable Logic Controller programs, given initially as Petri nets and later rewritten in Hardware Description Languages. The rule-based textual language input makes it possible to integrate the design system with existing formal logic based computer-based theorem proovers. The Petri net description in HDL provides the opportunity to integrate existing Petri net software with several commercial systems. Different Petri net places encoding methods are also discussed. Verilog-HDL is used for an intermediate representation of controller behavior on top of existing commercial synthesis tools. The implementation methods using D, JK and T flip-flops are presented.

Keywords: Logic Controller, Petri Net, Programmable Logic, FPGA, HDL, VHDL, Verilog, Place Encoding, Modeling, Synthesis, PNSF2

1. INTRODUCTION

1.1. MOTIVATION

To describe digital systems, designers frequently adapt a concurrent and distributed view of the modeled behavior. Petri nets [36, 37, 38, 41, 42, 55] provide a mechanism, which is suited to representing parallelism and hierarchy in complex digital processes. The Petri nets are used both as specification and synthesis models for Reconfigurable Logic Controller designs, which are frequently embedded inside modern, reactive microsystems [64].

The main aim of this paper is to demonstrate a practical, direct method of mapping Concurrent Digital Systems into Programmable Logic (PL), during the design process of controller design. The paper gives also an overview of selected papers, related to the hardware implementation of Petri nets. The experimental results have shown that the presented novel approach may produce economical Programmable Logic implementations of reconfigurable logic controllers.

A Petri net can be considered as a behavioral specification of a concurrent state machine (concurrent control automaton) as well as a formal model suitable for its rule-based description, transformed and optimized during logic design process step-by-step. The Petri net can serve also as a direct, immediate model for HDL description. both for rapid prototyping and optimized design by means of commercial tools, approved by electronic industry (Xilinx, Altera).

The behavioral rule-based textual descriptions of Control Interpreted Petri Nets (CIPN) are formally transformed into structured templates in XML, which are treated as shells for standard hardware description languages, such as VHDL or Verilog. The automatic model-driven design process is realized at first on the register transfer level (RTL) by means of experimental dedicated CAD tools developed at University of Zielona Gora. After local state encoding and structural logic synthesis the logic expressions are transformed into HDL statements. All procedures make it possible to obtain a compact and reliable implementation, considering the simplicity of design and limited size of array structures in SoC (*System-on-Chip*).

1.2. BACKGROUND

In general, Programmable Logic can be re-configured by the user to perform particular combinational or registered logic functions. The design process is greatly simplified by FPGA and CPLD compilers. The effective simulation allows the Logic Controller to be debugged before the device is programmed. If design change is needed, it is a simple matter to re-edit the original specification and then re-program or exchange the old device. FPGA can be dynamically reconfigured to perform many different logic control programs, serving as adaptive concurrent (parallel) state machine with data path.

While software implementation of logic controllers can be applied only to comparatively slow targets, hardware implementation of a Petri net is recommended for high-speed, parallel, dependable controllers, interacting with several concurrent processes. Other advantages of the method are reusability, fast prototyping, and testability, which are ensured because the reconfigurable controller fully implements a structure of discrete algorithm and its desired properties [8, 12, 53, 68].

2. LOGIC CONTROLLER – CASE STUDY

Logic controllers are related mainly to relatively simple embedded discrete systems, whose behavior is defined by interaction with its environment. As synthesis tools become more advanced and user friendly, the entry point in the design process is moving towards higher levels of specification. The proposed structured design is applied in many formalisms used to specify logic controllers programs, such as interpreted Petri net or Sequential Function Chart (SFC) UML state machine diagrams. Hardware Description Languages (VHDL or Verilog) are used for synthesis, verification, and documentation of design. The logic controller model (concurrent state machine with data path) may be implemented explicitly in hardware description language, or from the front-end entry (shell) from its rule-based description.

Digital embedded systems require real-time operations and concurrent processing. Reconfigurable Application Specific Logic Controllers (ASLC) are very fast and flexible dedicated devices, implemented in array-based programmable logic. A discrete HDL model of Logic Controller (Fig. 1), which is derived from the control interpreted Petri net is implemented as a FPGA-based control unit, which is nested inside a discrete control system.

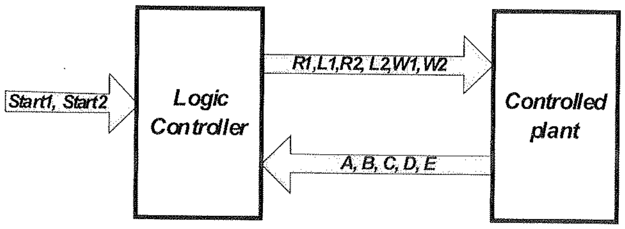


Fig. 1. Logic control system

The industrial logic control system consists of three parts: a Logic Controller, a Controlled Plant (mechatronic operational unit), and an environment, which involves a human operator.

The applicability of the presented approach is demonstrated by the solution to the discrete control problem [63]. This toy-example has been adapted by authors as an illustration of several different design methodologies [68].

Fig. 2 depicts the controlled part of a designed simple reactive system. The controlled system consists of two tankers ***1*** and ***2*** that go on the left (***L***) and right (***R***) sides. The tankers start on signals *Start1* and *Start2*, respectively. When both tankers go concurrently, they can reach points *D* and *E*. Because tanker ***1*** has higher priority, it goes to point *B*, and then goes back to *A*, where it waits. When it waits until button *Start 1* is pressed the next time, meanwhile tanker *2* waits in point *E* until tanker ***1*** reaches point *D* on its way back. Then the tanker *2* goes to *B* and back to emph *C*, where it waits for next pressing of bottom *Start2*. During this process, only oe tanker carn be located on any single track. When the full technological cycle is completed the system waits in the initial, idle states.

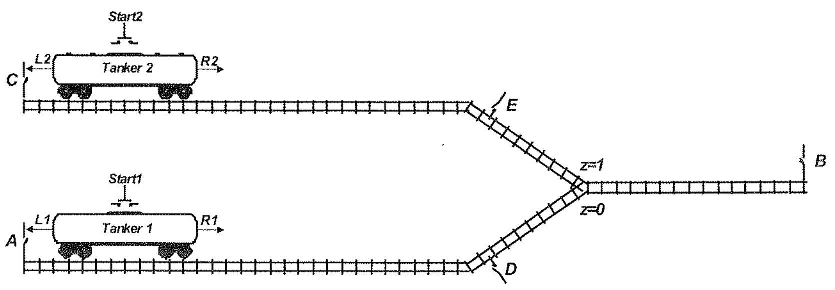


Fig. 2. Technological process

It is necessary to identify the inputs and outputs (Table 1) of Logic Controller (Fig. 1). The unique local states of the controller and their verbal descriptions are listed in Table 2.

Table 1

Description of inputs and outputs of the controller

Signal name		Description
Inputs	Start1	Start button for tanker 1
	Start2	Start button for tanker 2
	A	Tanker 1 in left starting position
	C	Tanker 2 in left starting position
	B	Tanker 1 or 2 in right final position
	D	Tanker 1 reaches rail switch
	E	Tanker 2 reaches rail switch
Outputs	R1	Tanker 1 goes right
	L1	Tanker 1 goes left
	R2	Tanker 2 goes right
	L2	Tanker 2 goes left
	W1	Tanker 1 on common track, z=0
	W2	Tanker 2 on common track, z=1

Table 2

Local states descriptionh

Local state	Description	Local state	Description
p1	Initial state (for tanker 1)	p8	Initial state (for tanker 2)
p2	Tanker 1 goes right on private track	p9	Tanker 2 goes right on private track
p3	Tanker 1 waits for permission	p10	Tanker 2 waits for permission
p4	Tanker 1 goes right on common track	p11	Tanker 2 goes right on common track
p5	Tanker 1 goes left on common track	p12	Tanker 2 goes left on common track
p6	Tanker 1 goes left on private track	p13	Tanker 2 goes left on private track
p7	Common track is free	p14	Common track is occupied by tanker 1
		p15	Common track is occupied by tanker 2

3. PETRI NET AS A BEHAVIORAL SPECIFICATION OF LOGIC CONTROLLER

3.1. PETRI NETS AND LOGIC CONTROLLERS

The sequence control problem is represented in or structural manner, showing the various actions (y) to be taken in each total discrete step (M) and indicating the conditions (x), which need to be satisfied before the next step. In concurrent systems a total discrete step (macrostate) is a collection of simultaneously held partial states. Designing the discrete controller as a digital subsystem involves the generation of a Petri net based behavioral specification by analyzing, the properties of the controlled object (plant) and its desired functionality (Fig. 3).

Control interpreted Petri net represents the behavior of a discrete controller as concurrent sequences of places and transitions. Each place p is related with an action, that is active (y = 1) when it is marked, or inactive (y = 0) if the place is empty. If the transition label (guard, predicate) is true, the marked input places of transition become empty and the next output places become marked. The required sequence of transition firing is shown by directed edges (arcs), pointing in the direction of the intended flow of tokens. In such a way interpreted Petri net encapsulates concurrent input and output sequences that the controller should accept and produce. Safe PNs can be viewed as a natural extension to linked Finite State Machine (FSM) specifications. A Concurrent State Machine (CSM) allows data (inputs, outputs) to be exchanged with the external environment, according to its current global state M. Each place of the Petri net (Fig. 3) is viewed as a local control state (Table 2). The global states $M = [p1, \ldots, p15]$ of the controller are given implicitly by the set of all possible Petri net markings, and they can be eventually derived from the reachability graph of the net, as distribution of tokens by the places during the evolution of the net: *M1* = *Marking1*, *M2* = *Marking2*, *M3* = *Marking3*. The logical expressions (*Start1*, *Start2*, ..., *E*) from Table 1, which

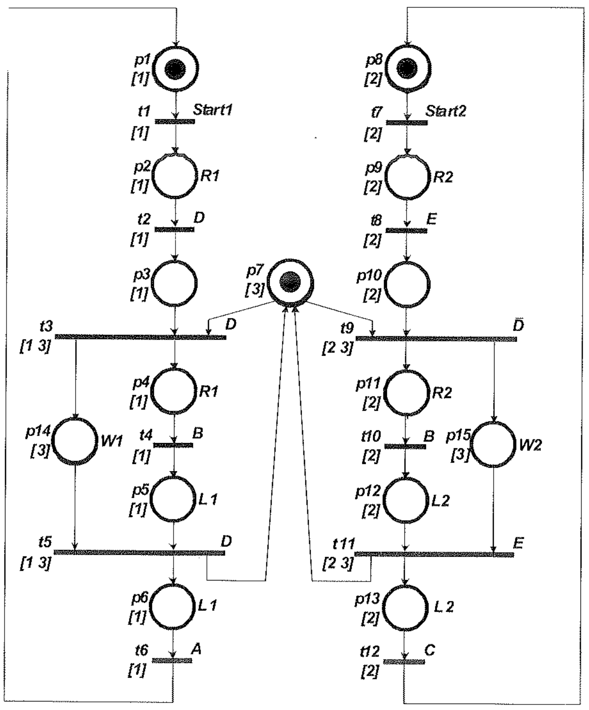


Fig. 3. Petri net model

are called guards, are associated with transitions *t1÷t12*. Respectively, to represent the controller actions, the output signals (*R1*, *L1*, . . . , *W2*) are associated with the places *p1÷p13*.

The virtual Sequential State Machines (SSM), which are included in CSM, interact with other, by means of a shared memory (internal state register). The colors *[1]*, *[2]*, *[3]* demonstrate the proper covering of the Petri net by three P-subnets, representing concurrently related state machines (SM). The strict rules of SM-coloring of the Petri net are given in the papers [16, 32].

It should be noted that the considered in the paper Coloured Control Interpreted Petri Net (CCIPN) is a subset of the general coloured Petri net (CPN), invented by Jensen [44] with restricted rules for allowed colouring of places, transitions and arcs [16, 76]. It makes possible to use efficiently well developed theory of CPN and CPN tools for formal analysis of model properties as well as animation of Logic controller behaviour during its evaluation [66].

The Petri net which is drawn according to standard CPN is presented in Fig. 4. In the colored Petri net (Fig. 3) the marking is represented by several colored tokens. Directed implicitly colored arcs connect the explicitly colored places and the implicitly colored transitions. Transitions are allowed to or prevented from occurring with respect to a particular color if the attached Boolean expression is respectively true or false.

In the standard CPN specification the coloring of the net must be explicitly defined (Fig. 4).

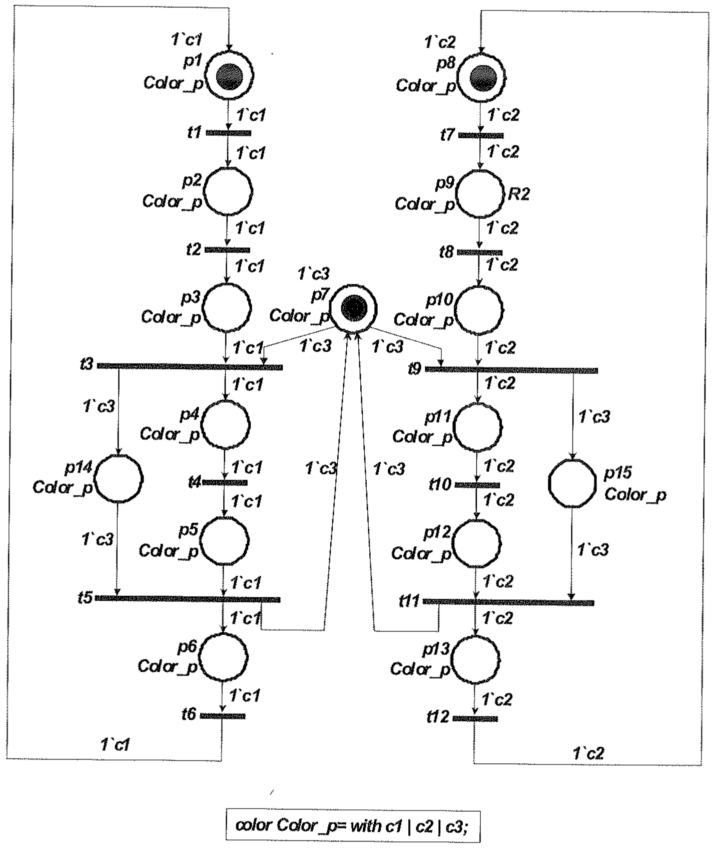


Fig. 4. Petri net drawn in Jensen CPN style

3.2. CONCURRENT STATE MACHINE

The Petri net is directly mapped into the Boolean equations (decision rules in propositional logic) without explicit enumeration of all possible global states and all possible global state changes. The specification is given in terms of local state changes, related with Petri net transitions (Table 3). Moore type output signals are generated by places (Table 4). The decision table format is very close to the state tables for sequential automata used in [25, 13, 27, 72, 73].

Table 3

Decision table for the control unit

Transition	Current local states	Conditions	Next local states
t1	p1	M1	p2
t2	p2	D	p3
t3	p3, p7	D	p4, p14
t4	p4	B	p5
t5	p5, p14	D	p6, p7
t6	p6	A	p1
t7	p8	M2	p9
t8	p9	E	p10
t9	p7, p10	!D	p11, p15
t10	p11	B	p12
t11	p12, p15	E	p7, p13
t12	p13	C	p8

Table 4

Decision table for outputs

Local state	Outputs	Local state	Outputs
p2	R1	p11	R2
p4	R1	p12	L2
p5	L1	p13	L2
p6	L1	P14	W1
P9	R2	P15	W2 (z)

Concurrent Logic Controller can be presented using a modular and hierarchical view of the modeled system behavior. It retains the natural partitioning of the behavior imposed by the designer, depicted by colors (*[1], [2], [3]*). A given Petri net is transformed into a hierarchical macronet, a net having structured macroplaces, which represent Petri net subnets, particularly State Machine subnets (Fig. 5). The functionality is represented as a set of concurrent blocks of a manageable size that communicate using few signals (Fig. 6).

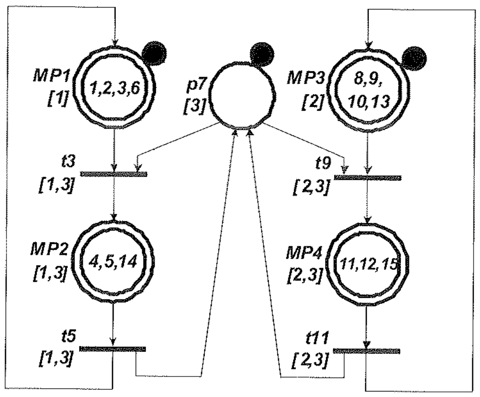


Fig. 5. Colored macronet: SM sub-components in hierarchical Petri net model

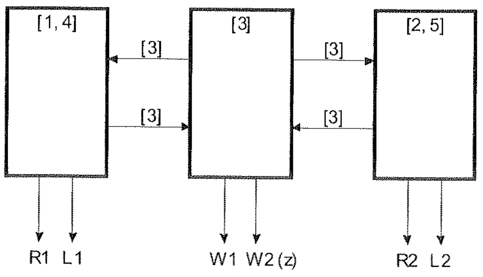


Fig. 6. Distinguished processes in Logic Controller

Each color represents a sequential process and in considered approach selected color can be related only to one process. The total number of the colored tokens indicates the number of concurrent processes being active at any global state. Such colored Petri net can be decomposed into several Linked State Machine (LSMs) [28].

3.3. TEXTUAL SPECIFICATION OF PETRI NETS

In [4, 7, 14, 20] the digital system is considered as an abstract reasoning system (rule based system) implemented in hardware. The mapping between inputs and outputs of the system is described in a formal manner by means of logic rules (represented as sequents) with some temporal operators, especially operator 'next' @. The rule-based description [5], supported by means of logic deduction techniques (Gentzen natural logic calculus [29]), is used in Programmable Logic Controller design context [9]. Place oriented declarative specification is as follows:

Preconditions:

```
  p1 * Start1 |- t1;
  p2 * D |- t2;
  ...
  p13 * C |- t12;
```

Next markings:

```
  t6 + p1 * !t1 |- @p1;
  t1 + p2 * !t2 |- @p2;
  t5 + t11 + p7 * (!t3 + !t9) |- p7;
  ...
```

Sequents may be roughly treated as more general forms of clauses with conjunctive antecedents and disjunctive consequents and they represent assertions [4]. In the paper [47] the Petri Net Specification Format (PNSF) for VLSI design was introduced as a simplified version of rule based description of Petri net using sequent language. One of its extended improved versions is called PNSF2 (Fig. 7) [67].

```
.clock CLK

.inputs Start1 Start2 A B C D E
.comb_outputs L1 L2 R1 R2 W1 W2

.part TankersControl
.places p1 p2 p3 p4 p5 p6 p7 p8
.places p9 p10 p11 p12 p13 p14 p15
.transitions t1 t2 t3 t4 t5 t6
.transitions t7 t8 t9 t10 t11 t12

.net
 t1: p1 * Start1 |- p2;
 t2: p2 * D |- p3;
 t3: p3 * p7 * D |- p4 * p14;
 t4: p4 * B |- p5;
 t5: p5 * p14 * D |- p6 * p7;
 t6: p6 * A |- P1;
 t7: p8 * Start2 |- p9;
 t8: p9 * E |- p10;
 t9: p10 * p7 * !D |- p11 * p15;
 t10: p11 * B |- p12;
 t11: p12 * p15 * E |- p13 * p7;
 t12: p13 * C |- p8;

.MooreOutputs
 p2 |- R1;
 p4 |- R1;
 p5 |- L1;
 p6 |- L1;
 p9 |- R2;
 p11 |- R2;
 p12 |- L2;
 p13 |- L2;
 p14 |- W1;
 p15 |- W2;

.marking p1 p7 p8
.end
```

Fig. 7. Petri net specification in PNSF2

3.4. RAPID PROTOTYPING

In dealing with concurrency the designer is confronted with some problems that will not arise in the logic synthesis of sequential systems. To keep a very strict

```
module TankersControl (CLK, Reset, A, B, C, D, E, Start1, Start2,
                       L1, L2, R1, R2, W1, W2);
input CLK, Reset;
input A, B, C, D, E, Start1, Start2;
output L1, L2, R1, R2, W1, W2;
reg p1, p2, p3, p4, p5, p6, p7, p8, p9, p10, p11, p12, p13, p14, p15;
wire t1, t2, t3, t4, t5, t6, t7, t8, t9, t10, t11, t12;

assign t1 = p1 & Start1;
assign t2 = p2 & D;
assign t3 = p3 & p7 & D;
assign t4 = p4 & B;
assign t5 = p5 & p14 & D;
assign t6 = p6 & A;
assign t7 = p8 & Start2;
assign t8 = p9 & E;
assign t9 = p7 & p10 & ~D;
assign t10 = p11 & B;
assign t11 = p12 & p15 & E;
assign t12 = p13 & C;

assign L1 = p5 | p6;
assign L2 = p12 | p13;
assign R1 = p2 | p4;
assign R2 = p9 | p11;
assign W1 = p14;
assign W2 = p15;

always @(posedge CLK)
    if (Reset) p1 <= 1'b1; else p1 <= t6 | (p1 & ~t1);
always @(posedge CLK)
    if (Reset) p2 <= 1'b0; else p2 <= t1 | (p2 & ~t2);
always @(posedge CLK)
    if (Reset) p3 <= 1'b0; else p3 <= t2 | (p3 & ~t3);
always @(posedge CLK)
    if (Reset) p4 <= 1'b0; else p4 <= t3 | (p4 & ~t4);
always @(posedge CLK)
    if (Reset) p5 <= 1'b0; else p5 <= t4 | (p5 & ~t5);
always @(posedge CLK)
    if (Reset) p6 <= 1'b0; else p6 <= t5 | (p6 & ~t6);
always @(posedge CLK)
    if (Reset) p7 <= 1'b1; else p7 <= (t5 | t11) | (p7 & ~t3 & ~t9);
always @(posedge CLK)
    if (Reset) p8 <= 1'b1; else p8 <= t12 | (p8 & ~t7);
always @(posedge CLK)
    if (Reset) p9 <= 1'b0; else p9 <= t7 | (p9 & ~t8);
always @(posedge CLK)
    if (Reset) p10 <= 1'b0; else p10 <= t8 | (p10 & ~t9);
always @(posedge CLK)
    if (Reset) p11 <= 1'b0; else p11 <= t9 | (p11 & ~t10);
always @(posedge CLK)
    if (Reset) p12 <= 1'b0; else p12 <= t10 | (p12 & ~t11);
always @(posedge CLK)
    if (Reset) p13 <= 1'b0; else p13 <= t11 | (p13 & ~t12);
always @(posedge CLK)
    if (Reset) p14 <= 1'b0; else p14 <= t3 | (p14 & ~t5);
always @(posedge CLK)
    if (Reset) p15 <= 1'b0; else p15 <= t9 | (p15 & ~t11);
endmodule
```

Fig. 8. Verilog model (with encoding option in synthesis tool

correspondence between an initial specification as Petri net and hardware description languages, such as VHDL, the rule-based textual form is considered [6]. It was developed as a bridge between PN and its VHDL models. The VHDL style and template type, introduced by Bolton, was continued and modified by several researchers [21, 40, 57, 77]. The structural version of rapid prototyping is presented in [23].

The one-hot-encoding of Petri net is treated as the simpliest case of more general mapping. The one-hot method [2, 40, 45, 57, 62, 67, 78, 79] produces fast designs with a simple combinational part, especially for rapid implementations in FPGA. It is not assumed that all flip-flops, except one, are set to 0 since several places can be marked simultaneously.

The concurrent one-hot encoding is a modification of a popular one-hot state assignment of sequential (non-concurrent) state machine, in which one flip-flop is used for each global state. After such local state encoding of concurrent state machine, all flip-flops related with simultaneously marked places are set to one at the same time. The total number of flip-flops is equal to the number of places:

code (p1) = p1
code (p2) = p2
...
code (p15) = p15

The Verilog model suitable for one-hot encoding is shown in Fig. 8.

3.5. PETRI NET COLORING AND RELATION OF CONCURRENCY

It has been previously mentioned that places in a Petri net are marked sequentially or concurrently with respect to each other. If the local state space of Petri net or Petri macronet is explicitly given (Fig. 5), it is straightforward to construct the concurrency graph. It can be performed by means of inspection of cliques related with vertices in the reachability graph of macronet (Table 5).

Table 5

Global states matrix

$$\begin{array}{r} \\ Marking1 \\ Marking2 \\ Marking3 \end{array} \begin{array}{c} \begin{array}{ccccc} MP1 & MP2 & p7 & MP3 & MP4 \end{array} \\ \left[\begin{array}{ccccc} 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 \end{array}\right] \end{array}$$

scription
as deve-
template
ners [21,

e general
designs
FPGA. It
s can be

state as-
o is used
chine, all
me time.

uentially
or Petri
currency
rtices in

Table 5

Every two simultaneously marked macroplaces *MPi*, *MPj* (Fig. 9) are represented by vertices (*MPi*, *MPj*) connected by edge in the *concurrency graph* (*GC*) (Fig. 10a). The complement of the concurrency graph *GC* forms a *non-concurrency graph* (*GN*) (Fig. 10b). In the non-concurrency graph edges connect pairs of the macroplaces, which are not simultaneously marked. The graphs *GC* and *GN* are frequently represented as adjacency matrices. The adjacency matrix of graph *GC*, which is supplemented with numbers 1 on the main diagonal, is called a concurrency matrix (Table 6) [22, 57].

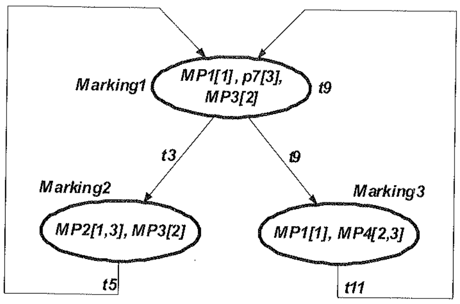


Fig. 9. Reachability graph

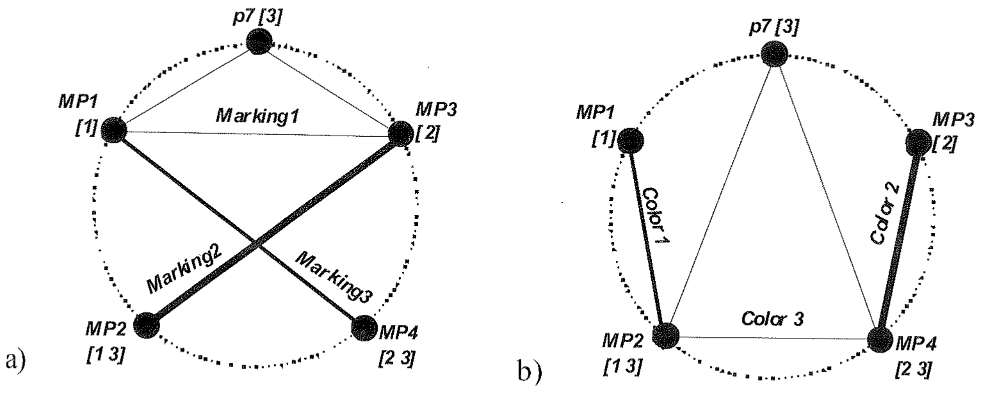


Fig. 10. Concurrency (a) and non-concurrency (b) graphs for macronet

Table 6

Concurrency matrix

	MP1	*MP2*	*p7*	*MP3*	*MP4*	*Superposition of*
MP1	1	0	1	1	1	*Marking1 + Marking3*
MP2	0	1	0	1	0	*Marking2*
p7	1	0	1	1	0	*Marking1*
MP3	1	1	1	1	0	*Marking1 + Marking2*
MP4	1	0	0	0	1	*Marking3*

An entry *Cij* in the concurrency matrix *C* equals 1 if places corresponding to the row *i* and the column *j* may hold tokens simultaneously (they belong to the same marking of the net), otherwise it equals 0. It should be noted that the main diagonal of the matrix only contains the numbers 1. The concurrency matrix may be used for the several analysis or synthesis techniques, including hierarchical, sequential, parallel decomposition and place encoding. The complementary matrix, which represents relation of non-concurrency is referred as *non-concurrency matrix*) (Table 7).

Table 7

Non-concurrency matrix

$$\begin{array}{r|ccccc|}
 & MP1 & MP2 & p7 & MP3 & MP4 \\
MP1 & 0 & 1 & 0 & 0 & 0 \\
MP2 & 1 & 0 & 1 & 0 & 1 \\
p7 & 0 & 1 & 0 & 0 & 1 \\
MP3 & 0 & 0 & 0 & 0 & 1 \\
MP4 & 0 & 1 & 1 & 1 & 0
\end{array}$$

4. CONTROLLER SYNTHESIS

4.1. CONCURRENT LOCAL STATE ASSIGNMENT

It is possible to reduce the global number of flip-flops, but usually with increasing the complexity of the combinatorial circuits per particular flip-flop [17, 34]. Adding additional state variables for the encoding of a particular place multiplies the number of expressions in flip-flop excitation functions to be realized in LUT (Look-Up Table). The basic methods [3, 6] were improved and developed by Bolton and Amroun [22], Biliński [30], Kozłowski, *et al.* [47], Pardey and Bolton [57], Węgrzyn [70, 71] and Zakrevskij [78, 79]. The codes of particular places are as follows:

*code (p1) = !Q1 * !Q2 * !Q3*
*code (p2) = !Q1 * Q2 * !Q3*
*code (p3) = Q1 * !Q2 * !Q3*
*code (p4) = Q1 * !Q2 * Q3*
*code (p5) = Q1 * Q2 * Q3*
*code (p6) = Q1 * Q2 * !Q3*
*code (p7) = !Q3 * !Q6*
*code (p8) = !Q4 * !Q5 * !Q6*
*code (p9) = !Q4 * Q5 * !Q6*
*code (p10) = Q4 * !Q5 * !Q6*
*code (p11) = Q4 * !Q5 * Q6*
*code (p12) = Q4 * Q5 * Q6*
*code (p13) = Q4 * Q5 * !Q6*
*code (p14) = Q3 * !Q6*
*code (p15) = !Q3 * Q6*

Some advanced techniques of concurrent state encoding developed by Adamski, Cheremisinova, Pottosin and Zakrevskij are presented in the book [19].

The encoded decision rules for the transitions of the control unit are presented in Table 8. The Petri net with encoded places is presented in Fig. 11. The decision rules describing the controller outputs according to marked places are given in Table 9.

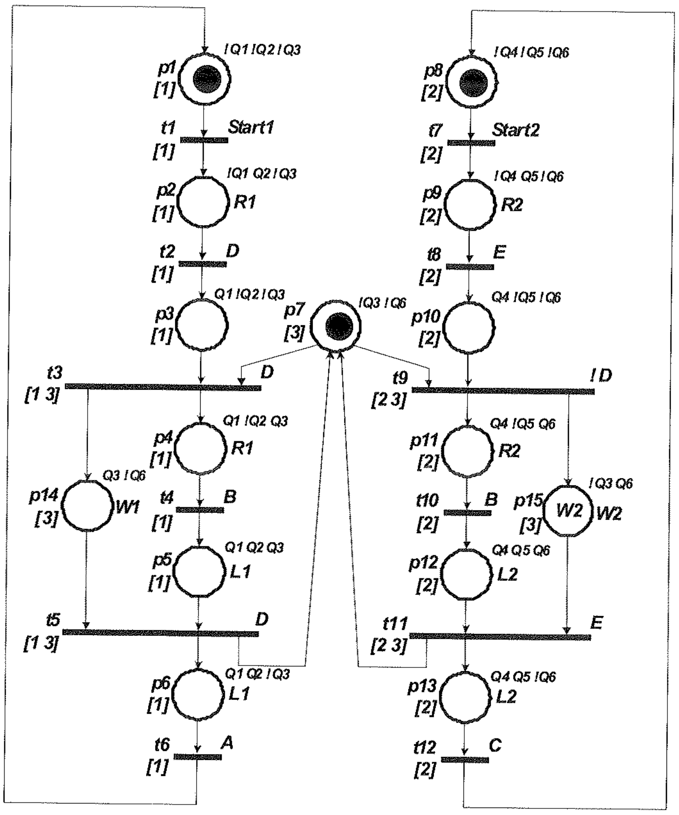


Fig. 11. Encoded Petri net

Table 8

Decision table for the encoded control unit

Transition	Current states	Current states' code	Conditions	Next states	Next states' code
t1	p1	!Q1*!Q2*!Q3	Start1	p2	!Q1* Q2*!Q3
t2	p2	!Q1* Q2*!Q3	D	p3	Q1*!Q2*!Q3
t3	p3 * p7	Q1*!Q2*!Q3*!Q6	D	p4 * p14	Q1*!Q2* Q3*!Q6
t4	p4	Q1*!Q2* Q3	B	p5	Q1* Q2* Q3
t5	p5 * p14	Q1* Q2* Q3*!Q6	D	p6 * p7	Q1* Q2*!Q3*!Q6
t6	p6	Q1* Q2*!Q3	A	p1	!Q1*!Q2*!Q3
t7	p8	!Q4*!Q5*!Q6	Start2	p9	!Q4* Q5*!Q6
t8	p9	!Q4* Q5*!Q6	E	p10	Q4*!Q5*!Q6
t9	p7 * p10	!Q3* Q4*!Q5*!Q6	!D	p11 * p15	!Q3* Q4*!Q5* Q6
t10	p11	Q4*!Q5* Q6	B	p12	Q4* Q5* Q6
t11	p12 * p15	!Q3* Q4* Q5* Q6	E	p7 * p13	!Q3* Q4* Q5*!Q6
t12	p13	Q4* Q5*!Q6	C	p8	!Q4*!Q5*!Q6

Table 9

Decision table for the encoded outputs

Local state	Local state code	Outputs	Local state	Local state code	Outputs
p2	!Q1* Q2*!Q3	R1	p11	Q4*!Q5* Q6	R2
p4	Q1*!Q2* Q3	R1	p12	Q4* Q5* Q6	L2
p5	Q1* Q2* Q3	L1	p13	Q4* Q5*!Q6	L2
p6	Q1* Q2*!Q3	L1	P14	Q3*!Q6	W1
P9	!Q4* Q5*!Q6	R2	P15	!Q3* Q6	W2 (z)

4.2. MAPPING OF CONCURRENT STATE MACHINE (CSM) INTO PROGRAMMABLE LOGIC

The implementation of the control algorithm represented by Petri net is fixed usually at the design stage. Petri net together with related inputs and outputs is mapped into a network of interconnected logic blocks. The direct mapping of Petri net into an FPGA device is based on the correspondence between a transition and a simple combinational circuit and the correspondence between a place and a clearly defined subset of state register [6]. A recent overview of Petri net based direct implementation of logic controllers is given in [19, 65]. The different coding styles for concurrent Logic Controllers are summarized in [19, 59, 60, 61, 75]. The general structure for mapping concurrent Logic Controllers in programmable Logic was introduced in [1]. The modified architecture is shown in Fig. 12. Global state register is implemented using JK-style flip-flops. Declarative specification for transitions encoder, excitation function for JK flip-flops (Table 10) and decoder outputs (Table 11) is as follows:

t1 = AND(!Q1, !Q2, !Q3, Start1)
t2 = AND(!Q1, Q2, !Q3, D)
...
t12 = AND(Q4, Q5, !Q6, C)

J1 = OR(t2)
J2 = OR(t1, t4)
...
K5 = OR(t8, t12)
K6 = OR(t11)

L1 = OR(AND(Q1, Q2, Q3), AND(Q1, Q2, !Q3))

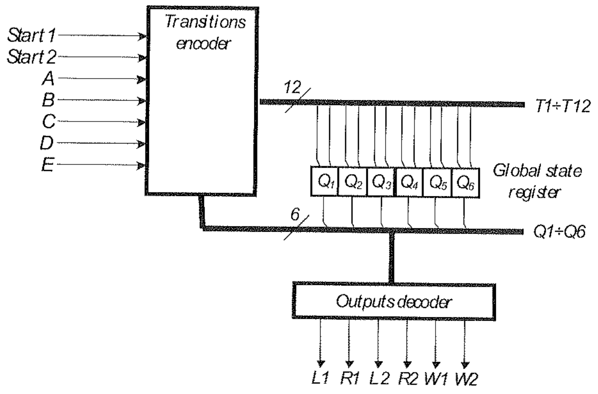


Fig. 12. Structure of the control unit (binary encoding)

Table 10

Decision table for the control unit (for Set/Reset or JK-style flip-flops)

Transition No.	Current states code	Conditions	Next states code	Set/Reset (JK)
1	!Q1*!Q2*!Q3	Start1	!Q1*Q2*!Q3	J2
2	!Q1* Q2*!Q3	D	Q1*!Q2*!Q3	J1 K2
3	Q1*!Q2*!Q3*!Q6	D	Q1*!Q2* Q3*!Q6	J3
4	Q1*!Q2* Q3	B	Q1* Q2* Q3	J2
5	Q1* Q2* Q3*!Q6	D	Q1* Q2*!Q3*!Q6	K3
6	Q1* Q2*!Q3	A	!Q1*!Q2*!Q3	K1 K2
7	!Q4*!Q5*!Q6	Start2	!Q4* Q5*!Q6	J5
8	!Q4* Q5*!Q6	E	Q4*!Q5*!Q6	J4 K5
9	!Q3* Q4*!Q5*!Q6	!D	!Q3* Q4*!Q5* Q6	J6
10	Q4*!Q5* Q6	B	Q4* Q5* Q6	J5
11	!Q3* Q4* Q5* Q6	E	!Q3* Q4* Q5*!Q6	K6
12	Q4* Q5*!Q6	C	!Q4*!Q5*!Q6	K4 K5

The direct implementation of concurrent controllers in FPGA is similar to the realizations of logic controllers based on FSM presented in the books [13, 25, 26, 27]. The main essential difference is concurrent state assignment (place encoding of Petri net). The logic controller contains a concurrent local states register, serving also as

Table 11

Encoded outputs table

Outputs	Local state code
R1	!Q1* Q2*!Q3 + Q1*!Q2* Q3
R2	!Q4* Q5*!Q6 + Q4*!Q5* Q6
L1	Q1* Q2* Q3 + Q1* Q2*!Q3
L2	Q4* Q5* Q6 + Q4* Q5*!Q6
W1	Q3*!Q6
W2	!Q3* Q6

a global state register. The combination of the code words of individual local states produces a unique configuration encoding. The superposition of codes of any two concurrent local states can share logic variables, but must be represented by words (ternary Boolean vectors), with non-overlapping, complete independent parts. Local states, which can never be concurrent, may also share a part of logic variables, but they must have a common overlapping part, with different values of logic variables.

When a Petri net is used to model a Concurrent State Machine (CSM), places represent its local states. A maximal subset of simultaneously marked places determines the global state of the controller. Transitions describe the local state changes, mostly forced by the external inputs. For simplicity, it will be considered that the CSM is implemented as a sequential circuit with a common internal clock. The controller, whose output depends on both internal state and external inputs, is modeled as a Mealy State Machine. The Moore type output by definition is implemented by a combinational cell as a function of state variables. On the other hand, the Moore type-output may be produced in advance in a registered output cell, because it must be stable for the entire clock period. Registered outputs can be eventually used for local state encoding [71, 75].

The local state encoding (place encoding) guarantees that all enabled transitions can fire independently, in any allowed order, not necessary exactly with the same edge of the clock.

An undesirable situation in interpreted Petri net occurs when two (or more) transitions, such as *t3* and *t9*, attempt to simultaneously unmark the same shared input place *p7* (Fig. 3). It is considered that the behavior of the net is deterministic since such possible conflict is previously eliminated by the consistent labeling of transitions by guards *D* and *!D*, respectively.

On the other hand, the global state register can be implemented using T-style flip-flops. The specification for the excitation function for T flip-flops (Table 12) is as follows (for avoiding of ambiguity, T input is denoted as QT):

QT1 = OR(t2, t6)
QT2 = OR(t1, t2, t4, t6)
...
QT6 = OR(t9, t11)

Table 12

Decision table for the control unit (for T-style flip-flops)

Transition No.	Current states code	Conditions	Next states code	Q changes (T)
1	!Q1*!Q2*!Q3	Start1	!Q1*Q2*!Q3	QT2
2	!Q1* Q2*!Q3	D	Q1*!Q2*!Q3	QT1 QT2
3	Q1*!Q2*!Q3*!Q6	D	Q1*!Q2* Q3*!Q6	QT3
4	Q1*!Q2* Q3	B	Q1* Q2* Q3	QT2
5	Q1* Q2* Q3*!Q6	D	Q1* Q2*!Q3*!Q6	QT3
6	Q1* Q2*!Q3	A	!Q1*!Q2*!Q3	QT1 QT2
7	!Q4*!Q5*!Q6	Start2	!Q4* Q5*!Q6	QT5
8	!Q4* Q5*!Q6	E	Q4*!Q5*!Q6	QT4 QT5
9	!Q3* Q4*!Q5*!Q6	!D	!Q3* Q4*!Q5* Q6	QT6
10	Q4*!Q5* Q6	B	Q4* Q5* Q6	QT5
11	!Q3* Q4* Q5* Q6	E	!Q3* Q4* Q5*!Q6	QT6
12	Q4* Q5*!Q6	C	!Q4*!Q5*!Q6	QT4 QT5

4.3. HDL MODELING AND SYNTHESIS OF ENCODED PETRI NET

The encoded Petri net model is converted into Verilog conserving the initial rule-based specification. It provides a path to commercial simulation and synthesis tools. The model with the specification of excitation functions for JK flip-flops is presented in Fig. 13. Because of the fact that in the considered FPGA devices there are D flip-flops, therefore the specification of global state register (with necessary conversion JK into D) is modeled as always process. Fig. 14 shows differences between models based on JK and T flip-flips: excitation function for T flip-flop and process with the specification of global state register (with necessary conversion T into D).

The Verilog models were simulated in the professional environment Mentor Graphics ModelSim v.6.4a. The considered controller was implemented as a simple example into an FPGA device using the standard Xilinx ISE 10.1.3 CAD/CAE tool. The results of the implementations are shown in Table 13. In addition, the set of benchmarks considering five different styles of place encoding [75] is under development.

```
module TankersControl_Coded_JK (CLK, Reset, A, B, C, D, E, Start1, Start2,
                                L1, L2, R1, R2, W1, W2);
input CLK, Reset;
input A, B, C, D, E, Start1, Start2;
output L1, L2, R1, R2, W1, W2;

wire [1:12] t;
reg  [1:6] Q;

assign  t[1] = ~Q[1] & ~Q[2] & ~Q[3]          &  Start 1;
assign  t[2] = ~Q[1] &  Q[2] & ~Q[3]          &  D;
assign  t[3] =  Q[1] & ~Q[2] & ~Q[3] & ~Q[6]  &  D;
assign  t[4] =  Q[1] & ~Q[2] &  Q[3]          &  B;
assign  t[5] =  Q[1] &  Q[2] &  Q[3] & ~Q[6]  &  D;
assign  t[6] =  Q[1] &  Q[2] & ~Q[3]          &  A;
assign  t[7] = ~Q[4] & ~Q[5] & ~Q[6]          &  Start 2;
assign  t[8] = ~Q[4] &  Q[5] & ~Q[6]          &  E;
assign  t[9] = ~Q[3] &  Q[4] & ~Q[5] & ~Q[6]  & ~D;
assign t[10] =  Q[4] & ~Q[5] &  Q[6]          &  B;
assign t[11] = ~Q[3] &  Q[4] &  Q[5] &  Q[6]  &  E;
assign t[12] =  Q[4] &  Q[5] & ~Q[6]          &  C;

wire [1:6] J, K;
assign J[1] = t[2];
assign J[2] = t[1] | t[4];
assign J[3] = t[3];
assign J[4] = t[8];
assign J[5] = t[7] | t[10];
assign J[6] = t[9];
assign K[1] = t[6];
assign K[2] = t[2] | t[6];
assign K[3] = t[5];
assign K[4] = t[12];
assign K[5] = t[8] | t[12];
assign K[6] = t[11];

integer i;
always @(posedge CLK)
             if (Reset) Q <= 6'b000000;
             else for (i = 1; i <= 6; i = i+1)
                 Q[i] <= (~Q[i] & J[i])  |  (Q[i] & ~K[i]);

assign L1 =  Q[1] &  Q[2] &  Q[3]  |   Q[1] &  Q[2] & ~Q[3];
assign L2 =  Q[4] &  Q[5] &  Q[6]  |   Q[4] &  Q[5] & ~Q[6];
assign R1 = ~Q[1] &  Q[2] & ~Q[3]  |   Q[1] & ~Q[2] &  Q[3];
assign R2 = ~Q[4] &  Q[5] & ~Q[6]  |   Q[4] & ~Q[5] &  Q[6];
assign W1 =  Q[3] & ~Q[6];
assign W2 = ~Q[3] &  Q[6];

//only for testing purpose (removing during opimization)
wire [1:15] p;
assign p[1]  = ~Q[1] & ~Q[2] & ~Q[3];
assign p[2]  = ~Q[1] &  Q[2] & ~Q[3];
assign p[3]  =  Q[1] & ~Q[2] & ~Q[3];
assign p[4]  =  Q[1] & ~Q[2] &  Q[3];
assign p[5]  =  Q[1] &  Q[2] &  Q[3];
assign p[6]  =  Q[1] &  Q[2] & ~Q[3];
assign p[7]  = ~Q[3] & ~Q[6];
assign p[8]  = ~Q[4] & ~Q[5] & ~Q[6];
assign p[9]  = ~Q[4] &  Q[5] & ~Q[6];
assign p[10] =  Q[4] * ~Q[5] * ~Q[6];
assign p[11] =  Q[4] & ~Q[5] &  Q[6];
assign p[12] =  Q[4] &  Q[5] &  Q[6];
assign p[13] =  Q[4] &  Q[5] & ~Q[6];
assign p[14] =  Q[3] & ~Q[6];
assign p[15] = ~Q[3] &  Q[6];

endmodule
```

Fig. 13. Verilog model with place encoding (for JK-type flip_flop)

```
module TankersControl_Coded_T (CLK, Reset, A, B, C, D, E, Start1, Start2,
                               L1, L2, R1, R2, W1, W2);
...

wire [1:6] QT;
assign QT[1] = t[2] | t[6];
assign QT[2] = t[1] | t[2] | t[4] | t[6];
assign QT[3] = t[3] | t[5];
assign QT[4] = t[8] | t[12];
assign QT[5] = t[7] | t[8] | t[10] | t[12];
assign QT[6] = t[9] | t[11];

integer i;
always @(posedge CLK)
                if (Reset) Q = 6'b000000;
                else for (i = 1; i <= 6; i = i+1)
                    Q[i] = (!Q[i] & QT[i])  |  (Q[i] & !QT[i]);

...
endmodule
```

Fig. 14. A part of Verilog model with place encoding (for T-type flip-flop)

Table 13

Synthesis summary (Xilinx FPGA – XC3S200pq208-5)

Model	Slices #	FF #	LUT4 #
TankersControl	18	15	19
TankersControl_Coded_JK	9	6	18
TankersControl_Coded_T	9	6	18

5. RELATED WORKS

The paper is concentrated on behavioral specification of Reconfigurable Logic Controller programs, mapped from Petri nets into HDL [11, 40, 54, 57, 69, 77]. The work introduced at the University of Zielona Góra was extended at several universities abroad [2, 5]. The result of collaboration with the University of Bristol, UK, is reported for example in papers [31, 47, 57, 58]. Some parts of the work have been realized at The University of Minho, Braga, Portugal [15, 40, 74], Fern University in Hagen [43], the Technical University of Ilmenau, Germany [39] and Academy of Science of Byelorussia [18].
The current research is especially related to Petri net-based structured state assignment for Concurrent State Machines, different kinds of Petri net decompositions, the symbolic exploration of Petri net state space, etc. [11, 24, 46, 48, 49, 52, 66, 69].

The formats PNSF [47], PNSF2 [67], PNSF3 [66] support the structured, hierarchical designs with Petri nets and FPGAs. The CONPAR specification format [40] is consistent with the previously introduced rule-based specification languages and was also created mainly as a bridge between the textual logic description of Petri nets and their VHDL models. Transition rules in PARIS and CONPAR are treated as production rules ('if-then' non procedural statements). Petri nets can be also specified in the newer textual formats – PNSF3 (Petri Net Specification Format v.3) and CCPNML (Concurrent Control PNML) [33]. PNSF3 represents interpreted, synchronous, hierarchical and colored Petri nets, and it is specified in the XML language.

The book [51] makes the connection between digital electronic design with Programmable Logic Devices (PLDs) and Programmable Logic Controllers (PLCs). The design of Petri net based controllers is summarized in [19, 35, 65]. Several aspects related to hardware design with Petri net can be found in books [36, 79]. The methodology for digital design of concurrent (parallel) controllers from Sequential Function Charts (SFC) and related Petri nets has been in development for several years and is presented in papers [8, 15, 68]. Hierarchical specification mechanism is introduced by means of macroplaces to allow the encapsulation of subnets as macronodes, which decreases the size of the specification, improves its readability and introduces modularity [10, 19]. Reconfigurable architectures for controllers based on Petri nets are described also in [56].

6. CONCLUSIONS

The paper concentrates on the behavioral specification of RLC programs, given initially as Petri nets and later rewritten in Hardware Description Languages. Some engineering notations like Sequential Function Chart (IEC 61131-3), as well as those mainly used by computer scientists, like Petri nets are integrated into a unified design methodology. By formally verifying the structural properties of Petri net the behavioral properties of control program such as reversibility, liveness and safeness are tested.
Hardware description languages, such as VHDL or Verilog, are used for an intermediate representation of controller behavior on top of existing commercial synthesis tools.
The rule-based textual language input makes it possible to integrate the design system with existing formal logic based computer-based theorem proovers. The Petri net description in HDL that is devoted to provide the opportunity to integrate existing Petri net software with several commercial systems.
The more advanced research, among other topic, would concentrate on:

- Unified Design of Concurrent Logic Controllers with Data Path [26];
- Effective structured state assignment and decomposition techniques devoted to the mapping of Petri net-based controllers into embedded modern microsystems as SoPCs (*System-on-Programmable-Chips*).
- Extensive application of formal methods for the analysis and synthesis of Concurrent State Machines, which are implemented in dynamically reconfigurable arrays.

7. REFERENCES

1. M. A d a m s k i: *PLA-realization by means of connecting and control algorithms*. Proc. of the 29th International Scientific Colloquium, IWK'84, Ilmenau, Germany, 1984, pp.43-45.

2. M. Adamski: *Heuristic method of structural encoding of Petri Net places*. Zeszyty Naukowe WSI, No. 78, Technical University of Zielona Gora, 1986, (in Polish).
3. M. Adamski: *Direct Implementation of Petri Net Specification*. Proceedings of the 7th International Conference on Control Systems And Computer Science CSCS'87, Bucharest, Romania, 1987, Vol.3, pp.74-85.
4. M. Adamski: *Digital System Design by Formal Transformation of Specification*. Proc. of the 35th International Scientific Colloquium, IWK'90, Ilmenau, Germany, 1990, Heft 3, pp.62-65.
5. M. Adamski: *Digital Systems Design by Means of Rigorous and Structural Method*. Technical University of Zielona Gora Press, Zielona Góra, 1990 (in Polish).
6. M. Adamski: *Parallel Controller Implementation using Standard PLD Software*. in: FPGAs (W.R.Moore, W.Luk (Ed)), Abingdon EE&CS Books, Abingdon, England, 1991, pp.296-304.
7. M. Adamski: *Petri Nets in ASIC Design*. Applied Mathematics and Computer Science, 1993, Vol.3, No.1, pp.169-180.
8. M. Adamski: *Application Specific Logic Controllers for Safety Critical Systems*. Proc. of the Triennial IFAC World Congress, Beijing, China, Pergamon Press, 1999, Vol. Q, pp. 519-524.
9. M. Adamski: *Specification and Synthesis of Petri Net based Reprogrammable Logic Controller*. Programmable Devices and Systems 2001 (PDS 2001): a proceedings volume from the 5th IFAC Workshop, Pergamon, London, 2002, pp.95-100.
10. M. Adamski: *Behavioural Specification of Programs for Modular Reconfigurable Logic Controllers*. Proceedings of the Mixed Design of Integrated Circuits and Systems, MIXDES 2006, Gdynia, Poland, pp.239-244.
11. M. Adamski: *Reconfigurable logic controller for embedded applications*. In: Discrete-Event System Design 2006, A Proceedings volume from the IFAC Workshop, DESDes'06, (M.Adamski, L.Gomes, M.Węgrzyn, G.Łabiak (Ed)), pp.147-152, University of Zielona Góra Press, Zielona Góra, 2006.
12. M. Adamski: *Logic design of reconfigurable controllers*. Proceedings of the IEEE Symposium on Industrial Embedded Systems, SIES 2007, Lisbon, 4-6 July 2007, pp.373-376.
13. M. Adamski, A. Barkalow: *Architectural and sequential synthesis of digital devices*. University of Zielona Góra Press, Zielona Góra, 2006.
14. M. Adamski, J. L. Monteiro: *Rule-based formal specification and implementation of Logic Controllers programs*. Proc. of the IEEE International Symposium on Industrial Electronics, ISIE'95, Athens, Greece, 1995, Vol.2, pp.700-705.
15. M. Adamski, J. L. Monteiro: *Declarative Specification of System Independent Logic Controller Programs*. Proc. of the IEEE International Symposium on Industrial Electronics ISIE'96, Warsaw, Poland, 1996, pp.305-310.
16. M. Adamski, M. Węgrzyn: *Hierarchically Structured Coloured Petri Net Specification and Validation of Concurrent Controllers*. Proc. of the 39th International Scientific Colloquium, IWK'94, Ilmenau, Germany, 1994, Band 1, pp.517-522.
17. M. Adamski, M. Węgrzyn: *Field Programmable Implementation of Concurrent State Machine*. Proceedings of the 3rd international conference on Computer-Aided Design of Discrete Devices, CAD DD'99, Minsk, Belarus, 1999, Vol.1, pp.4-12.
18. M. Adamski, A. Zakrevskij: *Formal specification of reactive logical control devices*. Proc. of the World Multiconference on Systemics, Cybernetics and Informatics, SCI 2001, Orlando, USA, 2001, Vol.14, Computer Science and Engineering, pp.428-433.
19. M. Adamski, A. Karatkevich, M. Węgrzyn (Ed): *Design of Embedded Control Systems*. Springer, New York, 2005.
20. M. Adamski, M. Węgrzyn, A. Węgrzyn: *Safe reconfigurable logic controllers design*. In: Measurement, Models, Systems and Design, J.Korbicz (Ed), WKiŁ, Warszawa, 2007, pp.343-367.
21. M. Adamski, M. Węgrzyn, P. Wolański: *Simulating and Synthesising of Reconfigura-*

ble Logic Controllers using VHDL. Proc. of the 42nd International Scientific Colloquium, IWK'97, Ilmenau, Germany, 1997, Band 1, pp.522-527.

22. A. Amroun, M. Bolton: *Synthesis of controllers from Petri net descriptions and application of Ella.* Proceedings of the IMEC-IFIP International Workshop on Applied Formal Methods for Correct VLSI Design, North Holland, 1989, pp.57-74.
23. D. Andreu, G. Souquet, T. Gill: *Petri Net based rapid prototyping of digital complex system.* Proceedings of the 2008 IEEE Computer Society Annual Symposium on VLSI, 2008, pp.405-410.
24. G. Andrzejewski: *Program model of interpreted Petri net for digital microsystems design.* Ph.D. Thesis, Szczecin University of Technology, Faculty of Information Technology, Szczecin, 2002 (in Polish).
25. S. Baranov: *Logic Synthesis for Control Automata.* Kluwer Academic Publishers, Boston, 1994.
26. S. Baranov: *Logic and System Design for Digital Systems.* Tallin University Press, Tallin, 2008, SiB Publishers, Toronto, 2008.
27. A. Barkalow, M. Węgrzyn: *Design of control units with programmable logic.* University of Zielona Góra Press, Zielona Góra, 2006.
28. H. Belhadj, L. Gerbaux, M.-C. Bertrand, G. Saucier: *Specification and Synthesis of Communicating Finite State machines.* Synthesis for Control Dominated Circuits. (A-22), Elsevier Science Publishers B.V. (North Holland), 1993, pp.91-101.
29. M. Ben-Ari: *Mathematical Logic for Computer Science.* Springer, London, 2001.
30. K. Biliński: *Application of Petri Nets in parallel controllers design.* Ph.D. Thesis, University of Bristol, Electrical and Electronic Engineering Department, Bristol, 1996.
31. K. Biliński, M. Adamski, J. M. Saul, E. L. Dagless: *Petri net based algorithms for parallel controller synthesis.* IEE Proceedings, Part E: Computers and Digital Techniques, 1994, Vol.141, No.6, pp.405-412.
32. K. Biliński, M. Adamski, J. M. Saul, E. L. Dagless: *Parallel controller synthesis from a Petri net specification.* Proceedings of EDAC'94, IEEE, 2004, pp. 96-101.
33. J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, M. Weber: *The Petri Net Markup Language: Concepts, Technology, and Tools.* Proceedings of the ICATPN 2003, (W.M.P. van der Aalst, E.Best (Ed)), LNCS, 2003, Vol. 2679, Eindhoven, Netherlands, Springer-Verlag, pp.483-505.
34. P. Bubacz, M. Adamski: *Heuristic algorithm for an effective state encoding for reconfigurable matrix-based logic controller design.* Proceedings of the IFAC Workshop on Programmable Devices and Embedded Systems, PDeS 2006, Brno, Czech Republic, 2006, pp.236-241.
35. N. Chang, W. H. Kwon, J. Park: *Hardware implementation of real time Petri-net-based controllers.* Control Engineering Practice, 1998, Vol.6, No.7, pp.889-895.
36. J. Cortadella, A. Yakovlev, G. Rozenberg: *Concurrency and Hardware Design.* Advances in Petri Nets, LNCS, Vol.2549, Springer, Berlin, 2002.
37. R. David, H. Alla: *Petri Nets and Grafcet.* Prentice Hall Int., USA, 1992.
38. P. Eles, K. Kuchciński, Z. Peng: *System Synthesis with VHDL.* Kluwer Academic Publishers, Boston, 1998.
39. W. Fengler, A. Wendt, M. damski, J. L. Monteiro: *Petri Net based Program Design and Implementation for Controller Systems.* Proc. of the 1996 IFAC Triennial World Congress, San Francisco, CA, USA, 1996, Vol.J, pp.425-429.
40. J. M. Fernandes, M. Adamski, A. J. Proença: *VHDL Generation from Hierarchical Petri Net Specifications of Parallel Controllers.* IEE Proceedings, Part E: Computers and Digital Techniques, 1997, Vol.144, No.2, pp.127 137.
41. L. Gomes, J. P. Barros, A. Costa: *Modeling Formalisms for Embedded System Design,* In: R. Zurawski, Embedded Systems Handbook. CRC Press, Inc., 2006, pp.5.1-5.34.
42. C. Girault, R. Valk: *Petri Nets for System Engineering.* Springer, Berlin, 2003.
43. W. Halang, M. Adamski: *A Programmable Electronic System for Safety Related Control*

Applications. Proc. of the International Conference on Safety and Reliability, ESREL'97, Lisbon, Portugal, 1997, pp.349 356.

44. K. Jensen: *Coloured Petri Nets. Basic Concept, Analysis Methods and Practical Use*, Volume 1, Basic Concepts, Springer-Verlag, Berlin, 1992.
45. J. Kalinowski, T. Łuba: *Metodą syntezy logicznej układów cyfrowych opisywanych sieciami Petriego*. Rozprawy Elektrotechniczne, 1986, T.32, Z.4, pp.1253-1263.
46. A. Karatkevich: *Dynamic Analysis of Petri Net-Based Discrete Systems*, Lecture Notes in Control and Information Sciences, Vol. 356, Springer, 2007.
47. T. Kozłowski, E. L. Dagless, J. M. Saul, M. Adamski, J. Szajna: *Parallel controller synthesis using Petri nets*. IEE Proceedings, Part E: Computers and Digital Techniques, 1995, Vol.142, No.4, pp.263-271.
48. G. Łabiak: From UML statecharts to FPGA – the Hicos approach. Forum on Spec. Design Languages, Frankfurt, 2003.
49. G. Łabiak: *The Use of Hierarchical Model of Concurrent Automaton in Digital Controller Design*. Ph.D. Thesis, Warsaw University of Technology, Faculty of Electronics and Information Technology, Warsaw, 2003 (in Polish).
50. J. R. Machado, J. M. Fernandes, A. J. Proenca: *Specification of Industrial Controllers with Object-Oriented Petri Nets*. Proc. of the IEEE International Symposium on Industrial Electronics, ISIE'97, Guimaraes, Portugal, 1997, pp. 77-82.
51. E. Mandado, J. Marcos, S. A. Perez: *Programmable Logic Devices and Logic Controllers*. Prentice Hall, London, 1996.
52. P. Miczulski, M. Adamski: *Analyses of safeness, liveness and persistence properties of Petri nets by means of monotype logic functions*. In: Discrete-Event System Design 2006, A Proceedings volume from the IFAC Workshop, DESDes'06, (M.Adamski, L.Gomes, M.Węgrzyn, G.Łabiak (Ed)), pp.137-142, University of Zielona Góra Press, Zielona Góra, 2006.
53. A. Milik, E. Hrynkiewicz: *Reconfigurable Logic Controller, Architecture, Programming, Implementation*. Proc. of the IFAC Workshop on Programmable Devices and Systems, PDS 2001, Gliwice, 2001.
54. P. Minns, I. Elliott: *FSM based Digital Design using Verilog HDL*. John Wiley & Sons, Ltd., Chichester, England, 2008.
55. T. Murata: *Petri Nets: Properties, Analysis and Applications*. Proceedings of the IEEE, 1989, Vol.77, No.4, pp.541-580.
56. P. S. B. Nascimento, P. R. M. Maciel, M. E. Lima, R. E. Santana, A. G. S. Filho: *A Partial Reconfigurable Architecture for Controllers based on Petri Nets*. Proc. of the SBCCI 2004, pp.16-21.
57. J. Pardey, M. Bolton: *Parallel controller synthesis for concurrent data paths*. Proc. of the IFIP Workshop Control Dominated Synthesis From a Register Transfer Level Description, 1992, pp.16-19.
58. J. Pardey, T. Kozłowski, J. Saul, M. Bolton: *State Assignment Algorithms for Parallel Controller Synthesis*. Proceedings of the IEEE International Conference on Computer Design, IEEE Computer Society Press, 1992, pp.316-319.
59. J. Pardey, A. Amroun, M. Bolton, M. Adamski: *Parallel Controller Synthesis for Programmable Logic Devices*. Microprocessors and Microsystems, 1994, Vol.18, No.8, pp.451 458.
60. E. Pastor, J. Cortadella: *Efficient Encoding Schemes for Symbolic Analysis of Petri Nets*. Design, Automation and Test in Europe,1998 , pp.790-795.
61. E. Pastor, J. soCortadella, J. Roig: *Symbolic Analysis of Bounded Petri Nets*. IEEE Transactions on Computers, 2001, Vol.50, No.5, pp.432-448.
62. M. Patel: *Random Logic Implementation of Extended Timed Petri nets*. Microprocessing and Microprogramming, 1990, Vol.30, No.1-5, pp.313-319.
63. J. Pleyber, M. Silva: *Software specification language for sequential process*. IFAC-IFIP Workshop on Real-Time Programming, Eindhoven, June 1977, pp.67-73.

64. L. Scheffer, L. Lavagno, G. Martin: *EDA for IC System Design, Verification, and Testing* (Electronic Design Automation for Integrated Circuits Handbook). CRC Press, Inc., 2005.
65. A. Yakovlev, L. Gomes, L. Lavagno (Ed): *Hardware Design and Petri Nets*. Kluwer Academic Publisher, Boston, 2000.
66. A. Węgrzyn: *Symbolic Analysis of Logical Control Devices using Selected Methods of Petri Net Analysis*. University of Zielona Góra Press, Zielona Góra, 2003 (in Polish).
67. M. Węgrzyn: *Hierarchical implementation of Logic controllers by means of Petri nets and FPGAs*. Ph.D. Thesis, Warsaw University of Technology, Faculty of Electronics and Information Technology, Warsaw, 1998 (in Polish).
68. M. Węgrzyn: *Implementation of Safety Critical Logic Controller by Means of FPGA*. Annual Reviews in Control, 2003, Vol.27, pp.55-61.
69. M. Węgrzyn: *Petri Net Decomposition Approach for Partial Reconfiguration of Logic Controllers*. In: Discrete-Event System Design 2006, A Proceedings volume from the IFAC Workshop, DESDes'06, (Adamski, M., L.Gomes, M.Węgrzyn, G.Łabiak (Ed)), pp.323-328, University of Zielona Góra Press, Zielona Góra, 2006.
70. M. Węgrzyn: *PLD-based implementation of concurrent controllers*. Electronics and Telecommunications Quarterly, 1996, T.42, Z.2, pp.235-251 (In Polish).
71. M. Węgrzyn, M. Adamski: *Synthesis of Logic Controller Uisng Standards FPLD Compilers*. Electronics and Telecommunications Quarterly, 1997, T.43, z.3, pp.353-372 (in Polish).
72. M. Węgrzyn, M. Adamski: *Hierarchical Approach for Design of Application Specific Logic Controller*. Proc. of the IEEE International Symposium on Industrial Electronics, ISIE'99, Bled, Slovenia, 1999, Vol.3, pp.1389-1394.
73. M. Węgrzyn, M. Adamski, J. L. Monteiro: *The Application of Reconfigurable Logic to Controller Design*. Control Engineering Practice, Special Section on Custom Processes, 1998, Vol.6, No.7, pp.879-887.
74. M. Wegrzyn, M. Adamski, J. L. Monteiro: *The application of reconfigurable logic to controller design*. Control Engineering Practice, 1998, Vol.6, No.7, pp.879-887.
75. M. Węgrzyn, P. Wolański, M. Adamski, J. Monteiro: *Field programmable device as a Logic Controller*. Proceedings of the 2nd Conference on Automatic Control, Control '96. Oporto, Portugalia, 1996, Vol. 2, pp. 715-720.
76. M. Węgrzyn, P. Wolański, M. A. Adamski, J. L. Monteiro: *Coloured Petri Net Model of Application Specific Logic Controller Programs*. IEEE International Symposium on Industrial Electronics ISIE'97, Guimaraes, Portugal, 07-11.07.1997, Vol.I, pp.SS158-SS163.
77. P. Wolański, M. Węgrzyn, M. Adamski: *VHDL modelling of industrial control systems*. Proc. of the 42nd International Scientific Colloquium, IWK'97, Ilmenau, Germany, Band 1, 1997, pp.528-533.
78. A. Zakrevskij: *CAD of discrete devices implementing parallel logical control algorithms*. Proc. of the Second International Symposium on Methods and Models in Automation and Robotics, Miedzyzdroje, Poland, 1995, pp.803-808.
79. A. Zakrevskij: *Parallel Algorithms for Logical Control*. Institute of Engineering Cybernetics of NAS of Bielarus, Minsk, 1999 (in Russian).

and Testing
.
ets. Kluwer
of Petri Net
and FPGAs.
Technology,
GA. Annual
Controllers.
DESDes'06,
Góra Press,
Telecommu-
Compilers.
ecific Logic
E'99, Bled,
ble Logic to
998, Vol.6,
ble logic to
able device
96. Oporto,
i Net Model
n Industrial
rol systems.
nd 1, 1997,
ithms. Proc.
otics, Mie-
bernetics of

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 183–200

Synthesis of finite state machines for implementation with programmable structures*

TADEUSZ ŁUBA, GRZEGORZ BOROWIK, ANDRZEJ KRAŚNIEWSKI

Warsaw University of Technology
Institute of Telecommunications, Poland
{luba, G.Borowik, andrzej}@tele.pw.edu.pl

Received 2009.01.06
Authorized 2009.04.01

Sensible application of programmable structures to the realization of digital systems cannot take place without computer aided design systems. It is particularly important when the design is intended for novel programmable structures containing LUT-based cells and embedded memory blocks, since traditional methods for technology mapping are oriented towards gate structures and based on minimization and factorization of Boolean functions.

This article focuses on finite state machine synthesis including logic optimization techniques, the technology mapping techniques, and the techniques that provide the resulting circuits with concurrent error detection capability. It is shown that a considerably more effective method of synthesis intended for CPLD and FPGA structures is based on the decomposition scheme.

Keywords: decomposition, state encoding, sequential circuit, finite state machine, FPGA, embedded memory block, logic cell, multi-graph

1. INTRODUCTION

In today's technologies, the density of elements is reaching hundreds of millions of transistors per digital circuit, and 100 millions of gates per circuit.[1] Similar densities characterize programmable structures – up to millions of logic gates. This, along the possibility of reprogramming and reconfiguration, gives unprecedented possibilities of

* This paper was supported by Ministry of Science and Higher Education financial grants: SINGA-PUR/31/2006, N517 003 32/0583.

[1] According to the Altera company, FPGA Stratix devices have from 4 up to 43 million of elements when converting these into the number of logic gates.

implementing digital circuits using such structures. However, sensible application of programmable structures to the realization of digital systems cannot take place without computer aided design systems.

For a typical digital system, the design process consists of compilation, translation, synthesis, logic optimization and technology mapping. Although the final result of that process is a structure built of standard cells, logic cells, macroblocks and similar components, the characteristics of the system (the silicon area, speed, power etc.) depend considerably on the logic model of the digital system obtained from the translation of the specification in hardware description language. Therefore, the synthesis and logic optimization (taking place between compilation technology mapping) has a significant impact on the quality of the implementation.

This turns out to be especially important in the case of user programmable devices: CPLDs (Complex Programmable Logic Devices) and FPGAs (Field Programmable Gate Arrays). However, many full custom and semi-custom circuits are also characterized by a great susceptibility to "logic transformations". A tendency towards using logic synthesis not only to minimize logic resources required to implement the circuit, but also to solve other typical design problems, such as signal delay reduction or power consumption reduction, can be observed [IP99], [YSL05].

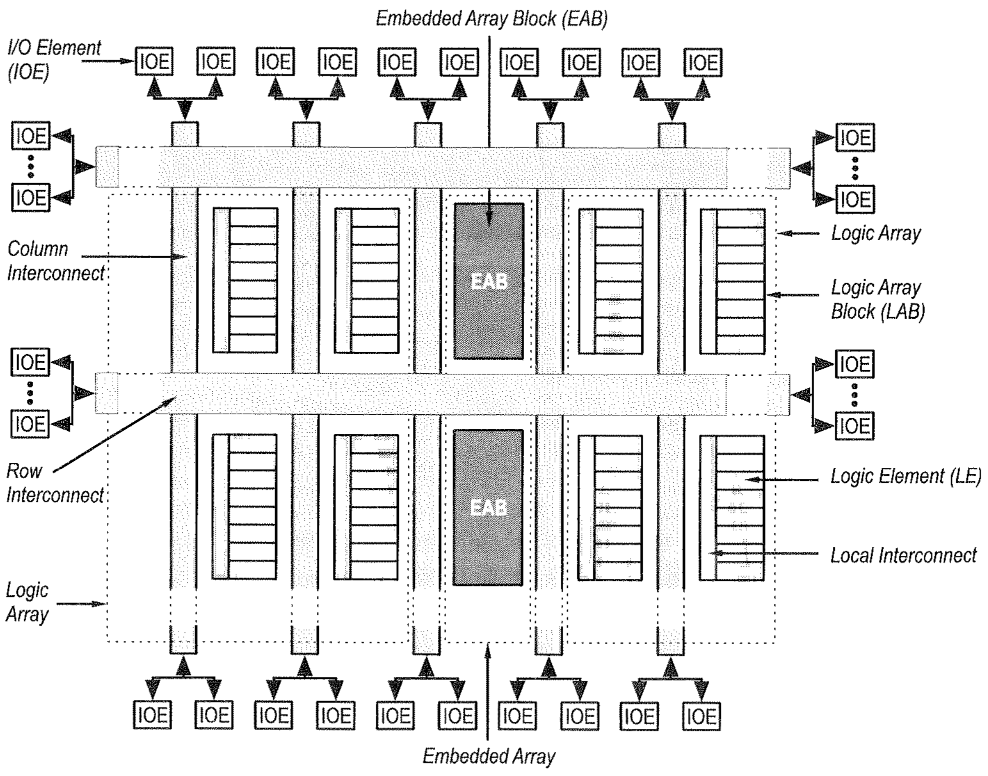


Fig. 1. FPGA with embedded memory blocks

The impact of advanced procedures of logic synthesis on the quality of implementation of digital circuits is particularly significant when the design is intended for programmable structures (Fig. 1) containing LUT-based cells (LE – Logic Elements) and embedded memory blocks, such as EABs (Embedded Array Blocks).

This is caused by the problems with the traditional methods for technology mapping, oriented towards gate structures and based on minimization and factorization of Boolean functions [DeM94]. These methods transform sum-of-product Boolean expressions into multilevel forms of strongly factorized expressions, which only then are realized in LUT cells. This approach is inefficient in the case when the implementation is based on logic cells designed to implement arbitrary logic functions.

For these reasons, a considerably more effective method of synthesis intended for CPLD and FPGA structures is based on the decomposition of Boolean functions. In this process a Boolean function is synthesized into a multilevel structure composed of logic blocks of the LUT type, specified by truth tables. The effectiveness of the functional decomposition was confirmed in many publications on the synthesis of combinational circuits [BL03], [CMSH96], [HK04], [HSB02], [Kan04], [LuS95], [LPP96], [PMG99], [RJL01], [Sch01], [SSP01]. A relatively small number of articles and papers describe the application of functional decomposition to the synthesis of sequential circuits [JC01], [JSC01], [JS00]. The reason for this is the computational complexity of functional decomposition procedures. Therefore, an effective application of functional decomposition in synthesis of finite state machines for implementation with user programmable devices requires a new design methodology that should be focused on:

- developing new algorithms for serial decomposition into subfunctions (tables) that fit embedded memory blocks,
- developing new methods for encoding internal states of sequential machines, suitable for implementation using programmable structures with embedded memory blocks.

The remainder of the paper is organized as follows: In Section 2 we summarize various techniques for sequential logic synthesis. This section contains two-level synthesis methods and multilevel synthesis methods from the literature. In Section 3, we illustrate ROM-based FSM synthesis methods which are especially efficient for today's programmable structures, particularly for FPGA devices with embedded memory blocks. In the next section we present self-testability and fault tolerance techniques for finite state machines implemented in considered programmable structures. Finally, in Section 5 we present our conclusions.

2. SYNTHESIS OF SEQUENTIAL MACHINES

In modern logic synthesis, regardless of the implementation technology (programmable devices, Gate Array or Standard Cell structures), the problem of finite state machine synthesis (in particular – the problem of internal state encoding) is an issue of significant practical importance. The internal state encoding affects both the structure of the FSM realization (i.e., connections between the combinational block and the memory block) and the complexity of the combinational block.

Many methods for structural synthesis of FSMs have been reported in the literature. Their diversity is a consequence of different assumptions taken to simplify calculations, as well as different types of intended target components. Thus, different methods of FSM synthesis have been developed for PLA structures [DBSV85], [DeM86], [VSV90], for ROM memories [Bor04], [RSL05], and for PLD modules [CK05].

Methods of state encoding that assume an FSM implementation with a PLA structure (including the case when PLA matrices are used as macrocells of ASIC circuits) are especially significant. For such a structure, the state (excitation) and output functions are treated as Boolean expressions with possible shared products (terms). In this case, the solution for state encoding is a binary representation of internal states of the FSM which results in the smallest total number of product terms in Boolean expressions (which represent all of its state and output functions). In the case of sequential machines, when solving this minimisation problem (common in combinational circuit synthesis) we have an additional degree of freedom in the form of selection of state encoding. However, due to its computational complexity, this optimization task is reduced to searching for internal state encodings that result in minimal usage of the PLA area.

A distinctive feature of traditional methods of FSM synthesis is the application of logical minimization before the process of state encoding. This minimization is possible when the inputs and outputs of the combinational part of the sequential circuit is represented with multi-valued symbolic variables. Unfortunately, such methods are limited to two-level structures. For other implementation styles different methods are needed. The research in this area goes into two directions: one concerns the implementation with multilevel gate structures, while the other embraces implementations with cellular FPGA and CPLD structures.

In the first case, like for two-level structures, the starting point of the synthesis process is a structure in which the combinational circuit is connected to the inputs of a register operating as state memory (Fig. 2a), whereas in the other case, the combinational circuit is connected to the outputs of such a register (Fig. 2b).

Until recently, mainly the first model (Fig. 2a) was used in synthesis of sequential machines. The optimization of the selection of state encoding was done for two-level or multilevel gate structures and was aimed at the reduction of hardware resources (silicon area).

The second model (Fig. 2b) was used in microprogrammed control circuits, with the combinational circuit implemented with ROM memory [AB06]. In the microprogrammed version of the sequential circuit, the fixed ROM memory was a separate element – separated from the rest of the circuit. The advantage of this structure was an ability to program the microcode memory, which was the only possible way to reconfigure the circuit at that time. These advantages made the capacity of the memory to be a non-critical factor, although the reduction of this capacity was a common optimization criterion. A typical approach was to construct the circuit with a special memory addressing unit called microprogram sequencer, connected to the microprogram memory through the Address Register (AR). The main function of the sequencer is to determine the address of the microinstruction to be executed. This address is transferred to the address register and is used to read the next microinstruction from ROM.

Microprogrammed control has been a very popular alternative implementation technique for control units. However, as systems have become more complex and new programmable technologies have appeared, the concept of classical microprogramming has become less attractive for control unit implementations. But the main idea of Microprogrammed Control Units, i.e. implementation of combinational part of the sequential circuit with a ROM, has gained new motivation after the appearance of programmable logic devices [BW06], [BT08]. In particular, the growing interest in ROM-based synthesis of finite state machines has been caused by the inclusion of embedded memory blocks in modern FPGAs.

In this situation, the main research work in the field of FSM synthesis can be classified into three distinctive areas, corresponding to three implementation styles for the combinational part of the sequential circuit:

- implementation with a two-level gate (cell) structure,
- implementation with a multilevel gate (cell) structure,
- implementation with a ROM.

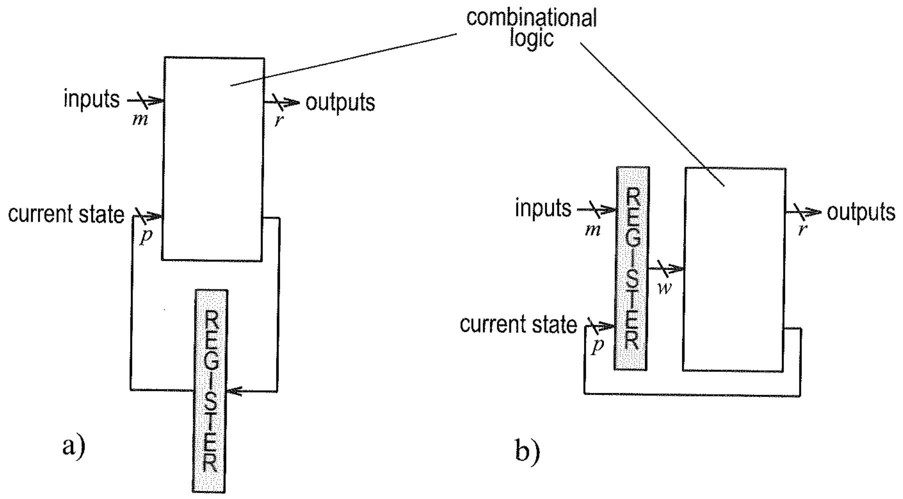


Fig. 2. Two models of a sequential circuit: a) classical, b) with microprogramming capability

Two-level synthesis

The two-level synthesis of finite state machines is based on symbolic minimization; its essence lies in the representation of the logic circuit with multi-valued variables.

In the symbolic (multi-valued) description, the binary realization is obtained only after the minimization of the multi-valued function, and the minimal multi-valued coverage determines the relations of minimal-length binary input codes.

The symbolic minimization was introduced by De Micheli in [DBSV85], where it was applied to encoding of multi-valued inputs symbols (the input encoding). In [DeM86] a heuristic algorithm is described, which can also consider the limitation resulting from the encoding of the output symbols. In [CY92], the problem of the symbolic minimization was reduced to graph coloring. The distinctive feature of this approach is the concurrent introduction of input and output limitations, which makes it possible to solve the tasks of input and output encoding. A similar approach was also introduced in [SB93], where an algorithm for input and output encoding of linear complexity is presented (the complexity of the algorithms described earlier is at least $O(n^2)$).

One of the most important benefits of symbolic minimization is its applicability to the internal state encoding of FSMs implemented with PLA structures. The essence of this approach lies in that logic minimization is done before state encoding. This minimization is possible when the symbolic representation of the combinational part of a sequential circuit is used.

In this case, a "symbolic coverage" is a set of basic elements called "symbolic implicants". A symbolic implicant consists of $n > 2$ component fields. Each field corresponds to a multi-valued variable, whose values are sequences of characters. In sequential circuits, the symbolic implicants have four fields ($n = 4$) that correspond to input, current state, next state and output. A symbolic implicant is represented by a quadruple $< x, s, s', y >$, where the first two fields (x, s) are the inputs of the symbolic implicant, while the other two (s', y) are its outputs.

Since only internal state encoding is considered, fields x and y have a binary representation and only fields s and s' have a symbolic representation – of the current state and the next state, respectively.

The essence of symbolic minimization lies in grouping together those states, which – in response to a given input – have the same next state and the same output. Thus, if the symbolic coverage is an implicant $(x, s, \delta(x, s), \lambda(x, s))$, where s represents a single state, the minimal multi-valued coverage can contain these symbolic implicants where s represents a set of states.

In [DN91], Devadas and Newton presented an exact algorithm for minimization of the number of terms and the realization of FSM combinational part with a PLA structure.

An exact algorithm for symbolic minimization with simultaneous input and output encoding was described in [ADN92]. This algorithm is executed in two stages: first, a set of generalized prime implicants is determined, and then the coverage of the matrix of encoding limitations is found.

In [SVBSV94], simultaneous input and output encoding is considered and it is shown that the problem is NP-hard. A polynomial-complexity algorithm for determining the existence of a solution that satisfies a given set of input and output limitations, and exact and heuristic algorithms for determining the minimum number of the code bits, for an encoding that satisfies all the limitations are also presented.

Coudert et al. [Cou98], [CS96] solve the problem of encoding for synchronous and asynchronous finite state machines. Their method is based on the theory of dichotomy and yields a safe solution for asynchronous machines and a minimal solution for synchronous ones.

The symbolic minimization was implemented in NOVA [VSV90] which is a part of the SIS package [SSL92].

In monograph [TKBSV98] new algorithms for symbolic minimization are presented, including those for the generation of generalized prime implicants, finding minimum symbolic coverage for given encoding limitations and finding the minimum-length codes. A distinctive feature of the presented algorithms is the application of BDD (Binary Decision Diagrams) and their modifications for the description of finite state machines.

Multilevel synthesis

The problem of internal state encoding for finite state machines whose combinational part is implemented as a multilevel structure was first discussed and solved by Devadas et al. [DMNSV88]. The main objective of the synthesis is the reduction of the area of the combinational circuit (when compared to the two-level implementation). The proposed algorithm for internal state encoding maximizes the the number of common cubes in the encoded network and minimizes the number of literals in the FSM combinational part. After encoding of internal states, a multilevel Boolean optimization is carried out. This approach was further examined in [ADN92], [DHLN91], and [WKA89].

Unfortunately, an FSM synthesis considering only the multilevel realization of the combinational part is not efficient enough; the classic decomposition problem, formulated back in middle of the 20th century has been revisited increasingly more frequently to solve this synthesis problem.

The problem of finite state machine decomposition was first formulated by Hartmanis and Stearns in [HS66]. Their solution was based on the theory of partitions; in particular, the introduction of the closed partition allowed for the formulation of the conditions of existence of parallel and serial decompositions of finite state machines.

A good application of decomposition to FSM synthesis was presented in [DN89]. In this paper, Devadas and Newton proposed a method of FSM synthesis targeting the optimization of the area and performance of the final circuit. This method is based on factorization of finite state machines.

The idea of the factorization of a finite state machine is to separate some of the FSM components and implement them as separate sequential machines – factors. In [DN89] algorithms for finding factors for a given FSM transition graph are given; in addition the concept of accurate factorization is introduced – the one yielding the smallest number of states and transitions. Methods for internal state encoding aimed at two- and multilevel realizations of FSMs are also shown.

In monograph [DeM94] several methods for multilevel synthesis are presented – both for combinational and sequential circuits. Most of these methods were implemented in Mustang [DMNSV88], JEDI [LN89] and other parts of the SIS package [SSL92].

Recent years have brought a noticeable progress in methods for finite state machine synthesis targeting CPLD and FPGA devices.

In [CK05] and [CKK06] a method of FSM internal state encoding is presented which considers the number of macrocell terms of the PAL structure. This method is based on assigning binary representations that differ on a single bit position to certain pairs of states. Additionally, the most common next states are assigned codes containing more bits corresponding to disabled output states.

One of the most common concepts leading to a reduction of realization complexity of FSMs implemented with PLD structures is the application of output flip-flops available in PLD macrocells as FSM memory components. This method is applicable to Moore sequential machines with output vectors identical to their corresponding internal states. A method for synthesis of such machines intended for PAL structures with registers was shown in [Sol97]. A distinctive feature of the proposed algorithm is the use of unspecified values of output variables for solving the problem of internal state encoding. If this approach does not make it possible to encode all of the FSM internal states, then a minimal number of additional memory elements is used. This method yields efficient results when applied to the synthesis of complex finite state machines with a large number of outputs. A similar idea is presented in [For95].

In [SLBG94] a method for FSM synthesis, including state encoding and subsequent optimization for implementation with FPGA structures is presented. In this approach Multi-ROBDD diagrams are used to describe a sequential machine. In [RSLS06], Rawski et al. present a method for FSM synthesis targeting FPGA architectures with LUT structures. Their approach relies on symbolic functional decomposition. Encoding algorithms based on cover algebra presented in [BL03] facilitate the search for efficient decompositions.

3. ROM-BASED SYNTHESIS

Although the methods discussed above can be effectively used for synthesis of FSM implemented with gates and flip-flops, they are not efficient for today's programmable structures, particularly for FPGA devices with embedded memory blocks. Such implementations would benefit from a structure with a separate memory block which is common in microprogrammable circuits. However, an advanced apparatus for design of address modifier is required to support the synthesis based directly on the FSM transition table.

A limited size of embedded memory blocks available in FPGA devices is the main argument behind the application of this structure. For example, Altera FLEX family devices have 2048-bit EAB memory blocks. In [RSL05] it is demonstrated that the ROM-based implementation of an example sequential circuit – the tbk benchmark - requires 16,384 bits of memory; this considerably exceeds the resources available in the FLEX 10K device. An alternative implementation of this circuit with LUTs requires 895 logic cells (a result from the Altera Quartus II ver. 6.0 sp1 system); this also exceeds the resources available in the FLEX 10K device, as it has only 576 cells. Thus, the *tbk* implementation with this device must rely on the a new FSM architecture.

Clearly, a considerably larger number and size of embedded memory blocks in the newer programmable Stratix and Cyclone devices do not eliminate this problem, as there will always be FSMs whose implementation requires more memory than it is available in the state-of-the-art programmable devices.

In case when efficient memory utilization is essential, the FSM can be implemented in a structure that includes an address register and ROM memory, in which the reduction of ROM memory size is obtained by the introduction of an additional block for address modification (Fig. 3b).

The address modifier can be synthesized with advanced algorithms of functional decomposition, applied until recently exclusively to synthesis of combinational circuits. Such an approach to address modifier synthesis was proposed in [RSL05] (and extended in [Bor04], [Bor08]).

The implementation of an FSM shown in Fig. 3b can be seen as a serial decomposition of the memory block included in the structure of fig. 3a into two blocks: an address modifier and a memory block of smaller capacity than required for the realization of the structure of Fig. 3a. As a result, sequential circuits requiring large-capacity ROM memories (and thus not implementable in the architecture of Fig. 3a) can be implemented using a memory block with a smaller number of inputs and an additional combinational logic block – the address modifier.

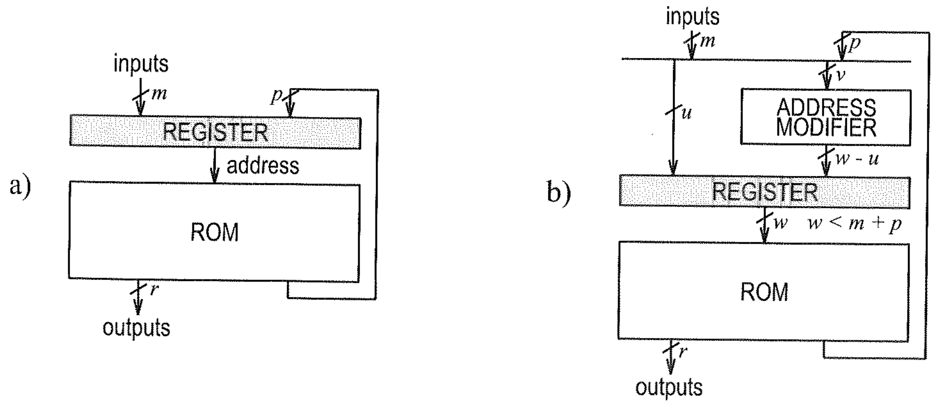


Fig. 3. FSM implementation: a) using ROM memory, b) with the addition of an address modifier

Assuming an FSM implementation with an FPGA device, the advantage of the proposed architecture is that the address modifier can be mapped into a network of LUT cells or into a PAL matrix, while the memory block can be mapped into the built-in EAB matrices. The application of this concept (without the optimization of the state encoding) to the synthesis of the earlier discussed benchmark *tbk* results in a design composed of 333 logic cells and a 4096-bit embedded memory block, which fits entirely in the limited resources of the FLEX structure.

The promising results of other design experiments reported in [Bor04], [RSL05] confirm the effectiveness of the architecture of Fig. 3b and indicate the need for further research. The results of the subsequent studies in this area are presented in [BFL07] and [Bor08].

Different strategies of decomposition

The idea of FSM synthesis presented above lies in the decomposition of the combinational section of the FSM into two modules: an address modifier and a ROM memory. In general, it is possible to view the address modifier and the memory as separate combinational blocks and implement them independently, applying different strategies for decomposition of these two components. In particular, an alternating application of serial and parallel decomposition has been shown to be an effective strategy to design a structure with both logic cells and EMBs.

To illustrate this approach, consider the earlier discussed benchmark *tbk*. In the first stage, *tbk* is decomposed into two blocks: the address modifier and ROM memory of 4096 bits. This decomposition results in the address modifier represented in the form of the truth table with 7 inputs and 5 outputs and the memory with the word length of 8 with a specified contents. Subsequently each of these two components is decomposed into a network of embedded memory blocks and logic cells. It is assumed that the EMB block has a built-in register and it can also be configured as a typical combinational structure.

Fig. 4a shows an implementation with a programmable device that has EMBs of 2048 bits. Two EMBs, configured to have the word length of 4 and operating in parallel, are needed to store the content of the ROM memory. The address modifier is implemented with a single EMB block configured to have the word length of 8. Some inputs and outputs of this block remain unused.

Fig. 4b shows another possible implementation of benchmark *tbk*, obtained under assumption that the programmable device has two types of EMBs with capacity of 512 bits and 4096 bits. Then, it is possible to implement ROM memory using a single 4096-bit EMB, configured to the word length of 8. For the address modifier, the parallel decomposition is applied which results in five single-output functions. The serial decomposition of these functions into logic cells results in the following solution: first function – one cell, second function – seven cells, third function – five cells, fourth function – six cells, fifth function – five cells. Finally, combining the second, third, fourth and fifth function results in one block implemented with the EMB of 512 bits with the word length of 4, and the second (first function) with a single logic cell.

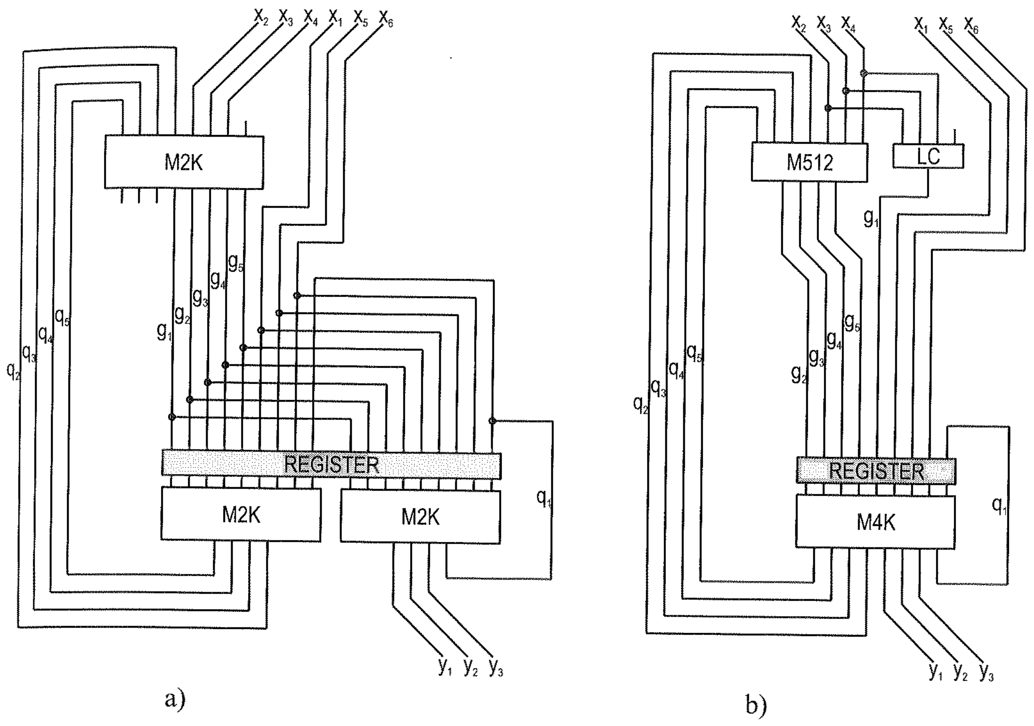

x_2 x_3 x_4 x_1 x_5 x_6
M2K
REGISTER
M2K
M2K
g_1 g_2 g_3 g_4 g_5
q_1 q_2 q_3 q_4 q_5
y_1 y_2 y_3
a)
x_2 x_3 x_4 x_1 x_5 x_6
M512
LC
REGISTER
M4K
g_1 g_2 g_3 g_4 g_5
q_1 q_2 q_3 q_4 q_5
y_1 y_2 y_3
b)

Fig. 4. *tbk* benchmark implementation; in programmable device a) with M2K built in memories, b) with M512 and M4K built in memories

4. SELF-TESTABILITY AND FAULT TOLERANCE FOR FSMS IMPLEMENTED USING PROGRAMMABLE STRUCTURES

As technology advances, digital circuits are becoming increasingly more susceptible to faults – both permanent faults (static faults and dynamic faults, including delay faults), which result primarily from imperfections in the manufacturing process, and soft faults (transient faults) induced primarily by various types of radiation.

To detect permanent faults various test procedures are applied. For circuits implemented with FPGAs, the concept of application-dependent testing has been proposed [Kras97] (this test strategy is also referred to as configuration-dependent testing [Qudd99] or application-oriented test [Reno03]). The idea is to thoroughly exercise only that particular configuration of the FPGA which represents the user-defined application (and not all possible configurations of a programmable device, as it is done by the FPGA manufacturer).

The implementation of application-dependent testing of FPGAs can be based on externally provided test patterns, as proposed in [Qudd99, Reno03], or on the built-in self-test (BIST) techniques. The idea of BIST-based application-dependent testing is to exercise the device in a number of self-test sessions. During each session, a selected part of an FPGA (configured to implement a user-defined function) is examined using the remaining portions of the device, temporarily reconfigured into modules that generate test patterns and analyze responses of the module under test. This strategy does not involve any circuitry overhead or performance penalty.

In [Kras97], a technique for application-dependent self-testing of FPGAs, based on the concept of C-exhaustive testing (combinationally-exhaustive testing), has been proposed. The key part of this technique is the self-test procedure for a sequential subcircuit that implements an FSM. In [Kras04a] it was shown how this basic self-test procedure can be applied to specific implementations of FSMs that include an address modifier and exploit embedded memory blocks available in FPGAs (the configuration shown in Fig. 3b). In addition, design guidelines to make such a circuit more suitable for self-testability were formulated.

The solutions presented in [Kras04a] provide only a "high-level" test strategy for the considered class of FSM implementations. Enhancements at the logic level were subsequently developed, relying on the observation that an appropriate extension of the memory (ROM) specification produced by the FSM synthesis procedure (specification of the contents of those memory words which are left undefined by the conventional FSM synthesis) can significantly improve the testability characteristics of the circuit (its susceptibility to randomly generated test patterns). In [Kras05a] algorithms for such testability-oriented optimization of an FSM implemented using embedded memory of an FPGA are presented. It has been demonstrated that as a result of such an optimization, the circuit becomes significantly more easier to test: for the largest of examined circuits, the self-test session required to achieve an acceptable level of fault detection for the optimized design, obtained using the proposed procedure, is almost 10^6 times shorter than for the non-optimized design. The proposed optimization is essentially cost-free.

A thorough testing of permanent faults resulting from the manufacturing process is not sufficient to guarantee reliable operation of a digital circuit. Error failure rates caused by soft (transient) faults that occur during the normal circuit operation will soon become unacceptable even for mainstream commercial applications [Cohe99]. Therefore, there is an increasing interest in designing digital systems to make them fault-tolerant, i.e. to protect them against such soft faults. SRAM-based FPGAs are particularly vulnerable to soft faults, as single event upsets (SEUs) induced by external radiation affect both functional memory (flip-flops, embedded memory blocks) and configuration memory of an FPGA.

There are different ways to make a digital circuit fault tolerant. Some techniques intended to achieve this goal are based on the concept of error masking and rely on massive hardware redundancy; therefore, they are very expensive and can be afforded only for critical applications. Alternative techniques are based on an on-line detection of errors and appropriate "recovery" actions. The key part of such techniques is effective concurrent error detection (CED).

Most techniques for concurrent error detection in sequential circuits (implementing FSMs) assume that at some stage of design the circuit is represented by a network of gates and flip-flips (or equivalent Boolean formulas) and that such a representation is mapped onto standard cells [AlDM06, BMSS00, DaTo98, ZeSM99]. Concurrent error detection techniques have also been proposed for alternative implementations of sequential circuits, in particular for:

- sequential circuits operating as microprogrammed control units [IyKi95, Wong83],
- sequential circuits implemented with PLAs [BoNT93].

Concurrent error detection techniques for FSMs implemented with programmable logic components have only recently become a subject of extensive studies. CED techniques intended for FSM implementations based on LUTs and flip-flops, available in programmable logic cells of FPGA devices have been proposed in [LOKS06, LeSi99].

A different group of CED techniques have been developed for sequential circuits implemented with memories embedded in FPGAs. The technique presented in [Kras04b] applies to the simplest FSM structure shown in Fig. 3a. For the FSM structure in which the combinational logic is divided into two parts: memory (ROM) and address modifier (as shown in Fig. 3b), two techniques have been proposed:

- the solution presented in [Kras06] is applicable when the ROM is implemented with embedded memory blocks and the address modifier is implemented with LUT-based programmable logic components;
- the solution presented in [Kras08] is applicable when both the ROM and the address modifier are implemented with embedded memory blocks.

The concurrent error detection schemes applicable to FSMs implemented with memories embedded in programmable devices have been proven to detect each permanent or transient fault associated with a single input or output of any component of the circuit that results in its incorrect state or output. The experimental results show that the circuitry overhead associated with concurrent error detection is quite low. For the set of benchmark circuits examined in [Kras08], the overhead calculated under pessimistic assumptions is in the range of 20.7% to 63.8%, with an average value of 32.2%. This compares favourably with the solutions applicable to conventional FSM designs based on gates and flip-flops for which an overhead exceeding 100% is quite typical. The overhead can be further reduced at the expense of the efficiency of fault detection – a design trade-off is possible [Kras05b]. These results indicate that memory-based structures of FSMs, obtained using dedicated synthesis tools, intended for an implementation in FPGAs with embedded memory blocks, are much more suitable for concurrent error detection, and thereby for applications in highly reliable systems, than conventional designs based on gates and flip-flops.

5. CONCLUSION

In modern digital circuit design, the problem of finite state machine synthesis is an issue of significant practical importance. The concept of FSM provides an excellent model for designing of complex Control Units. Many methods for structural synthesis of FSMs have been reported in the literature. Their diversity is a consequence of different styles of implementation. There is a large variety of logic building blocks that can be exploited in modern technologies. The standard cell libraries contains various types of gates; a lot of complex gates can also be generated in (semi-)custom CMOS design; and the field programmable logic families include different types of (C)PLDs and FPGAs.

This article summarizes various techniques for sequential logic synthesis, including logic optimization techniques, the technology mapping techniques, and the techniques that provide the resulting circuits with concurrent error detection capability. These techniques vary considerably in terms of quality and efficiency, and different techniques may be suitable for different types of design and/or different optimization objectives. We hope our systematic classification and review of these techniques will help the reader to choose the best combination of these techniques for a given application, and to develop new techniques to overcome the limitations in the existing technologies.

For FPLD technology, sequential logic synthesis is a very important step in design automation. However, the opportunities created by modern microelectronic technology are not fully exploited because of weaknesses in traditional logic design methods. Commercially available tools are immature and do not allow the designer to take advantage of all the architectural features available in modern programmable structures.

We believe that high-quality logic synthesis tools will play an increasingly more important role in FPLD design systems and their integration with the tools used for other steps in the design process may be a key to success in digital designing.

6. REFERENCES

[AB06] M. Adamski, A. Barkalov: *Architectural and Sequential Synthesis of Digital Devices* University of Zielona Góra Press, 2006.

[AlDM06] S. Almukhaizim, P. Drineas, Y. Makris: tion*Entropy-driven parity-tree* selecc for *low-overhead concurrent error detection in finite state machines*, IEEE Trans. on CAD vol. 25, no. 8, pp. 1547-1554, Aug. 2006,

[ADN92] P. Ashar, S. Devadas, A. R. Newton: *Sequential Logic Synthesis*, Kluwer Academic Publishers, 1992, Boston, MA, USA.

[BMSS00] C. Bolchini, R. Montandon, F. Self-Checking Finite Salice, D. Sciuto: *Design of VHDL-Based Totally -State Machine and Data Path Descriptions*, IEEE Trans. on VLSI Systems, vol. 8, pp. 98-103, Feb. 2000.

[BFL07] G. Borowik, B. Falkowski, T. Łuba: *Cost-Efficient Synthesis for Sequential Ciruits Implemented Using Embedded Memory Blocks of FPGA's*, Proc. of 10th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (2007), Kraków, Poland, 11-13 April 2007.

[BL03] J. A. Brzozowski, T. Łuba: *Decomposition of Boolean Functions Specified by Cubes*, Journal of Multi-Valued Logic and Soft Computing vol. 9 (2003), 377-417, Old City Publishing Inc., Philadelphia 2003.

[Bor04] G. Borowik: *Synthesis of Memory Addressing Circuits in FPGA-based Sequential Machines*, Proc. of V International Conference on Computer-Aided Design of Discrete Devices (2004), 31-38, Minsk, Belarus, 16-17 November 2004.

[Bor08] G. Borowik: *Improved State Encoding for FSM Implementation in FPGA Structures with Embedded Memory Blocks*, Electronics and Telecommunications Quarterly, vol. 54, no 1, 9-28, March 2008.

[BW06] A. Barkalov, L. Titarenko: *Logic Synthesis for Compositional Microprogram Control Units*, Springer, 2008.

[BW06] A. Barkalov, M. Węgrzyn: *Design of Control Units with Programmable Logic*, University of Zielona Góra Press, 2006.

[BoNT93] M. Boudjit, M. Nicolaidis, K. Torki: *Automatic generation algorithms, experiments and comparisons of self-checking PLA schemes using parity codes*, Proc. European Design Automation Conf., pp. 144-150, 1993.

[Cohe99] N. Cohen et al.: *Soft error considerations for deep-submicron CMOS circuit applications*, Dig. IEDM Int. Electron Devices Meeting, pp. 315-318, 1999.

[CK05] R. Czerwiński, D. Kania: *State Assignment for PAL-based CPLDs*, Proc. of 8th Euromicro Conference on Digital Systems Design, Architectures, Methods and Tools, IEEE Computer Society, Christophe Wolinski (Ed.) (2005), 127-134, Porto, Portugal, 30 August – 3 September 2005.

[CKK06] R. Czerwiński, D. Kania, J. Kulisz: *FSMs State Encoding Targeting at Logic Level Minimization*, Bulletin of the Polish Academy of Sciences vol. 54 (2006), no. 4.

[CMSH96] S. C. Chang, M. Marek-Sadowska, T. T. Hwang: *Technology Mapping for TLU FPGAs Based on Decomposition of Binary Decision Diagrams*, IEEE Trans. on CAD vol. 15 (1996), no. 10, 1226-1236, October 1996.

[Cou98] O. Coudert: *A New Paradigm for Dichotomy-based Constrained Encoding*, Proc. of Design, Automation and Test in Europe (1998), 830-834.

[CS96] O. Coudert, C. J. R. Shi: *Exact Dichotomy-based Constrained Encoding*, Proc. of the Int. Conf. on Computer Design (1996), 426-431.

[CY92] M. J. Ciesielski,. Y. Plade: *A Two-stage PLA Decomposition*, IEEE Trans. on CAD vol. 11 (1992), no. 8, 943-954.

[DaTo98] D. Das, N.A. Touba: *Synthesis of Circuits with Low-Cost Concurrent Error Detection based on Bose-Lin Codes*, Proc. VLSI Test Symp., pp. 309-315, 1998.

[DBSV85] G. De Micheli, R. K. Brayton A. Sangiovanni-Vincentelli: *Optimal State Assignment for Finite State Machines*, IEEE Transactions on CAD vol. CAD-4 (1985), no. 3, 269-284.

[DeM86] G. De Micheli: *Symbolic Design of Combinational and Sequentional Logic Circuits Implemented by Low-level Logic Macros*, IEEE Transactions on CAD vol. CAD-5 (1986), no. 4, 597-616.

[DeM94] G. De Micheli: *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, 1994.

[DHLN91] X. Du, G. Hachtel, B. Lin, R. A. Newton, *MUSE: A Multilevel Symbolic Encoding Algorithm for State Assignment*, IEEE Transactions on CAD vol.10 (1991), no. 1, 28-38.

[DMNSV88] S. Devadas, H. K. Ma, R. Newton, A. Sangiovanni-Vincentelli: *MUSTANG: State Assignment of Finite State Machines Targeting Multilevel Logic Implementations*, IEEE Transactions on CAD vol. 7 (1988), no. 12, 1290-1300.

[DN89] S. Devadas, R. Newton: *Decomposition and Factorization of Sequential Finite State Machines*, IEEE Transactions on CAD vol. 8 (1989), no. 11, 1206-1217.

[DN91] S. Devadas, R. Newton: *Exact Algorithms for Output Encoding, State Assignment and Four-level Boolean Minimization*, IEEE Transactions on CAD vol. 10 (1991), no. 1, 13-27.

[For95] J. Forrest, ODE: *Output Direct State Machine Encoding*, Proc. of the European Design Automation Conference (1995), 600-605.

[HK04] E. Hrynkiewicz, D. Kania: *Metody syntezy dedykowane dla struktur FPGA typu tablicowego* (in Polish), Kwartalnik Elektroniki i Telekomunikacji nr 50, z. 3 (2004), 325-342.

[HS66] J. Hartmanis, R. E. Stearns: *Algebraic Structure Theory of Sequential Machines*, Prentice-Hall, New York, 1966.

[HSB02] S. Hassoun, T. Sasao, R. Brayton (Eds.): *Logic Synthesis and Verification*, Kluwer Academic Publishers, 2002, New York.

[IP99] S. Iman, M. Pedram: *Logic Synthesis for Low Power VLSI Designs*, Kluwer Academic Publishers, 1999.

[IyKi95] V. S. Iyengar, L. L. Kinney: *Concurrent Fault Detection in Microprogrammed Control Units*, IEEE Trans. on Computers, vol. C-34, pp. 810-821, Sept. 1985.

[JC01] L. Jóźwiak, A. Chojnacki: *Effective and Efficient FPGA Synthesis through Functional Decomposition Based on Information Relatinship oMeasures*, Proc. of Euromicro Symposium on Digital Systems Design (2001), 30-37, Warsaw, Poland, 4-6 September 2001.

[JS00] L. Jóźwiak, A. Ślusarczyk: *A new state assignment method targeting FPGA implementations*, Proc. of the 26th Euromicro Conference vol. 1 (2000), 50-59, Maastricht, The Netherlands, 5-7 September 2000.

[JSC01] L. Jóźwiak, A. Ślusarczyk, A. Chojnacki: *Fast and Compact Sequential Circuits through the Information-Driven Circuit Synthesis*, Proc. of Euromicro Symposium on Digital Systems Design (2001), 46-53, Warsaw, Poland, 4-6 September 2001.

[Kan04] D. Kania: *The Logic Synthesis for the PAL-based Complex Programmable Logic Devices* (in Polish), Zeszyty Naukowe (2004), 16-19, Politechnika Śląska, Gliwice 2004.

[Kras97] A. Kraśniewski: *Design for Application-Dependent Testability of FPGAs*, Proc. Int. Workshop on Logic and Architecture Synthesis, pp. 245-254, Grenoble, Dec. 1977. Self-Testing of Sequential.

[Kras04a] A. Kraśniewski: *Circuits Designed for Implementation in FPGAs with Embedded Memory Blocks*, Proc. IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems, pp. 75-82, Tatranska Lomnica, April 2004.

[Kras04b] A. Kraśniewski: *Concurrent Error Detection in Sequential Circuits Implemented Using FPGAs with Embedded Memory Blocks*, Proc. IEEE Int. On-Line Testing Symp., pp. 67-72, Funchal (Madeira), July 2004.

[Kras05a] A. Kraśniewski: *Cost-Free Enhancement of Testability for Sequential Circuits Implemented Using Embedded Memory Blocks of FPGA's*, Proc. IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems, pp. 61-68, Sopron, April 2005.

[Kras05b] A. Kraśniewski: *A Pragmatic Approach to Concurrent Error Detection in Sequential Circuits Implemented Using FPGAs with Embedded Memory*, Proc. IEEE Int. On-Line Testing Symp., pp. 197-198, San Raphael, July 2005

[Kras06] A. Kraśniewski: *Low-Cost Concurrent Error Detection for FSMs Implemented Using Embedded Memory Blocks of FPGAs*, Proc. IEEE Workshop on Design and Diagnostics of Electronics Circuits and Systems, pp. 180-185, Praha, April 2006.

[Kras08] A. Kraśniewski: *Concurrent Error Detection for Finite State Machines Implemented* *with Embedded Memory Blocks of SRAM-Based FPGAs*, Microprocessors and Microsystems, vol. 32, no. 5-6, pp. 303-312, August 2008.

[LeSi99] I. Levin, V. Sinelnikov: *Self-checking of FPGA-based control units*, Proc. 9th Great Lakes Symposium on VLSI, pp. 292-295, 1999.

[LOKS06] I. Levin, V. Ostrovsky, O. Keren, V. Sinelnikov: *Cascade Scheme for Concurrent Errors Detection*, Proc. 10th EUROMICRO Conf. on Digital System Design, pp. 359-368, 2006.

[LN89] B. Lin, R. A. Newton: *Synthesis of Multiple Level Logic Symbolic High-level Description Languages*, Proc. of the Int. Conf. on VLSI (1989), 187-196.

[LPP96] Y. T. Lai, K. R. R. Pan, M. Pedram: *OBDD-Based Functional Decomposition: Algorithm and Implementation*, IEEE Trans. on CAD vol. 15 (1996), no. 8, 977-990, August 1996.

[LuS95] T. Łuba, H. Selvaraj: *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, 289-300, 1995.

[PMG99] M. Perkowski, R. Malvi, S. Grygiel, M. Burns, A. Mishchenko: *Graph coloring algorithms for Fast Evaluation of Curtis Decompositions*, Proc. of Design Automation Conference (1999), 225-230, New Orleans, 1999.

[Qudd99] Quddus, W., Jas, A., Touba, N.A.: *Configuration Self-Test in FPGA-Based Reconfigurable Systems.* In: Proc. ISCAS'99, 1999, pp. 97-100.

[Reno03] M. Renovell: *Some Aspects of the Test Generation Problem for an Application-Oriented Test of SRAM-Based FPGAs*, Journal of Circuits, Systems, and Computers, vol. 12(2003), no. 2, pp. 143-158.

[RJL01] M. Rawski, L. Jóźwiak, T. Łuba: *Functional Decomposition with an Efficent Input Support Selection for Sub-functions Based on Information Relationship Measures*, Journal of Systems Architecture 47 (2001), 137-155, Elsevier Science B.V., 2001.

[RSL05] M. Rawski, H. Selvaraj, T. Łuba: *An Application of Functional Decomposition in ROM-based FSM Implementation in FPGA Devices*, Journal of Systems Architecture vol. 51 (2005), 424-434, Elsevier 2005.

[RSLS06] M. Rawski, H. Selvaraj, T. Łuba, P. Szotkowski: *Multilevel Synthesis of Finite State Machines Based on Symbolic Functional Decomposition*, International Journal of Computational Intelligence and Applications, Vol. 6, No. 2, June 2006, pp. 257-271, Imperial College Press 2007.

[SB93] C. J. Shi, J. A. Brzozowski: *An Efficient Algorithm for Constrained Encoding and Its Applications*, IEEE Trans. on CAD vol. 12 (1993), no. 12, 1813-1826.

[Sch01] C. Scholl: *Functional Decomposition with Application to FPGA Synthesis*, Kluwer Academic Publisher, 2001, Boston.

[SLBG94] G. Sarwary, E. P. Lopes, L. Burgun, A. Greiner: *FSM Synthesis on FPGA Architectures*, Proc. of 7th Annual IEEE International ASIC Conference and Exhibit (1994), 178-181.

[Sol97] V. V. Solovjev: *Sintez konechnyh avtomatov na programmiruemyh matricah logiki*,Avtomatika i Vychislitelnaya Tehnika No. 2 (1997), 65-74.

[SSL92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, A. Sangiovanni-*-Vincentelli*, SIS: *A System for Sequential Circuit Synthesis, Memorandum*, no. UCB/ERL M92/41, Electronics Research Laboratory, Department of Electrical Engineering and Computer Science, University of California, Berkley, 1992.

[SSP01] P. Sapiecha, H. Selvaraj, M. Pleban: *Decomposition of Boolean Relations and Functions in Logic Synthesis and Data Analysis, Rough Sets and Current Trends in Computing*, Springer Verlag, W. Ziarko, Y. Yao (Eds.) (2001), 487-494, Berlin 2001.

[SVBSV94] A. Saldanha, T. Villa, R. K. Brayton, A. Sangiovanni-Vincentel-li: *Satisfaction of Input and Output Encoding Constraints*, IEEE Trans. on CAD vol. 13 (1994), no. 5, 589-602.

[TKBSV98] T. Villa, T. Kam, R. K. Brayton, A. Sangiovanni-Vincentelli: *Synthesis of Finite State Machines: Logic Optimization*, Kluwer Academic Publishers, Boston, 1998.

[VSV90] T. Villa, A. Sangiovanni-Vincentelli, NOVA: *State Assignment for Optimal Two-level Implementations*, IEEE Transactions on CAD vol. 9 (1990), no. 9, 905-924.

[WKA89] W. Wolf, K. Kautzer, J. Akella: *Addendum to A Kernel Finding State Assignment Algorithm for Multilevel Logic*, IEEE Transactions on CAD vol. 8 (1989), no. 8, 917-920.

[Wong83] C.Y. Wong et al.: *The Design of a Microprogram Control Unit with Concurrent Error Detection*, Proc. Fault Tolerant Computing Symp., pp. 476-483, 1983

[YSL05] S. N. Yanushkevich, V. P. Shmerko, S. E. Lyshevski: *Logic Design of NanoICs, Detection*, CRC Press, 2005.

[ZeSM99] C. Zeng, N. Saxena, E. J. McCluskey: *Finite State Machine Synthesis with Concurrent Error Detection*, Proc. IEEE Int. Test Conf., pp. 672-679, 1999

Opti

of
blo
str
cas
are

Ke

The
of the "
impleme
prise the
4] and e
an arbitr
memory
(CU) tha
tion of n
solution
because

Optimization of compositional microprogram control unit by modification of microinstruction format

LARYSA TITARENKO, JACEK BIEGANOWSKI

University of Zielona Góra
Institute of Computer Engineering and Electronics
ul. Podgórna 50, 65-246 Zielona Góra
Email: {L.Titarenko, J.Bieganowski}@iie.uz.zgora.pl

Received 2009.01.05
Authorized 2009.03.19

The methods of hardware amount decrease are proposed oriented on implementation of compositional microprogram control unit with PAL macrocells and embedded memory blocks of CPLD – based SoC. First method is based on introduction of additional microinstructions. Second method is based on expansion of the format of microinstructions. In both cases amount of EMBs is minimal one. The examples of application of proposed methods are given.

Keywords: compositional microprogram control unit, flow-chart of algorithm, CPLD, microinstruction

1. INTRODUCTION

The progress in microelectronics has resulted in appearance of integrated circuits of the "system-on-a-chip" (SoC) type [1]. The functional power of SoC is enough to implement a complex digital system using single chip [2]. The modern SoC can comprise the programmable array logic (PAL) macrocells based on CPLD conception [3, 4] and embedded memory blocks (EMB). The PAL macrocells are used to implement an arbitrary logic of digital system and EMBs implement the tables, such as control memory [5]. One of the most important blocks of any digital system is control unit (CU) that coordinates the cooperation of all system blocks [6, 7, 8]. The minimization of number of PAL macrocells in the circuit of CU is an actual problem and its solution permits to decrease the chip area occupied by this circuit [4]. It is important because the disengaged resources can be used to increase the power of the system.

The characteristics of both control algorithm to be implemented and logic elements in use should be taken into account to minimize hardware amount in the circuit of CU [9]. In this article we propose some methods of this task solution oriented on both linear control algorithms [9] and CPLD-based SoC. The peculiarities of CPLD are wide fan-in and very limited number of intermediate terms in its macrocells [3, 4]. Therefore, one of the ways of hardware optimization here is the decreasing of number of terms in the disjunctional normal forms (DNF) of implemented Boolean functions [10, 11]. The peculiarity of linear control algorithms is existence of long sequences of unconditional transitions between the microinstructions. In this case the model of compositional microprogram unit (CMCU) perfectly fits to implement such transitions [12]. In this article we propose the method of optimization of PAL macrocells in the circuit of CMCU, when interpreted control algorithm is represented as linear flow-chart of algorithm [13].

2. BACKGROUND OF CMCU

Let a control algorithm be represented as flow-chart of algorithm (FCA) $\Gamma = \Gamma(B, E)$, where $B = \{b_0, b_E\} \cup E_1 \cup E_2$ is set of the nodes, $E = \{\langle b_t, b_q \rangle \mid b_t, b_q \in B\}$ is set of the edges. The set B includes an initial node b_0, a final node b_E, the operational nodes $b_q \in E_1$ and the conditional nodes $b_q \in E_2$. The node $b_q \in E_1$ contains set $Y(b_q) \subseteq Y$, where $Y = \{y_1, \ldots, y_N\}$ is set of microoperations of data path of digital system [7, 8]. The node $b_q \in E_2$ contains single element of the set of logic conditions $X = \{x_1, \ldots, x_L\}$. The FCA Γ is named as linear FCA (LFCA), if number of its operational nodes M is not less than 75% from total number of the nodes [9], where $M = |E_1|$.

Let $C = \{\alpha_1, \ldots, \alpha_g\}$ be a partition of the set E_1 and $\alpha_g \in C$ be an operational linear chain (OLC) of LFCA Γ. The OLC $\alpha_g \in C$ is a sequence of operational nodes $\langle b_{g1}, \ldots, b_{gF_g} \rangle$, such that an edge $\langle b_{gi}, b_{gi+1} \rangle \in E$ exists for each pair of its adjacent nodes ($i = 1, \ldots, F_g - 1$) [12]. An OLC $\alpha_g \in C$ can have an arbitrary number of inputs and only one output $O_g \in O(\Gamma)$, where $O(\Gamma)$ is set of the outputs of OLC $\alpha_g \in C$. Let node $b_q \in E_1$ correspond to microinstruction MI$_q$ with address $A(b_q)$ and let $A(b_q)$ have bits.

$$R =]\log_2 M[\tag{1}$$

Let us execute the natural addressing of microinstructions [6, 12] to satisfy the following condition:

$$A\left(b_{gi+1}\right) = A\left(b_{gi}\right) + 1, \tag{2}$$

where i = 1,...,Fg-1; g = 1,...,G. In this case LFCA Γ can be interpreted by CMCU U_1 (Fig. 1).

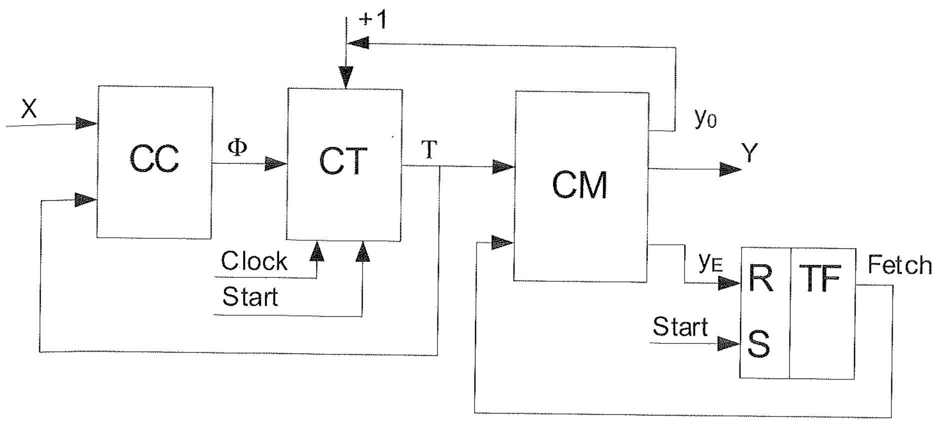


Fig. 1. Structural diagram of the CMCU

The CMCU U_1 operates in the following manner. If pulse *Start* = 1, then an address of the first microinstruction (MI) of microprogram MP(Γ) corresponding to LFCA Γ is loaded into counter of microinstruction address CT. In the same time the flip-flop TF is set up and *Fetch* = 1. It permits the fetching of the MIs from control memory CM. If CT contains an address $A(b_q)$, where $b_q \notin O(\Gamma)$, then signal y_0 is formed together with microoperations $Y(b_q) \subseteq Y$. If $y_0 = 1$, then pulse *Clock* causes increment of content of CT and it corresponds to mode (2). If $y_0 = 0$, then transition address is formed by combinational circuit CC that implements the system of excitation functions of the flip-flops of CT

$$\Phi = \Phi(T, X), \tag{3}$$

where $T = \{T_1,...,T_R\}$ is the set of address variables, $|\Phi| = R$. If CT contains an address $A(b_q)$ and $\langle b_q, b_E\rangle \in E$, then variable y_E is formed together with microoperations $y_n \in Y(b_q)$. If $y_E = 1$, then flip-flop TF is reset, $y_E = 0$ and operation of CMCU U_1 is terminated.

In case of CPLD-based SoC the circuit CC is implemented using PAL macrocells and control memory CM is implemented using EMBs. The positive feature of U_1 is minimal number of the outputs of the circuit CC in comparison with other known structures of CMCU [12]. It gives a potential possibility of optimization of the circuit CC in comparison to other models of CMCU. But CC and CT form Moore finite-state-machine (FSM) [1] of microinstruction addressing. It is well-known that as a rule Moore FSM has more transitions, than equivalent Mealy FSM [1]. Each transition corresponds to one term of the system (3). The minimization of this number can be reached due to partition of the set C by the classes of pseudoequivalent OLC (POLC) that correspond to pseudoequivalent states of Moore FSM [14].

The OLC $\alpha_i, \alpha_j \in C$ are named POLC, if their outputs are connected with the input of the same node of LFCA Γ [9]. Let $C' \subset C$ and let $\alpha_g \in C'$, if $\langle O_g, b_E\rangle \notin E$. Let us form the partition $\Pi_C = \{B_1,...,B_I\}$ of the set C', such that B_i is the class of POLC ($i = 1,...,I$). Let us encode the classes $B_i \in \Pi_C$ by binary codes $K(B_i)$ with

$$R_1 =]log_2 I[\tag{4}$$

bits and let us use the variables $\tau_r \in \tau = \{\tau_1,..., \tau_{R1}\}$ for such encoding. One of the ways of hardware optimization of the circuit of U_1 is usage of address transformer AT [5] that forms code of $B_i \in \Pi_C$ on the base of address $A(b_q)$, where $b_q = O_g, \alpha_g \in B_i$. But this approach drawback is a need of some chip resources to implement circuit of AT. It has sense, if total number of macrocells in both CC and AT is lower, than number of macrocells in circuit CC of CMCU U_1. In this article we propose the methods of optimization of the circuit CC based on modification of microinstruction format of CMCU U_1.

3. MAIN IDEAS OF PROPOSED METHODS

The main idea of proposed methods is the usage of redundant resources of EMBs to implement the transformation of the addresses $A(b_q)$ into the codes $K(B_1)$. The microinstruction format of CMCU U_1 is shown in Fig. 2, a.

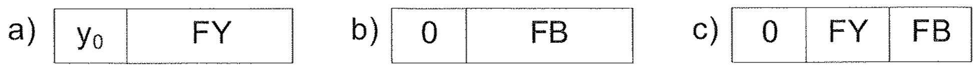


Fig. 2. Microinstruction format of CMCU U_1 (a) and proposed modifications (b, c)

The field FY contains the microoperations $Y(b_q)$ and variable y_E, where $b_q \in E_1$. We propose two approaches of microinstruction format modifications, when field FB is in use. The field FB contains the code $K(B_i)$, where $B_i \in \Pi_C$.

In first case an additional node O_g is inserted as a last component of OLC $\alpha_g \in B_i$. This node corresponds to MI with format (Fig. 2, b). In second case each node $b_q = O_g$ corresponds to MI with format (Fig. 2, c). The first bit of those formats corresponds to variable y_0. There are two new structures of CMCU (U_2, U_3) and both use format (Fig. 2, a) with $y_0 = 1$. The CMCU U_2 uses also format (Fig. 2, b) with $y_0 = 0$. The CMCU U_3 uses also format (Fig. 2, c) with $y_0 = 0$. Let us discuss these modifications in details.
The control memory of CMCU $U_1(\Gamma)$ has

$$\Delta_F = 2^R - M \tag{5}$$

free cells, here $U_i(\Gamma_j)$ means that LFCA Γ_j is interpreted by CMCU U_i. We propose to insert Δ_F additional MI with format F1 (Fig. 2, b) into microprogram MP(Γ). Let G_0 be amount of OLC $\alpha_g \in C'$ for the classes $B_i \in \Pi_C$ such that condition holds.

$$|B_i| > 1 \tag{6}$$

Let us discuss a case, when condition holds.

$$G_o > \Delta_F \tag{7}$$

In this case we propose to combine the modification of OLC $\alpha_g \in C'$ and transformation of output address.

Let us represent partition Π_C as $\Delta_1 \cup \Delta_2$, where set Δ_1 contains the classes B_i that satisfy to (6) and let $I_1 = |\Delta_1|$ is as nearer to Δ_F as it is possible. It is clear that $\Delta_2 = \Pi_C \Delta_1, |\Delta_2| = I_2 = I - I_1$.
Let us encode each class $B_i \in \Delta_1$ by binary code $C(B_i)$ with

$$R_2 =]\log_2 I_1[\tag{8}$$

bits and let us use the variables $z_r \in Z = \{z_1,...,z_{R2}\}$ for such encoding. Let us encode the classes $B_i \in \Delta_2$ by binary codes $K(B_i)$ with

$$R_3 =]\log_2 I_2[\tag{9}$$

bits and let us use the variables $\tau_r \in \tau = \{\tau_1,..., \tau_{R3}\}$ for such encoding. In this case the CMCU U_2 (Fig. 3) is proposed to interpret the LFCA Γ.

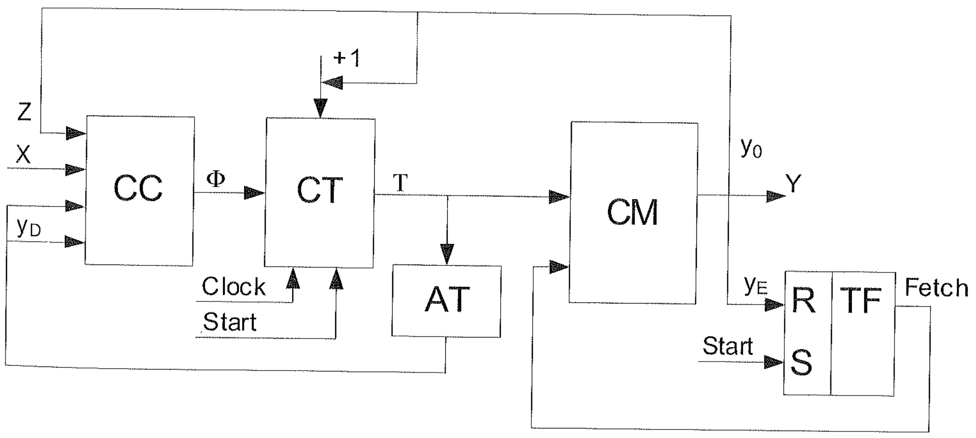


Fig. 3. Structural diagram of CMCU U_2

In CMCU U_2 an address of transition is formed by functions

$$\Phi = \Phi(Z, \tau, X, y_D), \tag{10}$$

where variable y_D points out the source of the code of the class $B_i \in \Pi_C$. Let us point out that variable y_D can be formed by any block of CMCU (CC, AT, CM). In case under discussion block AT forms both variable y_D and variables $\tau_r \in \tau$:

$$\tau = \tau(T), \tag{11}$$

$$y_D = y_D(T). \tag{12}$$

Let us point out that amount of inputs of PAL macrocells increased in CMCU U_2 in comparison with CMCU U_1. But it does not affect the hardware amount because of wide fan-in of CPLD [3, 4].

The drawback of CMCU U_2 is an appearance of idle cycles of data-path, when microinstruction with format F1 is executed. It leads to increase of the time of control algorithm interpretation. If such increase is undesirable, then microinstruction format F2 (Fig. 2, c) can be used to optimize the hardware amount.

The information in the field FY can be encoded using different strategies: one-hot encoding of microoperations, maximal encoding of the collections of microoperations, encoding of the fields of compatible microoperations [6, 12]. Let us discuss the one-hot case, when bit capacity of the field FY is determined as

$$m_y = N + 1. \tag{13}$$

If $y_0 = 0$ (format F2), then transition address is formed by functions

$$\Phi = \Phi(\tau, X), \tag{14}$$

where $|\Phi| = R_1$. Such approach permits to minimize hardware amount for circuit CC without decrease of performance of digital system.

The drawback of F2 is the increase of bit capacity of MI by R_1 bits that can lead to increase of the amount of EMBs in CM in comparison to CMCU U_1. If such increase is undesirable, then the following approach is proposed.

Let output word of EMB have t_M bits and let number of these words be not lower, than M. In this case

$$n_1 = \lceil](N + 2)/t_M[\rceil \tag{15}$$

EMBs is enough to implement the control memory of CMCU. In this case

$$\Delta_m = n_1 t_M - N - 2 \tag{16}$$

bits of the word can be used to represent the field FB. If condition

$$\Delta_m \geq R_1 \tag{17}$$

holds, then free bits of EMB word can be used to keep the field FB. Otherwise the collections of microoperations $Y(b_q)$ should be analyzed, where $b_q = O_g$ and $\alpha_g \in C'$. Let their microoperations form the set $Y^1 \subseteq Y$, where $|Y| = N_1$. It means that

$$\Delta_y = N + 1 - N_1 \tag{18}$$

bits of the field FY are not in use, when microinstructions MI_q are executed ($b_q = O_g$, $\alpha_g \in C'$). If condition

$$\Delta_m + \Delta_y \geq R_1 \tag{19}$$

holds, then field FB can be represented by Δ_m free bits of EMB word and by $(R_1 - \Delta_m)$ bits of the field FY. If condition (19) is violated, then $\Delta_m + \Delta_Y$ bits of $K(B_i)$ are represented by EMB word and

$$R_4 = R_1 - \Delta_m - \Delta_y \tag{20}$$

bits are formed by AT. In this case $\tau = \tau^1 \cup \tau^2$, where $|\tau'| = \Delta_m + \Delta_Y$ and $|\tau^2| = R_4$. In the most common case we propose CMCU U_3 (Fig. 4) to interpret the LFCA Γ.

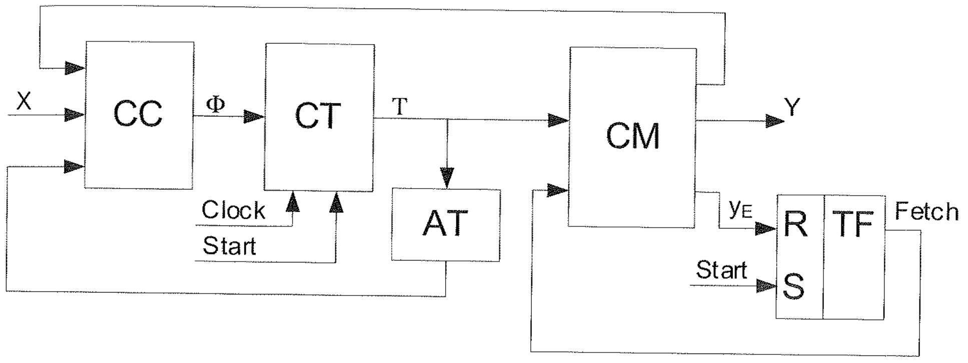


Fig. 4. Structural diagram of CMCU U_3

In this article we propose the methods of design of both CMCU U_2 and U_3. In both cases the proposed methods are illustrated using LFCA Γ_1 with $E_1 = \{b_1, ..., b_{27}\}$. Let set of OLC be formed, such that $C = \{\alpha_1, ..., \alpha_{11}\}$ where $\alpha_1 = \langle b_1, b_2 \rangle, \alpha_2 = \langle b_3, b_4, b_5 \rangle$, $\alpha_3 = \langle b_6, b_7 \rangle, \alpha_4 = \langle b_8, b_9, b_{10} \rangle, \alpha_5 = \langle b_{11}, b_{12} \rangle, \alpha_6 = b_{13}, b_{14}, b_{15} \rangle, \alpha_7 = \langle b_{16}, b_{17} \rangle$, $\alpha_8 = \langle b_{18} \rangle, \alpha_9 = \langle b_{19}, b_{20} \rangle, \alpha_{10} = \langle b_{21}, b_{22} \rangle, \alpha_{11} = \langle b_{23}, ..., b_{27} \rangle$. Let $\Pi_C = \{B_1, ..., B_5\}$, where $B_1 = \{\alpha_1\}$, $B_2 = \{\alpha_2, \alpha_3\}$, $B_3 = \{\alpha_4, \alpha_5\}$, $B_4 = \{\alpha_6, \alpha_7, \alpha_8\}$, $B_5 = \{\alpha_9, \alpha_{10}\}$, it is clear that $\alpha_{11} \notin C$.

4. SYNTHESIS OF CMCU U_2

We propose the following method of synthesis of CMCU U_2:

1. Construction of the set C of LFCA Γ.
2. Construction of the partition Π_C, and sets $\Delta_1\Delta_2$.
3. Modification of OLC $\alpha_g \in C'$.
4. Natural addressing of microinstructions.
5. Encoding of the classes $B_i \in \Delta_1 \cup \Delta_2$.
6. Construction of the content of control memory.
7. Construction of the table of transitions of CMCU.
8. Construction of the table of address transformer AT.
9. Implementation of the logic circuit of CMCU.

In the example under discussion we have $M = 27$, $R = 5$, $\Delta_F = 32\text{-}27 = 5$, $G_0 = 9$. Therefore, only $\Delta_F = 5$ OLC can be modified. It is clear that set Δ_1 includes class B_4 and one of the classes B_2, B_3, B_5. Let $\Delta_1 = \{B_2, B_4\}$, $\Delta_2 = \{B_1, B_3, B_5\}$, $I_1 = 2$, R_2=1, $Z = \{z_1\}$, $I_2 = 3$, $R_3 = 2$, $\tau = \{\tau_1, \tau_2\}$. Let us modify the OLC $\alpha_2, \alpha_3, \alpha_6, \alpha_7, \alpha_8 \in B_i$, where $B_i \in \Delta_1$, in the following manner: $\alpha_2 = \langle b_3, b_4, b_5, O_2\rangle$, $\alpha_3 = \langle b_6, b_7, O_3\rangle$, $\alpha_6 = \langle b_{13}, b_{13}, b_{15}, O_6\rangle$, $\alpha_7 = \langle b_{16}, b_{17}, O_7\rangle$, $\alpha_8 = \langle b_{18}, O_8\rangle$. The application of the methods from [12] leads to addresses of microinstructions of CMCU $U_2(\Gamma_1)$ (Fig. 5).

$T_4T_5 \backslash T_1T_2T_3$	000	001	010	011	100	101	110	111
00	b_1	b_5	O_3	b_{11}	b_{15}	O_7	b_{20}	b_{24}
01	b_2	O_2	b_8	b_{12}	O_6	b_{18}	b_{21}	b_{25}
10	b_3	b_6	b_9	b_{13}	b_{16}	O_8	b_{22}	b_{26}
11	b_4	b_7	b_{10}	b_{14}	b_{17}	b_{19}	b_{23}	b_{27}

Fig. 5. Addresses of microinstructions of CMCU $U_2(\Gamma_1)$

Let us encode the classes $B_i \in \Delta_1 \cup \Delta_2$ in the following manner: $K(B_1) = 00$, $C(B_2) = 0$, $K(B_3) = 01$, $C(B_4) = 1$, $K(B_5) = 10$. The content of CM should be formed in the following manner:

- if $\langle b_q, b_E\rangle \in E$, then variable y_E is written in the cell with address $A(b_q)$;
- cell with address $A(b_q)$ contains microoperations $y_n \in Y(b_q)$;
- if OLC $\alpha_g \in B_i$ was modified, then $y_0 = 1$ for all microinstructions corresponding to nodes $b_q \in E_1$, from this OLC;
- the cells corresponding to additional nodes contain code $C(B_i)$ and $y_0 = 0$, where $B_i \in \Delta_1$.

Let set Y of LFCA Γ_1 include $N = 5$ microoperations and let these microoperations be distributed as the following: $Y(b_1) = Y(b_{10}) = Y(b_{27}) = \{y_1, y_2\}$; $Y(b_2) = Y(b_{11}) = Y(b_{26}) = \{y_3\}$; $Y(b_3) = Y(b_{12}) = Y(b_{25}) = \{y_4\}$; $Y(b_4) = Y(b_{13}) = Y(b_{24}) = \{y_1, y_5\}$; $Y(b_5) = Y(b_{14}) = Y(b_{23}) = \{y_2, y_4\}$; $Y(b_6) = Y(b_{15}) = Y(b_{22}) = \{y_1, y_3, y_5\}$; $Y(b_7) = Y(b_{16}) = Y(b_{21}) = \{y_2, y_3\}$; $Y(b_8) = Y(b_{17}) = Y(b_{20}) = \{y_2, y_5\}$; $Y(b_9) = Y(b_{18}) = Y(b_{19}) = \{y_5\}$

The first 8 cells of control memory of CMCU $U_2(\Gamma_1)$ are shown in the Table 1. The record "y_1/z_q" means that this bit of the output word represents either microoperation y_1 (if $y_0 = 1$), or variable z_1 (if $y_0 = 0$). For MIs with format F1 only field FB has sense and other bits are ignored and they can have arbitrary values. It is shown by symbol "*" in the Table 1.

Table 1

Fragment of control memory of CMCU $U_2(\Gamma_1)$

Address	y_0	FY/FB						Remarks	
$T_1T_2T_3T_4T_5$		y_1/z_1	y_2	y_3	y_4	y_5	y_E	b_q/O_g	B_i
00000	1	1	1	0	0	0	0	b_1	B_1
00001	0	0	0	1	0	0	0	b_2	B_1
00010	1	0	0	0	1	0	0	b_3	B_2
00011	1	1	0	0	0	1	0	b_4	B_2
00100	1	0	1	0	1	0	0	b_5	B_2
00101	0	0	*	*	*	*	*	O_2	B_2
00110	1	1	0	1	0	1	0	b_6	B_2
00111	1	0	1	1	0	0	0	b_7	B_2

Let system of formulae of transitions [13] be formed for node b_0 and nodes $b_q \in O(\Gamma_1)$:

$$\begin{aligned}
&b_0 \rightarrow b_1;\\
&b_2 \rightarrow x_1b_3 \vee \bar{x}_1\, x_2b_4 \vee \bar{x}_1\, \bar{x}_2\, b_6;\\
&b_5, b_7 \rightarrow x_3b_8 \vee \bar{x}_3\, x_4b_9 \vee \bar{x}_3\, \bar{x}_4\, b_{11};\\
&b_{10}, b_{12} \rightarrow x_2x_{13} \vee \bar{x}_2\, x_3b_{16} \vee \bar{x}_2\, \bar{x}_3\, b_{18};\\
&b_{15}, b_{17}, b_{18} \rightarrow x_3b_{14} \vee \bar{x}_3\, x_5b_{19} \vee \bar{x}_3\, \bar{x}_5\, b_{21};\\
&b_{20}, b_{22} \rightarrow x_6b_{20} \vee \bar{x}_6\, b_{23};\;\; b_{27} \rightarrow b_E.
\end{aligned} \tag{21}$$

This system is the base to form the table of transitions of CMCU U_2, that has the columns B_i, $K(B_i)$, $C(B_i)$, b_q, $A(B_q)$, y_D, X_h, Φ_h, h. In case of CMCU $U_2(\Gamma_1)$ this table has $H_2(\Gamma_1) = 14$ lines and the first 6 lines are represented by the Table 2.

Table 2

Fragment of the table of transitions of CMCU $U_2(\Gamma_1)$

B_i	$K(B_i)$	$C(B_i)$	b_q	$A(b_q)$	y_D	X_h	Φ_h	h
B_1	00	—	b_3	00010	1	x_1	D_4	1
			b_4	00011	1	x_1x_2	D_4D_5	2
			b_6	00110	1	$\bar{x}_1\bar{x}_2$	D_3D_4	3
B_2	—	0	b_8	01001	0	x_3	D_2D_5	4
			b_9	01010	0	$\bar{x}_3x_4$	D_2D_4	5
			b_{11}	01100	0	$\bar{x}_3\bar{x}_4$	D_2D_3	6

The $y_D = 0$ corresponds to classes $B_i \in \Delta_1$ and $y_D = 1$ corresponds to classes $B_j \in \Delta_2$. This table is the base to form the system (10). One can form from the table 1, for example, the following DNF: $D_3 = \bar{y}_D \bar{\tau}_1 \bar{\tau}_2 \bar{x}_1 \bar{x}_2 \vee y_D \bar{z}_1 \bar{x}_3 \bar{x}_4$. Let us point out that table of transitions of CMCU $U_1(\Gamma_1)$ has $H_1(\Gamma_1 1) = 28$ lines. It means that the amount of terms in the system (10) is 2 times lower, than the amount of terms in the system (3).

The address transformer AT forms variables $\tau_r \in \tau$ and y_D as functions of the addresses of the outputs of OLC $\alpha_g \in B_i$, where $B_i \in \Delta_2$. The table of AT includes the columns b_q, $A(b_q)$, B_i, $K(B_i)$, τ_r, y_D, q. In case of CMCU $U_2(\Gamma_1)$ this table has $H^2_{\mathrm{AT}}(\Gamma_1) = 5$ lines (Table 3).

Table 3

Table of address transformer of CMCU $U_2(\Gamma_1)$

b_q	$A(b_q)$	B_i	$K(B_i)$	τ_r	y_D	q
b_2	00001	B_1	00	–	1	1
b_{10}	01011	B_3	01	2	1	2
b_{12}	01101	B_3	01	2	1	3
b_{20}	11000	B_5	10	1	1	4
b_{22}	11010	B_5	10	1	1	5

This table is the base to form the systems (11)-(12). After minimization we can get from Table 3 the following DNFs: $\tau_1 = T_1 T_2 \bar{T}_3 \bar{T}_5$, $\tau_2 = \bar{T}_1 T_2 \bar{T}_3 T_4 T_5 \vee \bar{T}_1 T_2 T_3 \bar{T}_4 T_5$. It is clear that variable y_D can be expressed using equations for $\tau_r \in \tau$. In our case we have: $y_D = \tau_1 \vee \tau_2 \vee \bar{T}_1 \bar{T}_2 T_3 \bar{T}_4 T_5$.

The implementation of the circuit of CMCU U_2 is reduced to implementation of the systems (10)-(12) using CPLD and implementation of control memory using EMBs. These methods are well-known [3, 4] and they are out the scope of our article.

Let us point out that additional MIs correspond to idle cycles of data path of digital system with CMCU U_2. To work it out, it is enough to ban the synchronization of data path, if MI with format F1 is read out the CM. Let it correspond to some variable $y_C = 1$. If condition

$$\Delta_m \geq 1 \tag{22}$$

holds, then free bit of EMB can be used to keep y_C. If condition (22) is violated, then variable $\mathrm{y_C}$ can be formed as

$$y_C = \bar{y}_0 \bar{y}_D. \tag{23}$$

As a preliminary conclusion, we can point out that the proposed modification of microinstruction format permits to decrease the number of PAL macrocells in the circuit CC in comparison to equivalent CMCU U_1. The number of PAL macrocells in the circuit

AT can be lower, than in other models of CMCU [12]. But this hardware optimization is connected with decrease of performance of digital system. Let us point out that block AT can be eliminated, if condition

$$\Delta_F \leq G_0 \tag{24}$$

holds and variable y_D is formed either by CC or CM.

5. SYNTHESIS OF CMCU U_3

The proposed method of synthesis of CMCU $U_3(\Gamma)$ includes the following steps.

1. Construction of the set C of LFCA Γ.
2. Construction of partition Π_C.
3. Encoding of the classes $B_i \in \Pi_C$.
4. Calculation of the values of parameters Δ_m, Δ_Y, R_4.
5. Natural addressing of microinstructions.
6. Construction of content of control memory.
7. Construction of the table of transitions of CMCU.
8. Construction of the table of address transformer AT.
9. Implementation of the logic circuit of CMCU.

In case of CMCU $U_3(\Gamma_1)$ we have $I = 5$, $R_1 = 3$, $\tau = \{\tau_1, \tau_2, \tau_3\}$. Let us encode the classes $B_i \in \Pi_C$ in a trivial way: $K(B_1) = 000,\ldots, K(B_5) = 100$.

An analysis of the collections $Y(b_q)$, where $b_q \in O(\Gamma_1)$, shows that $Y^1 = Y$ and only bit y_E of the field FY can be used to keep the variables $\tau_r \in \tau$. Let number of outputs $t_M = 4$ for EMBs in use, therefore, expression (15) gives us $n_1 = 2$. It means that two EMBs form the control memory of CMCU $U_3(\Gamma_1)$, Thus, according to (16), we can get $\Delta_m = 1$. Condition (19) is violated because $\Delta_Y = 1$, $\Delta_m + \Delta_Y = 2 < R_1 = 3$. It means that circuit of CMCU $U_3(\Gamma_1)$ should include AT and $R_4 = 1$. Let AT form variable $\tau_1 \in \tau^2$ and let variables τ_2, $\tau_3 \in \tau^1$ kept in CM.

The application of the methods from [12] gives the following addresses of microinstructions: $A(b_1) = 00000$, $A(b_2) = 00001,\ldots, A(b_{27}) = 11010$.

The content of control memory is formed in the following manner:

- if $b_q \in O(\Gamma)$, then field FB of microinstruction MI_q contains $(R_1 - R_4)$ junior bits of the code $K(B_i)$, where $b_q = O_g$, $\alpha_g \in B_i$, field FY contains microoperations $y_n \in Y(b_q)$;
- if $b_q \notin O(\Gamma)$, then $y_0 = 1$, FY $= Y(b_q)$, FB $= \emptyset$;

The first 8 cells of control memory of CMCU $U_3(\Gamma_1)$ are shown in the Table 4.

Table 4

Fragment of control memory of CMCU $U_3(\Gamma_1)$

Address	y_0	FY						FB	Remarks	
$T_1T_2T_3T_4T_5$		y_1	y_2	y_3	y_4	y_5	$y_E/2$	τ_3	b_q	B_i
00000	1	1	1	0	0	0	0	*	b_1	B_1
00001	0	0	0	1	0	0	0	0	b_2	B_1
00010	1	0	0	0	1	0	0	*	b_3	B_2
00011	1	1	0	0	0	1	0	*	b_4	B_2
00100	0	0	1	0	1	0	0	1	b_5	B_2
00101	1	1	0	1	0	1	0	*	b_6	B_2
00110	0	0	1	1	0	0	0	1	b_7	B_2
00111	1	0	1	0	0	1	0	*	b_8	B_3

The table of transitions of CMCU U_3 includes the following columns: B_i, $K(B_i)$, b_q, $A(b_q)$, X_h, Φ_h, h. This table is constructed on the base of the system of formulae of transitions for nodes $b_q \in O(\Gamma)$. Let us point out that transition from initial node b_0 is executed using pulse *Start*. If this transition is conditional one, then additional node $b_t \in E_1$ should be inserted into LFCA Γ. In the same time the edge $\langle b_0, b_t \rangle$ should be included into set E. This step is named as transformation of initial LFCA [6]. Such transformation should be executed for CMCU $U_1 - U_3$, but in case of LFCA Γ_1 this step is eliminated. Another peculiarity of CMCU is execution of transition $\langle b_q, b_E \rangle$, using $y_E = 1$. Because of it the table of transitions reflects only transitions for the outputs of OLC $\alpha_g \in C'$. To form this table the outputs of OLC $\alpha_g \in C'$ should be replaced by classes $B_i \in \Pi_C$, where $\alpha_g \in B_i$. In case of CMCU $U_3(\Gamma_1)$ this table has $H_3(\Gamma_1) = 14$ lines. The first 6 lines of this table are shown in the Table 5.

Table 5

Fragment of the table of transitions of CMCU $U_3(\Gamma_1)$

B_i	$K(B_i)$	b_q	$A(b_q)$	X_h	Φ_h	h
B_1	000	b_3	00010	x_1	D_4	1
		b_4	00011	x_1x_2	D_4D_5	2
		b_6	00100	$\bar{x}_1\bar{x}_2$	D_3	3
B_2	001	b_8	00111	x_3	$D_3D_4D_5$	4
		b_9	01000	$\bar{x}_3x_4$	D_2	5
		b_{11}	01010	$\bar{x}_3\bar{x}_4$	D_2D_4	6

This table is the base to form the system (14). For example, we can form from the Table 4 the expression $D_3 = \bar{\tau}_1\bar{\tau}_2\bar{\tau}_3x_1x_2 \vee \bar{\tau}_1\bar{\tau}_2\tau_3x_3$. As in case of CMCU $U_2(\Gamma_1)$, the number of the terms in system 14 is twice less, than in case of CMCU $U_1(\Gamma_1)$.

The table of address transformer AT has the columns b_q, $A(b_q)$, B_i, $K(B_i)$, τ_m, m. Here the column τ_m includes the variables $\tau_r \in \tau^2$, that are equal to 1 in the code $K(B_i)$ from the m-th line of the table. In case of CMCU $U_3(\Gamma_1)$ $\tau^1 = \{\tau_2, \tau_3\}$, $\tau^2 = \{\tau_1\}$ and table of AT has $H^3_{\mathrm{AT}} = (\Gamma_1) = 2$ lines (Table 6).

Table 6

Table of address transformer of CMCU $U_3(\Gamma_1)$

b_q	$A(b_q)$	B_i	$K(B_i)$	τ_m	m
b_{20}	10011	B_5	100	τ_1	1
b_{22}	10101	B_5	100	τ_1	2

This table is the base to form the system

$$\tau^2 = \tau^2(T). \tag{25}$$

For example, from the table 5 we can get $\tau_1 = T_1\bar{T}_2\bar{T}_3T_4T_5 \vee T_1\bar{T}_2T_3\bar{T}_4T_5$.
The implementation of the circuit of CMCU U_3 is reduced to implementation of the systems (14) and (25) using PAL macrocells and implementation of CM using EMBs. This step is out the scope of this article. If some bits of the field FY are used as variables $\tau_r \in \tau^1$, then corresponding microoperations should be formed , if $y_0 = 1$. This dependence is expressed by formula

$$y_n = y_0 FY\left[y_n\right] \ (n = 1, \ldots, N), \tag{26}$$

where FY$[y_n]$ is a bit of the field FY corresponding to $y_n \in Y \cup \{y_E\}$. If such approach is used, then time of cycle of CMCU U_3 is increased on the propagation time of one PAL macrocell in comparison to the time of cycle of equivalent CMCU U_1.

6. CONCLUSION

The proposed methods of modification of microinstruction format permit to decrease the number of PAL macrocells in the addressing circuit of CMCU. This effect is achieved due to the decrease of amount of terms in the system of excitation functions of the flip-flops of the counter of microinstruction addressing. The optimization is based on existence of free cells or free bits of embedded memory blocks that are used to implement the control memory of CMCU. In this case an address transformer can be used to form either the part of the code of the class of pseudoequivalent OLC or some codes of these classes. In both cases the hardware amount in the circuit of AT is lower than for known models of CMCU [6, 12]. If some conditions satisfy, then block AT is eliminated from CMCU. The volume of CM is the same for CMCU $U_1 - U_3$.

If additional microinstructions (format F1) are in use, then performance of resulted digital system is decreased due to idle cycles of data path. If additional field FB is used (format F2), then the number of cycles for both CMCU U_1 and U_3 are the same, but time of cycle of CMCU U_3 can be more than in case of CMCU U_1. Thus, hardware amount and time characteristics of CMCU depend on both control algorithm to be implemented and parameters of elements in use.

Our research showed that the best solution permits up to 18-26% decrease of hardware amount in comparison with CMCU U_1. Here term "the best" means "the best for given conditions". In this article we have discussed only case of field FY organization with one-hot encoding of microoperations. The proposed methods should be modified if some other strategy is used for field FY. The practical sense has such strategies as maximal encoding of the collections of microoperations and encoding of the fields of compatible mircrooperations [12].

7. REFERENCES

1. C. M a x f i e l d: *The design warrior's guide to FPGAs.* Academic Press, Inc., Orlando, FL, USA, 2004.
2. R. I. G r u s h n i t s k y, A. H. M u r s a e v, E. P. U g r j u m o v: *Design of the systems using the microchips of programmable logic.* BHV, Petersburg, 2002, (in Russian).
3. D. K a n i a: *Logic synthesis for Programmable PAL-based Structures.* Zeszyty Naukowe Politechniki Ślaskiej, Gliwice, 2004.
4. V. V. S o l o v j e v: *Design of digital systems using the programmable logic integrated circuits.* Hot line – Telecom, Moscow, 2001, (in Russian).
5. A. A. B a r k a l o v: *Synthesis of control units on PLDs.* Donetsk National Technical University, 2002, (in Russian).
6. M. A d a m s k i, A. B a r k a l o v: *Architectural and sequential synthesis of digital devices.* University of Zielona Góra Press, 2006.
7. G. D e M i c h e l i: *Synthesis and optimization of digital circuits.* McGraw-Hill, 1994.
8. D. G a j s k i: *Principles of digital design.* Prentice Hall, New York, 1997.
9. A. B a r k a l o v, M. W e g r z y n: *Design of control units with programmable logic.* University of Zielona Góra Press, 2006.
10. E. M c C l u s k e y: *Logic design principles.* Prentice Hall, Englewood Cliffs, 1986.
11. A. D. Z a k r e v s k i, U. V. P o t t o s i n, L. D. C h e r m i s i n o v a: *Background of logic design.* Book 2. Optimization in Boolean space. National Academy of Science of Belarusian, 2004.
12. A. A. B a r k a l o v, A. V. P a l a g i n: *Synthesis of microprogram control units.* IC NAS of Ukraine, Kiev, 1997, (in Russian).
13. S. I. B a r a n o v: *Logic synthesis of control automata.* Kluwer Academic Publishers, 1994.
14. A. A. B a r k a l o v: *Principles of optimization of logic circuit of Moore FSM.* no. 1, 1998, (in Russian)

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 215–241

Structural decomposition of microprogrammed controllers

REMIGIUSZ WIŚNIEWSKI, ALEXANDER BARKALOV

University of Zielona Góra
ul. Podgórna 50, 65-246 Zielona Góra, Poland
e-mail: R.Wisniewski@iie.uz.zgora.pl
A.Barkalov@iie.uz.zgora.pl

Received 2009.01.07
Authorized 2009.03.24

The paper focuses on the structural decomposition of control units. Eight methods of compositional microprogram control units are described and compared. Proposed solutions can be divided into two main groups. The first one deals with CMCUs with mutual memory, where the internal code of the controller is recognized by the microinstruction address. The second group of presented methods is based on control units with sharing codes, where the microinstruction address is formed as a concatenation of codes generated by the counter and by the register. The aim of all proposed solutions is to reduce the number of logic blocks of the destination programmable device.

Keywords: Control Units, Microprogrammed Controllers, Compositional Microprogram Control Units, Programmable Devices, Field Programmable Gate Arrays

1. INTRODUCTION

A control unit (CU) is one of the most important part of any digital system [1,2,3,4,5]. It can be found in almost all devices that contain microelectronics; such as computers (central processor unit, CPU), cellular phones, cars and even remote controllers. The control unit is responsible for managing all modules of the designed system – it sends adequate microinstructions that should be executed [6].

Most of control units that are available on the market are created as a single-level finite-state-machine (FSM). This means that the control unit is formed as a simple Moore or Mealy automaton [7,8]. Such a solution was good for small systems. But the size of devices grows very fast, and now complex digital systems can be implemented using one digital board such as system-on-a-chip (SoC) or system-on-a-programmable-chip

(SoPC). Especially SoPC systems, where logic functions are realized using programmable logic devices (PLDs), complex programmable logic devices (CPLDs) or field programmable gate arrays (FPGAs) are very popular nowadays. Such devices compacts all elements of the design on a single chip that contains built-in logic and dedicated memory blocks [9,10]. Therefore, traditional methods of control units prototyping evolve. One of effective methods of the CU realization is an application of the model of the compositional microprogram control unit (CMCU).

The compositional microprogram control unit is a multi-level device, where the control unit is decomposed into two main units [11,12,13]. The first is responsible for addressing of microinstructions that are kept in the control memory. It is a simple finite-state-machine. The role of the second part is to hold and generate adequate microinstructions. Such a solution may lead to minimization of the number of logic elements that are used for implementation of the CU. Therefore, wider areas of the target device can be accessed by other modules of the designed system. The CMCU memory can be implemented using either logic elements or dedicated memory blocks of a chip [9,14,15].

2. CURRENT STATE OF THE ART

A digital system may be represented by a composition of the *control unit (CU)* and operational unit (OU) also known as a data-path [1,6,16]. The idea of such a defined digital system is illustrated in the Fig. 1.

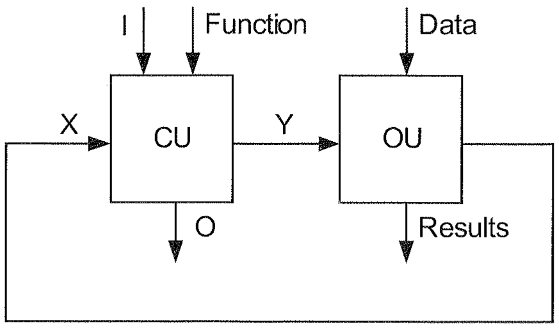


Fig. 1. The model of the digital system

Based on the set of input values (I) and set of logic conditions (X), the CU sends proper microoperations (Y) to the OU. Additionally, a set of output values (O) is generated. The set of inputs (I) and set of outputs (O) are used for communication with the environment of the digital system [17,18].
The operational unit executes microoperations (Y) by processing proper input (Data) and generating results (Results). Additionally the OU generates logic conditions (X) for the control unit.

2.1. SINGLE-LEVEL CONTROL UNITS (FINITE STATE MACHINES)

The most popular realization of control units nowadays is an finite state machine (FSM) also known as the finite state automaton [3,12,16,19,20,21,22]. The FSM is a model of behavior that consists of the set of states, set of transitions between states and set of actions (microoperations). Formally the FSM can be described as a 6-tuple vector:

$$M =< S, X, Y, f, h, s_0 >, \tag{1}$$

where:

- $S = \{s_0, s_1, \ldots, s_K\}$ is non-empty finite set of states;
- $X = \{x_0, x_1, \ldots, x_L\}$ is finite set of inputs;
- $Y = \{y_0, y_1, \ldots, y_N\}$ is finite set of outputs;
- $f : S \times X \to S$ is the transition function, this function determines the next state $s_s \in S$ depending on the current state $s_m \in S$ and on the value of input $x_i \in X$;
- $h : S \times X \to Y$ is the output function, this function determines the current output $y_n \in Y$, based on the current state (in case of Moore automaton) or depending on the current state and the current input (in case of Mealy automaton);
- $s_0 \in S$ is the initial state of automaton.

Figure 2 shows the typical realization of the finite state machine [12,16]. There are two main units in the FSM. The combinational circuit CC generates proper output values (microinstruction) and indicates excitation functions for the register RG which is in charge of holding internal state $s_m \in S$ of an FSM.

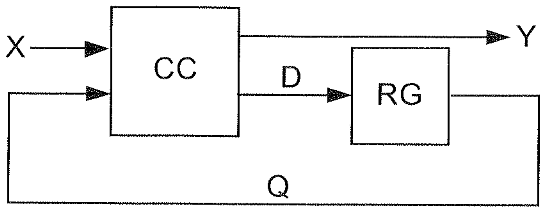


Fig. 2. The model of the finite state machine

The FSM can be realized as Moore or Mealy automaton. If the control unit is described as Moore FSM, then outputs depend on the current state of the automaton [8]. Microinstruction is represented as:

$$Y = f(s_m) \tag{2}$$

where Y_t means the value of the output and $s_m \in S$ represents the current state of the FSM. The main advantage of such a realization is simplification of the behaviour of the control unit. States are clearly tied with the action generating proper microinstruction while inputs (conditions) influence only transitions between states.

The second way of implementation of the FSM is Mealy automaton [7]. The value of outputs depends not only on the current state but also on input signals:

$$Y = f(s_m, X), \tag{3}$$

where X is a set of input values (conditions).

The main benefit offered by Mealy FSM is the reduction of the number of internal states of the automaton in comparison with Moore FSM. Both Moore and Mealy automata are classified as single-level control units.

The optimization of the FSM is one of the most popular tasks nowadays. There are many ideas focused on improving the prototyping process and encoding of internal states of the automaton [23,24,25,26,27,28,29,30,31]. Above researches benefited in appearance of computer-aided design systems, like *Sequential Circuit Synthesis, SIS* [32]. It contains algorithms for state assignments (*NOVA, JEDI*) and state minimization (*STAMINA*).

The next subsection deals with microprogram control units where outputs of the controller are organized in microinstructions.

2.2. MICROPROGRAM CONTROL UNITS

The idea of microprogramming was introduced by M. V. Wilkes in 1951 as an intermediate level to execute computer program instructions [33,18,19,34,35,36,37,38]. Microprograms were organized as a sequence of microinstructions and stored in the special control memory (CM). The algorithm for the MCU is usually specified by the flow-chart (FC) description [11,16]. Such a flow-chart algorithm consists of four main types of vertices (start, end, operational vertex, conditional vertex) that are interpreted by the control unit.

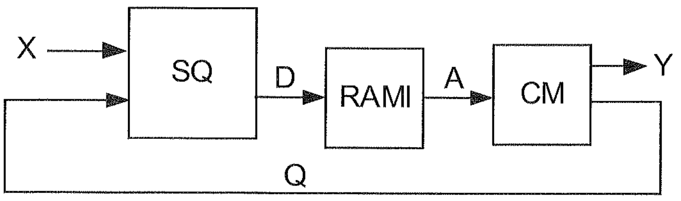


Fig. 3. The model of the microprogram control unit (MCU)

Typical structure of the MCU is presented in the Fig. 3. There are three main blocks that consist of the MCU: a sequencer SQ, a register of the microinstruction address RAMI and the control memory CM [11,16]. The sequencer is the combinational circuit that forms the excitation function for the RAMI:

$$D = f(X, T). \tag{4}$$

Here X is a set of logic conditions of the system and T is a feedback function generated by the control memory. Based on this function, the RAMI generates the proper microinstruction address A. The control memory CM holds microprogram that is further executed by the operational part of the system. There are different methods of microinstructions addressing [39,3], however in most cases the CM also keeps addresses of next microinstructions that should be executed. The feedback function T is analysed by the sequencer which based on the condition from the set X selects the proper address A.

There are many designing ways of the MCU [11,40]. One of the most popular is to perform the sequencer as the multiplexer, and the RAMI as the counter [11]. Then the structure of the MCU may be interpreted as the system shown in the Fig. 4.

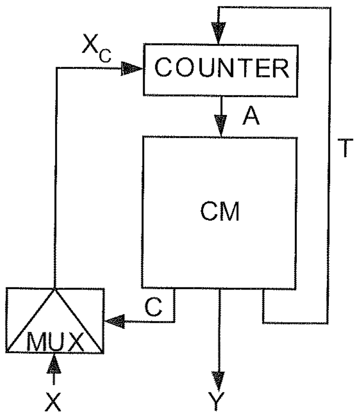


Fig. 4. The model of the microprogram control unit with counter

In the MCU presented in the Fig. 4, the CM generates two feedback functions – T for the counter and C for the multiplexer. Such a realization is especially fruitful in case of long segments (chains) of microinstructions [17]. Then the chain of microinstructions that are not separated by the condition may be replaced by one state (block).

The main advantage of the microprogram control unit is simplicity of its structure. Outputs of the controller are organized in microinstructions and they can be easily replaced. Additionally, the control memory may be implemented using dedicated memory blocks of the FPGA reducing the area of used logic elements.

Apart from its benefits, the MCU has some disadvantages. The control memory holds not only microoperations but also information for calculation of the address of the next microinstruction. Very often the size of the control memory exceeds the size of the dedicated memory block of an FPGA. To eliminate these disadvantages, the control unit may be decomposed.

The control unit may be decomposed in two ways. The first one is functional decomposition. Here the controller is decomposed basing on its internal functions and states. The second method is structural decomposition where the task of the decomposition is reached thanks to the modification of the structure of the control unit. Such a method leads to the compositional microprogram control unit.

3. FUNCTIONAL DECOMPOSITION OF CONTROL UNITS

Functional decomposition is the process that splits the complex function into the smaller sub-functions [41,11,42,43,44,45,46]. Such a realization is often used as a part of logic synthesis of designs implemented with programmable devices. Functional decomposition is widely expanded especially by academic organizations [32,47,48,49,50]. This paper focuses on the decomposition of control units implemented on the FPGA. Optimization of SPLDs and CPLDs can be found in [51,52,53,42,54,55,43,56,57].
In the FPGA, the limited number of inputs and only one output of LUT elements make functional decomposition very effective [46,11,29,58,59]. The idea of functional decomposition is widely used either by commercial (Xilinx, Altera, Synplicity, etc.) and non-commercial organisations (Universities). It should be pointed out that the best results are achieved by non-commercial projects such as DEMAIN (Technical University of Warsaw) or SIS (University of Berkeley).

Functional decomposition may be realized as serial decomposition or parallel decomposition. In the first one, the set X of input variables is split into two subsets U and V [3].

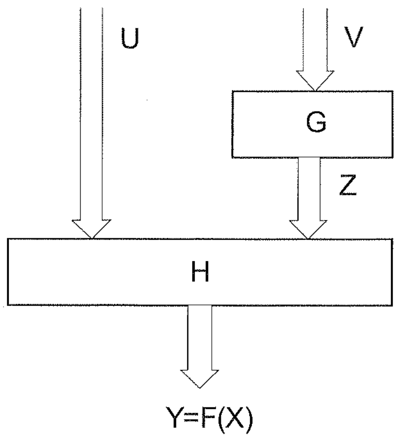


Fig. 5. The idea of serial functional decomposition

The set V forms inputs for the function G which generates the set of outputs $Z = G(V)$. Of course the method has sense only if the number of outputs of the function G is fewer than the number of its inputs. Furthermore, outputs Z generated by G and the set U form inputs for the function H (Fig. 5). Finally, function F is represented as follows:

$$F = H(U, G(V)). \tag{5}$$

The aim of parallel decomposition is to decompose the initial function F into two separate sub-functions G and H [11]. The main idea is to split the set of outputs Y into two subsets Y_G and Y_H. Here Y is the set of outputs of the function F. Similarly Y_G is the set of outputs of the function G and Y_H – the set of outputs of H. The method has sense if either one of functions G or H has fewer input variables than the initial function F. The idea of parallel decomposition is illustrated in the Fig. 6.

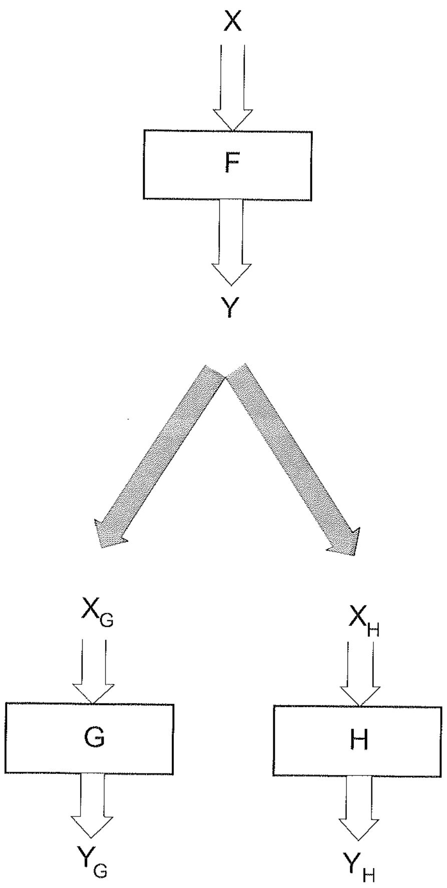


Fig. 6. The idea of parallel functional decomposition

Serial and parallel decompositions are very often combined. Balanced decomposition joins both methods [11,60,61]. The whole process may be divided into steps. In each step either serial or parallel decomposition is performed. The process is repeated until the satisfactory result is reached [11].

Presented methods of decomposition are fruitful for combinational blocks of the system. However they can also be used in decomposition of control units [11,29,62,63]. The controller may be realized as the sequential circuit shown in the Fig. 7. The main idea is to use the control memory to hold microinstructions. Such a memory is implemented with dedicated memory blocks of the FPGA, which reduces the logic resources of the device.

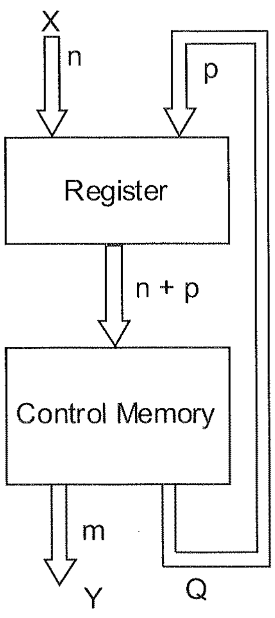


Fig. 7. The control unit realized as the sequential circuit

Each microinstruction of the control unit presented in the Fig. 7 consists of two fields. The first one holds the code Q of internal states of the automaton, while the second contains output variables from the set Y. The next state of the controller is determined by input variables X and by the current state of the control unit. The width of the address of the control memory may be calculated as $|A| = n + p$, where n means the number of the input variables and p represents the number of bits that are used for encoding internal states of the controller [11]. The volume of the control memory depends on the width of its address. Each additional bit doubles the volume of the memory. Thus, very often such a volume exceeds the volume of dedicated memory blocks of the FPGA. The solution to this problem may be functional decomposition of the control unit (Fig. 8).

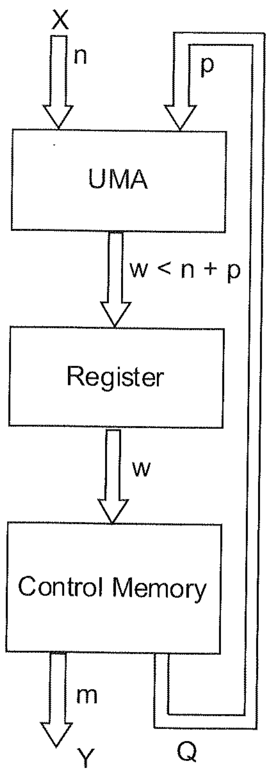


Fig. 8. Functional decomposition of the control unit

In the system shown in the Fig. 8 the control memory is decomposed into two parts: block of the address modification (CAM) and smaller memory that may be realized using the dedicated memory block of the FPGA. There are many variants of such a decomposition [29,62,63]. The main aim of all methods is to decrease the size of the control memory using the minimum number of logic blocks of the FPGA.

The main benefit of the functional decomposition of the control unit is very high effectiveness. The memory may be decomposed in such a way that dedicated memory blocks of the FPGA are used to the maximum. In the other words the minimum number of logic blocks are used to realize the circuit of the address modification. On the another hand, only a part of a microinstruction is held in the memory after the decomposition.

4. STRUCTURAL DECOMPOSITION OF CONTROL UNITS – COMPOSITIONAL MICROPROGRAM CONTROL UNITS

The structural decomposition of control units lead to the new microcontrollers structures, known as Compositional Microprogram Control Units (CMCUs). Any flow-chart of algorithm can be interpreted as the compositional microprogram control unit [13]. In the CMCU the control unit is decomposed into two main parts. The first is responsible for addressing microinstructions that are kept in the control memory. The role of the second part is to hold and generate adequate microinstructions.

Let's introduce some definitions that will be needed later in order to describe the CMCU more formally.

4.1. MAIN DEFINITIONS

Let the control algorithm [1] of a digital system [6,16,40] be represented as the flow-chart Γ [12] with a set of operational vertices $B = \{b_1, \ldots, b_K\}$ and a set of edges E. Each vertex $b_k \in B$ contains microoperations $Y(b_k) \in Y$, where $Y = \{y_1, \ldots, y_N\}$ is the set of microoperations. Each conditional vertex of the flow-chart contains one element from the set of logic conditions $X = \{x_1, \ldots, x_L\}$.

Defininition 1. ***The operational linear chain (OLC)*** of the flow-chart Γ is a finite sequence of operational vertices $\alpha_g = \langle b_{g1}, \ldots, b_{gF_g} \rangle$ such that for any pair of adjacent components of the vector α_g there is an edge $\langle b_{g1}, b_{gi+1} \rangle \in E$, where i is the number of the component in the vector α_g (i=1,...,F_g-1).

Defininition 2. The vertex $b_q \in B$ is called as an ***input of the OLC*** α_g if there is the edge $\langle b_t, b_q \rangle \in B$, where b_t is either initial or conditional vertex of the flow-chart Γ or operational vertex that does not belong to the OLC α_g.

Defininition 3. The vertex $b_q \in B$ is named as an ***output of the OLC*** α_g if there is the edge $\langle b_q, b_t \rangle$, where b_t is either conditional or final vertex of the flow-chart Γ or operational vertex that does not belong to the OLC α_g.

Defininition 4. The parameter M_1 is equal to the number of vertices in the longest OLC α_g of the flow-chart Γ.

Defininition 5. The minimum number of bits required to encode the variable M_1 is represented as R_1 and it is equal to: $R_1 =]log_2 M_1[$.

Defininition 6. The parameter M_2 is equal to the number of all operational chains presented in the flow-chart Γ.

Defininition 7. The minimum number of bits required to encode the variable M_2 is represented as R_2 and it is equal to: $R_2 =]log_2 M_2[$.

Defininition 8. The parameter M_3 is equal to the number of all operational vertices in the flow-chart Γ. This parameter also indicates the total number of microinstructions of the CMCU.

Defininition 9. The minimum number of bits required to encode the variable M_3 is represented as R_3 and it is equal to: $R_3 = \rceil log_2 M_3 \lceil$.

4.2. THE CMCU WITH BASE STRUCTURE

Let D^g be a set of operational vertices that are included in the chain $R_2 = \alpha_g$. Then let $C = \{\alpha_1, \ldots, \alpha_G\}$ be a set of OLCs of the flow-chart Γ satisfied to the condition:

$$\begin{aligned} &D^g \cap D^q = \emptyset (g \neq q; g, q \in \{1, ..., G\}); \\ &B = D^1 \cup D^2 \cup ... \cup D^G; \\ &D^g \neq \emptyset (g = 1, ..., G). \end{aligned} \tag{6}$$

Let natural addressing of microinstructions is executed for each OLC $\alpha_g \in C$:

$$A(b_{gi+1}) = A(b_{gi}) + 1 (i = 1, ..., F_{g-1}), \tag{7}$$

where $A(b_g)$ is an address of microinstruction corresponding to the vertex $b_g \in B$. In this case flow-chart Γ can be interpreted by CMCU U_{BS} with mutual memory (Fig. 9).

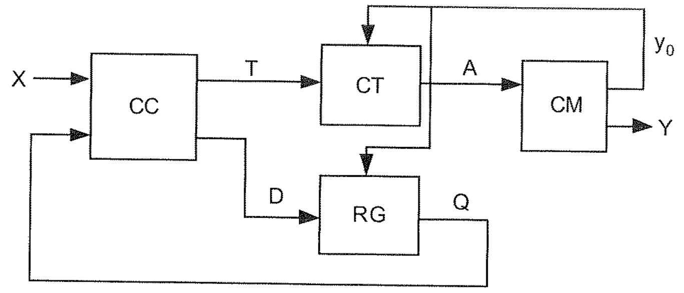


Fig. 9. The compositional microprogram control unit with base structure

There are four main modules in the CMCU U_{BS}: the combinational circuit CC, the register RG, the counter CT and the control memory CM. The combinational circuit and the register represent the simplified FSM of microinstructions addressing S_1. Furthermore, the counter and the control memory form the microprogram control unit S_2. The RG keeps a code $K(a_m)$ of the current state $s_m \in S$ of the CMCU, where $S = \{s_1, \ldots, s_M\}$ is a set of internal states. The register has $\rceil log_2 M_2 \lceil$ flip-flops and their outputs $q_r \in Q$ are used to encode states $s_m \in S$, here $Q = \{q_1, \ldots, q_{R2}\}$, $|Q| = R_2$. The transition from the state $s_m \in S$ to the state $s_s \in S$ is executed by switching the register from the code $K(a_m)$ to the code $K(a_s)$. Such a switching is determined by the excitation function $Q_r \in Q$. The CT keeps the address $A(b_t)$ of the microinstruction $Y(b_t)$ that is executed by a data-path [13]. Variables $a_r \in A$ are used for the representation of the addresses $A(b_k)$. Microinstructions are kept in the CM having 2^{R1} words. Each word (microinstruction) has N+2 bits in a case of one-hot encoding of microoperations

[64,65]. One of additional bits is used to keep an variable y_0 to organize the mode of addressing (7). The second bit keeps a variable y_K to terminate the fetching of microinstructions from the control memory.

The presented CMCU U_{BS} operates in the following manner: at the beginning the register is set to the value that corresponds to the initial state of the FSM. Similarly, the counter is set to the address of the first microinstruction. If transitions are executed inside the OLC $\alpha_g \in C$, then $y_0 = 0$ which causes the increment of the CT and forbids changing the state of the CMCU. When the output of the OLC $\alpha_g \in C$ is reached then $y_0 = 1$ and the combinational circuit forms the excitation function for the register setting it into the proper state [66,67,68]. Similarly the counter is set with the proper value as well:

$$D = f(Q, X), \tag{8}$$

$$T = f(Q, X). \tag{9}$$

Here X means the set of conditions, Q is the set of internal variables used to encode the current state of the CMCU, D is a set of variables that form an excitation function for the register $D = \{d_1, \ldots, d_{R2}\}$ and T is a set of variablesthat form an excitation function for the counter $T = \{t_1, \ldots, t_{R2}\}$.

Functions (8) and (9) form a code $K(s_m)$ of the state of the transition in the register and an address of the input of the next OLC $\alpha_g \in C$. If the CT contains the address of the microinstruction $Y(b_k)$ such as $\langle b_t, b_q \rangle \in E$, then $y_K = 1$. In this case the operation of the CMCU UBS is finished [69,70,71].

The main benefit of the realization of the controller as the compositional microprogram control unit U_{BS} is a possibility of implementation of the circuit CM using dedicated memory blocks [72]. Other blocks of the prototyping system U_{BS} are implemented with the logic blocks (flip-flops and LUT elements) of the FPGA [3,73,74]. Such an idea lead to reduction of the number of logic blocks in comparison with the realization of the controller as a traditional finite state machine and thus, the designer can allocate wider area of the FPGA for another blocks of the prototyping system. The effectiveness of the CMCU is especially high if the controller interprets the linear flow-chart. Such flow-chart contains 75% of operational vertices or includes long linear chains (segments) of operational vertices.
The second advantage of the CMCU is the possibility of selecting the implementation method of the control memory. The designer can decide if the circuit CM should be realized with logic blocks or with dedicated memory blocks. It is important especially incase of designs, which consumes large area of the memory. Then the whole CMCU is implemented with logic blocks of the FPGA.

In opposition to functional decomposition, structural decomposition of a control unit permits to apply the idea of partial reconfiguration [14,75,76]. In this case, only a part of the controller (the control memory) can be replaced while the rest of the system remains untouched.

4.3. WITH THE CMCU MUTUAL MEMORY

The structure of the CMCU U_{MM} with mutual memory is presented in the Fig. 10. There are three main blocks in the CMCU U_{MM}: the combinational circuit CC, the counter CT and the control memory CM [77].

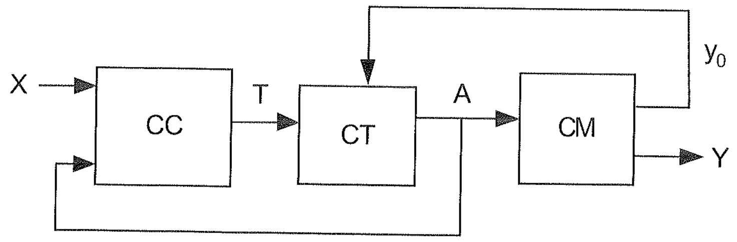


Fig. 10. The structure of the CMCU with mutual memory

Distinct from the CMCU U_{BS} with base structure, in the CMCU U_{MM} the combinational circuit generates the excitation function only for the counter:

$$T = f(Q, X). \tag{10}$$

where X means the set of conditional vertices and A means the code that is determined by the counter. Such a code is also the address of the microinstruction that is kept in the control memory. The number of logic functions is decreased in comparison with the CMCU U_{BS}, because the circuit CC doesn't generate the excitation function for the register. Thus the number of logic blocks of the destination programmable device is reduced [40,78,79,80].

In the CMCU U_{MM} transitions between internal states of the controller are performed in the different way than it is in the CMCU with base structure. Here the address generated by the counter is used to recognize the proper state of the control unit.
The controller operates as follows: at the beginning, the counter is set to the value that corresponds to the initial state of the FSM which is equal to the address of the first microinstruction. If transitions are executed inside the $\alpha_g \in C$, then $y_0 = 0$. It causes the incrementation of the CT and forbids changing the current state of the control unit. When the output of the $\alpha_g \in C$ is reached, $y_0 = 1$ and the circuit CC forms the excitation function for the counter (10). This function forms the code $K(s_s)$ of the state of transition and the address of the input of the next OLC $\alpha_g \in C$ as well. If the controller reaches an address of the microinstruction $Y(b_k)$ such as $\langle b_k, b_E \rangle \in E$, then $y_K = 1$. In this case, operation of the CMCU U_{MM} is finished.

4.4. THE CMCU WITH FUNCTION DECODER

The microprogram control unit with function decoder U_{FD} is an extended structure of the CMCU with mutual memory [78,81]. In comparison to the controller U_{MM} there is additional circuit (function decoder, FD) introduced. Figure 11 illustrates the CMCU with function decoder.

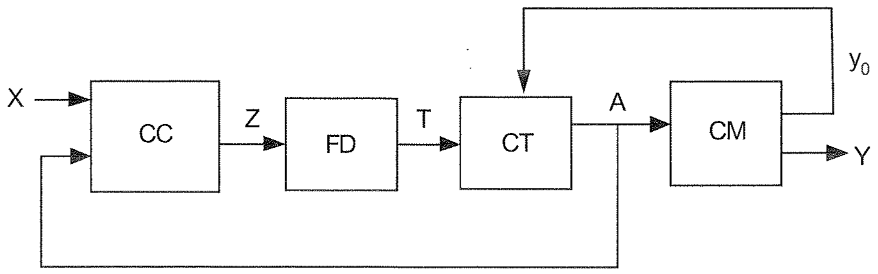


Fig. 11. The structure of the CMCU with function decoder

The main idea of the method is to reduce the number of logic blocks of the destination FPGA due to the usage of additional block (function decoder) which may be implemented using dedicated memories. Therefore, fewer LUT elements are used during the realization of the control unit in comparison with the CMCU with mutual memory.

In the CMCU U_{FD} variables that form excitation function for the counter are encoded with the minimum number of bits. To solve this task all inputs of operational linear chains ought to be encoded. Now the circuit CC generates the function Z:

$$Z = f(X, A). \tag{11}$$

Function Z contains encoded addresses $E(I)$ of all inputs in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter:

$$T = f(Z). \tag{12}$$

The number of bits that are required to encode all inputs can be calculated as $R_Z = \rceil log_2 M_Z \lceil$, where $M_Z = |I|$ is equal to the number of all inputs in the set of OLCs.
Presented solution permits to reduce the number of outputs generated by the circuit CC. Additional block of the function decoder is implemented with dedicated memories of FPGAs. Therefore, the number of logic elements that are needed to implement whole controller is reduced.

4.5. THE CMCU WITH OUTPUTS IDENTIFICATION

The structure of the CMCU U_{OI} with outputs identification is illustrated by the Fig. 12. The main idea is to use the part of the address A for the identification of the internal states of the control unit. Now the set of variables Q ($Q \subset A$) represents the code of the current state of the controller [80,82].

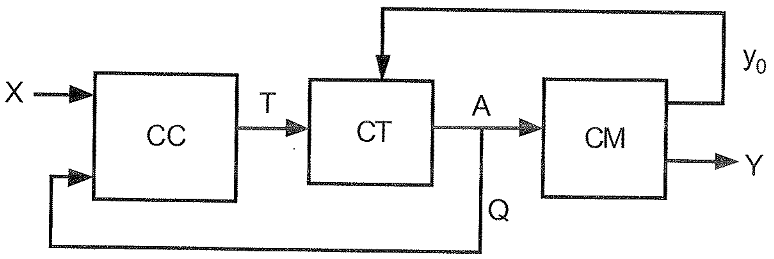


Fig. 12. The structure of the CMCU with outputs identification

In the CMCU U_{OI} the set of feedback variables A that are used for the identification of the current state of the controller is reduced to the minimum. Outputs of the operational linear chains may be recognized using R_{OI} bits thanks to the special encoding of microinstructions [83,84,85,86,87]. Therefore, the combinational circuit generates the function T for the counter [83,88]:

$$T = f(X, Q). \tag{13}$$

4.6. THE CMCU WITH OUTPUTS IDENTIFICATION AND FUNCTION DECODER

The CMCU U_{OD} with outputs identification and function decoder (Fig. 13) is a conjunction of two structures presented in previous sections. There is a special addressing of microinstructions used in the CMCU U_{OD}. Moreover, maximal encoding of the set of variables A is performed as well.

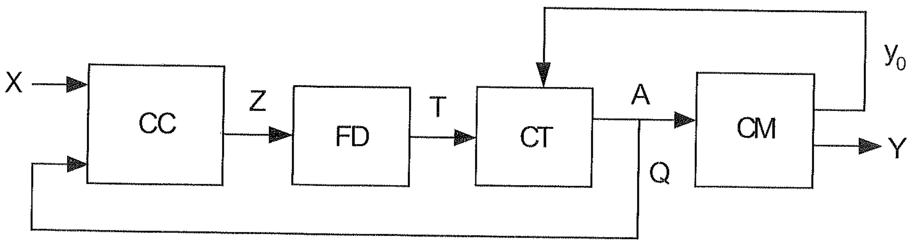


Fig. 13. The structure of the CMCU with outputs identification and function decoder

To improve the reduction of LUT elements of the implementation of CMCUs U_{FD} and U_{OI}, both methods may be combined. Now the combinational circuit generates the excitation function Z for the circuit FD:

$$Z = f(X, Q). \tag{14}$$

where X means the set of input variables of the CMCU (conditional vertices) and $Q \subseteq A$ is a feedback function generated by the counter. The function decoder generates proper addresses of microinstructions:

$$T = f(Z). \tag{15}$$

where T means the set of variables that form the excitation function for the counter.

4.7. COMPOSITIONAL MICROPROGRAM CONTROL UNITS WITH SHARING CODES

In a CMCU with sharing codes the microinstruction address is determined by both codes generated by the counter and by the register. The aim of such a method is the reduction of logic blocks of the FPGA. Figure 14 shows the CMCU U_{SC} with sharing codes [89,90,91,92]. The main idea is to use both codes generated by the counter and by the register to form the microinstruction address. Therefore, the number of variables that are used for encoding of the excitation functions for the counter is reduced in comparison to the CMCU U_{BS}.

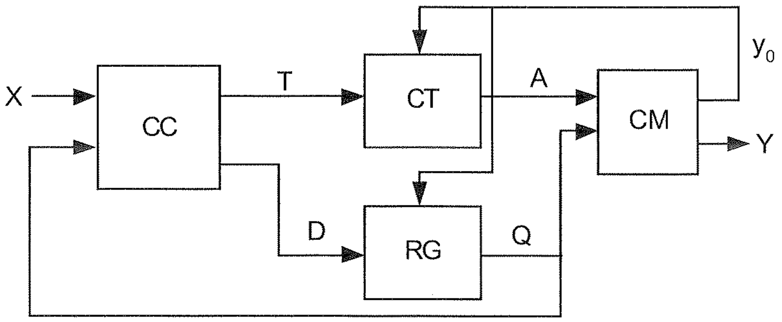


Fig. 14. The structure of the CMCU with sharing codes

In the CMCU with sharing codes the microinstruction address $A(b_t)$ is represented as a concatenation [39]:

$$A(b_t) = K(\alpha_g) * K(b_t). \tag{16}$$

Here $K(\alpha_q)$ is a code of the OLC $\alpha \in C$ with $R_2 =]log_2M_2[$ bits, where M_2 defines the number of OLCs in the initial flow-chart Γ; $K(b_t)$ is a code of a component of the OLC $\alpha_g \in C$ corresponding to the vertex $b_t \in B$. Code $K(b_t)$ has $R_I=]log_2M_I[$ bits, where M_I is equal to the maximum amount of components in the OLC $\alpha_g \in C$. Sign (*) in (15) is used for concatenation operation.

In the CMCU U_{SC} the combinational circuit CC generates excitation functions for the counter CT and for the register RG:

$$T = f(X, Q), \tag{17}$$

$$D = f(X, Q), \tag{18}$$

The RG is in charge of holding the code of the current OLC. Additionally it generates an upper part of the microinstruction address. The CT keeps only the number of the active component (block) in the current OLC. Therefore it determines the lower part of the microinstruction address.

4.8. THE CMCU WITH SHARING CODES AND FUNCTION DECODER

The CMCU U_{SD} with sharing codes and function decoder is shown in the Fig. 15. The main idea is to reduce the number of outputs of the combinational circuit thanks to the encoding of the excitation functions for the counter and the register. Therefore, the number of logic blocks required for implementation of the CMCU is reduced. The additional block – function decoder – decodes and sends proper values for the counter and for the register. Function decoder can be implemented with dedicated memory blocks.

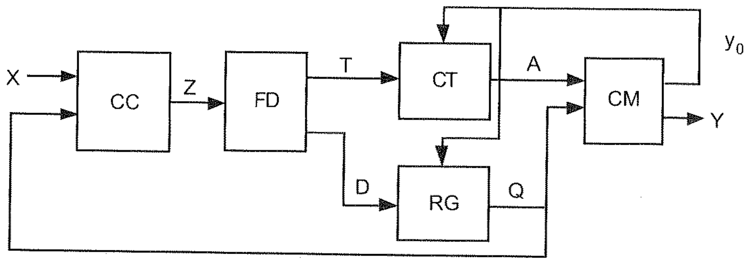


Fig. 15. The CMCU with sharing codes and function decoder

In the CMCU U_{SD}, the set of variables that form the excitation function T for the counter and the set of variables that form the excitation function D for the register are encoded. Similarly to the CMCUs U_{FD} and U_{OF} shown in the previous section, all inputs of the set of OLCs are encoded. The combinational circuit generates the excitation function Z for the function decoder:

$$Z = f(X, Q), \tag{19}$$

Function Z contains encoded addresses Q of all inputs I in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter and for the register:

$$T = f(Z), \tag{20}$$

$$D = f(Z), \tag{21}$$

4.9. THE CMCU WITH ADDRESS CONVERTER

The method of sharing codes makes sense only if the size of codes generated by the register RG and by the counter CT is equal to the width of the microinstruction address [13]. Then the following condition is fulfilled:

$$R_1 + R_2 = R_3. \tag{22}$$

In most cases the total number of bits generated by the register and by the counter exceeds the width of the microinstruction address. The condition (22) is violated because $R_1 + R_2 > R_3$ and the volume of the control memory grows drastically. The minimum volume of the memory can be calculated as:

$$S_{CM} = (N + 2) * 2^{R3}. \tag{23}$$

where S_{CM} means the total volume of the control memory, N+2 counts the total number of microoperations kept in the control memory (N is the number of microoperations while two additional bits are formed by y_0 and y_K), and R_3 defines the minimum width of the address. It is clear that each additional bit in the microinstruction address doubles the total volume of the memory.

To solve such a problem the CMCU with address converter can be used. The method is based on the application of the additional block (address converter) in the CMCU structure (Fig. 16). Such an approach has sense only if the condition (22) is violated and the total quantity of codes generated by the register and by the counter is greater than the width of the address of the control memory.

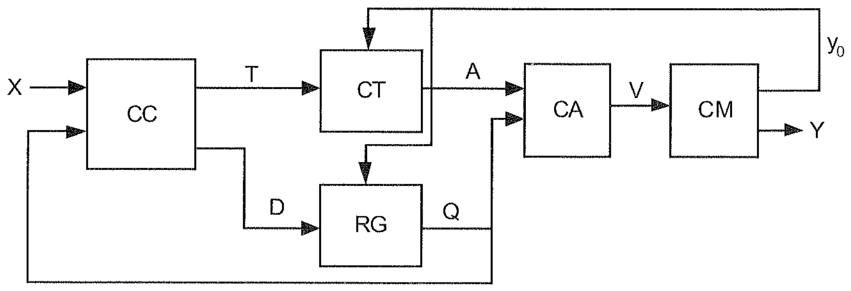


Fig. 16. The structure of the CMCU with address converter

Let $K(\alpha_g)$ be the state code of the register and $K(b_t)$ the state code of the counter. According to the (14), the microinstruction address $A(b_t)$ is calculated as the concatenation of these codes:

$$A(b_t) = K(\alpha_g) * K(b_t).$$

In the CMCU U_{CA} the address generated by the register and by the counter is converted by the address converter.
Now the circuit CC forms the system of functions:

$$T = f(X, Q), \tag{24}$$

$$D = f(X, Q), \tag{25}$$

and the circuit CA converts generated addresses, forming the new function V:

$$V = V(Q, X), \tag{26}$$

Here $V = \{v_1, \ldots, v_{R3}\}$ is the set of addresses of the control memory.
Presented solution permits to combine the positive features of the traditional CMCU with base structure (U_{BS}) and with sharing codes (U_{SC}) such as:

- minimal number of inputs and outputs of the combinational circuit CC (compared with the U_{SC});
- minimal width of an address of the control memory (in comparison with the U_{SC}).

It is clear that application of a given method makes sense only if the implementation of the CMCU with additional address converter requires fewer memory blocks of the destination FPGA than CMCUs based on the standard structure U_{SC}.

4.10. THE CMCU WITH ADDRESS CONVERTER AND FUNCTION DECODER

This section presents the last method of synthesis of the CMCU with sharing codes that is proposed in the paper – the CMCU U_{CD} with address converter and function decoder. Such a controller combines two ideas presented in previous sections. Application of the address converter permits to minimize the volume of control memory if the condition (22) is violated, while the additional function decoder reduces required logic elements for implementation of the CMCU.

The CMCU U_{CD} with address converter and function decoder is shown in the Fig. 17. Excitation functions T for the counter and D for the register are encoded with the minimum number of bits. Now the combinational circuit CC generates the excitation function Z for the function decoder:

$$Z = f(X, Q), \tag{27}$$

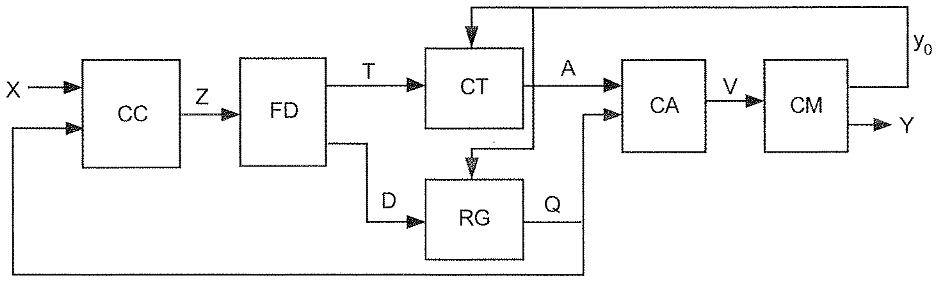


Fig. 17. The CMCU with address converter and function decoder

Function Z contains encoded addresses Q of all inputs I in the set of OLCs. They are further decoded by the circuit FD which indicates the proper code for the counter and for the register:

$$T = f(Z), \tag{28}$$

$$D = f(Z), \tag{29}$$

Finally the address indicated by the counter and by the register is converted via the circuit CA:

$$V = f(T, D), \tag{30}$$

5. RESULTS OF EXPERIMENTS

This section presents results that were achieved during the logic synthesis and implementation of CMCUs. All synthesis methods were verified with over 70 benchmarks. Additionally, there was an FSM model prepared for each test. The automaton was created according to the rules presented in [93,94]. It should be pointed out that all FSMs were prepared in such a way, that during implementation, all microoperations were realized with dedicated memory blocks of the FPGA.

The prototyping process for each benchmark was similar. Based on the flow-chart description (*.fc* file), the controller was structurally decomposed with all 8 synthesis methods presented in the article. Additionally, there was an equivalent FSMproduced. Achieved Verilog codes were finally synthesised and implemented with the Xilinx XST tool.

Table 1 presents average results of the CMCU implementation designed with the particular synthesis method in comparison to the FSM and to the traditional CMCU with mutual memory. As the destination, the FPGA XC2VP30 (Xilinx Virtex-II Pro family) was selected. The device contains 27392 Flip-Flops, 27392 LUTs (13696 Slices) and 136 dedicated memory blocks (Block-RAMs). The device was selected because of its structure (it can be partially reconfigured) and its availability at University of Zielona Góra.

Table 1

Average results of experiments

	FPGA	Designing method								
	resources	FSM	MM	FD	OI	OD	SC	SD	CA	CD
	Slices	100%	91%	73%	76%	60%	57%	51%	53%	50%
Comparison	FF	100%	100%	105%	102%	108%	120%	127%	122%	125%
to the FSM	LUTs	100%	91%	71%	78%	60%	57%	50%	54%	49%
	BRAMs	100%	100%	136%	102%	126%	279%	320%	151%	186%
Comparison	Slices	110%	100%	82%	84%	68%	62%	57%	60%	57%
to the CMCU	FF	100%	100%	105%	102%	108%	120%	127%	122%	125%
with mutual	LUTs	110%	100%	81%	86%	68%	63%	57%	62%	57%
memory	BRAMs	100%	100%	136%	102%	126%	279%	320%	151%	186%

The meaning of abbreviations:
- FSM – realization of the controller as the FSM;
- MM – realization of the controller as the CMCU with mutual memory;
- FD – realization of the controller as the CMCU with function decoder;
- OI – realization of the controller as the CMCU with outputs identification;
- OD – realization of the controller as the CMCU with outputs identification and function decoder;
- SC – realization of the controller as the CMCU with sharing codes;
- SD – realization of the controller as the CMCU with sharing codes and function decoder;
- CA – realization of the controller as the CMCU with address converter;
- CD – realization of the controller as the CMCU with address converter and function decoder.

6. CONCLUSIONS

The detailed analysis of results of investigations shows, that the number of logic blocks that are required for implementation of the controller in the FPGA is strongly tied with the number of microinstructions that are held in the control memory.

In case of relatively small devices where the control memory may be implemented with one dedicated memory block, the realization of the controller as the CMCU U_{SD}

with sharing codes and function decoder gives the best results. Firstly, it requires average the fewest number of logic blocks among all presented methods. Furthermore, the control memory is implemented with one dedicated memory block, thus there is no need for application of the address converter. Obviously application of the function decoder is optional – its usage reduces the number of logic blocks but increases the number of dedicated memories.

According to the (21), if the total number of bits generated by the register and counter exceeds the width of the microinstruction address, the CMCU U_{CD} with address converter and function decoder ought to be selected. It should be pointed out that results gained during realization of the controller as the CMCU U_{SD} are similar to the values achieved for the CMCU U_{CD}. The number of required logic blocks for implementation of both controllers are almost the same. However, in case of control units that contain memories that ought to be decomposed (their volume exceeds the volume of one dedicated memory block), the CMCU with address converter requires on average by 46% fewer dedicated memory blocks than the CMCU with sharing codes. These results prove the effectiveness of application of the address converter in case of CMCUs, where the address indicated by the counter and by the register is wider than the minimum number of bits needed for microinstructions addressing.

The CMCU U_{SD} with address converter and function decoder consumes the fewest number of logic blocks of the destination FPGA in case of controllers where the control memory is decomposed (which means that more than one BRAM is used). Such a realization requires only 49% LUTs in comparison to the FSM and 57% in comparison to the CMCU with mutual memory. It means that the proposed synthesis method with address converter and function decoder reduces the number of logic blocks that are used for implementation of the controller over two times in comparison to the traditional automaton. On the other hand, there are more dedicated memory blocks required for realization of the control unit. The number of dedicated memory blocks increases on average by 86%, therefore the CMCU U_{CD} is the best solution for implementation of the controller in FPGAs that contain enough dedicated memory blocks. Finally, among controllers that produce more than 150 microinstructions, the CMCU U_{OD} with outputs identification and function decoder gives the best results. In this case, such arealization on average requires the fewest dedicated memory blocks and usually the fewest logic blocks as well.

The criteria of all experiments were to reduce the number of logic blocks that are required for the controller implementation. The detailed analysis of the results of experiments showed that selection of the proper synthesis method may be tied with the structure of the CMCU. There are three typical situations when the proper synthesis algorithm can be proposed:

- In case of relatively small systems (where the number of microinstructions does not exceed 150 and the control memory can be implemented with one dedicated memory block), the CMCU with sharing codes and function decoder seems to be the best solution. However, it should be pointed out that such a realization consumes at least two dedicated memory blocks of the FPGA. Therefore, if a number of available dedicated memory blocks is limited, the method with outputs identification should be used.

- In case of controllers where the volume of the control memory exceeds the volume of one dedicated memory block and the total number of microinstructions is fewer than 150, the CMCU with address converter and function decoder gives the bestresults.
- In case of controllers where the total number of microinstructions exceeds 150, the CMCU with outputs identification and function decoder ought to be selected.

7. REFERENCES

1. G. D e M i c h e l i: *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, New York, 1994.
2. A. C l e m e n t s: *The principles of computer hardware*. Oxford University Press, New Jersey, 2000.
3. T. Ł u b a: *Synteza układów cyfrowych*. WKŁ Warszawa, 2003.
4. M. B o l t o n: *Digital Systems Design with Programmable Logic*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1990.
5. D. B u r s k y: *Embedded logic and memory find a home in FPGAs*. Electronic Design, 1999, vol. 47, no 14, pp. 43-56.
6. D. G a j s k i: *Principles of Digital Design*. Prentice Hall, New Jersy, 1997.
7. G. M e a l y: *A method for synthesizing sequential circuits*. BSTJ, 1955, vol. 34, pp. 1045-1079.
8. E. M o o r e: *Gedanken experiments on sequential machines*. In C. E.Shannon and J. McCarthy, editors, Automata Studies, 1956, pp. 129-153.
9. A l t e r a: Embedded memory in Altera FPGAs. Altera, http://www.altera.com/technology/memory/embedded/mem-embedded.html, 2006.
10. X i l i n x: Using Block SelectRAM+ Memory in Spartan-II FPGAs. www.xilinx.com/bvdocs/appnotes/xapp130.pdf, 2000.
11. T. Ł u b a: *Synteza układów logicznych*. Oficyna Wydawnicza PW, Warszawa, 2005.
12. S. I. B a r a n o v: *Logic Synthesis for Control Automata*. Kluwer Academic Publishers, Boston, 1994.
13. A. B a r k a l o v: *Synthesis of Control Units on PLDs*. DonNTU, Donetsk, 2002.
14. R. W i ś n i e w s k i: *Częściowa rekonfiguracja mikroprogramowanych układów sterujących implementowanych z wykorzystaniem struktur FPGA*. Proceedings of PTETIS, 2005, Vol.21, ss. 239-242.
15. X i l i n x: Two flows for partial reconfiguration. http://direct.xilinx.com/bvdocs/appnotes/xapp290.pdf, 2004.
16. A. B a r k a l o v, M. W ę g r z y n: *Design of Control Units with Programmable Logic*. University of Zielona Góora Press, Zielona Góra, 2006.
17. T. Ł u b a: *Synteza układów logicznych*. WSISiZ, Warszawa, 2001.
18. M. M o l s k i: *Modułowe i mikroprogramowalne układy cyfrowe*. WKŁ, Warszawa, 1986.
19. W. T r a c z y k: *Układy cyfrowe. Podstawy teoretyczne i metody syntezy*. WNT, Warszawa, 1982.
20. M. A d a m s k i, M. W ę g r z y n, A. W ę g r z y n: *Safe reconfigurable logic controllers design*. In ed. by J. Korbicz, editor, Measurements models systems and design, WKŁ, 2007, pp. 343-370.

21. M. Adamski and M. Węgrzyn: *Reprogrammable controllers for reactive embedded systems*. Proceedings volume from the 26th IFAC/IFIP/IEEE Workshop, Elsevier, Oxford, 2003, pp. 39-44.
22. M. Adamski: *Programowane asynchroniczne układy sterujące z samosynchronizacją*. KKA'80, 1980, Szczecin, Polska, ss. 203-208.
23. E. Sentovich, K. J. Singh, Cho W. Moon, H. Savoj, R. K. Brayton, A. L. Sangiovanni-Vincentelli: *Sequential circuit design using synthesis and optimization*. Proceedings of the ICCD '92, 1992, Washington, DC, USA, pp. 328-333.
24. E. Hrynkiewicz, K. Pucher, D. Kania: *The input partitioning and coding problem in PAL -based CPLD* s. XX National Conference Circuit Theory and Electronic Networks, 1997, pp. 145-152.
25. P. Ashar, S. Devadas, A. R. Newton: *A unified approach to the decomposition and re-decomposition of sequential machines*. Proceedings of the 27th ACM/IEEE conference on DAC, New York, NY, USA, 1990, pp. 601-606.
26. P. Ashar, S. Devadas, A. R. Newton: *Sequential Logic Synthesis*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.
27. H. Kubatova: *Finite state machine implementation in FPGAs.*, Design of Embedded Control Systems, Springer, New York, 2005, pp. 177-187.
28. M. Perkowski, L. Jóźwiak, W. Zhao: *Symbolic two-dimensional minimization of strongly unspecified finite state machines*. Journal of Systems Architecture, Vol. 47, 2001, pp. 15-28.
29. M. Rawski, H. Selvaraj, T. Łuba: *An application of functional decomposition in rom-based fsm implementation in FPGA devices*. Proceedings of the DSD '03, Washington, DC, USA, 2003, p. 104.
30. A. Barkalov: *Principles of optimization of logical circuit of Moore finite state-machine*. Cybernetics and System Analysis, no. 1, 1998, pp. 65-72.
31. I. Ahmad, F. Ali, and R. Ul-Mustafa: *An Integrated State Assignment and Flip-Flop for Selection Technique FSM Synthesis*. Microprocessors and Microsystems, 2000, pp. 141-152.
32. E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli: *SIS: A system for sequential circuit synthesis*. Technical Report UCB/ERL M92/41, U.C. Berkeley, 1992.
33. M. V. Wilkes: *The best way to design an automatic calculating machine*. Manchester University inaugural conference, Manchester, England, 1951.
34. S. S. Husson: *Microprogramming – Principles and Practices*. Prentice Hall, New York, 1970.
35. L. Kravcov, G. Chernicki: *Design of microprogram control units*. Energia, Leningrad, 1976 (in Russian).
36. P. Misiurewicz: *Podstawy techniki cyfrowej*. WNT, Warszawa, 1982.
37. Ch. A. Papachristou: *A scheme for implementing microprogram addressing with programmable logic arrays*. Digital Processes, vol. 5, no. 3-4, 1979, pp. 235-256.
38. W. Stalings: *Computer organization and architecture. Prentice Hall*, New Jersey, 1996.
39. A. Barkalov. A. Palagin: *Synthesis of Microprogram Control Units*. IC NAC of Ukraine, Kiev, 1997.
40. M. Adamski, A. Barkalov: *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, Zielona Góra, 2006.
41. D. Kania, J. Kulisz, A. Milik, R. Czerwieński: *Modele dekompozycji przeznaczone dla struktur matrycowych*. RUC'2005, ss. 77-84, 2005.
42. S. Devadas, A. R. Wang, A. R. Newton, and A. L. Sangiovanni-Vincentelli: *Boolean decomposition of programmable logic arrays*. CICC'88, 1988.

43. T. Sasao: *Totally undecomposable functions: Applications to efficient multiple-valued decompositions.* ISMVL '99: Proceedings of the ISMVL '99, Washington, DC, USA, 1999, p. 59.
44. E. McCluskey: *Logic design principles.* Prentice Hall, Englewood, 1986.
45. M. Rawski, T. Łuba, Z. Jachna, P. Tomaszewicz: *The influence of functional decomposition on modern digital design process.* Design of Embedded Control Systems, Springer, Boston, 2005, pp. 193-206.
46. C. Scholl: *Functional decomposition with application to FPGA synthesis.* Kluwer Academic Publishers, 2001.
47. D. Kania, J. Kulisz: *Logic synthesis for PAL-based CPLD-s based on two-stage decomposition.* J. Syst. Softw., 80(7), 2007, pp. 1129-1141.
48. D. Kania: *A new approach to logic synthesis of multi-output boolean functions on PAL -based CPLD* s. Proceedings of the GLSVLSI '07, New York, NY, USA, 2007, pp. 152-155.
49. T. Łuba, M. Rawski, Z. Jachna: *Functional decomposition as a universal method of logic synthesis for digital circuits.* Proceedings of the MixDes'02, Wrocław, Poland, 2002, pp. 285-290.
50. M. Rawski, L. Jóźwiak, T. Łuba: *Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures.* Journal of Systems Architecture, vol. 47, 2001, pp. 137-155.
51. D. Kania: *Synteza logiczna przeznaczona dla matrycowych struktur programowalnych typu PAL.* Zeszyty Naukowe Politechniki Śląskiej, Gliwice, 2004.
52. D. Kania: *Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition.* Procedings of 25-th Euromicro Conference. IEEE Computer Society Press, 1999, pp. 278-281.
53. M. Ciesielski, S. Yang: *PLA de: a two-stage PLA decomposition.* IEEE Trans. on CAD of Integrated Circuits and Systems, vol. 11(8), 1992, pp. 943-954.
54. S. Devadas, A. Wang, R. Newton, and A. L. Sangiovanni-Vincentelli: *Boolean decomposition in multilevel logic optimization.* IEEE Journal of solid-state circuits, 1989, pp. 399-408.
55. V. Muthukumar, R. J. Bignall, H. Selvaraj: *An efficient variable partitioning approach for functional decomposition of circuits.* J. Syst. Archit., no. 53 (1), 2007, pp. 53-67.
56. E. M. Sentovich: *Sequential Circuit Synthesis at the Gate Level.* PhD thesis, 1993. (Chair-Robert K. Brayton).
57. V. Solovjev: *Design of the Functional Units of Digital Systems Using Programmable Logic Devices.* Bestprint, Minsk, 1996.
58. J. Lach, E. Sapiecha, B. Zbierzchowski: *Synteza układów sekwencyjnych w strukturach FPGA z wbudowanymi blokami pamięci.* Przegląd Telekomunikacyjny i Wiadomości Telekomunikacyjne, nr 2-3, 2003, ss. 81-86.
59. J. Pasierbiński, P. Zbysiński: *Układy programowalne w praktyce.* WKŁ, Warszawa, 2001.
60. M. Rawski, P. Tomaszewicz, H. Selvaraj, T. Łuba: *Efficient implementation of digital filters with use of advanced synthesis methods targeted FPGA architectures.* Proceedings of DSD '05, Washington, DC, USA, 2005, pp. 460-466.
61. H. Selvaraj, T. Łuba: *A balanced multilevel decomposition method.* Proceedings of EDTC '95, Washington, DC, USA, 1995, p. 594.
62. G. Borowik: *Synteza układów sekwencyjnych w sieciach wbudowanych matryc logicznych struktur FPGA.* Proceedings of OWD'04, vol. 19, Wisa, Polska, 2004, ss. 361-366.
63. G. Borowik: *FSM coding for optimal serial decomposition.* Proceedings of OWD'05, vol. 21, Wisa, Polska, 2005, pp. 243-248.
64. A. Barkalov, L. Titarenko, R. Wiśniewski: *Synthesis of compositional microprogram control units with transformation of the numbers of inputs.* Proceedings of CADSM 2005, Lviv – Polyana, Ukraina, 2005, pp. 181-184.

65. A. B a r k a l o v, R. W i ś n i e w s k i: *Design of compositional microprogram control units with maximal encoding of inputs*. Radioelektronika i Informatika, no 3, 2004, pp. 79-81.
66. A. B a r k a l o v, R. W i ś n i e w s k i: *Design of compositional microprogram control units with transformation of the number of transactions*. Proceedings of MIXDES'04, Szczecin, Polska, 2004, pp. 172-175.
67. A. B a r k a l o v, R. W i ś n i e w s k i, R. B a b a k o v: *Optimizacija kompozicionnogo mikroprogramnogo ustrojstva upravlenija s elementarnymi operatornymi linejnymi celjami*. Naukovi Praci DNTU: Obcisljuval'na Technika ta Avtomatizacija, nr 77, 2004, pp. 210-216 (in russian).
68. A. B a r k a l o v, R. W i ś n i e w s k i: *Optimization of compositional microprogram control unit with elementary operational linear chains*. Upravljuscije Sistemy i Masiny, no. 5, 2004, pp. 25-29.
69. A. B a r k a l o v, R. W i ś n i e w s k i: *Synthesis of compositional microprogram control units with transformation of the numbers of inputs*. Proceedings of DESDes' 04, Zielona Góra, 2004, pp. 145-148.
70. A. B a r k a l o v, R. W i ś n i e w s k i: *Design of control units with transformation of the number of transaction*. Radiotechnika, Charkivskij nacionalnij universitet radioelektroniki, Charkiv, no 138, 2004, pp. 110-113.
71. A. B a r k a l o v, A. B u k o w i e c, R. W i ś n i e w s k i: *Sintez mikroprogrammnogo avtomata s predstavleniem termov funkcij vozbuzdenija kak par mikrokomand*. Radiotechnika, Charkivs'kij nacional'nij universitet radioelektroniki, Carkiv, 2005, no 142, pp. 92-96.
72. R. W i ś n i e w s k i: *Projektowanie układów mikroprogramowanych z wykorzystaniem wbudowanych bloków pamięci w matrycach programowalnych*. Proceedings of KNWS' 05, Złotniki Lubańskie, Polska, 2005, ss. 33-38.
73. X i l i n x: Using Block RAM in Spartan-3 Generation FPGAs. www.xilinx.com/bvdocs/appnotes/xapp463.pdf, 2005.
74. A l t e r a: Altera Devices Website. http://www.altera.com/products/devices/dev-index.jsp, 2008.
75. A. B a r k a l o v, M. W ę g r z y n, R. W i ś n i e w s k i: *Partial reconfiguration of compositional microprogram control units implemented on FPGAs*. Proceedings of PDeS 2006, Brno, Czech Rep., 2006, pp. 116-119.
76. D. M e s q u i t a, F. M o r a e s, J. P a l m a, L. M o l l e r, N. C a l a z a n s: *Remote and partial reconfiguration of FPGAs*: Tools and trends. Proc. Of IPDPS'03, 2003, pp. 177-185.
77. A. B a r k a l o v, L. T i t a r e n k o, R. W i ś n i e w s k i: *Optimization of the amount of LUT-elements in compositional microprogram control unit with mutual memory*. Proc. of EWDTW '05, Odessa, Ukraine, 2005, pp. 75-79.
78. R. W i ś n i e w s k i, A. B a r k a l o v: *Synthesis of compositional microprogram control units with function decoder*. Proc. of IWCIT 2007, Ostrava, Czech Rep., 2007, pp. 229-232.
79. A. B a r k a l o v, M. W ę g r z y n, R. W i ś n i e w s k i: *Optimization of LUT-elements amount in cotrol unit of system-on-chip*. Proceedings of DESDes '06, Rydzyna, Poland, 2006, pp. 143-146.
80. R. W i ś n i e w s k i, A. B a r k a l o v, L. T i t a r e n k o: *Synthesis of compositional microprogram control units with OLC output identification*. Proceedings of CAD DD 2007, vol 2, Minsk, Belarus, 2007, pp. 81-86.
81. A. B a r k a l o v, L. T i t a r e n k o, R. W i ś n i e w s k i: *Synthesis of compositional microprogram control units with function decoder for telecommunication systems*. Radiotehnika: Problemy telekommunikacij, no 151, Charkivskij nacionalnij universitet radioelektroniki, Charkiv, 2007, pp. 106-111.
82. A. B a r k a l o v, R. W i ś n i e w s k i: *Optimization of compositional microprogram control units with sharing of codes*. Avtomatizacija proektirovanija diskretnych sistem, vol 1, Minsk, Bialorus, 2004, pp. 16-22.

83. R. Wiśniewski, A. Barkalov, L. Titarenko: *Optimization of address circuit of compositional microprogram unit*. Proceedings of EWDTW '06, Sochi, Rosja, 2006, pp. 167-170.
84. A. Barkalov, L. Titarenko, R. Wiśniewski: *Optimization of the circuit of compositional microprogram control unit with mutual memory*. Proceedings of CADSM 2007, Lviv -Polyana, Ukraina, 2007, pp. 251-255.
85. A. Barkalov, K. Efimenko, R. Wiśniewski: *Optimizacia shemy adresacii kompozicionnogo ustrojstva upravlenia*. Naukovi Praci DNTU: Problemi Modeljuvannja ta Avtomatizacii Proektuvannja Dinamicnich Sistem, Doneck, Ukraine, no 5, 2006, pp. 156-161.
86. A. Barkalov, R. Wiśniewski: *Optimization of compositional microprogram control units implemented on system-on-chip*. Informatyka Teoretyczna i Stosowana, no 9, 2005, pp. 7-22.
87. R. Wiśniewski: *Design of compositional microprogram control units with elementary operational linear chains*. Proceedings of DESDes '06, Rydzyna, Polska, 2006, pp. 191-194.
88. A. Barkalov, R. Wiśniewski, S. Kovalyov, K. Efimenko: *Optimizacia eisla LUT-elementov v ustrojstve upravlenia sistemy na kristalle*. Sbornik trudov XIII medunarodnoj naueno-tehnieeskoj konferencii, vol 1, Sevastopol, Ukraine, 2006, pp. 75-80.
89. A. Barkalov, R. Wiśniewski: *Optimization of compositional microprogram control units with sharing of codes*. Avtomatizacija proektirovanija diskretnych sistem: materialy pjatoj mezdunarodnoj konferencii; vol. 1, Minsk, Bialorus, 2004, pp. 16-22.
90. R. Wiśniewski: *Design of compositional microprogram control units with sharing of the codes*. Proceedings of OWD 2004; vol. 19, Wisła, Poland, 2004, pp. 217-220.
91. R. Wiśniewski: *Synteza mikroprogramowanych układów sterujących ze współdzieleniem kodów z wykorzystaniem dekodera adresów*. Pomiary Automatyka Kontrola, no 6, 2006, ss. 38-40.
92. R. Wiśniewski, A. Barkalov, L. Titarenko: *Synthesis of compositional microprogram control units with sharing codes and address decoder*. Proceedings of MIXDES 2006, Gdynia, Polska, 2006, pp. 397-400.
93. IEEE Standard Verilog Hardware Description Language 1364-2001. New York, 2001.
94. D. Thomas, P. Moorby: *The Verilog hardware description language*. Kluwer Academic Publishers, Norwell, MA, USA, 5th edition, 2002.

lo
lo
th
p
o
su
is
T
m
us
us
re

K

The
in appea
lity to ir
ble-Chi
and a c
tunity fo
(IP Cor
only a c

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 243–267

Structural decomposition of finite state machines

ARKADIUSZ BUKOWIEC, ALEXANDER BARKALOV

Institute of Computer Engineering and Electronics, University of Zielona Góra, Podgórna 50, 65-246 Zielona Góra, POLAND,
E-mails: a.bukowiec@iie.uz.zgora.pl, a.barkalov@iie.uz.zgora.pl

Received 2009.01.09
Authorized 2009.03.25

New architectures of FPGA devices combine different type of logic elements like look-up tables, flip-flops and memory blocks. But standard synthesis methods utilize only look-up tables and flip-flops and it makes that device utilization is not optimal one. Methods of synthesis and implementation of Mealy finite state machines into FPGAs there are presented in this article. Synthesis methods are based on the architectural decomposition of logic circuit of FSM and multiple encoding of some its parameters. Architectures of such designed structures are based on existence of decoders as second-level circuits. There is also proposed hardware implementation into FPGAs of developed multi-level structures. The hardware implementation is based on an implementation with use of look-up tables and memory blocks together. The combinational circuit and the register are implemented with use of logic blocks, like in standard realizations. While, decoders are implemented with use of memory blocks. Such realization leads to balanced and rational usage of hardware resources of modern FPGA devices.

Keywords: control unit, decomposition, FSM, FPGA, synthesis

1. INTRODUCTION

The silicon product development grows very fast. This rapid evolution has resulted in appearance of very large scale integration (VLSI) chips and circuits. It makes possibility to implement a complex digital system in a single chip as a System-on-Programmable-Chip (SoPC) [21], [28]. Such digital system can be also represented as a data path and a control unit [7], [23]. The representation with this decomposition gives opportunity for reuse of early designed components or for use of intellectual property cores (IP Cores), that are available on the silicon market, for data processing. It means, that only a control unit has to be designed from the beginning.

Finite state machines (FSMs) [4], [23] are still one of the most popular ways of realization of an algorithm of a control unit. Because a control unit is a part of almost any digital system, optimization of a synthesis process of its digital circuit is a very important subject of many works.

Nowadays, FPGA devices are one of the most popular for realization of whole digital devices as SoPC. It creates new needs of fit a control unit into available hardware resources after implementation of a data patch. Because new FPGAs have different kind of logic elements, like look-up tables (LUTs), registers, and embedded memory blocks, it makes that not only reduction of hardware resources required for implementation of a finite state machine is a goal but also possibility to balanced usage of different types of resources.

The methods of synthesis proposed in this article are based on the structural decomposition of FSM logic circuit and multiple encoding of some parameters of FSM divided into subsets based on a current state or a currently executed microinstruction [17], [19]. This encoding allows to decrease a number of logic functions implemented by the combinational circuit of an FSM. Internal parameters are decoded in the second level circuit based on the multiple code and the code of a current state or the code of a currently executed microinstruction. Because this system is regular it can be implemented with embedded memory blocks. It leads to decreasing a number of LUTs required for implementation of a logic circuit of an FSM and balanced usage of different resources of an FPGA device.

2. MAIN DEFINITIONS

2.1. FINITE STATE MACHINE DEFINITION

A finite state machine is a mathematical model of behavior composed of a finite set of input symbols, a finite nonempty set of states, a finite set of output symbols, transitions and actions [1], [4]. This model can be represented as six tuple:

$$S = \langle X, Y, A, a_1, \delta, \omega \rangle \tag{1}$$

where:

- X is a finite set of input Boolean variables, $X = \{x_1, \dots, x_L\}$;
- Y is a finite set of output Boolean variables, called microoperations (μO), $Y = \{y_1, \dots, y_N\}$;
- A is a finite, nonempty set of states, $A = \{a_1, \dots, a_M\}$;
- a_1 is the initial state of the FSM, $a_1 \in A$;
- δ is a transition function, defined as a function of a state and affirmation or negation of some input variables:

$$\delta : A \times X \to A; \tag{2}$$

- ω is an output function, and in case of Mealy model it is defined as a function of a state and affirmation or negation of some input variables:

$$\omega : A \times X \to Y. \tag{3}$$

In case of Moore model it is defined only as a function of a state:

$$\omega : A \to Y. \tag{4}$$

2.2. DIRECT STRUCTURAL TABLE DEFINITION

Such defined a Mealy FSM can be set up by a direct structural table (DST) [4]. DST is a one-dimensional state transition table extended with some additional columns, and it has following columns: a_m, $K(a_m)$, a_s, $K(a_s)$, X_h, Y_h, Φ_h, h. Here a_m is a current state of an FSM, $a_m \in A; K(a_m)$ is a binary code of the state a_m with $R = \lceil \log_2 M \rceil$ bits, the internal Boolean variables $q_r \in Q = \{q_1,...,q_R\}$ are used to encode states a_m; a_s is the next state, $a_s \in A$; $K(a_s)$, is a code of the state a_s; X_h is a condition of transition $\langle a_m, a_s \rangle$, it consists of conjunction of affirmation or negation of some logic elements from the set $X; Y_h$ is the microinstruction (μI) which is formed during the transition $\langle a_m, a_s \rangle$, $Y_h \subseteq Y; \Phi_h$ is the set of memory excitation functions that are equal to 1 to switch an FSM from $K(a_m)$ to $K(a_s)$, $\Phi_h \subseteq \Phi = \{D_1,...,D_R\}$ as a rule D type flip-flops are used to form a memory – there are considered only D type flip-flops because typical FPGAs are build only from such flip-flops; h is a number of the DST line, $h = 1,...,H$.

2.3. SINGLE-LEVEL STRUCTURE

The DST table is used as the base to form the system of functions:

$$Y = Y(Q, X), \tag{5}$$

$$\Phi = \Phi(Q, X). \tag{6}$$

This system corresponds to functions (3) and (2) and it describes a single-level circuit of Mealy FSM (Fig. 1). This structure is called P Mealy FSM. Here the circuit P implements system of functions (5), (6) and it is implemented with the use of LUTs in FPGA technology. The number of required LUTs strongly depends on complexity of these systems and possibility of its functional decomposition [25] and there is no possibility to correctly estimate this value. The register RG represents the memory of FSM and it is build from R D type flip-flops. The RTL schematic of this single-level structure for FPGA technology is presented in Figure 2. One of the drawbacks of the structure P is a big number of LUTs required for implementation of combinational circuit P. It is caused also by big number of logic functions implemented by this circuit:

$$n_p(\mathrm{P}) = N + R. \tag{7}$$

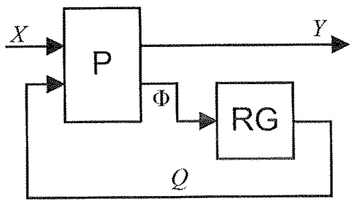


Fig. 1. Structural diagram of P Mealy FSM

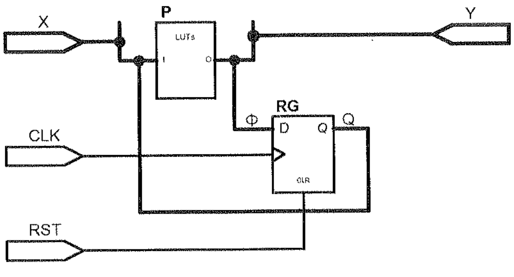


Fig. 2. RTL schematic of P Mealy FSM

2.4. BASE DOUBLE-LEVEL STRUCTURE

One of the known methods of decreasing the number of required LUTs is application of analytical methods of a functional decomposition [24], [27]. This decomposition operates on Boolean functions obtained during the synthesis process and it is preformed in its final phase of synthesis. These algorithms do not affect the total number of functions realized by a combinational circuit of an FSM.

In other ways, the decreasing of number of required LUTs can be achieved by reduction of the number of implemented logic functions. This reduction can be made by application of structural decomposition [4], [6] of logic circuit of an FSM. The structural decomposition follows on a system level and it is applied in early stage during the synthesis process. It refers to the process by which a complex circuit is broken down into parts that are easier to implement. In case of finite state machine it splits the combinational circuit into several circuits which together have the same function but each of them has different nature. The system after decomposition has a multi-level nature because data is processed serially and passed from one circuit to next one. It means that the both decomposition methods should not be treated as competitive ones and, what more, they can be applied together in the synthesis process.

Let the DST contain T different microinstructions $Y_t \subseteq Y$. Encode microinstructions with maximal encoding method by assigning to each set Y_t the binary code $K(Y_t)$ with $N_1 = \lceil \log_2 T \rceil$ bits ($t = 1,...,T$). Use variables $z_r \in Z = \{z_1,...,z_{N_1}\}$ for representation of these codes. In this case a Mealy FSM can be implemented as double-level circuit (Fig. 3) named as PY Mealy FSM [7]. The register RG is exactly the same as in single-level structure. The circuit Y implements the system of functions:

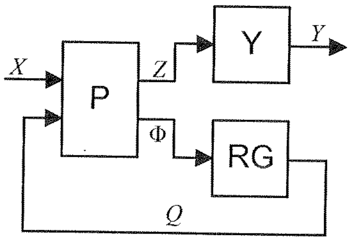


Fig. 3. Structural diagram of PY Mealy FSM

$$Y = Y(Z) \tag{8}$$

and transforms the code $K(Y_t)$, represented by variables z_r, into the microinstruction Y_t, built from microoperations y_n. Because system (8) has a regular structure this circuit can be effectively implemented using embedded memory blocks. Now the circuit P implements systems (6) and:

$$Z = Z(Q, X), \tag{9}$$

it is implemented with use of LUTs like for single-level structure. The RTL schematic of double-level PY structure for FPGA technology is presented in Figure 4.

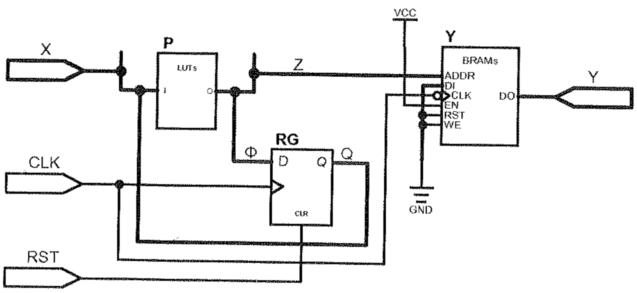


Fig. 4. RTL schematic of PY Mealy FSM

This structure permits to reduce the number of Boolean function to:

$$n_p(\mathrm{PY}) = N_1 + R. \tag{10}$$

But this number is still relatively big and it makes such a structure need still relatively big number of LUTs for implementation of the circuit P. It makes that application of this structure in a process of an FPGA implementation is not grateful. Additionaly the usage of embedded memory blocks of FPGAs is not effective because of application of maximal encoding method. However it does not disqualify structural decomposition. The application of different encoding method gives prospective results.

3. STRUCTRURAL DECOMPOSITION WITH MULTIPLE ENCODING

The double-level structure and method of its synthesis, presented in previous section, can be adopted into an FPGA technology. It required application of special methods of encoding [10], [11] and modification of a logic circuit structure. Proposed methods are based on a multiple encoding [15] of some parameters of a state machine. The structure of logic circuits depends on which parameter is multiple encoded and which parameter is used as a partial code.

A multiple encoding can be applied for some parameters of a state machine, like microinstructions or internal states [14]. The set of these parameters is partitioned into several subsets. Then parameters are encoded separately in each subset. The same codes are used for different subsets. The partition into subsets is made base on other parameter, like a current state or a currently executed microinstruction. The logic circuit of such designed state machine required special structure. Type of blocks and their connections depend on which parameter is multiple encoded and which parameter is used as a partitioning set. Generally, such circuit is realized in a double-level structure with a combinational circuit on a first level and a decoder on a second level.

3.1. MULTIPLE ENCODING OF MICROIMSTRUCTIONS

The method with a maximal encoding of microinstructions can be modified by applying the multiple encoding for a set of microinstructions [8], [13], [14]. Let partition a set of all microinstructions $\Upsilon = \{Y_1,...,Y_T\}$ into subsets based on a current state a_m. It leads to existence of M subsets $\Upsilon(a_m) \subseteq \Upsilon$ and a microinstruction $Y_t \in \Upsilon(a_m)$ if it is executed during any transition from the state a_m. Let

$$T_m = |\Upsilon(a_m)| \tag{11}$$

and

$$T_0 = \max(T_1,...,T_M). \tag{12}$$

Let encode each microinstruction $Y_t \in \Upsilon(a_m)$ by a binary code $K_m(Y_t)$ with bits.

$$N_2 = \lceil \log_2 T_0 \rceil \tag{13}$$

Because $\Upsilon(a_m) \subseteq \Upsilon(T_0 \leq T)$ then $N_2 \leq N_1$. But for typical control algorithm $\Upsilon(a_m) \subset \Upsilon$ and $T_0 < T$ and in this case also $N_2 < N_1$ and this condition has to be satisfied for benefits from application of this method [14]. Let use variables $\psi_n \in \Psi = \{\psi_1,\ldots,\psi_{N_2}\}$ for representation of codes $K_m(Y_t)$. In this case the code of a microinstruction $K(Y_t)$ is represented by concatenation of the multiple code of the microinstruction $K_m(Y_t)$ and the code of the current state $K(a_m)$:

$$K(Y_t) = K_m(Y_t) * K(a_m). \tag{14}$$

A digital circuit of an FSM with such encoding can be implemented as a double-level structure PY_0 (Fig. 5). The RTL schematic of this structure for FPGA technology is presented in Figure 6. This structure permits to decrease the number of outputs of the circuit P in comparison to the structure PY. Here the circuit P implements the system (6) and the system

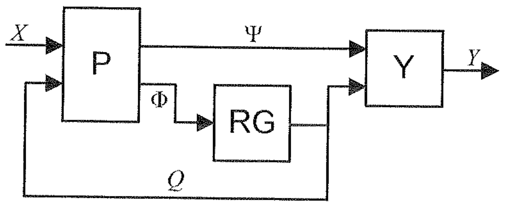


Fig. 5. Structural diagram of PY_0 Mealy FSM

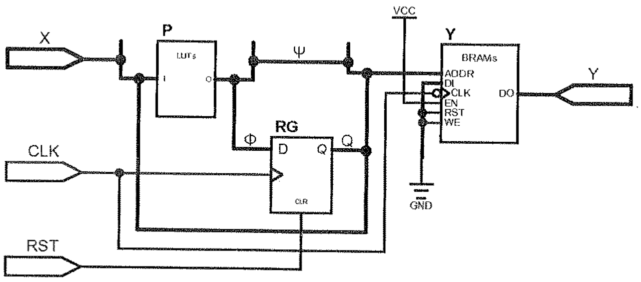


Fig. 6. RTL schematic of PY_0 Mealy FSM

$$\Psi = \Psi(X, Q). \tag{15}$$

It has to implement

$$n_p(PY_0) = R + N_2 \tag{16}$$

logic functions. The circuit Y implements a decoding system

$$Y = Y(\Psi, Q), \tag{17}$$

where the variables from the set Ψ are used to detect a adequate microinstruction for current state that is identified by variables from the set Q.

The entering point for synthesis process with structural decomposition is a formatted DST and it consists of the following steps:

- a multiple encoding of microinstructions,
- a formation of the transformed direct structural table,
- a formation of the system of Boolean functions,
- a formation of the decoder table,

- an implementation of the logic circuit of the FSM.

The multiple encoding of microinstructions is based on binary encoding of microinstructions Y_t in each subset $\Upsilon(a_m)$. It means that if one microinstruction Y_t belongs to several subsets $\Upsilon(a_m)$ it also receives several codes $K_m(Y_t)$.

The formation of the transformed direct structural table is base for formation of systems (6) and (15). It is created from the original DST by replacing the column Y_h by the column Ψ_h. The column Ψ_h contains variables $\psi_n \in \Psi$ ($n = 1,...,N_2$), that are equal to 1 in the code $K_m(Y_t)$ of the microinstruction Y_t from the h-th line of the original DST.

The formation of the system of Boolean functions is base for obtaining systems (6) and (15). The system (6) is defined as:

$$D_r = \bigvee_{h=1}^{H} (C_{rh} \wedge F_h), \tag{18}$$

where $r = 1,...,R$; C_{rh} is a Boolean variable equal to 1 iff the h-th line of a DST contains the function D_r in the column Φ_h;

$$F_h = A_m^h \wedge X_h, \tag{19}$$

where A_m^h is a conjunction of internal variables $Q_r \in Q$ corresponding to the code $K(a_m)$ of the state $a_m \in A$ from the h-th line of the DST

$$A_m^h = \bigwedge_{r=1}^{R} Q_r^{l_{mr}}, \tag{20}$$

where $l_{mr} \in \{0, 1\}$ is a value of the r-th bit of the code $K(a_m)$: $Q_r^0 = \overline{Q}_r$ and $Q_r^1 = Q_r$ [5]. Based on the similar way system (15) is defined as:

$$\psi_n = \bigvee_{h=1}^{H} (C_{nh} \wedge F_h), \tag{21}$$

where $n = 1,...,N_2$; C_{nh} is a Boolean variable equal to 1 iff the h-th line of the transformed DST contains the function ψ_n in the column Ψ_h.

The formation of the decoder table. This step forms the table that describes behavior of the circuit Y based on the system (16). This table has four columns:

- $K(a_m)$ is a binary code of the current state a_m;
- $K_m(Y_t)$ is a binary code of the microinstruction Y_t from the subset $\Upsilon(a_m)$;
- Y is a binary representation of the microinstruction Y_t, $y_n = 1$, iff $y_n \in Y_t$ and $y_n = 0$ iff $y_n \notin Y_t$, $n = 1,...,N$;
- t_0 is a number of the line, $t_0 = 1,...,\Sigma_{m=1}^{M} T_m$.

The implementation of the logic circuit of the FSM in FPGA. The combinational circuit P, represented by systems (18) and (21), is implemented by LUTs, and the register RG is implemented by D flip-flops. The decoder Y is implemented using an embedded memory blocks (Fig. 6) with $2^{(R+N_2)}$ words of N bits and the content of the memory is described by the decoder table where the concatenation of a binary code of a current state and a binary code of a microinstruction (13) is an address and the binary representation of a microinstruction is a value of word. There can be assigned any (*don't care*) values for addresses omitted in decoder tables because such concatenations of both codes are never used. Memory blocks in popular FPGAs are synchronous ones [2], [30]. The clock signal for memory blocks is the same as for the register but memory blocks are trigged by opposite edge (in this case falling edge). It cause that data is ready to read after one cycle and there is no need to wait one clock cycle until data is stable [17]. They additionally work also as an output register. Such registers are needed in each digital system with Mealy's outputs to stabilize its operation [7], [21]. Other input signals of memory blocks are connected to logic 1 or logic $\emptyset$, according to specification [30], to satisfy read-only mode.

3.2. MULTIPLE ENCODING OF INTERNAL STATES

The synthesis method with multiple encoding of internal states [9], [11], [19] applies multiple encoding for the set of internal states. In this case, current states [13], [15] or microinstructions [14] can be treated as a partitioning set.

3.2.1. *CURRENT STATES AS A PARTITIONING SET*

In first approach, let partition the set of internal states $a_s \in A = \{a_1,...,a_m\}$ into subsets based on a current state $a_m \in A$. It leads to existence of M subsets $A(a_m) \subseteq A$ and an internal state $a_s \in A(a_m)$ if it is the state of transition from the state a_m. Let

$$M_m^A = |A(a_m)| \tag{22}$$

and

$$M_0^A = \max(M_1^A, \ldots, M_M^A). \tag{23}$$

Let encode each internal state $a_s \in A(a_m)$ by a binary code $K_m(a_s)$ with bits.

$$R_1 = \lceil \log_2 M_0^A \rceil \tag{24}$$

In a theoretical case $A(a_m) \subseteq A$ $(M_0^A \leq M) \Rightarrow R_1 \leq R$. But in a typical state machine $A(a_m) \subset A$ and $M_0^A < M$ and of course $R_1 < R$. This condition has to be satisfied for benefits from application of this method. Let use variables $\tau_r \in \mathrm{T} = \{\tau_1,...,\tau_{R_1}\}$ for representation of $K_m(a_s)$ codes. In this case the code of internal state $K(a_s)$ is represented by concatenation of the multiple code of the internal state $K_m(a_s)$ and the code of the current state $K(a_m)$:

$$K(a_s) = K_m(a_s) * K(a_m). \tag{25}$$

A digital circuit of a FSM with such encoding can be implemented as the double-level structure PAY (Fig.7). The RTL schematic of this structure for FPGA technology is presented in Figure 8. Here the circuit P implements the system (9) and the system

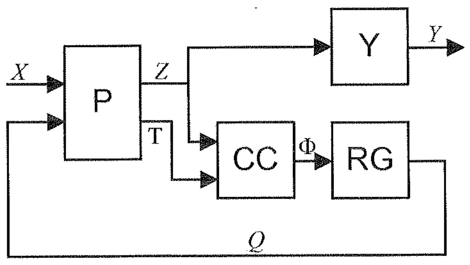


Fig. 7. Structural diagram of PAY Mealy FSM

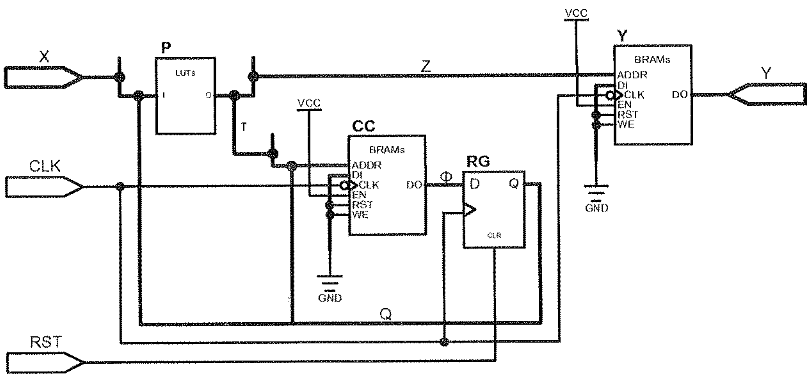


Fig. 8. RTL schematic of PAY Mealy FSM

$$\mathrm{T} = \mathrm{T}(X, Q), \tag{26}$$

and it implements

$$n_{\mathrm{p}}(\mathrm{PAY}) = R_1 + N_1. \tag{27}$$

Boolean functions. The circuit Y implements a decoding of microinstructions system (8). There is additional circuit CC that decodes internal states and generates a excitation functions:

$$\Phi = \Phi(\mathrm{T}, Q), \tag{28}$$

where the variables from the set T are used to detect a next state for current state that is identified be variables from the set Q.

The starting point for synthesis with structural decomposition is the formated DST and it consists of the following steps:

- an encoding of microinstructions,
- a multiple encoding of internal stares,
- a formation of the transformed direct structural table,
- a formation of the system of Boolean functions,
- a formation of the microoperation decoder table,
- a formation of the internal state code converter table,
- an implementation of the logic circuit of the FSM.

The encoding of microinstructions is based on a trivial way of a binary encoding. Let us encode each microinstruction $Y_t \in \Upsilon$ by a binary code $K(Y_t)$ with N_1 bits.

The multiple encoding of internal states is based on assigning a binary code $K_m(a_s)$ to internal states a_s in each subset $A(a_M)$.

The formation of the transformed direct structural table is base for formation of systems (9) and (26). It is created from the original DST by replacing the column Y_h by the column Z_h and columns $K(a_s)$ and Φ_h with columns $K_m(a_s)$ and T_h. The column $K_m(a_s)$ contains the multiple code of the internal state. The column T_h contains variables $\tau_r \in \mathrm{T}$, $r = 1,\dots,R_1$, that are equal to 1 in the code $K_m(a_s)$ from the same line of the DST.

The formation of the system of Boolean functions is base for obtaining systems (9) and (26). The system (9) is defined as:

$$z_n = \bigvee_{h=1}^{H} (C_{nh} \wedge F_h) \tag{29}$$

and the system (26) is defined as:

$$\tau_r = \bigvee_{h=1}^{H} (C_{rh} \wedge F_h), \tag{30}$$

where $n = 1,\dots,N_1$; $r = 1,\dots,R_1$; C_{nh} is a Boolean variable equal to 1 if the h-th line of the transformed DST contains the function z_n in the column Z_h; C_{rh} is a Boolean variable equal to 1 if the h-th line of the transformed DST contains the function τ_r in the column T_h.

The formation of the microoperation decoder table. This step forms the table that describes behavior of the Y circuit. This table has three columns:

- $K(Y_t)$ is a binary code of the microinstruction Y_t;
- Y is a binary representation of the microinstruction Y_t, $y_n = 1$ iff $y_n \in Y_t$ and $y_n = 0$ iff $y_n \notin Y_t$, $n = 1,\dots,N$;
- t is a number of the line, $t = 1,\dots,T$.

The formation of the internal state code converter table. This step forms the table that describe behavior of the circuit CC. This table has four columns:

- $K(a_m)$ is a binary code of the current state a_m;
- $K_m(a_s)$ is a binary code of the internal state a_s from the subset $A(a_m)$;
- Φ is a binary representation of excitation functions that switches the memory of the FSM from $K(a_m)$ to $K(a_s)$, in case of D type flip-flops $D_r = Q_r^*$, $r = 1,\dots,R$;

- m_0 is a number of the line, $m_0 = 1,..., \Sigma_{m=1}^{M} M_m^A$.

The implementation of the logic circuit of the FSM. The combinational circuit P is implemented with LUTs and the register RG with D type flip-flops (Fig. 8). The decoder Y is implemented using embedded memory blocks as for the structure PY. The internal state converter CC is also implemented with embedded memory blocks with $2^{(R+R_1)}$ words of R bits and the content of the memory is described by the internal state code converter table where the concatenation of the binary code of the current state and the binary code of the internal state (25) is an address and the binary representation of excitation functions is a value of the word. There can be assigned any (*don't care*) values for addresses omitted in the table because such concatenations of both codes are never used. Becouse memory block is trigged by opposite edge of clock the next state is latched in the register RG exactly after the same numer of cycles like for single-level structure [19].

3.2.2. *MICROINSTRUCTIONS AS A PARTITIONING SET*

In second approach, let partition the set of internal states $a_s \in A = \{a_1,...,a_m\}$ into subsets based on currently executed microinstruction $Y_t \subseteq Y$. It means that there is also required application of the maximal encoding of microinstructions because a usage of microinstructions codes only makes sense – set of microoperations create too long vector. It leads to existence of T subsets $A(Y_t) \subseteq A$ and the internal state $a_s \in A(Y_t)$ if it is the state of transition when the microinstruction Y_t is executed. Let

$$M_t^Y = |A(Y_t)| \tag{31}$$

and

$$M_0^Y = \max(M_1^Y, \ldots, M_T^Y). \tag{32}$$

Let encode each internal state $a_s \in A\ (Y_t)$ by a binary code $K_t(a_s)$ with bits.

$$R_2 = \lceil \log_2 M_0^Y \rceil \tag{33}$$

In theory $A(Y_t) \subseteq A$ and $(M_0^y \leq M) \Rightarrow R_2 \leq R$, but for implementation of typical algorithms $A(Y_t) \subset A$ and $M_0^y < M$ and it leads to $R_2 < R$. This condition has to be satisfied for the benefits of application of this method. Let use variables $\tau_r \in \mathrm{T} = \{\tau_1,...,\tau_{R_2}\}$ for representation of codes $K_t(A_s)$. In this case the code of the internal state $K\ (a_s)$ is represented by concatenation of the multiple code of the internal state $K_t\ (a_s)$ and the code of the currently executed microinstruction Y_t:

$$K(a_s) = K_t(a_s) * K(Y_t). \tag{34}$$

A digital circuit of a FSM with this encoding can be implemented as a double-level structure PYY (Fig. 9) [14]. The RTL schematic of this structure for FPGA technology is presented in Figure 10. Here the circuit P implements systems (9) and (26) and it realizes

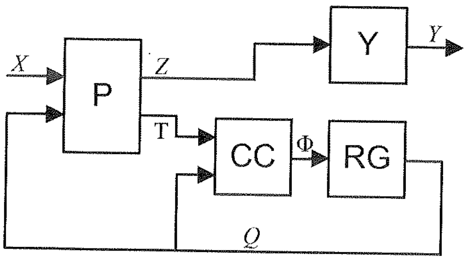


Fig. 9. Structural diagram of PYY Mealy FSM

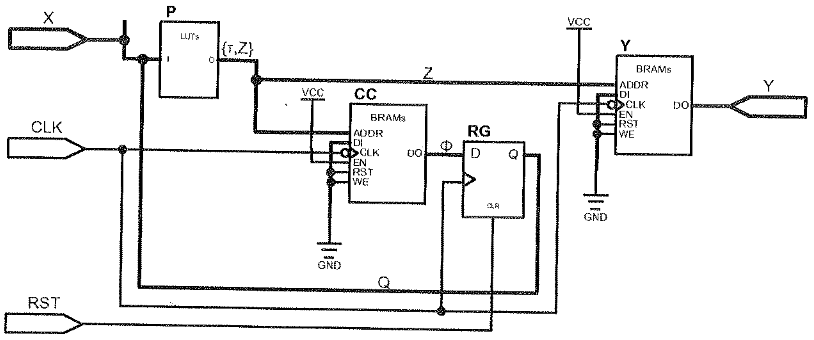


Fig. 10. RTL schematic of PYY Mealy FSM

$$n_p(\text{PYY}) = R_2 + N_1 \qquad (35)$$

logic functions. The circuit Y implements a decoding of microinstruction system (8). There is also the circuit CC that decodes internal states and generates an excitation function system:

$$\Phi = \Phi(\text{T}, Z), \qquad (36)$$

where the variables from the set T are used to detect a next state for currently execute microinstruction that is identified by its code represented by variables from the set Z.

The starting point for synthesis with structural decomposition is the formatted DST and it consists from following steps:

- an encoding of microinstructions,
- a multiple encoding of internal stares,
- a formation of the transformed direct structural table,
- a formation of the system of Boolean functions,
- a formation of the microoperation decoder table,

- a formation of the internal state code converter table,
- an implementation of the logic circuit of the FSM.

The encoding of microinstructions. This step is exactly the same as for the previous method of synthesis.

The multiple encoding of internal states is based on assigning a binary code $K_t\ (a_s)$ to internal states a_s in each subset $A(Y_t)$.

The formation of the transformed direct structural table is base for formation of systems (9) and (26). It is created from the original DST by replacing the column Y_h by the column Z_h and columns $K(a_s)$ and Φ_h with columns $K_t(a_s)$ and T_h. The column $K_t(a_s)$ contains the multiple code of the internal state for the microinstruction Y_t. The column T_h contains variables $\tau_r \in \mathrm{T}$, $r = 1,...,R_2$, that are equal to 1 in the code $K_t(a_s)$.

The formation of the system of Boolean functions is base for obtaining systems (9) and (26). These systems are defined as (29) and (30).

The formation of the microoperation decoder table. This step is exactly the same as for the previous synthesis method.

The formation of the internal state code converter table. This step forms the table that describes behavior of the circuit CC. This table has four columns:

- $K(Y_t)$ is a binary code of the microinstruction Y_t;
- $K_t(a_s)$ is a binary code of the internal state a_s from the subset $A(Y_t)$;
- Φ is a binary representation of excitation functions that switches the memory of a FSM from $K(a_m)$ to $K(a_s)$, in case of D type flip-flops $D_r = Q_r^*$, $r = 1,...,R$;
- t_0 is a number of the line, $t_0 = 1,..., \Sigma_{t=1}^{T} M_t^Y$.

The implementation of the logic circuit of the FSM. The idea of implementation is similar to implementation of a logic circuit where current states are used as the partitioning set. The only difference is a size and an addressing method of a memory block implementing the circuit CC (Fig. 10). There are $2^{(R+R_2)}$ words of R bits and the content of the memory is described by the internal state code converter table where the concatenation of the binary code of the microinstruction and the binary code of the internal state (34) is an address.

3.3. MULTIPLE ENCODING OF MICROINSTRUCTIONS AND INTERNAL STATES

Because internal states can be used as a partitioning set also for the multiple encoding of microinstructions and the multiple encoding of internal states this two encodings can be applied together in one method of synthesis [15]. It leads to existence of the structure PAY_0 (Fig. 11). The RTL schematic of this structure for FPGA technology is presented in Figure 12. The partitioning and the encoding of microinstructions are exactly the same as for the method with the multiple encoding of microinstructions and the partitioning and the encoding of internal states are also exactly the same as for the method with the multiple encoding of internal states with current states as partitioning set. It means that the code of the microinstruction $K(Y_t)$ is represented as (14) and the code of the internal state $K(a_s)$ is represented as (25).

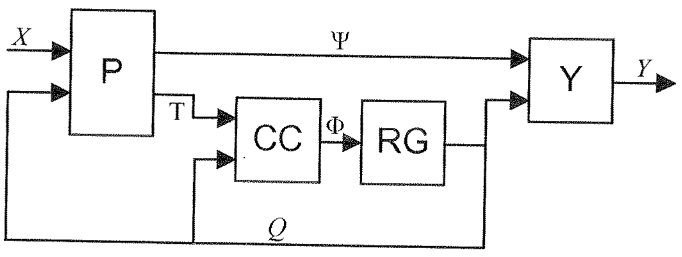


Fig. 11. Structural diagram of PAY$_0$ Mealy FSM

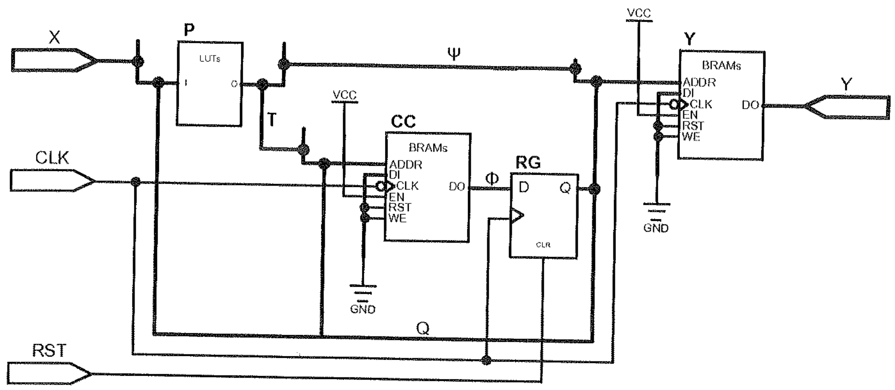


Fig. 12. RTL schematice of PAY$_0$ Mealty FSM

In this structure the combinational circuit P implements systems (15) and (26) and it implements

$$n_p(\text{PAY}_0) = R_1 + N_2 \tag{37}$$

Boolean functions in total. The circuit Y implements a decoding of microinstruction system (17) and the circuit CC implements the decoding system of intarnal state (28) and it generates the excitation functions.

The starting point for synthesis with structural decomposition is the formatted DST and it consists of the following steps:

- a multiple encoding of microinstructions,
- a multiple encoding of internal stares,
- a formation of the transformed direct structural table,
- a formation of the system of Boolean functions,
- a formation of the decoder table,
- a formation of the internal state code converter table,
- an implementation of the logic circuit of the FSM.

The multiple encoding of microinstructions. This step is exactly the same as for the structure PY$_0$.

The multiple encoding of internal states. This step is exactly the same as for the structure PAY.

The formation of the transformed direct structural table is base for formation of systems (15) and (26). It is created from the original DST by replacing the column Y_h by the column Ψ_h and columns $K(a_s)$ and Φ_h with columns $K_m(a_s)$ and T_h.

The formation of the system of Boolean functions is base for obtaining systems (15) and (26). These systems are defined as, respectively, (21) and (30).

The formation of the decoder table. This step is exactly the same as for the structure PY_0.

The formation of the internal state code converter table. This step is exactly the same as for the structure PAY.

The implementation of the logic circuit of the FSM. The idea of implementation is the same as for previous methods with the multiple encoding. Based on the same rules of clocking of memory blocks there can be drawn the RTL schematic for FPGA architecture (Fig. 12).

3.4. MULTIPLE SHARED ENCODING OF MICROINSTRUCTIONS AND INTERNAL STATES WITH COMMON MEMORY

Shared multiple encoding of microinstruction and internal states is a further improvement of the multiple encoding of theses parameters [16]. In this approach systems (15) and (26) are replaced by one system

$$\Psi = \Psi(X, Q), \tag{38}$$

which is used for encoding of microinstructions and internal states, represented by identifiers, and it is implemented by the combinational circuit P.
Becouse now, deocoder Y and CC have exactly the same input signals they can be replaced by one decoder YCC. It leads to existence of a new structure $\mathrm{PAY}_{\mathrm{SC}}$ (Fig. 13). The RTL schematic of this structure for FPGA technology is presented in Figure 14.

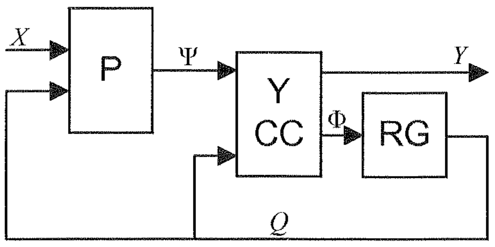


Fig. 13. Structural diagram of $\mathrm{PAY}_{\mathrm{SC}}$ Mealy FSM

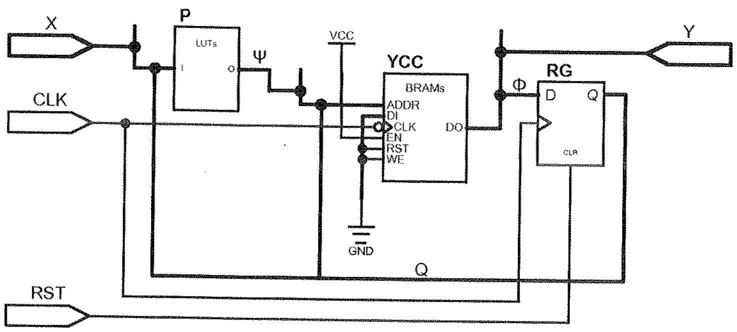


Fig. 14. RTL schematic of $\mathrm{PAY_{SC}}$ Mealy FSM

Create the identifier I_s^t that represents the pair $\langle a_s, Y_t \rangle$ where a_s is an internal state and Y_t is a microinstruction executed during the transition to this state. All identifiers create a set of identifiers I. The set of identifiers should be partitioned into subsets base on a current state $a_m \in A$ in order to make suitable encoding of identifiers. It leads to existence of M subsets $I(a_m) \subseteq I$ and identifier $I_s^t \in I(a_m)$ if there is transition from the state a_m to the state a_s and the microinstruction Y_t is executed during this transition. Now,

$$U_m = |I(a_m)| \tag{39}$$

and

$$U_0 = \max(U_1, ..., U_M). \tag{40}$$

And each identifier $I_s^t \in I(a_m)$ can be encoded by a binary code $K_m(I_s^t)$ with bits.

$$R_3 = \lceil \log_2 U_0 \rceil \tag{41}$$

Now, the combinational circuit P implements only

$$n_{\mathrm{p}}(\mathrm{PAY_{SC}}) = R_3 \tag{42}$$

logic functions. In this case, the decoder YCC is used for decoding of both microoperations and internal states and it implements systems:

$$Y = Y(Q, \Psi), \tag{43}$$

$$\Phi = \Phi(Q, \Psi). \tag{44}$$

Both codes, of microinstructions and of internal states, are represented by concatenation of the multiple code of the identifier $K(I_s^t)$ and the code of the current state $K(a_m)$:

$$K(Y_t) = K_m(I_s^t) * K(a_m), \tag{45}$$

$$K(a_s) = K_m(I_s^t) * K(a_m). \tag{46}$$

The starting point for the synthesis with structural decomposition into the structure PAY$_{SC}$ is the formatted DST and it consists of the following steps:

- a multiple encoding of identifiers,
- a formation of the transformed direct structural table,
- a formation of the system of Boolean functions,
- a formation of the common decoder table,
- an implementation of the logic circuit of the FSM.

The multiple encoding of identifiers. In application of this method of synthesis the simple binary encoding can be used.

The formation of the transformed direct structural table is base for formation of the system (38). It is created from the original DST by replacing columns a_s, $K(a_s)$, Y_h and Φ_h by columns I_s^t, $K_m(I_s^t)$ and Ψ_h. The column Ψ_h contains variables $\psi_r \in \Psi$ that are equal to 1 in the code $K_m(I_s^t)$from the h-th line of the transformed DST.

The formation of the system of Boolean functions is base for obtaining the system (38). This system is defined as (21).

The formation of the common decoder table. This table describes the behavior of the circuit YCC. It includes columns:

- $K(a_m)$ is a binary code of the current state a_m;
- $K_m(I_s^t)$ is a binary code of identifiers I_s^t from the subset $I(a_m)$;
- Y is a binary representation of the microinstruction Y_t, $y_n = 1$ iff $y_n \in Y_t$ and $y_n = 0$ iff $y_n \notin Y_t$, $n = 1,\dots,N$;
- Φ is a binary representation of excitation functions that switches the memory of the FSM from $K(a_m)$ to $K(a_s)$, in case of D type flip-flops $D_r = Q_r^*$, $r = 1,\dots,R$;
- m_0 is a number of the line.

The implementation of the logic circuit of the FSM. The idea of the implementation is the same as for previous methods with the multiple encoding. The decoding memory, built up of memroy blocks, is trigged by opposit edge of clock signal like in previous cases. It leads to existance of RTL schematic presented in Figure 14.

4. AUTOMATA SYNTHESIS SYSTEM

There was designed a prototype of system for structural synthesis of FSMs with use of proposed methods of synthesis. In order to apply these synthesis methods the design flow for FPGAs have to be modified (Fig.15). This system is named *the Automata Synthesis* (A♠S) [17], [18], [19]. In case of future implementation of discussed methods in commercial synthesis systems the design flow does not have to be modified and proposed method of structural synthesis can be included in the synthesis step.

In the proposed design flow the entry point for the structural synthesis step is the behavioral description of an FSM in the KISS2 format [32]. The output of the logic synthesis step is the structural description of FSM. It is represented by the set of files in

Verilog. Then these files can be the entry point for further synthesis and implementation into selected FPGA device with use of comercial tools. The set of these files consists of one top-level module (Fig.16) – it describes connections between blocks of the logic circuit – and group of files that describe particular blocks. The combinational circuit is described as a set of Boolean equations using continue assignments (Fig.17). The register is described as R-bit D type flip-flop with asynchronous reset (Fig.18) using typical synthesis template [22], [31]. Decoders (circuits Y, CC and YCC – depends on structure) are described using case statement (Fig.19). The address is placed as a selector of the case statement and the content of the memory is described by choices of the case statement [29]. Because it should by synthesized as synchronous ROM memory this statement is placed in always block. To ensure that such described module is synthesized as a memory block it is required to set a value of special synthesis attribute bram_map to "YES" [31]. This is synthesis attribute of Xilinx devices and it is ignored in case of synthesis into other vendors FPGA devices. But each vendor supplies similar attributes or directives, for example, the attribute romstyle specify the type of memory block to be used in Altera devices [3].

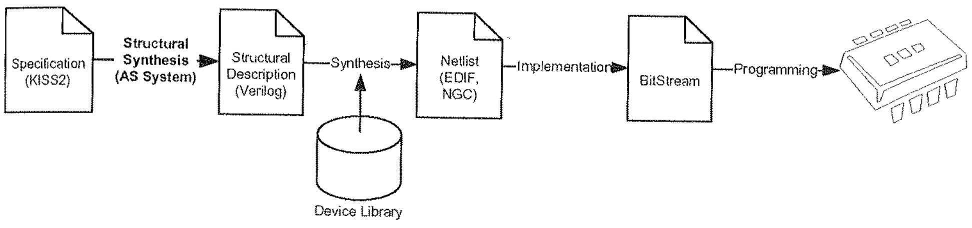


Fig. 15. The design flow for FPGAs with use of the AS System

```
module dk14 (clk, res, x, y);
   input clk, res;
   input [1:3] x;
   output [1:5] y;
   wire [1:3] d;
   wire [1:3] q;
   wire [1:2] psi;
   wire [1:3] tau;

   dk14_RG UD (.clk(clk), .res(res), .D(d), .Q(q));
   dk14_P  UP (.x(x), .Q(q), .psi(psi), .tau(tau));
   dk14_Y  UY (.clk(clk), .psi(psi), .Q(q), .y(y));
   dk14_CC UC (.clk(clk), .tau(tau), .Q(q), .D(d));
endmodule
```

Fig. 16. The top-level module of the Mealy FSM dk14 with the structure PAY_0

```
module dk14_P (x, Q, psi, tau);
   input [1:3] x;
   input [1:3] Q;
   output [1:2] psi;
   output [1:3] tau;

   assign psi[1] = x[1] & x[2] & ~x[3] & Q[1] & ~Q[2] & ~Q[3]
                 | x[1] & x[2] & ~x[3] & Q[1] & ~Q[2] & Q[3]
                 (...);
   assign psi[2] = x[1] & x[2] & x[3] & Q[1] & ~Q[2] & ~Q[3]
                 | x[1] & x[2] & x[3] & Q[1] & ~Q[2] & Q[3]
                 (...);
   assign tau[1] = x[1] & ~x[2] & x[3] & Q[1] & Q[2] & ~Q[3]
                 | ~x[1] & x[2] & ~x[3] & Q[1] & Q[2] & ~Q[3];
   assign tau[2] = x[1] & x[2] & x[3] & Q[1] & Q[2] & ~Q[3]
                 | x[1] & x[2] & x[3] & ~Q[1] & ~Q[2] & Q[3]
                 (...);
   assign tau[3] = x[1] & ~x[2] & ~x[3] & ~Q[1] & ~Q[2] & Q[3]
                 | x[1] & ~x[2] & ~x[3] & Q[1] & ~Q[2] & ~Q[3]
                 (...);

endmodule
```

Fig. 17. The part of the combinational circuit module of the Mealy FSM dk14 with the structure PAY_0

The example set of files is shown in figures 16, 17, 18 and 19. These files are generated for the Mealy FSM dk14 from the library *LGSynth91* [32] synthesized into the structure PAY_0 by the A♠S *System*.

```
module dk14_RG (clk, res, D, Q);
   input clk, res;
   input [1:3] D;
   output [1:3] Q;
   reg [1:3] Q;

   always @(posedge clk or posedge res)
   begin
       if (res)
          Q <= 3'b0;
       else
          Q <= D;
   end

endmodule
```

Fig. 18. The register module of Mealy FSM dk14 with the PAY_0 structure

```
module dk14_Y (clk, psi, Q, y);
   input clk;
   input [1:2] psi;
   input [1:3] Q;
   output [1:5] y;
   reg [1:5] y;

// synthesis attribute bram_map of dk14_Y is yes
always @(negedge clk)
   case ({Q,psi})
      5'b00000: y = 5'b00010;
      5'b00001: y = 5'b01010;
      5'b00010: y = 5'b01000;
      5'b00100: y = 5'b01001;
      (...);
      default: y = 5'b00000;
   endcase

endmodule
```

Fig. 19. The part of the microoperations decoder module of the Mealy FSM dk14 with the structure PAY_0

The *Automata Synthesis System* in version *1.6.2.* [18] is able to perform the logic synthesis into following structures:

- P – standard single-level structure,
- PY – standard double-level structure with maximal encoding of microinstructions,
- PY_0 – double-level structure with multiple encoding of microinstructions,
- PAY – double-level structure with multiple encoding of internal states (partitioning by current state) and maximal encoding of microinstructions,
- PAY_0 – double-level structure with multiple encoding of internal states and microinstructions,
- PA – double-level structure with multiple encoding of internal states (partitioning by current state),
- PYY – double-level structure with multiple encoding of internal states (partitioning by microinstructions) and maximal encoding of microinstructions,
- PAY_{SC} – double-level structure with shared multiple encoding of internal states and microinstructions with common memory.

The A♠S *System* works in command line of the *Windows XP* operating system. It is executed as follow:

synth[.exe] file.kiss2 -Method [–ImplementationSystem device]

where:

- synth[.exe] is the name of the executable file of the A♠S *System*.
- file.kiss2 is a name of a file to be synthesized.
- -Method is the name of a method of synthesis.
- -ImplementationSystem device is a optional argument that allows to generate synthesis macro for third party commercial synthesis system. At this stage only

XST from Xilnix is supported. Instead of device word there have to be placed a correctly symbol of device.

For example:

```
synth dk14.kiss2 -PAY0 -xst xcv50-bg256-6.
```

Output files are saved in newly created directory. The name of this directory is the name of synthesized file with the suffix _Method, where instead of the word Method is placed the name of the method of synthesis, for example, dk14_PAY0. Besides, structural description in Verilog of synthesized FSM there is also created a raport file with the .rep extension. This file indudes codes of encoded parameters, number of inputs, outputs, states and variables required for encoding, the number of logic functions realized by the combinational circuit and an estimated size of ROM memory.

Additionally, there can be generated files to run synthesis with *XST* if -xst device option is included. There is created the XST project file (extension .prj), the XST command file (extension. xst) and the batch file to invoke the synthesis process with *XST* (extension .bat).

5. SYNTHESIS

To analyze a gain of application of proposed synthesis methods there was performed the synthesis of benchmarks form the library *LGSynth91*. Benchmarks were synthetised with use of the A♠S *System* and then obtained files was synthetised with *Xilinx XST 8.1i* [31] from *Xilinx ISE 8.1i* into Xilinx Virtex v50 (xcv50-bg256-6) device [30] according to proposed design flow (Fig.15).

The obtained results of synthesis (Tab. 1) with use of standard methods (P and PY) and proposed methods have been compared and with results of synthesis of behavioral description in VHDL and Verilog. Because, there is no possibility to implement the behavioral description of an FSM in the KISS2 format it also was converted into behavioral description in VHDL [20] and Verilog [26].

Table 1

Average results of the synthesis of benchmarks

Type of resources	Structure								
	VHDL	Verilog	P	PY	PY_0	PAY	PYY	PAY_0	PAY_{SC}
Slices	39,67	44,51	51,89	50,38	45,55	34,66	42,04	25,13	23,6
LUTs	4,26	4,38	91,98	89,98	81,43	61,21	4,72	44,34	41,45
FFs	70,21	79,00	4,53	4,7	4,72	4,38	4,53	4,34	4,34
BRAMs	0	0	0	1,04	1,49	2,15	2,53	2,6	2,4
Slices	76%	86%	100%	97%	88%	67%	81%	48%	45%
LUTs	94%	97%	100%	98%	89%	67%	81%	48%	45%
FFs	76%	86%	100%	104%	104%	97%	100%	96%	96%
BRAMs	0%	0%	0%	43%	62%	89%	105%	108%	100%

It should be mentioned that XST performs synthesis of behavioral description of FSMs with use of standard single-level P structure. The synthesis of behavioral description was performed with compact encoding of states and default settings of other parameters. Differences in hardware utilization between behavioral description and the structure P can be caused by different state assignment. The *XST* also has implemented the algorithm of minimization of unreached states which improve results in some cases. The results also depend on the scheme of description of FSM in HDLs [12], [22]. The description in VHDL obtained from the *KISS2VHDL* converter [20] is recognized as FSM by *XST* and the minimization of unreached states the state re-assignment can be performed. The description in Verilog obtained from the *Kiss2vl* converter [26] has wrong interpretation of transitions from *any state* and *XST* remove whole state machine during synthesis process.

As it can be seen the standard method with the maximal encoding of microinstructions (PY) reduces the number of slices only by 3%. The other important parameter is the number of BRAMs. The conducted research showed that application of the standard method with the maximal encoding of microinstructions does not give benefits in case of implementation of control unit into an FPGA device – the number of LUTs is weakly reduced or even not reduced and additionally it assumes usage of memory blocks.

The multiple encoding of microinstructions (PY_0) in most cases diminishes the number of LUTs. This method is used as a base of further methods and it can be also used as an alternative balanced method of synthesis when outcomes of other methods exceeded number of available BRAMs because this method required relatively smallest number of memory blocks.

Average results obtained for the multiple encoding of internal states based on a current state (PAY) are better than the results obtained for the multiple encoding of microinstructions. The multiple encoding of internal states based on a microinstruction (PYY) gives better results than the multiple encoding of internal states based on a current state. These both structures required implementation of two decoders which means that it required the bigger number of BRAMs.

The multiple encoding of microinstructions and internal states (PAY_0) is a further improvement of the method PAY and it gives better results. The shared multiple encoding of microinstructions and internal states (PAY_{SC}) gives the best results of synthesis in most number of cases. Additionally the application of the common memory for both decoders reduce the number of required BRAMs for small FSMs.

6. SUMMARY

There were presented five methods of synthesis and five double-level structures of a digital device implementing an FSM. Each structure is dedicated to one adequate synthesis method. The synthesis methods are based on the multiple encoding of some parameters of a state machine and structural decomposition of its logic circuit. All methods are adapted for synthesis process into FPGA devices. They take advantage of features of new FPGAs like embedded memory blocks. The utilization of such resources leads to reducing the number of required standard logic blocks, like LUTs, for implementation of a control unit.

The choice from variety of synthesis methods gives opportunity to fit a control unit exactly into unused hardware resources by other components of the whole digital system. It makes that all blocks of device can be used equable, what means that synthesis process is more effective.

7. REFERENCES

1. M. Adamski, A. Barkalov (2006): *Architectural and Sequential Synthesis of Digital Devices*. Zielona Góra: University of Zielona Góra Press.
2. Altera (2007). *Embedded Memory in Altera FPGAs*, available at http://www.altera.com/technology/memory/embedded/mem-embedded.html.
3. Altera (2008). Synthesis. In: *Design and Synthesis vol. 1 of Quartus II Development Software Handbook (v8.0)*. User Guide. pp. 8-1-8-98. San Jose: Altera.
4. S.I. Baranov (1994): *Logic Synthesis for Control Automata*. Boston: Kluwer Academic Publishers.
5. A. Barkalov, A. Palagin (1997): *Synthesis of Microprogram Control Units*. Kiev: IC NAC of Ukraine (in Russian).
6. A. Barkalov (2002): *Synthesis of Control Units on PLDs*. Donetsk: DonNTU (in Russian).
7. A. Barkalov (2003): *Synthesis of Operational Units*. Donetsk: DonNTU (in Ukrainian).
8. A. Barkalov, A. Bukowiec (2004): *Synthesis of control unit with multiple encoding of the sets of microoperations. In: Proceedings of the 2nd International Workshop on Discrete-Event System Design DESDes'04*, pp. 75-78, Dychów, Poland: University of Zielona Góra Press.
9. A. Barkalov, A. Bukowiec (2004): *Synthesis of Mealy FSM with transformation of system of microoperations in excitation functions*. Radioelectronics and Computer Science, No. 3, 82-85.
10. A. Barkalov, A. Bukowiec (2005): *Optimization of Mealy FSM with decoding of the microoperations system*. Control Systems and Computers, No 5, 51-56.
11. A. Barkalov, A. Bukowiec (2007): *Realization of Mealy automata with transformation of microoperations in the registers excitation functions*. In: Proceedings of the 6th International Conference on Computer-Aided Design of Discrete Devices CAD DD'07 vol. 2, pp. 34-38, Minsk, Belarus.
12. S. Brown, Z. Vernesic (2005): *Fundamentals of Digital Logic with VHDL Design*. New York: McGraw-Hill, 2nd edition.
13. A. Bukowiec (2004): *Synthesis Mealy finite state machines with multiple encoding of internal states or sets of microoperations*. In: Proceedings of the VI International Workshop for Candidates for a Doctor's Degree OWD'04 vol. 19 of Conference Archives PTETiS, pp. 367-372, Wisła, Poland.
14. A. Bukowiec, (2004): *Synthesis of Mealy automata with multiple encoding of internal states*. In: Proceedings of Scientific Conference Computer Science – Art or Craft KNWS'04, pp. 29-34, Zamek Czocha, Poland: University of Zielona Góra Press.
15. A. Bukowiec, (2005): *Automata synthesis with application of multiple encoding*. In: Proceedings of 2nd Scientific Conference Computer Science - Art or Craft KNWS'05, pp. 17-22, Złotniki Lubańskie, Poland: University of Zielona Góra Press.
16. A. Bukowiec (2006): *Synthesis of Mealy FSM with multiple shared encoding of microinstructions and internal states*. In: Proceedings of IFAC Workshop on Programmable Devices and Embedded Systems PDeS'06, pp. 95-100, Brno, Czech Republic.
17. A. Bukowiec (2008): *Synthesis of Finite State Machines for Programmable Devices Based on Multi-Level Implementation*. Ph.D. Thesis, University of Zielona Góra, Faculty of Electrical Engineering, Computer Science and Telecommunications. Supervisor Prof. A. Barkalov, Ph.D. D.Sc.

18. A. B u k o w i e c, (2008): *Automata Synthesis System*, available at http://willow.iie.uz.zgora.pl/~abukowie/AS/as.htm
19. A. B u k o w i e c, A. B a r k a l o v, L. T i t a r e n k o, (2008): *FSMs implementation into FPGAs with multiple encoding of states.* In: Proceedings of IEEE East-West Design & Test Symposium EWDTS'08, pp. 72-75, Lviv, Ukraine: IEEE.
20. K. F i g l e r, (2006): *Analysis of Formal Methods of Synthesis of One-Level Finite State Machines.* Master's thesis, University of Zielona Góra, Faculty of Electrical Engineering, Computer Science and Telecommunications. Supervisor: Prof. A. Barkalov, Ph.D. D.Sc., co-supervisor: A. Bukowiec, M.Sc. (in Polish).
21. A. J a n t s c h (2003): *Modeling Embedded Systems and SoC's: Concurrency and Time in Models of Computation.* San Francisco: Morgan Kaufmann.
22. J. M. Lee, (1999): *Verilog QuickStart: A Practical Guide to Simulation and Synthesis in Verilog.* Norwell, MA: Kluwer Academic Publishers.
23. T. Ł u b a, (2001): *Synthesis of Logic Circuits.* Warszawa: Warsaw Information Technology (in Polsih).
24. T. Ł u b a, M. R a w s k i, Z. J a c h n a, (2002): *Functional decomposition as a universal method of logic synthesis for digital circuits.* In: Proceedings of the 9th International Conference Mixed Design of Integrated Circuits and Systems MixDes'02, pp. 285-290, Wrocław, Poland.
25. T. Ł u b a, M. R a w s k i, P. T o m a s z e w i c z, B. Z b i e r z c h o w s k i, (2003): *Synthesis of Digital Circuits.* Warszawa: Transport and Communication Publishers (in Polsih).
26. C. P r u t e a n u, (2004): *Kiss to Verilog FSM Converter*, available at http://codrin.freeshell.org.
27. M. R a w s k i, P. M o r a w i e c k i, H. S e l v a r a j, (2006): *Decomposition of combinational circuits described by large truth tables.* In: Proceedings of the 8th International Conference on Systems Engineering ICSE'06, pp. 401-406, Coventry, United Kingdom.
28. Z. S a l c i c, (1998): *VHDL and FPLDs in Digital Systems Design, Prototyping and Customization.* Boston: Kluwer Academic Publishers.
29. D. T h o m a s, P. M o o r b y, (2002): *The Verilog Hardware Description Language.* Norwell, MA: Kluwer Academic Publishers, 5th edition.
30. Xilinx (2002). *Virtex 2.5V Field Programmable Gate Arrays.* Data Scheet. San Jose: Xilinx.
31. Xilinx (2005). *XST User Guide (8.1i).* User Guide. San Jose: Xilinx.
32. Y a n g, S. (1991). *Logic Synthesis and Optimization Benchmarks User Guide. Version 3.0.* Technical Report, No. 1991-IWLS-UG-Saeyang. Microelectronics Center of North Carolina. North Carolina.

Fast O

e-n

cyclic
to pr
block
Propo
soluti
comp
block
is pla
chang

Keyw

Time r
that deterr
one measu
parameters
ment of a
short time
such a un
make poss
effective p

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 269–286

Fast Operating PLC Based on Event-Driven Control Program Tasks Execution

MIROSŁAW CHMIEL*, EDWARD HRYNIEWICZ**, ADAM MILIK***

Institute of Electronics, Silesian University of Technology, Gliwice, Poland
*e-mail: mchmiel@polsl.pl), ** e-mail: ehrynkiewicz@polsl.pl) , *** e-mail: amilik@polsl.pl*

Received 2009.01.08
Authorized 2009.03.26

The paper presents modified idea of program execution in PLCs. Instead of serial cyclic execution of control program event-driven execution is proposed. Suggested approach to program execution allows for selective execution of program parts or tasks. Only these blocks from entire program are executed whose variables have changed since last calculation. Proposed method can be implemented as software modification or as hardware accelerated solution. The most important part of the idea is task or subprogram triggering condition computation. Methods of program optimization are discussed. In order to determine program blocks that require recalculation in current program scan execution specific hardware support is planned to be researched. Memory content change detection unit allows to determine changes in memory content since last program block execution.

Keywords: Programmable Logic Controller; Central Processing Unit; Control Program; Process Image Memory; Scan Time; Throughput Time

1. INTRODUCTION

Time needed to execute one thousand of instructions is one of the basic parameters that determine performance of Programmable Logic Controllers (PLCs). The second one measure of a PLC performance is throughput time [10]. If the values of these parameters are low a possible range of PLC application is wider. Design and development of a CPU that would enable execution of a control program during extremely short time is becoming a very important task. Owing to its constructional features, such a unit should not only cover all the possible functional requirements but also make possible to take maximum benefits from these features through possibly most effective programming techniques [1, 4, 5, 8].

A PLC during standard processing performs following operations [10]:

1. Reading of input signals,
2. Control program execution,
3. Updating of output signals.

Apart from the presented operations that are visible for user the CPU is responsible for performing tasks required by its operating system (Fig.1).

In the presented approach entire control program is executed during each scan and it is independent of inputs and outputs signal changes. Large number of calculations performed by the CPU does not work out new output signals values because input arguments remain unchanged since last calculation.

Typical PLC executes control program in serial-cyclic manner. Instructions are executed one after another. After execution point reaches the end of instruction sequence cycle starts over. Each execution cycle (Fig.1) begins with Cycle Initialization and ends with Communication and Diagnostics.

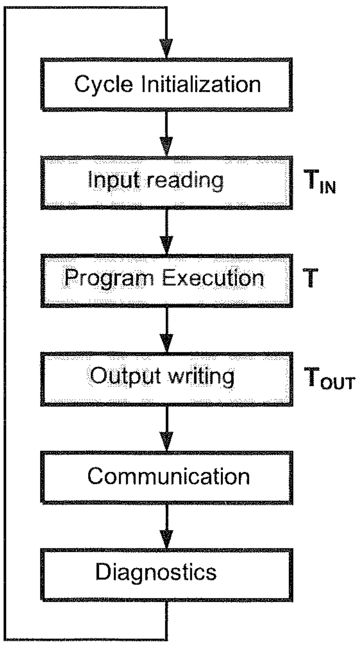


Fig. 1. A PLC standard operation cycle

Even though variables haven't changed between consecutive program executions PLC behaves in the same way. In case of variables whose state has not changed since the previous computation cycle CPU executes tasks whose results are already known - they have been derived during previous program scan. This is very important observation that allows to introduce a new approach in construction and implementation of program execution in PLC. This new idea is based on event driven program execution. The program blocks executed conditionally are triggered by the changes of input variables since last time. This approach allows to reduce calculation overhead by execution of only required parts of program instead of entire control algorithm processing.

Above observation shows that control algorithm can be executed partially. Only blocks whose input variables have changed since last scan should be selected for execution. This approach allows to eliminate excessive program execution overhead arising from code execution whose results are already known. This modification allows to increase controller performance and reduce its response time.

Such way of control program execution one can obtain in standard PLC but it should be put in nonstandard operation. The solution may be software supported or hardware supported.

2. SOFTWARE SUPPORTED SOLUTION

Even though commercially available PLC CPUs are not adapted to work in event driven fashion approach, it is possible to write a program that takes benefits from event observation.

Modicon TSX controllers' family can be programmed using two different program structures. Control programs can be developed in form of:

1. Single threaded main program that consists of segments and subprograms,
2. Multi threaded structure that consists of a main task, quick tasks and alarm interrupt tasks.

In multithreaded approach each task has assigned priority. The lowest priority is assigned to main task that can be executed in cyclic way (typical PLC execution) or periodically. Quick tasks have higher priority assigned. A quick task should be a short piece of program due to its periodical execution. Mainly it is executed for monitoring purposes like monitoring of rapid changes of controller digital inputs. Its execution is triggered by event like a counter overflow or change on a digital input. In opposite to described tasks (main and quick) an interrupt task is executed neither periodically nor continously.

In other types of controllers it is also possible to write a program in a such way that only selected blocks are executed when input variables have changed since last calculation. Dividing a program into blocks, macros and subprograms an is given opportunity for partial execution of entire program. All those mechanisms must be implemented in software. In such approach it is required to implement additional fragment of program that is responsible for execution of condition checking for program blocks.

A Sequential Function Chart (SFC) offers a programmatic mechanism to describe tasks whose execution is conditional. This method enables to create subfunctions that execution is suspended until given conditions are met. The subfunction can be executed once or in cyclic way in parallel with other running tasks. A program designer is responsible for defining transition and logical statements that activate appropriate step. Steps and transitions contain standard program blocks.

3. WHAT IS AN EVENT?

To be able to talk about event-driven program execution two questions should be discussed:
What is “an event” from the control program’s point of view?
How to divide control program to achieve optimal solution?

Discussion presented in the paper is focused on a proposition of an idea of a programmable controller that is able to execute control program in an event-driven mode, releasing the user from the effort of preparing the conditions for the blocks triggering. These tasks should be completed by the program compiler according to appropriately defined task partitioning algorithms. During execution of a program PLCs make use of external signals: inputs, outputs and internal variables: markers, counters, timers which can also be treated as markers in this case. According to analysis presented in [6] change of state of a variable or signal requires evaluation of a new output value.

In general controller function can be put down as following formula:

$$Y_n = \lambda(X, Q, Y_{n-1})$$

Where:
Y_n – Calculation result,
X – Input variables,
Q – Marker variables,
Y_{n-1} – Previous value of output.

Dividing the program into blocks is the main part of an event-driven program execution concept. Through an analysis a set of function arguments can be determined, which can be referred as the sensitivity list of the function λ for each block. For purpose of conditional execution recording of variable changes is required. Processed triggering conditions are stored in conditional execution marker. Before starting execution of a block its conditional execution marker (flag) is checked. If in the set of the controlling function arguments no changes have appeared since last calculation the flag remains inactive and the block does not undergo recalculation.

The most reasonable way of the program partitioning is outputs based method - as many blocks as outputs are controlled. For each output a list of variables (sensitivity list) is created whose state, or rather change of state, will affect the output. Change of one of these variables will impose execution of the corresponding block. Size of each block will directly depend on the complexity of the function evaluating the corresponding output state, whereas size of the triggering function will depend on the sensitivity list length. The first aspect is typical for a standard controlling program while the second results from the program division into the event-triggered blocks. As presented in Fig.2, organization block of the program will contain the cyclically executed part of the program – the fragments that are not suitable for event-driven execution and the part of the program that will be responsible for detection of changes in the sensitivity list of the block. The part of the program, that is intended for event-driven execution consists of operational part from original program and conditional entry.

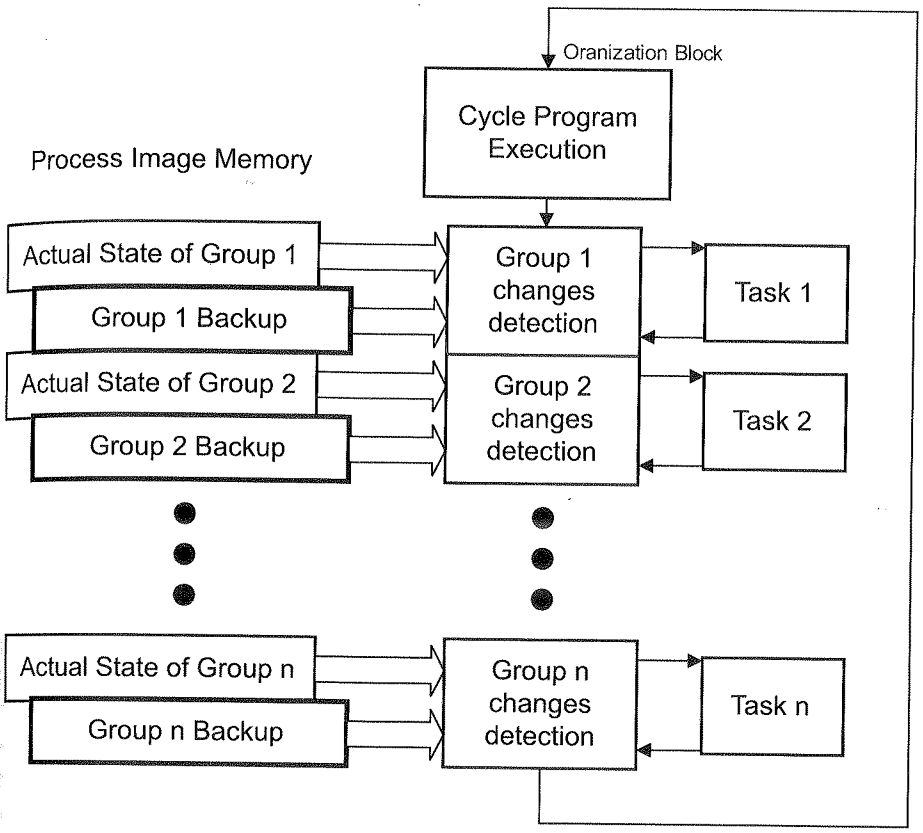


Fig. 2. Program block execution

What should be done when a state of one output affects another one state, or when several output states depend on one internal variable state? After closer look at the variables deciding about execution of a given program block, it can be found that only two kinds of variables can have direct influence on the given function's state: input signals coming directly from the input modules (in most cases their change initiate change of other variables state) and state of the timers - they are the only internal variables which changes independently from the input signals. It results from the fact that the time is counted independently of cycles of the controller loop. All variables processed by a PLC are shown in the Fig.3. With dashed line are marked sets of variables that are responsible for calculation triggering. Above observation reduce the size of the additional memory necessary for storing information about variable states.

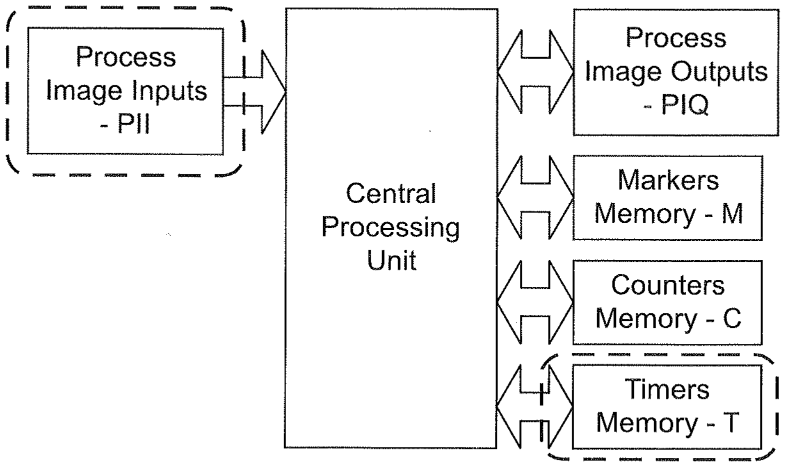


Fig. 3. A PLC standard variables ranges

4. HOW TO BUILD AN ORGANIZATION BLOCK?

When the decision has been already made that it is sufficient to monitor changes of inputs and timers, capabilities of the controllers should be investigated for the purpose of writing procedures checking state of mentioned variables. It is necessary to check the state of all variables, that given block depends on. It should be noticed that changes of any number of variables should schedule a single execution of a block. There's no need for a multiple execution of a block while input variables are updated at the beginning of the program scan.

Let's consider two examples of programmatic implementation of event driven processing. First example is schematically presented in Fig.4. Before execution of program block current value of all variables that are used is compared with their previous value. If current state of at list one variable differs from previous one the block must be executed. There is a great inconvenience in implementation of change checker block. This block consists of large amount of instructions that are responsible for checking processing condition of a program block only. Exemplary program listing and execution times are presented in Fig.5 [2].

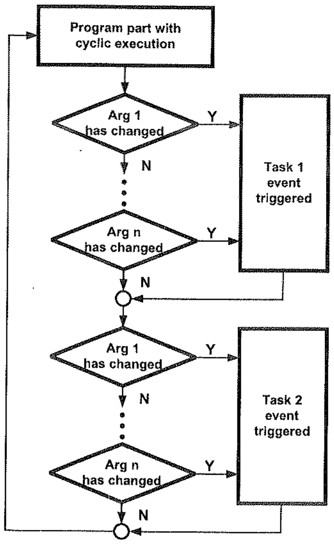


Fig. 4. Program block diagram with event-driven tasks execution

```
        a.  „S7-314 PLC"
X       I0.0     //0.3µs
X       M10.0    //0.6µs
O                //0.3µs
           ...
X       I0.7
X       M10.7
CC      FC1      //5.3µs
A       I0.0     //0.2µs
=       M10.0    //1.4µs
A       I0.7
=       M10.7
Execution Time://27.4µs
```

```
        b.  „S7-214 PLC"
LD        I0.0   //0.22µs
EU               //8.00µs
LD        I0.0
ED               //8.00µs
            ...
OLD              //0.22µs
LD        I0.7
EU
OLD
LD        I0.7
ED
OLD
CALL      SBR_0  //1.30µs
Execution Time://113.5µs
```

Fig. 5. Listing of change detection for one block

In the presented program example it was necessary to detect the changes from 0 to 1 and from 1 to 0 for each variable. Detection a variable change in Simatic S7-314 requires at least 5 instructions (twice XOR, OR, AND, and „=") and additional marker should be used for storing previous value of a signal. Time needed to execute this program part for 8 inputs is 27.4μs. The Simatic S7-214 (b) PLC requires 113.5μs, even though less instructions are executed. The program execution in the S7 314 CPU is almost four times faster than in S7 214 CPU.

Complexity of program implementation is growing very quickly for blocks that process several signals. Partially this problem can be overcome by variable grouping in adjacent memory cells to form longer words that allow processing several bits at once. This will allow to use byte or word instructions that can greatly accelerate detection process. Block diagram of optimized solution is presented in Fig.6.

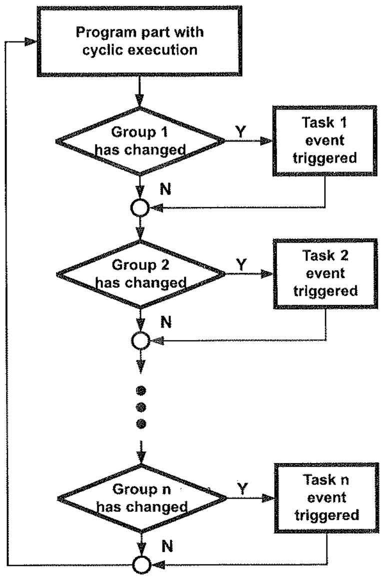


Fig. 6. Variables grouping in event detection procedure

Part of exemplary program in STL representation and calculated execution times for Siemens S7-214 and S7-314 CPU of PLC is presented in Fig.7. The example presents solution for eight variables, but it can be easy extended up to 32 bits (variables) – double world – grouped in area of input, output variables or internal markers. In case of checking larger number of variables or variables that belong to different areas presented fragment of code must be repeated for all triggering variables. Final result is logical sum of all partial results that allow to determine if change has occurred. As it can be seen in the example, in case of both S7-214 and S7-314 controllers, program execution time decreases significantly. For the S7-314 unit 3 times faster execution is observed. Increasing efficiency is expected for a bigger number of grouped variables.

Implementing conditional execution of block with modified organization of data cause program growing. For blocks with moderate number of parameters that are gathered in coherent memory area such solution gives quite good results. Performance is reduced when number of parameters is growing or parameters belong to different

```
                a. „S7-314 PLC"
L       Grl_Memory                  //0.7µs
L       Grl_Signals                 //0.6µs
T       Grl_Memory                  //0.3µs
XOW                                 //0.5µs
L       0                           //0.6µs
<>I                                 //1.6µs
CC  FC1                             //5.3µs

                b. „S7-214 PLC"
LD       SM0.0                      //0.80µs
LPS                                 //0.24µs
MOVB     Grl_S, Grl_M2              //15.0µs
AENO                                //0.40µs
XORB     Grl_M1, Grl_M2             //19.0µs
AENO                                //0.40µs
AB<>     Grl_M2, 0                  //18.0µs
CALL     SBR_0                      //9.00µs
LPP                                 //0.22µs
MOVB     Grl_S, Grl_M1              //15.0µs
Totally Execution Time:            //78.06µs
```

Fig. 7. Listing of change detection for grouped variables for one block

areas (input, output or markers). Other inconvenience of this approach is additional marker allocation for holding previous value of variables.

5. AN EXAMPLE OF EVENT DRIVEN PROGRAM EXECUTION

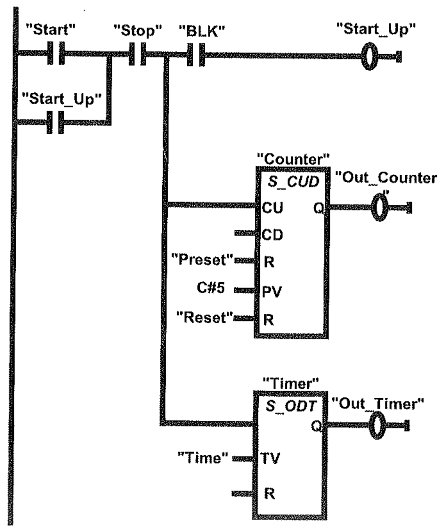


Fig. 8. An Example of LAD Program

The above considerations are illustrated by an exemplary program presented in Fig.8 in a well-known form of LAD diagram [3]. The example is very simple and it is intended to clearly demonstrate the subject of this paper. There are two program blocks (Counter and Timer) that are triggered with the same condition. The counter and the timer have also inputs and outputs that do not directly depend on this condition – Preset, Reset, Out_Counter, Out_Timer. It does not pay off to test these signals and process them with event-driven method, which can be clearly seen in the program listing (Fig.9).

As it can be noticed the instructions 1-9 and 17-19 are directly linked. The element associating these instructions is, among other things, additional variable L 0.0 included during the process of program compilation. The rest of the instructions form two- or three-instruction groups not directly connected with each other. This part of the program will be coded in unchanged form and will be executed by the controller in serial cyclic mode. The program written in the standard way is executed by the S7-314 CPU in time of 27.10μs. This is the time of presented program execution in each loop cycle. In order to compare execution time with event driven execution method second version of program is written.

```
1.    A     „Start"          //0.2μs
2.    O     „Start_Up"       //0.2μs
3.    A     „Stop"           //0.2μs
4.    =     L 0.0            //0.8μs

5.    A     L 0.0            //0.8μs
6.    A     „BLK"            //0.2μs
7.    =     „Start_Up"       //0.2μs

8.    A     L 0.0            //0.8μs
9.    CU    „Counter"        //2.6μs

10.   A     „Preset"         //0.2μs
11.   L     C#5              //0.6μs
12.   S     „Counter"        //6.0μs
13.   A     „Reset"          //0.2μs
14.   R     „Counter"        //1.7μs
15.   A     „Counter"        //0.6μs
16.   =     „Out_Counter"    //0.2μs

17.   A     L 0.0            //0.8μs
18.   L     S5T#5s           //0.6μs
19.   SD    „Timer"          //9.0μs

20.   A     „Timer"          //0.8μs
21.   =     „Out_Timer"      //0.2μs


Event Execution Time:        //16.60μs

Cycle Execution Time:        //10.50μs
Totally Execution Time:      //27.10μs
```

Fig. 9. An Example of STL Program

```
A.     X         „Start“         //0.3µs
B.     X         „M_Start“       //0.6µs
C.     O                         //0.3µs
D.     X         „Stop“          //0.3µs
E.     X         „M_Stop“        //0.6µs
F.     O                         //0.3µs
G.     X         „BLK“           //0.3µs
H.     X         „M_BLK“         //0.6µs
I.     JCN       J               //2.3µs

         1.    A        „Start“
        2.    O       „Start_Up“
         3.    A        „Stop“
          4.    =        L0.0

          5.    A        L0.0
         6.    A        „BLK“
        7.    =        „Start_Up“

          8.    A        L0.0
        9.    CU       „Counter“

          10.   A        L0.0
         11.   L        S5T#5s
         12.   SD       „Timer“
J.     A         „Start“         //0.2µs
K.     =         „M_Start“       //1.4µs

Cycle Execution Time 1:      //10.50µs
Cycle Execution Time 2:      //07.20µs
Event Execution Time:        //16.60µs
MIN Execution Time:          //17.70µs
MAX Execution Time:          //34.30µs
```

Fig. 10. An Example of STL Program

The program after application of event-driven control is presented in Fig.10. The listing is limited to the event-triggered part of the program. The rest – executed cyclically - has been omitted. As can be seen the maximum loop cycle time has increased, what could be obviously predicted, as the part responsible for the detection of the signals Start, Stop, and BLK has been added. A decision had to be made which version of the changes detection should be used, as the number of the tested inputs is only three.

Maybe for three tested variables the solution from Fig.5a will prove to be better. Application of this solution made the cyclically executed program longer by 7.20µs. As it is seen in Fig.7, application of the grouped variables change checking would make the program 6.60µs longer – after replacing the block calling instruction CC FC1 with the jump to label instruction JNC J. The difference is not significant. It should be noticed, that even in case of three tested signals the application of change checking in grouped variables is executed faster. However for the sake of clarity of the programs and convenience of their comparison this method has not been used in this example. The minimum program scan time (i.e. no signal changes observed) is 17.70µs, what is about 9µs less then in case of the standard solution. It should be pointed out that it means about 1/3 time saving for those loop cycles for which signals from the sensitivity list remain unchanged.

The average program execution time in function of input signal frequency should be determined. Let us take into account "N" control program scans which contain "n" scans without changes of observed inputs. The average time of control program scan can be calculated as:

$$\frac{17.70n + 34.40(N - n)}{N} < 27.10$$

Therefore:

$$\frac{n}{N} > 0.434$$

It means that for the considered example it pays to use event-driven control program execution if rate of input changes is not grater then one per two control program scans.

That is highly improbable for control systems dealing with real automation systems like motor drives. Sometimes changes of the Start or Stop signals happen in few hours intervals, and blocking signals state can change every few days or even much more rarely. In typical operation condition the average time of program scan for the event-driven program will be shorter then for the program written using the standard techniques. It should be stressed here that reducing this time results in reduction of access time to the input/output signals, while the CPU executes less instructions in each program loop.

6. HARDWARE SUPPORTED SOLUTION

As it was presented software solution in area of event driven program execution give very permissible results but requires execution of additional fragment of program that detects execution condition for program blocks. Based on experience and observation from software implementation all those inconvenience can be overcome by appropriate hardware construction [9, 11]. Merging features of a CPU with custom hardware implemented in an FPGA gives opportunity to construct systems with set of features that are not commercially available. Main limitation in detecting of execution conditions is connected with memory access. In order to determine difference in signal group additional memory access is required. Current value of signal or variable must be compared with previous one. In this case comparison operation is limited to difference detection that can base on XOR operation between current and previous value of memory. This operation involves additional instructions as well it consumes some CPU time to execute this conditional entry to program block.

Is it possible to construct a memory controller that will support memory content change detector? Further part of this chapter presents developed solution that allows to detect memory content changes during operation and accelerates controller operation. Memory is used to store operation arguments of the PLC CPU. In order to accelerate calculation input variables are transferred from input modules to process image memory before calculation starts. Calculation results are accumulated in separated space of process image memory. When execution of control program is completed controller transfers content of output part of process image memory to output modules. To be able to perform complex calculation additional space for internal variables is required. This additional area is called marker memory.

In event driven system calculation are performed only if changes are detected in function arguments e.g. (X, Q, Yn-1). This requires to monitor three separate areas input, output and markers space of process memory. Independently from destination area only operation that can alter memory content is write operation. In order to implement difference detection circuit of memory content during write operation current and new cell content must be compared. While this memory must be connected to CPU, additional read or write cycles should not be performed. Problem can be solved by use of memory with separated data inputs and outputs. Schematic diagram of memory system with data change detection is shown in Fig. 11. From a memory cell with separated data input and output is formed a memory block that is connected to bidirectional data bus (D). The memory block is typical component that can be implemented with use of distributed or block memories available in an FPGA device. Implementation take benefits from synchronous write operation. It can be compared with writing data to the set of addressable D-type edge triggered registers. Independent and concurrent read and write operations allow accessing new and current memory cell contents. Those two data items are compared against changes with use of XOR gate. Variable change detector is responsible for catching any changes. Detector register is set during write cycle at the first detected difference. The HC line becomes active notifying that difference has been observed. For initialization purposes there is an input (HC_CLR) that allows to clear HC flag. In order to make available this signal to the CPU it is mapped in marker space. The HC signal can be read and processed by the CPU as marker flag.

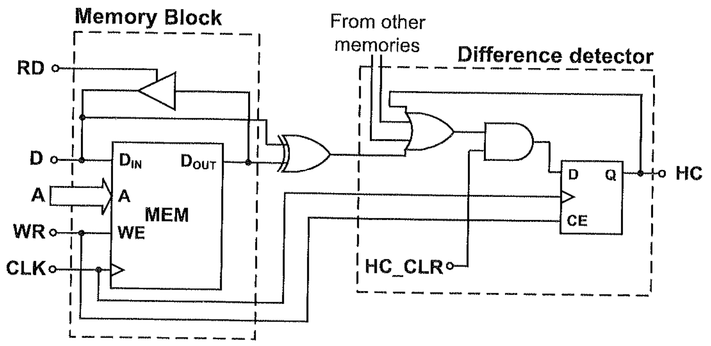


Fig. 11. Memory system with change detection

Diagram shows only single bit construction. Circuit can be easily extended for detecting changes in memories with word longer than one bit. For those purpose each data line is equipped with difference detector (XOR gate) and difference results are accumulated by multiple input OR-gate.

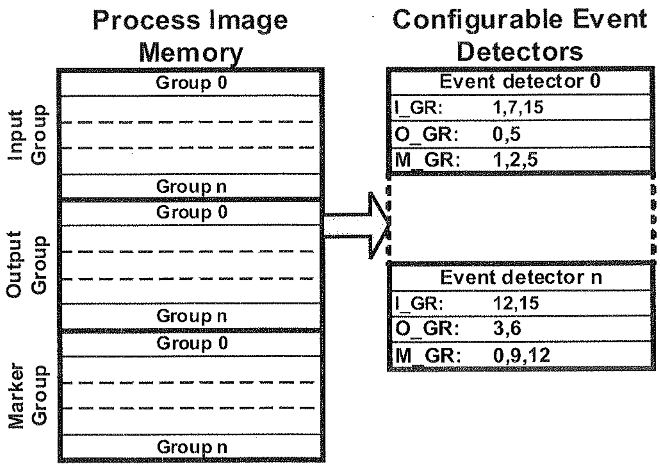


Fig. 12. Process image memory mapping to difference detection unit

Proposed change detector covers problem of change detection inside given space of memory (according to memory space assignment). In order to determine condition for program block execution several change markers should be checked. This approach has been considered as inefficient solution. Variable grouping reduces overhead connected with condition checking (see Fig. 6). Further optimization are possible in hardware detectors. Hardware detectors can integrate functionality that allow to selectively guard required memory regions covering desired variable set into change detection system. Exemplary diagram of process memory regions assignment is shown in Fig. 12. Process image memory is divided into three spaces: input, output and internal marker. Each space consist from n groups that can be individually included to selected difference detector . Presented idea allows for hardware detection of changes in selected memory areas. The change detector can accommodate different requirements of watched area by configurable memory mapping in so called membership look-up table. Detailed construction of memory watching system is shown in Fig. 13. In the diagram there is a main memory that is used by the PLC CPU as process image memory. To the memory is connected value change detector. Whenever data in memory cells are modified difference notification is passed to configurable watch points. The configurable watch point is surrounded by rectangle. Inside watch point there are three look up tables. If write cycle falls into watched area, look-up table output is active. Each look-up table is assigned to different memory space (input, output, markers). Number of cells in look up table determines possible number of group that each space can consists of. Content of look-up tables describes membership of memory spaces and groups to given task. This membership look-up tables are loaded by the PLC CPU during initialization of system. Variable membership data should be determined by program compiler based on user program.

Each task that is supposed to be executed conditionally requires one watch point. There are two requirements that should be met by watch point system:

1. Large number of independent watch points (determines number of conditionally executed program blocks)
2. Large number of groups covered by one watch point determines precision of variable localization in memory space

There exist a trade off between number of groups and memory space that allow to store membership information. If number of groups is growing triggering condition can be very precisely determined as number of discrete or byte variables inside group is relatively small. Usually each task is using variables that belong to several groups. In order to detect changes all those regions must be watched This will require relatively large storage space that keeps variables assignment information.

The number of conditionally handled tasks depends on number of change detectors implemented inside an FPGA device. From a PLC user point of view it is important that number of this blocks is relatively high in order to allow flexible program execution. Watch points architecture should be as simple as possible and perfectly fit into FPGA architecture.

Those opposing requirements cause that number of group and number of watch points must be reduced to a number that can fit inside selected FPGA device.

The configurable watch point was implemented with use of an HDL language and widely examined. For implementation purposes was chosen Spartan 3 FPGA family from Xilinx [12]. Those are symmetrical FPGA devices with integrated block memories. One block memory can store up to 16kb. Very important feature of those memories is dual gate access system with configurable bus width from 1-32 bits with step of power of 2.

Table 1

Change detector logic resource utilization

Number of groups	LUTs	Distributed RAMs
3 x 16	20	3
3 x 32	33	6
3 x 64	66	12
3 x 128	110	24

In the first approach the difference detector with membership memories was implemented according to block diagram shown in Fig. 13. There was implemented versions with number of groups from 16 – 128 for each memory space. Membership look-up tables are implemented as distributed RAM blocks formed from LUTs. Implementation results are collected in Table 1.

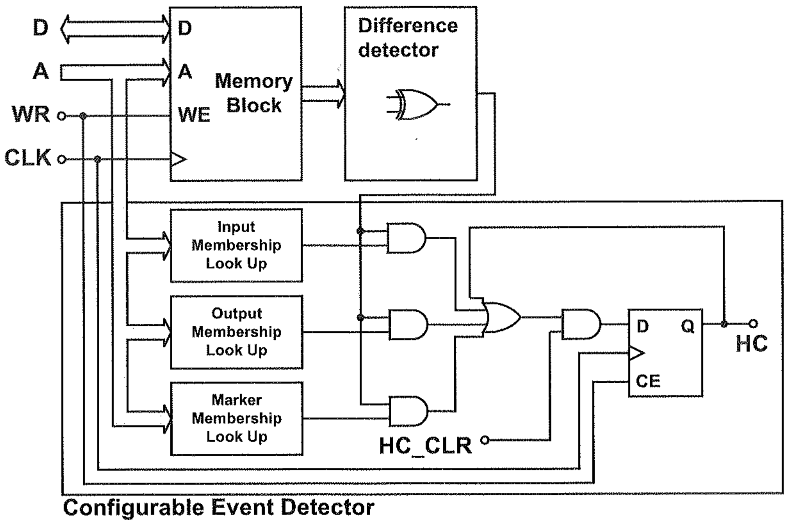


Fig. 13. Difference detector with guarded memory regions assignment

Difference detector instead of using distributed RAM cells for membership implementation can use block RAM that is also available in FPGA device. This modification allow to save logic resources that are used for membership function implementation. Block RAM can be configured to operate with 32 + 4 bit data words. This allow to implement up 36 independent difference detectors using one block RAM. The 32+4 bit data word configuration allows defining up 512 separate regions inside memory. Finally two implementation of process memory have been created with different approaches in implementation of difference detection systems. Final implementation results are gathered in Table 2. In the first line are gathered results for implementation based on distributed RAMs while second line describes unit designed with use of block RAMs. The table presents logic requirements, possible number of groups that can be defined in memory and number of independent detectors implemented.

Table 2

Process image memory with difference detection

Channels	Number of groups	LUTs	Distr. RAMs	Block RAMs
8	3x16	158	24	1
36	512	147	0	2

As it is presented logic complexity of both circuits is comparable. They consume almost the same number of general purpose components. The only difference is in membership look-up table implementation. Using block RAM resource (second line of Table 2) instead of distributed RAMs (first line of Table 2) it is possible to:

1. Increase number of monitored groups from 48 (3x16) to 512 (8 x increase)
2. Increase number of independent channels from 8 to 36 (4.5 x increase)
3. Slightly reduction of logic resources allocation from 158 to 147 LUTs (about 7%)

Finally it can be noticed that with use of the smallest FPGA device from Spartan 3 family (XC3S50 – 4xRAMB16) it is possible to implement 2kB process image memory (1xRAMB16) with up to 108 independent watch points (3xRAMB16).

7. CONCLUSIONS

Studies on the optimized program execution in CPU of the PLC have shown the grate improvement in operation speed. The presented solutions bring especially many benefits in case of control of the processes for which input signal changes happen relatively rarely, as in such situation typical controller executes "empty" loops, while the event driven controller will only perform testing for changes, what will significantly reduce access time to the input/output signals whose change happens during such an "empty" cycle. The authors' experience show that many industrial application satisfy the above condition. Most industrial processes feature great number of inputs/outputs, whose state changes very rarely – controller executes then many instructions uselessly.

This time could be utilized for other important tasks as for example network communication, or simply more frequent checking of the object state what would significantly reduce response time for the signal changes of strategic importance. Because in real applications, especially using event-driven execution, control program scan time varies from cycle to cycle, it seems that it would be better to introduce a new measure of a PLC performance - average time of one control program scan. Such parameter seems to be more useful for determining a real performance of a PLC system.

The presented discussion concerns the attempt to apply standard CPU solutions met in programmable controllers for nonstandard tasks, however, research is also done relating to hardware support of the changes detection, what should lead to even greater response time reduction. In this approach selective program execution takes benefits from specific memory controller that compares memory contents during write operation. Through configurable group membership tables it is possible to determine if given write operation cause condition change for given program task or block. All those operation are performed in parallel with CPU operations. There are not additional processing required for condition calculation just conditional execution flag is checked for given program block. Even relatively small and cheap FPGA device can implement up to 108 independently triggered processes with possible definition of 512 variables groups.

Further research and development will concentrate on detailed tests and implementation improvement of proposed solution. An optimization in autonomous input and output data collecting and processing is also considered. Concurrently we investigate possibility of operating speed improving of typical bit-byte structure of a PLC CPU [4,7]. We hope that results of the presented work will be useful for this structure too.

8. REFERENCES

1. N. Aramaki, Y. Shimokawa, S. Kuno, T. Saitoh, H. Hashimoto: *A new Architecture for High-performance Programmable Logic Controller*, Proc. of the IECON'97 23rd Int. Conf. on Industrial Electronics, Control and Instrumentation, IEEE, 1997, volume 1, 187-190, New York, USA.
2. H. Berger: *Automating with STEP 7 in STL and SCL – SIMATIC S7-300/400 Programmable Controllers*, Siemens AG, Germany, 2001
3. H. Berger: *Automating with STEP 7 in LAD and FBD – SIMATIC S7-300/400 Programmable Controllers*, Siemens AG, Germany, 2001
4. M. Chmiel, E. Hrynkiewicz, A. Milik: *Remarks on Improving of Operation Speed of The PLCs*, Preprints of the 16th IFAC World Congress, Prague, Czech Republic, July 3-8, 2005.
5. M. Chmiel and E. Hrynkiewicz: *Remarks on Parallel Bit-Byte CPU structures of Programmable Logic Controllers*. In Adamski M. A., Karatkevich A. Węgrzyn M. (ed.): Design of Embedded Control Systems, Section V, 231-242. Springer Science+Business Media, 2005.
6. M. Chmiel, E. Hrynkiewicz, A. Milik: *Compact PLC With Event-Driven Program Tasks Execution*, Preprints of the 3RD IFAC Workshop On Discrete-Event System Design, DESDes'06, pp. 99-104, Rydzyna near Leszno, Poland, September 26-28, 2006.
7. M. Chmiel, and E. Hrynkiewicz: *Fast Operating Bit-Byte PLC, Preprints of the 17th IFAC World Congress, Seoul*, Korea, July 6-11, 2008.
8. J. Donandt: *Improving response time of Programmable Logic Controllers by use of a Boolean coprocessor*. IEEE Comput. Soc. Press., no 4, Washington, DC, USA, 1989, pp.167-169.
9. E. Hrynkiewicz, A. Milik, J. Mocha: *Dynamically Reconfigurable Concurrent Control Program Execution*, Proceedings of National Electronic Conference, Poland, Darłowko Wschodnie, June, 2008.
10. G. Michel: *Programmable Logic Controllers, Architecture and Applications, John Wiley & Sons, West Sussex*, England, 1990.
11. A. Milik: *High Level Synthesis – Reconfigurable Hardware Implementation of Programmable Logic Controller*, PDeS 2006 Programable Devices and Embedded Systems, Brno 14-16 February 2006
12. Xilinx: XAPP 430: Spartan-3 Advanced Configuration Architecture, 2006.

Logic synthesis dedicated for CPLD circuits

DARIUSZ KANIA*, ADAM MILIK*, JÓZEF KULISZ*, ADAM OPARA**, ROBERT CZERWIŃSKI*

Politechnika Śląska, Instytut Elektroniki, Instytut Informatyki***
e-mail: dkania@.pdsi.pl

Received 2009.01.05
Authorized 2009.04.03

The paper presents synthesis strategies for PAL-based devices. All component methods used in presented strategies are originally developed. In this paper the essentials of all methods have been presented. Exact algorithms descriptions can be found in referenced materials. The optimization of synthesis methods were aimed toward required areas minimization or propagation delay minimization (reducing number of levels).

A low computation complexity of synthesis methods that use tri-state output buffers or output graphs make them useful as additional steps of complex synthesis strategies. Application of those methods can radically reduce areas or propagation delay. Without doubt the best results in terms of required surface can be obtained by methods that use decomposition components. Decomposition methods that extend classical model of functional decomposition (Curtis' decomposition – row based and column based decompositions) are computing demanding procedures. The binary decision diagram was taken into consideration in order to increase computation performance/efficiency. The experience that has been gained in implementation of column and row based decomposition allows to implement efficient partitioning procedures for the BDD. Decomposition results for the BDD methods are slightly worse as referenced to previous approaches. The synthesis process is computation efficient and allows to decompose complex logic circuits in reasonable amount of time. The exploration of BDD decomposition methods shows their undiscovered potential that still can be developed especially for decomposition of function consisting of few hundred of input and output variables.

Several years' of experience in design of decomposition procedures for CPLD allows developing complex synthesis strategies that have been presented as summary of the paper. They are dedicated for different CPLD families addressing different features (e.g. three-state output buffers) and requirements (e.g. propagation time constraint).

Keywords: logic synthesis, CPLD, decomposition, technology mapping

1. INTRODUCTION

Technological progress drives the necessity of constant improvement of logic synthesis algorithms. In recent years Field Programmable Gate Array (FPGA) structures have gained the greatest popularity. Many various synthesis methods were developed for them [1-7]. A number of synthesis methods, dedicated strictly for Complex Programmable Logic Devices (CPLD-s), also appeared in the literature [8-18]. They sometimes utilize elements of synthesis dedicated for FPGA devices. In such a case certain steps of the algorithm are modified to include constraints characteristic for resources available in CPLD-s.

A very interesting approach was presented in [9,11]. Generally in the synthesis process methods developed for FPGA devices were employed, but Look-up Table (LUT) blocks were replaced by Programmable Logic Array (PLA) cells, characteristic for CPLD architectures. The optimal structure of the cells was selected basing on experiments presented in [19-20]. The authors prove, that it would be more efficient with respect to area occupied in silicon, to build logic blocks as small PLA arrays, instead of LUT blocks [13-16, 19-22]. On the other hand synthesis methods dedicated for PLA structures are well mastered, and known for a long time [23-27]. These two observations let us suppose that it is quite likely that devices, in which arrays characteristic for Simple Programmable Logic Devices (SPLD-s) are the main building blocks, can regain their popularity.

One of the basic problems in logic synthesis dedicated for Programmable Logic Devices (PLD-s) is project decomposition. The goal of decomposition is to partition the entire design into parts that can be directly mapped onto logic blocks available in the target structure. Most of contemporary CPLD circuits consist of logic blocks with internal structures that resemble architectures of simple PAL devices (Fig. 1). Further on in this paper such devices will be referred to as PAL-based CPLD-s. A characteristic feature of a PAL-based CPLD is that its basic building block (a PAL block) contains a limited, and small number of AND gates (product terms).

The classical method of logic synthesis, dedicated for PAL-based CPLD-s, and implemented in great majority of vendor tools, consists of two steps. First a two-level minimization is applied separately to every single-output function, next implementation of the minimized functions in PAL-based blocks, containing a predefined number of product terms, is performed. If the number of implicants Δ_f, representing a function after minimization, is greater than the number of product terms k, available in a logic block (Fig. 1), a greater number of logic blocks has to be utilised to implement the function. The classical product term expansion method utilize feedback lines to build a multi-level cascaded structure, which increases propagation delays significantly.

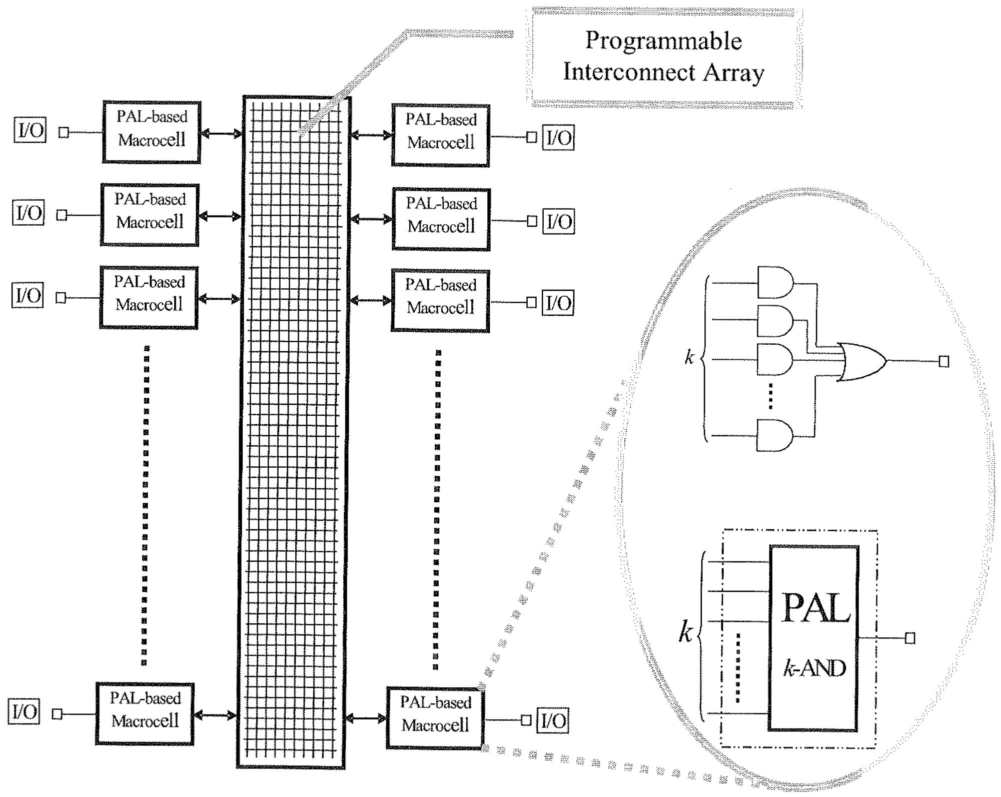


Fig. 1. An idealized CPLD architecture with PAL-based logic blocks consisting of k product terms

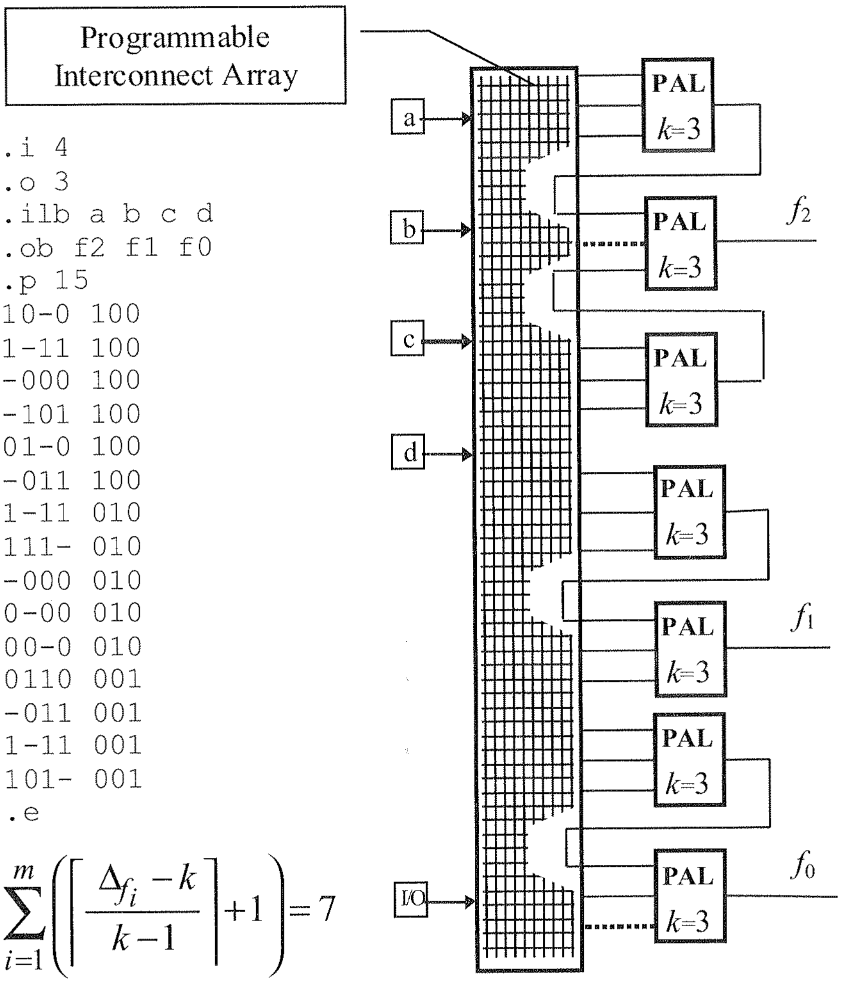
An implementation of a minimized function f, which can be represented as a sum of Δ_f implicants, requires δ_f PAL-based logic blocks containing k product terms (Eq. 1).

$$\delta_f = \left\lceil \frac{\Delta_f - k}{k - 1} \right\rceil + 1 \tag{1}$$

Similarly, the classical implementation of a $f : B^n \rightarrow B^m$ function requires δ_f^1 PAL-based logic blocks (Eq. 2).

$$\delta_f^1 = \sum_{i=1}^{m} \delta_{f_i} = \sum_{i=1}^{m} \left(\left\lceil \frac{\Delta_{f_i} - k}{k - 1} \right\rceil + 1 \right) \tag{2}$$

As an example a classical implementation of a function $f : B^4 \rightarrow B^3$, utilising PAL blocks containing three ($k = 3$) product terms, is presented (Fig. 2).



```
.i 4
.o 3
.ilb a b c d
.ob f2 f1 f0
.p 15
10-0 100
1-11 100
-000 100
-101 100
01-0 100
-011 100
1-11 010
111- 010
-000 010
0-00 010
00-0 010
0110 001
-011 001
1-11 001
101- 001
.e
```

$$\delta_f^1 = \sum_{i=1}^{m} \left(\left\lceil \frac{\Delta_{f_i} - k}{k-1} \right\rceil + 1 \right) = 7$$

Fig. 2. A classical implementation of a function $f : B^4 \rightarrow B^3$, utilising PAL-based logic blocks containing 3 product terms

The purpose of this paper is to present more effective methods of function implementation in PAL-based CPLD structures, and propose synthesis strategies appropriate for various device architectures, and optimisation goals. The synthesis strategies proposed in the paper make use of various methods and algorithms, developed by the authors during several years of research work.

The paper is structured as follows: the first chapters briefly presents the synthesis algorithms, their properties, and possible applications. Section 2 introduces some new concepts of product term expansion. Section 3 focuses on multi-level synthesis based on the Graph of Outputs. Various decomposition models for PAL-based devices are presented in Section 4. Section 5 gathers this information, and presents a complex synthesis strategy dedicated for PAL-based CPLD-s. The strategy comprises all of the methods presented before. Basic information about interface to vendor tools, and some experimental results are reported in Section 6. The paper is concluded with a summary in Section 7.

2. LOGIC SYNTHESIS METHODS BASED ON THE CONCEPT OF EXPANSION EXPLOITING TRI-STATE OUTPUTS

Logic blocks contained in CPLD structures feature usually additional logic resources that can facilitate product term expansion. These resources include parallel expanders, folded NAND feedback lines, often referred to as shared expanders, logic allocators, and tri-state output buffers.

The expanders enable unequal distribution of product terms between macrocells, and extending the number of products available for one function beyond the limit of k terms contained in one PAL block. Anyway they can only move the limit to a greater value, and they do not provide feasibility of implementation for every function. Additional expansion of the number of terms is thus necessary.

Product term expansion that exploits tri-state output buffers seems to be the most attractive solution, as it doesn't lead to expansion of logic levels. The idea is presented in Fig. 3.

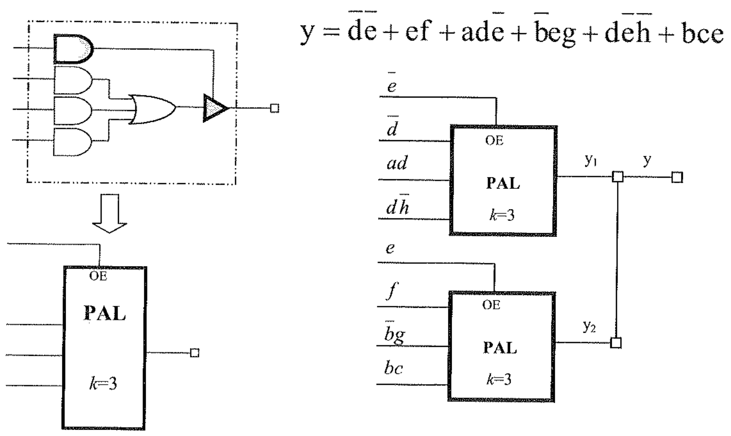

$y = \bar{d}\bar{e} + ef + ad\bar{e} + \bar{b}eg + d\bar{e}\bar{h} + bce$
OE
PAL
k=3
$\bar{e}$
$\bar{d}$
ad
$d\bar{h}$
y_1
y
e
f
$\bar{b}g$
bc
y_2

Fig. 3. The essence of product term expansion exploiting three-state output buffers

The concept of product term expansion utilizing tri-state buffers lies in the background of an original synthesis method, dedicated for CPLD-s featuring three-state terminals. The set of multioutput implicants of a Boolean function $f : B^n \rightarrow \{0, 1, -\}^m$ serves as the starting point for a two-level synthesis. It has been assumed that the number of inputs of a PAL block is big enough, so the proposed synthesis model does not comprise an algorithm of input partitioning.

The two-level synthesis consists of Two-level Splitting Minimization, PAL-oriented term partitioning, and PAL mapping.

The synthesis process starts with the Two-level Splitting Minimization. Then partitioning of individual minimized functions is performed. As a result of the two procedures, the set of implicants of a Boolean function is divided into subsets with cardinality less than the number of terms available in one PAL block.

The detailed criterion of minimization may vary slightly, according to the implementation style. The idea of the Two-level Splitting Minimization is presented in Fig. 4. The objective of the classical two-level minimization is to reduce both the number of products in the Boolean formula representing a function, and the number of literals in a product. Because of a limited number of multi input terms available in a PAL block, the primary goal of the Two-level Splitting Minimization is to reduce the number of products. Reduction of literals is inessential. As the result of the Two-level Splitting Minimization, we obtain a set of terms of minimum cardinality. The terms contain a reduced number of "-" elements.

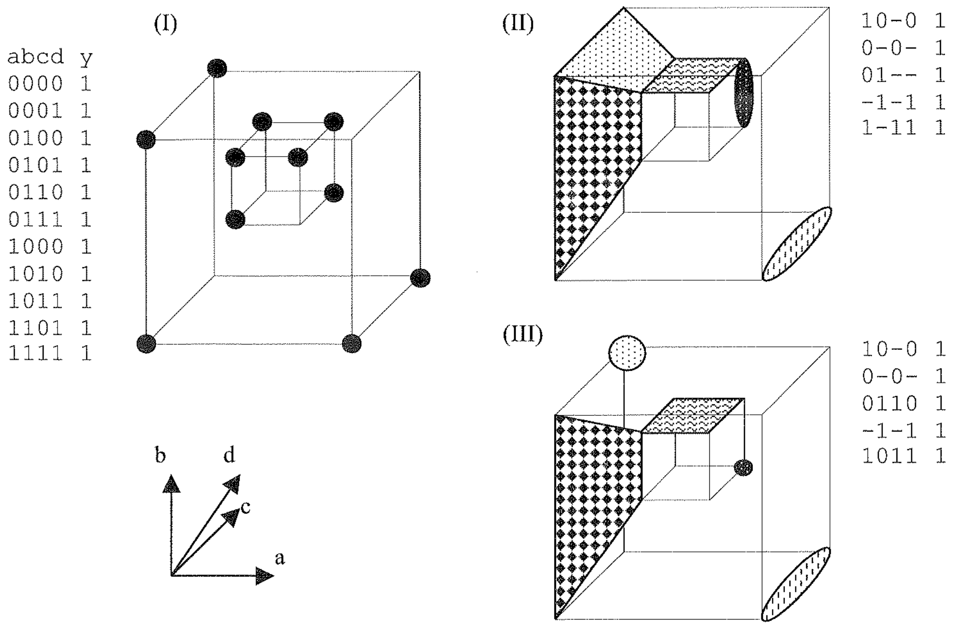


Fig. 4. The essence of the Two-level Splitting Minimization: (I) minterms of the exemplary function; (II) results of the classical two-level minimization, (III) results of the Two-level Splitting Minimization

The objective of the PAL-oriented term partitioning procedure is to subdivide the set of implicants into subsets, for which cardinality is less or equal to the number of terms (k) available in a PAL-based block. Different concepts of the algorithm are presented in [28-31]. Sometimes the PAL-oriented term partitioning procedure doesn't lead to a one-level structure, but to a p-level[1] structure, where p is a parameter of the synthesis algorithm [32].

The idea of the PAL-oriented term partitioning, and PAL-based logic block mapping is presented in Fig. 5.

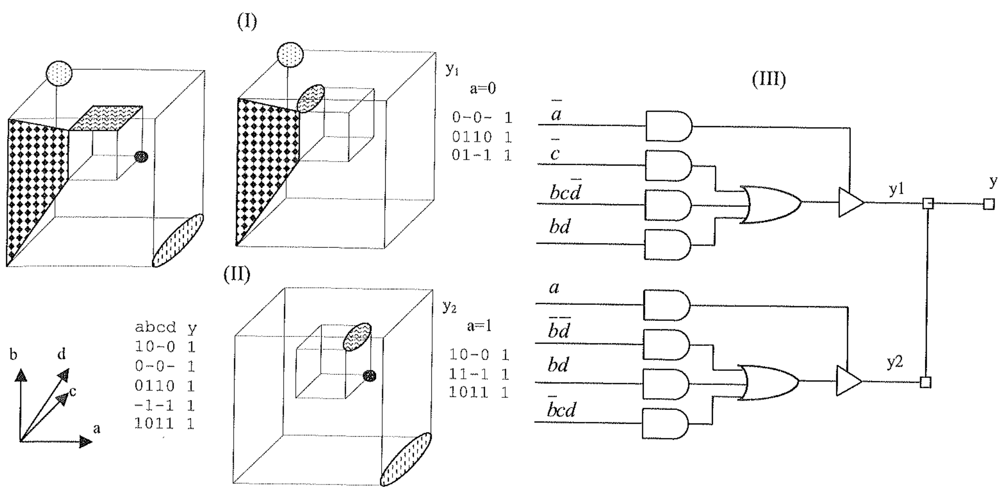


Fig. 5. The idea of PAL-oriented term partitioning: (I) partitioning vector a = 0, (II) partitioning vector a = 1, (III) mapping onto PAL blocks featuring 3 terms

The synthesis algorithms exploiting tri-state buffers were implemented in software, and are elements of the PALDec synthesis system. Results of synthesis for different PAL-based devices are presented in [28-33]. The methods are especially attractive with respect to dynamic parameters. The algorithms discussed above can be used as independent synthesis methods, or constitute an extension for other tools, improving dynamic properties of final solutions.

3. MULTI-LEVEL SYNTHESIS FOR PAL-BASED DEVICES BASED ON THE GRAPH OF OUTPUTS

Another approach to logic synthesis consists in analyzing the set of implicants for a multi-output function, and extracting the groups, that are common for several outputs. The method is based on analysis of the so-called Graph of Outputs.

[1] Hereafter in this paper, we will interpret the term "*number of logic levels*" (and consequently "*one-level structure*", or "*p-level structure*") as the number of cascaded PAL blocks in the longest signal path from the inputs to the outputs in the circuit of concern. The exception to this rule will be the terms "*two-level minimization*", and "*two-level synthesis*". These terms are well established in the literature, and we will use them in their traditional meaning, i. e. "*two-level*" = two levels of logic gates.

A minimized multi-output function $f : B^n \rightarrow B^m$ can be described by a set of multi-output implicants, consisting of the input part, containing the {0,1,-} symbols, and the output part, containing the {0,1} symbols. Let us assume, that $G < Y, \vec{U} >$ is a directed graph, where Y is the set of the graph nodes, and $\vec{U}$ is the set of the graph edges. An example function, and the graph corresponding to it are presented in Fig. 6. Every node of the graph represents a different output part of the multi-output implicants. Edges of the graph connect nodes, for which the output parts have at least one common 1 symbol at the same position, and simultaneously the code distance between the output parts is minimal [33-34].

```
.i 4
.o 3
.ilb a b c d
.ob f3 f2 f1
.p 8
-000 010
0110 111
100- 110
-010 001
0011 111
1111 111
110- 001
010- 110
.e
```

$$Y = \begin{Bmatrix} 001 \\ 010 \\ 110 \\ 111 \end{Bmatrix}_{f_3, f_2, f_1}$$

$$\vec{U} = \begin{Bmatrix} (001,111) \\ (010,110) \\ (110,111) \end{Bmatrix}$$

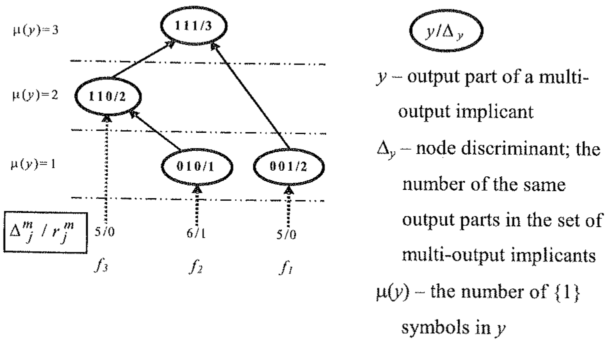

μ(y)=3
μ(y)=2
μ(y)=1
111/3
110/2
010/1
001/2
Δ_j^m / r_j^m
5/0
6/1
5/0
f_3
f_2
f_1
y/Δ_y
y – output part of a multi-output implicant
Δ_y – node discriminant; the number of the same output parts in the set of multi-output implicants
μ(y) – the number of {1} symbols in y

Fig. 6. Representation a minimized multi-output function $f : B^4 \rightarrow B^3$ by means of the directed graph G

Every node in the lowest range of the graph corresponds to a function output. Every output can be assigned the Δ_j^m, and r_j^m parameters, where j is the index of the given output, and m is the number of outputs of the multi-output function f. The value of Δ_j^m is equal to the sum of node discriminants that belong to the directed path, starting from the j-th node in the lowest range of the graph, and ending in a node in the highest possible range for the j-th output. As an example: $\Delta_2^3 = \Delta_{010}+\Delta_{110}+\Delta_{111} = 1+2+3 = 6$ – see Fig. 6. The r_j^m parameter, referred to as the remainder, is a number calculated from the following relation:

$$\Delta_j^m - 1 \equiv r_j^m(\mathrm{mod}(k-1)) \qquad (3)$$

where k stands for the number of terms available in the PAL blocks used for implementation.

A graph built according to the procedure described above is referred to as the Graph of Outputs. Basing on analysis of the Graph of Outputs, solutions that require less logic blocks than the classical approach, can be found. Theoretical backgrounds for the method are presented in [33-34]. The algorithm is founded on the Theorem of

Selection of a Node of the Graph. Proof of the theorem can be found in [33], together with many further details.

The essence of the technology mapping consists in analysis of the graph, and searching for the node discriminants that are associated with possibly large groups of common implicants. These groups of implicants are implemented by common PAL-based logic blocks.

The synthesis algorithm consists in iterative selection of the graph nodes. After selecting in step i a graph node, a new PAL block is introduced to the synthesized multi-level structure, starting from the inputs, and growing towards the outputs.

The rules for selecting a node can be deducted directly from the Theorem of Selection of a Node of the Graph [33-34]. They can be described by the following procedure:

1. From the set of all graph nodes the node ${}^{i}\Delta_y$ is chosen, for which $\mu\,({}^{i}\Delta_y) = \max$ (i. e. the node, which is placed on the highest range of the graph).
2. If several nodes lie in the same range, further selection is carried out, depending on values of the discriminants:

2a. if there are nodes, for which ${}^{i}\Delta_y \geq k$, the node is selected, for which the discriminant ${}^{i}\Delta_y = \max$;

2b. if for all discriminants ${}^{i}\Delta_y < k$, the node is selected, for which within the set of remainders $\mathbf{R} = \left\{ r_j^{\mu({}^{i}\Delta_y)}; j \in \left\langle 1, \mu({}^{i}\Delta_y) \right\rangle \right\}$ the maximum number of remainders $r_x^{\mu({}^{i}\Delta_y)}$ satisfies the condition $0 < r_x^{\mu({}^{i}\Delta_y)} >^{i} \Delta_y < k$

3. If no nodes meet the criteria 1 and 2, implementation of the implicants assigned to the other nodes is carried out by means of the classical method.

The Graph of Outputs for the example function from Fig. 6, together with the associated values of discriminants Δ_j^m, remainders r_j^m, and the implementation using PAL-based blocks featuring 3 product terms, is shown in Fig. 7.

The classical implementation of this function requires 16 product terms (5, 6, and 5 for each output respectively). If using PAL-based logic blocks containing 3 product terms, a 3-level structure consisting of 7 blocks is obtained.

The synthesis method presented above was implemented in software as a module of the PALDec synthesis system. Results of synthesis for different PAL-based devices are presented in [33-39]. The method is especially efficient with respect to chip area. The algorithm presented above can be used as an independent synthesis method, or constitute an extension for other methods, used as a tool for optimising resources usage.

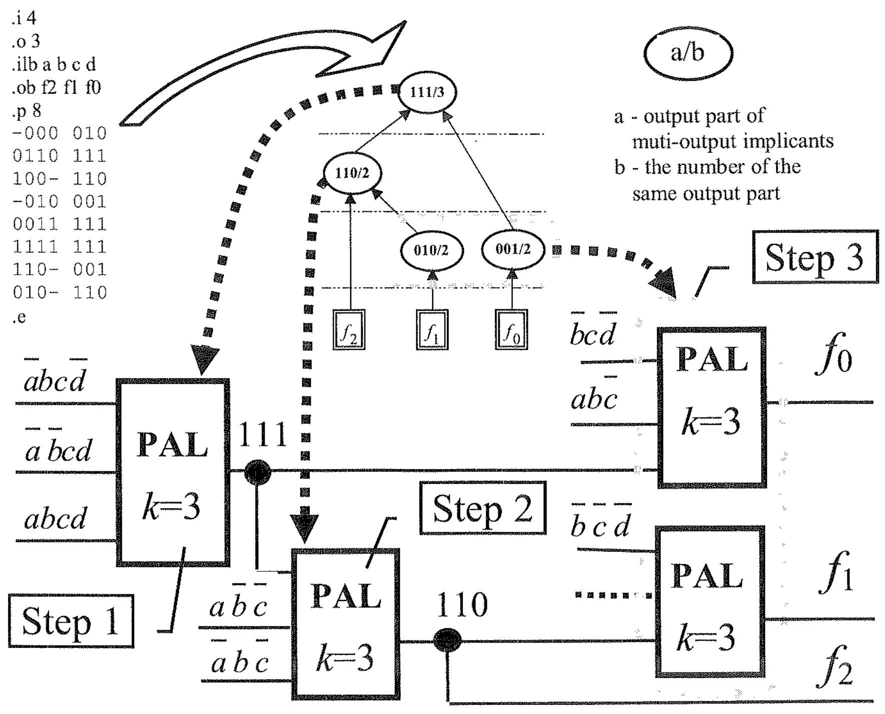

```
.i 4
.o 3
.ilb a b c d
.ob f2 f1 f0
.p 8
-000 010
0110 111
100- 110
-010 001
0011 111
1111 111
110- 001
010- 110
.e
```

Fig. 7. A multi-level implementation of the example multioutput function, utilising PAL-based logic blocks containing 3 product terms

4. DECOMPOSITION MODELS DEDICATED FOR PAL-BASED DEVICES

Decomposition plays an extremely important role in modern logic synthesis. In spite of rapid and multifarious progress in the field, the optimal means of partitioning a digital circuit into logic blocks characteristic for programmable structures is still not known at present.

In great majority of applications decomposition is used in synthesis dedicated for FPGA circuits. It is an essential part that enables partitioning of a designed circuit, and mapping it onto configurable logic blocks (CLB-s).

Decomposition is hardly ever used in synthesis dedicated for other types of PLD-s. There are few known algorithms, dedicated for PLA structures [9, 21-23, 25-27]. In some of them decomposition methods developed for LUT-based FPGA-s were directly transferred [9, 11]. Input and output assignment is a characteristic feature of those methods, that significantly influences the number of products in blocks of minimized functions, obtained after the decomposition process [40-41]. The problem of proper encoding of inputs and outputs is widely discussed in connection with state assignment for FSM-s in [42-43]. Those problems are related to symbolic state encoding, dichotomy theory, multi-value function minimization, and the concept of output dominance. There are also works devoted to binding input and output coding process with decomposition problems.

The main constraint characteristic of a PAL-based logic block is not the number of inputs, but the number of multi-input product terms. This results in an observation, that the objective of decomposition dedicated for PAL-based structures can be defined in the form of the following two major tasks:

1. Minimizing the number of PAL-based blocks used;
2. "Fitting" the designed circuit into PAL-based logic blocks best.

Further on three models of decomposition, directly concerning the above-mentioned issues, will be presented.

4.1. THE COLUMN DECOMPOSITION

A function $f : B^n \rightarrow B^m$ can be decomposed if, and only if, the column multiplicity of the partition matrix $\nu(X_2 \mid X_1)$ is lower or equal to 2^p, i.e.

$$\nu(X_2|X_1) \leq 2^p \quad \Leftrightarrow \quad f(X_2, X_1) = F\,[g_1(X_1), g_2(X_1), ..., g_p(X_1), X_2] \tag{4}$$

where the $\mathbf{X_1}$ and $\mathbf{X_2}$ sets should satisfy the conditions $\mathbf{X}_1 \cup \mathbf{X}_2 = \mathbf{I} = \{i_n, ..., i_2, i_1\}$, and $X1 \cap X2 = \phi$ [44]. The $\mathbf{X}_1$ and $\mathbf{X}_2$ sets are respectively called the bound and the free set.

Analysis of the classical decomposition model proposed by Curtis shows that the partitioning expands the total number of outputs in the circuit. At least p additional blocks are required for implementation of the bound block. Employing decomposition is thus efficient only in the case, when the classical approach leads to a cascaded structure that requires a greater number of a PAL-based block to implement the same function.

In the Column Decomposition model presented hereafter, the number of AND gates required by the implementation is minimized. As a consequence, the number of PAL-based blocks is minimized simultaneously. The process of the proposed column decomposition can be divided into the following steps:

- Selection of the bound variables;
- Calculation of column multiplicity;
- Column pattern code assignment;
- Implementation of the bound and the free blocks.

All the steps are optimized for PAL-based devices with a predefined number of product terms. The steps of the decomposition process listed above will be briefly explained below.

Selection of the bound variables: Selecting variables for the bound set is based on searching for such a variable partitioning, which implies splitting of the analyzed circuit into two subcircuits of similar complexity. The circuit complexity is estimated by multiplying the number of inputs and the number of outputs of a block. The set of input variables $\mathbf{I} = \{i_n, \ldots, i_2, i_1\}$ is partitioned into subsets $\mathbf{X}_1$ and $\mathbf{X}_2$ in such a way, that the value of $\overline{\overline{\mathbf{X}_1}} \cdot p + (\overline{\overline{\mathbf{X}_2}} + p) \cdot m$ is minimal (see Fig. 8).

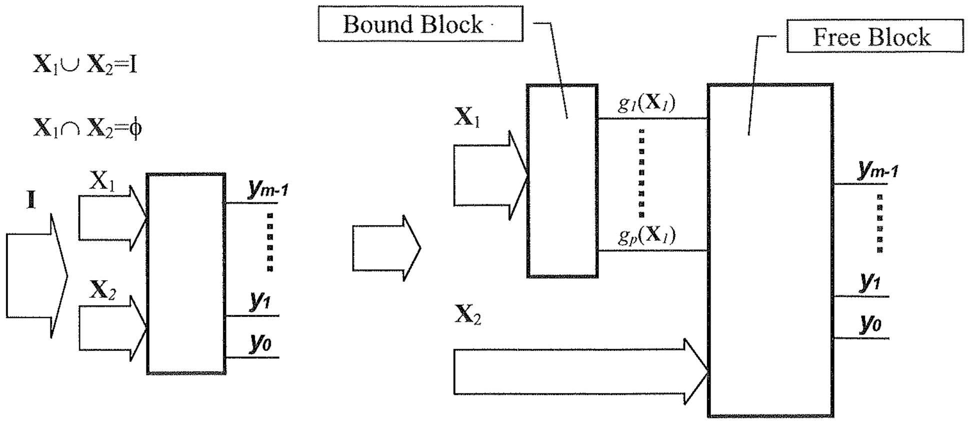


Fig. 8. The essence of selection of variables for the bound set

Calculation of column multiplicity: The column multiplicity is obtained as a result of colouring of the Column Incompatibility Graph. A special colouring algorithm was proposed for this purpose. The algorithm introduces elements of two-level minimization at early stages of the decomposition process. The method is presented in detail in [45].

Column pattern code assignment: The column multiplicity of the partition matrix is equal to the chromatic number of the Column Incompatibility Graph $\gamma(G)$. The number of bits required to distinguish all column patterns is equal to $\lceil \lg_2 \gamma(G) \rceil$. The bound block works in fact as a specific code translator. Its output code is determined by column pattern assignment. The way the codes are assigned to each of the columns can significantly influence complexity of the bound and the free blocks after minimization. Various techniques of output and input coding are described in [23, 27, 40]. Usually the coding methods are closely related to FSM state assignment. Methods of product minimization after primary circuit partitioning, and appropriate input-output coding for PLA structures, are also considered. Algorithms of this class can directly be implemented in synthesis process dedicated for PAL-based structures, but the results obtained are rather unsatisfactory. Column pattern assignment for CPLD-s should take into account properties of the target architecture. In the proposed pattern assignment method, an attempt is made to anticipate the results of two-level logic minimization. The columns that appear more often in the partition matrix, are assigned codes that require less products. As a consequence for the whole set of column patterns the codes are selected, that provide the minimal number of products after minimization. The proposed code assignment method employs uniform coding [46-48], coverage pattern coefficients, column code coefficients, pattern pair coverage coefficients, and neighbourhood pattern coefficients. The detailed algorithm of the pattern code assignment is described in [33, 49].

***Implementation of bound and the free blocks*:** After partitioning of the $\mathbf{I}=\{i_n,\ldots,i_2,i_1\}$ set into subsets $\mathbf{X}_1=\{i_{nz},\ldots,i_2,i_1\}$ and $\mathbf{X}_2=\{i_n,\ldots,i_{nz+2},i_{nz+1}\}$, and encoding column patterns using method described above, a two level function minimization is carried out. Minimization is performed for the functions that describe the bound and the free blocks. Following that, the number of PAL-based logic blocks required for implementation δ_{zw} is calculated, using Eq. (5). The number is a sum of the number of logic blocks required for implementing the bound (δ_z) and free (δ_w) blocks. Symbols z_i and w_i describe the numbers of product terms required for implementation of the i-th function of respectively the bound and the free blocks.

$$\delta_{zw}=\delta_z+\delta_w=\sum_{i=1}^{p}\left(\left\lceil\frac{z_i-k}{k-1}\right\rceil+1\right)+\sum_{i=1}^{m}\left(\left\lceil\frac{w_i-k}{k-1}\right\rceil+1\right) \tag{5}$$

If the δ_{zw} value is greater or equal to number of logic block required in classical approach, the function is implemented using the classical method. In the other case the next step is performed, during which the free and bound blocks are subject to further column decomposition. Additionally, after the two-level minimization of single-output functions (Espresso-Dso), the condition $\Delta_{fi}<2k$ is checked (where Δ_{fi} stands for the number of implicants representing the function f_i – see Eq. 1). When this inequity is true, the function is implemented using the classical method. This approach reduces the total number of outputs analysed in next decomposition steps.

The column decomposition method was implemented in the PALDec synthesis system. Results of synthesis for different PAL-based devices are presented in [33, 50-51]. The method is the most attractive in respect of area, but unfortunately is characterized by a quite high computational complexity.

4.2. THE ROW DECOMPOSITION

Decomposition is usually carried out for "fitting" the number of inputs of a synthesized blocks to the number of inputs available in configurable LUT-based blocks of a target programmable structure. After introducing certain additional elements, decomposition can also lead to minimization of the total number of products, indirectly influencing minimization of the number of PAL-based blocks. Incorporating into the decomposition elements of "fitting" the synthesized functions into PAL-based blocks could constitute a valuable extension of the method.

The proposed Row Decomposition (two-stage decomposition) directly focuses on "fitting" the designed circuit into PAL-based logic blocks best. The essence of the proposed concept (Fig. 9) consists in finding such a design partitioning (function decomposition), which enables implementation of the free block in one PAL-based logic block featuring a predefined number of product terms.

Additionally, minimization of the number of the bound block outputs is carried out. As for a given Karnaugh map the number of the bound block outputs is equal to the row multiplicity ($p=\mu(\mathrm{X}_2\mid\mathrm{X}_1)$), one of the main problems occurring at the initial

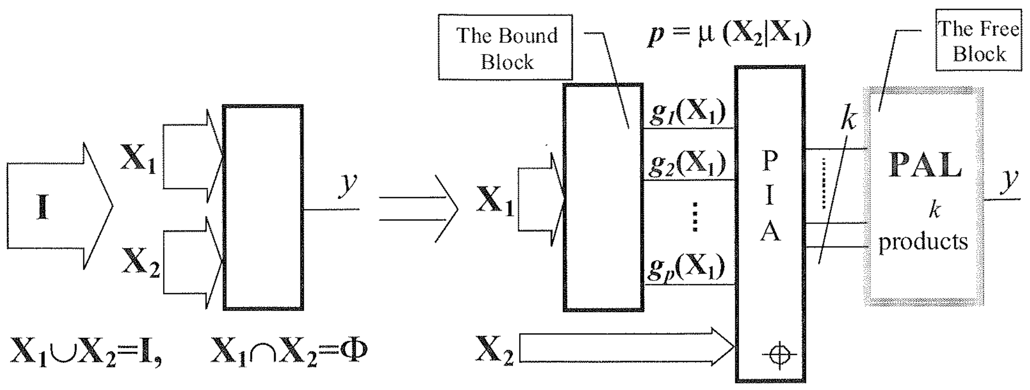


Fig. 9. The concept of Row Decomposition

stage of the appropriate decomposition search is evaluation of row multiplicities for subsequently analysed input variables partitionings. The algorithm for determining the row multiplicities utilises a new idea of the so-called Incompatibility and Complement Graph [33, 52].

For a given variable partitioning, a function $f(\mathbf{X}_2,\mathbf{X}_1)$ can be presented in the following form:

$$f(\mathbf{X}_2,\mathbf{X}_1) = H_0(\mathbf{X}_2) + \sum_{i=1}^{\mu(\mathbf{X}_2|\mathbf{X}_1)} \left[H_i(\mathbf{X}_2)g_i(\mathbf{X}_1) + H_i^{|}(\mathbf{X}_2)\overline{g_i(\mathbf{X}_1)}\right] \tag{6}$$

where $g_i(\mathrm{X}_1)$ denote functions, which describe row patterns.

The number of products necessary to implement the free block is equal to the number of implicants of the function represented in the above form.

The row multiplicity μ $(\mathbf{X}_2\,|\mathbf{X}_1)$, determined as a result of the Row Incompatibility and Complement Graph colouring, is related to the minimum number of outputs of the bound block. If the number of implicants required for implementing each of the g_i $(\mathbf{X}_1)$ functions is less or equal to k, then the number of PAL based logic blocks necessary for implementing the bound block amounts to $\delta_z = \mu$ $(\mathbf{X}_2|\,\mathbf{X}_1)$. This observation suggested a simplified method of the δ_z coefficient (Eq. 5) minimization. A heuristic rule of restricting the search area to solutions, for which μ $(\mathbf{X}_2|\,\mathbf{X}_1) = \min$, is included in the algorithm. More details are presented in [53-55].

The synthesis method described here was thoroughly examined in comparison to the Column Decomposition model, presented in the previous chapter. The experiments show that:

- The Row Decomposition is better with respect to the number of logic levels, than the algorithms based on the Column Decomposition;
- The Row Decomposition can be useful in cases, for which reducing the chip area is of main concern, but without degrading the chip dynamic properties significantly;

- If reducing the number of logic levels is an important factor in the synthesis, the Row Decomposition algorithm is especially efficient for structures consisting of PAL-based blocks containing 2^i (a power of 2) product terms.

4.3. BINARY DECISION DIAGRAMS IN DECOMPOSITION DEDICATED FOR CPLD-S

Complexity of most of decomposition algorithms grows dramatically with the number of function arguments. This impedes their implementation in commercially available EDA tools. Great hopes for prevailing those obstacles are recently pinned on Binary Decision Diagrams (BDD-s).

The classical decomposition model, in which column patterns are analysed, is equivalent to an appropriate BDD transformation, based on horizontal diagram partitioning (cutting). It can be proved that the number of nodes that are left below the cut line, but directly connected with some nodes above the cut line, is equal to column multiplicity of the partition matrix [56]. Fig. 10 presents an example of decomposition for a function represented by a Binary Decision Diagram. The edges from the upper part of the diagram (above the cut line), reach directly to two nodes below the cut line. This means that the bound block, which will implement the function represented by the upper part of the graph, should have one output.

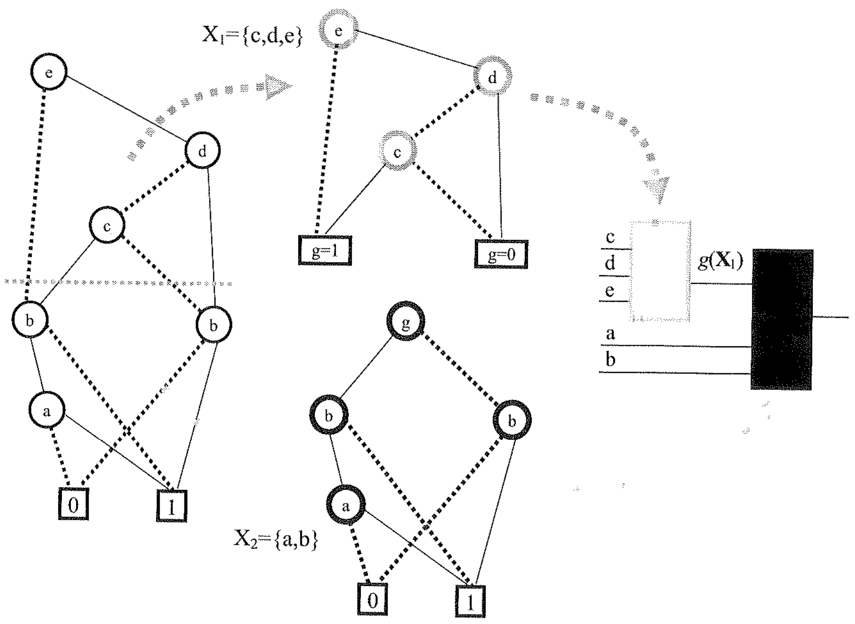


Fig. 10. An example of decomposition utilising Reduced Ordered BDD-s (ROBDD-s)

The presented approach, described e. g. in [56], became the base for a number of decomposition methods dedicated for LUT based FPGA-s.

In the case of CPLD structures, the main constraint regards the number of product terms, while the number of inputs is relatively high. So another approach to the graph partitioning is needed. One of such algorithms was presented in [57]. The essence of the method consists in searching for such a subdiagram in the BDD, that can be implemented in a PAL block containing a predefined number of product terms. For determining the number of products required to implement a subfunction, the number of implicants associated with the subdiagram needs to be found. The main criterion for a decomposition attempt is the minimal number of implicants Δ_f, necessary to implement the function before the decomposition. If Δ_f is known, also the number of PAL-based blocks required can be evaluated (Eg. 1). Decomposition of the diagram is justified only if $\Delta_f \geq 2k$. If the condition is not met, the classical approach is used.

The example of function decomposition with the use of the subdiagram search is presented in Fig. 11. It is assumed, that the number of terms available in PAL blocks used for implementation amounts to 3. The presented diagram uses the so-called attributed edges. If an edge ends up with dot, this means that the expression represented by the pointed subtree has to be inverted. The circuit obtained as the result of the decomposition is presented in Fig. 12.

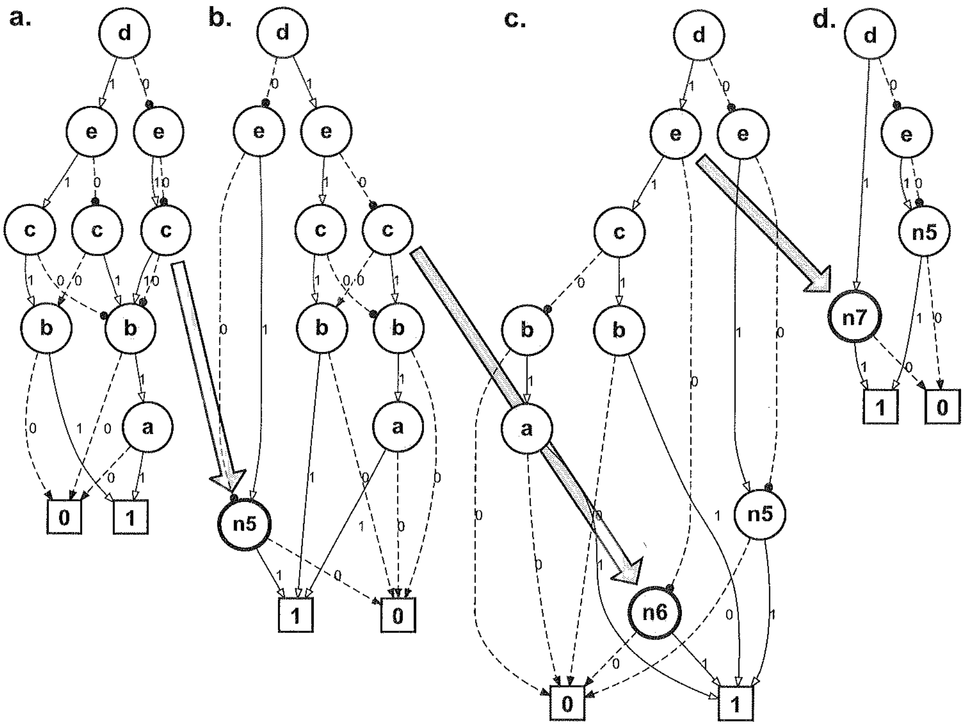


Fig. 11. Transformations of a BDD corresponding to a function decomposition dedicated for PAL-based circuits

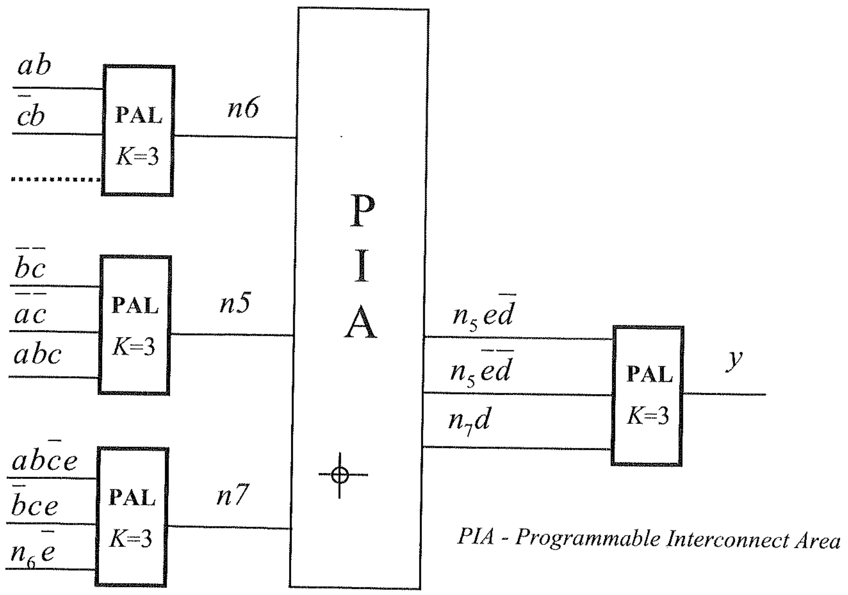
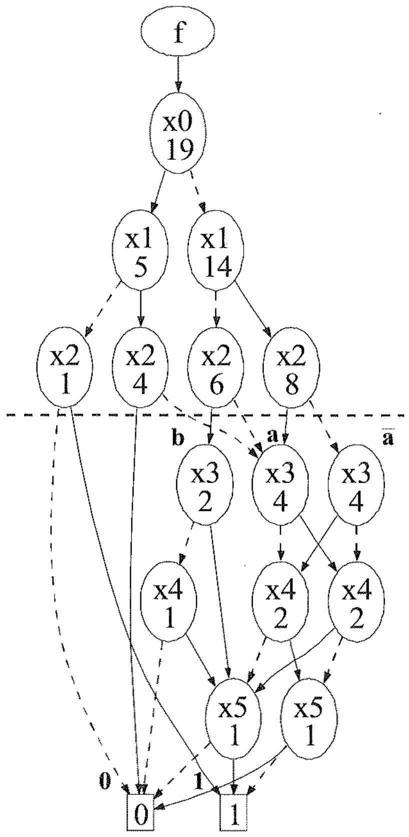


Fig. 12. The synthesized structure obtained as the result of decomposition

The next method of function decomposition, utilising Binary Decision Diagrams, like the first one, is oriented towards fitting the partitioning of the primary circuit into the PAL-based logic block structure. It can be considered as an extension of the Row Decomposition concept discussed in chapter 4.2. In the process of search for the best partitioning of the set of input variables, an attempt is made to add as many variables to the free set, as possible, without violating the condition of feasibility of the free block in one PAL block. The partitioning of the set of variables can represented in a ROBDD as a horizontal cut. The variables associated with nodes lying above the cut line correspond to the free set, and the variables below the cut line – the bound set.

In this approach a ROBDD with edge complement attributes is used to evaluate the row multiplicity $\mu(X_2|X_1)$, instead of the Row Incompatibility and Complement Graph (see p. 4.2). The row multiplicity can be efficiently computed by counting the number of nodes cut off. Different partitionings are obtained by changing order of variables in the ROBDD, and by moving the level of the cut line.

The decomposition algorithm consists of several phases. During each phase cardinality of free set, which corresponds to the current cut level, is determined. A variable partitioning is searched, which satisfies the condition of feasibility of implementation of the free block in one PAL block, and for which the bound block has the lowest number of outputs. If a solution is found for given cut level, the cut line is lowered.



$$
\begin{aligned}
&\mathbf{a} \rightarrow g_1(x_3, x_4, x_5)\\
&\mathbf{b} \rightarrow g_2(x_3, x_4, x_5)\\
f_h &= x_0\bar{x}_1 x_2\\
&+ \bar{x}_0\bar{x}_1\bar{x}_2\, g_1\\
&+ \bar{x}_0 x_1 x_2\, g_1\\
&+ x_0 x_1 \bar{x}_2\, g_1\\
&+ \bar{x}_0 x_1 \bar{x}_2\, \bar{g}_1\\
&+ \bar{x}_0 \bar{x}_1 x_2\, g_2
\end{aligned}
$$

Fig. 13. Function diagram with annotated path number

In the Row Decomposition model discussed above, only disjunctive partitionings are considered. In order to reduce the length/depth of the critical path, also non-disjunctive partitionings can be employed. Let's consider the function defined by the diagram presented in Fig 14.

In the first step of the non-disjunctive decomposition, a possibly good disjunctive partitioning is found. Let's assume, that we are going to implement the function using PAL blocks containing 3 product terms. For a given variable ordering only x_0, can be included into the free set. In this case, the free block is described by the following formula:

$$f = x_0 g_2(x_1, x_2, x_3, x_4) + \bar{x}_0 g_1(x_1, x_2, x_3, x_4) \tag{7}$$

tionings
disjunc-
diagram

junctive
on using
, can be
llowing

(7)

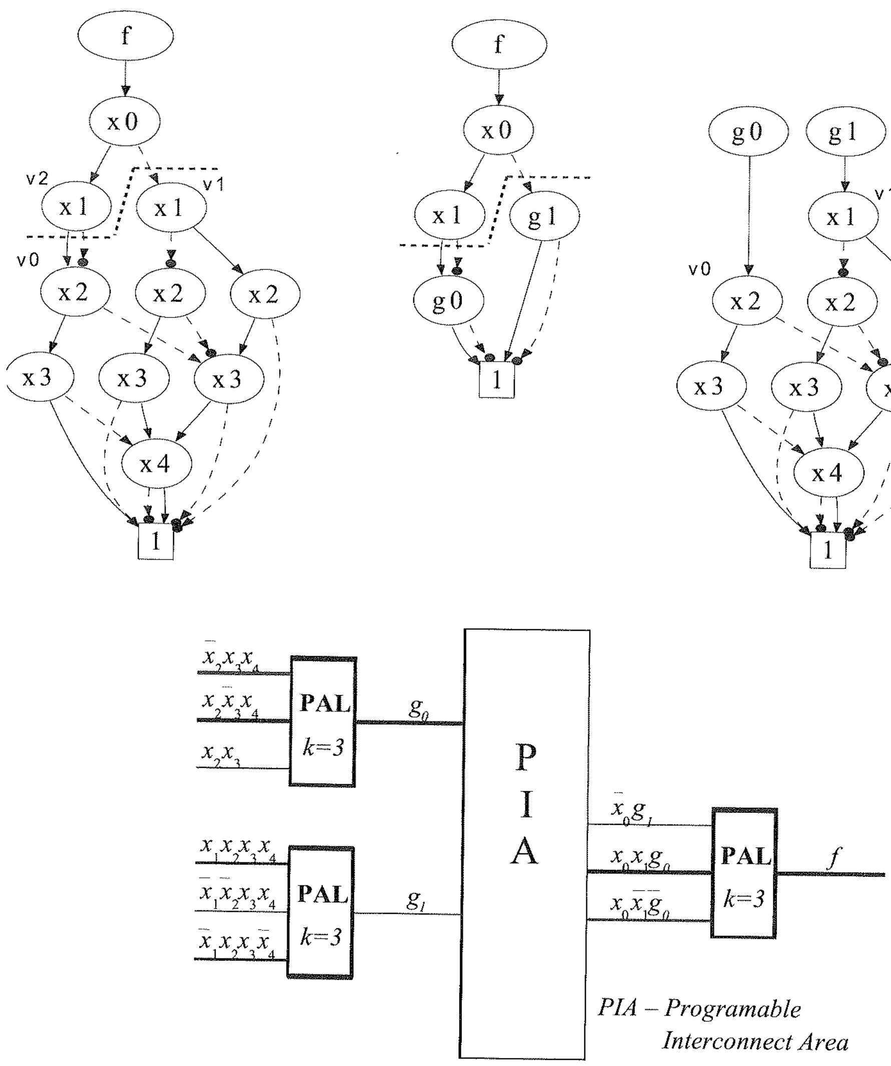


Fig. 14. The diagram cut method corresponding to a non-disjunctive decomposition, and the resulting circuit structure

The free block is implemented using two product terms. Function g_2 describes a diagram rooted by node $\mathbf{v}_2$, and g_1 – by $\mathbf{v}_1$ respectively. Including the variable x_1 into the disjunctive free set increases implementation requirements to 4 product terms. This exceeds limit of 3 terms in a PAL block. Function g_1 is realized by one PAL block, and g_2 by two blocks, respectively. Finally, using the disjunctive decomposition, the circuit can be implemented with the use of 4 blocks, and the maximum path length is 3 levels.

The non-disjunctive decomposition allows to include the variable x_1 both into the free and the bound sets. The free block is described by the formula

$f = x_0 x_1 g_0 + x_0 \bar{x}_1 \bar{g}_0 + \bar{x}_0 g_1$, and utilizes 3 product terms. The whole circuit is built of 3 PAL-based logic blocks, and the maximum path length is reduced to 2 levels (Fig. 14).

According to the observations presented above, the basic decomposition algorithm, was modified [58]. After finding a proper disjunctive partitioning, the procedure tries to add a child node, located below cut line, to the free set. In the presented example, $\mathbf{v}_0$ is chosen as a child of $\mathbf{v}_1$. The node is accepted, if the resulting implementation of the free block fits into a single PAL-based logic block (Fig.14).

5. COMPLEX STRATEGY OF LOGIC SYNTHESIS FOR PAL-BASED CPLD-S

Synthesis algorithms presented in previous chapters were carefully and extensively verified and tested. The test methodology was based on synthesizing benchmark circuits [59]. The results obtained were compared against results published by other authors, and against results generated by using commercial tools. A number of program modules, implementing the algorithms presented above, were developed. They make up together the PALDec ("*PAL Decomposition*") synthesis system. The number of experiments carried out in recent few years reaches tens of hundreds. The results obtained were published in [28-36, 38-39, 49-51, 53-55, 58, 60-62,]. Analysis of the results makes it possible to compare the algorithms with respect to logic resources consumed, and the number of logic levels.

Without doubt we can appoint the decomposition methods that utilise tri-state buffers, as the best for obtaining the shortest paths (and propagation delays). The one-level method with tri-state buffers (1_TBW) gives the best results [28, 33] in terms of propagation delays. Finding a solution with this method is however not guaranteed. If a solution cannot be found using the one-level method, the p-level method with tri-state buffers (p_TBW) can be applied instead [32-33]. This method is able to find a solution providing the smallest possible number of logic levels. The synthesis strategy combining both methods (1p_TPW), allows to obtain implementations with the smallest possible number of logic levels.

The 1p_TPW strategy consumes however significantly more logic resources than the other presented methods. The average increase in logic blocks usage is around 1.5 times, in comparison to the classical approach [33]. A unique feature of the proposed p-level function implementation is, that the number of logic levels p can be set as a parameter for the method. It was observed, that solutions, for which the number of logic levels allowed is higher, are found faster, and they consume less logic resources. This way it is possible to control propagation delays vs. resources usage in the obtained solutions. Simplicity of the methods (1_TBW, p_TBW) enables them to be successfully used in complex synthesis strategies dedicated for any PAL-based devices featuring tri-state output buffers.

The greatest advantages of the multiple-output function synthesis methods utilising the Graph of Outputs are simplicity, and short computation time. By combining the method basing on analysis of the Graph of Outputs, with the classical function implementation, a several percent reduction in the number of logic blocks used can be achieved. This is however paid with a similar increase in the number of logic levels.

In most cases, independently from the size of the PAL logic blocks, applying the methods utilising Graphs of Outputs leads to expansion of the number of logic levels, if compared to the classical method. This drawback can however be compensated in circuits that contain tri-state output buffers. In the methods referenced above, the groups of multiple-output implicants extracted in subsequent steps of synthesis, were implemented using the classical method. There is nothing to prevent the groups of implicants from being synthesised with one of methods exploiting tri-state buffers (1_TBW or p_TBW) instead.

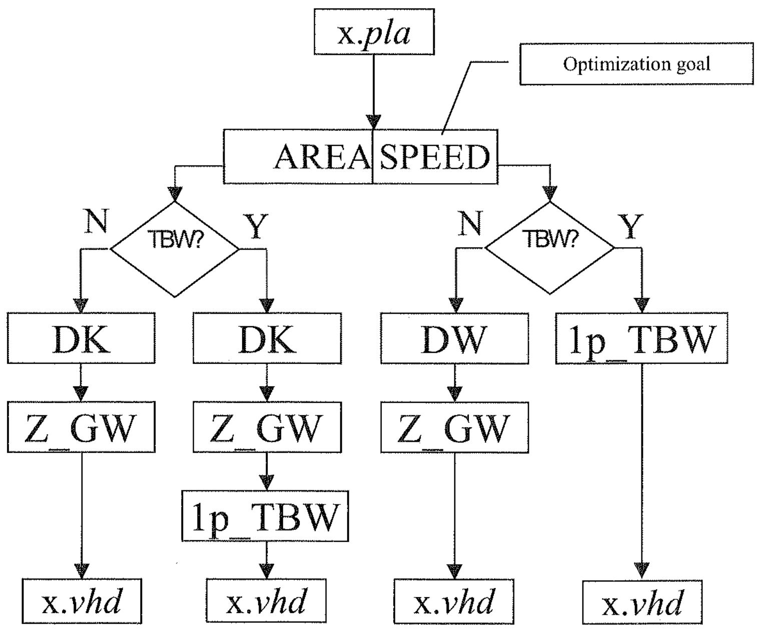
The most efficient, in respect of logic resources used, synthesis methods are the methods employing decomposition. The best results were obtained for the method based on the Column Decomposition of multiple-output functions (DK) [33, 45]. Using this method, a reduction of the number of logic blocks between 20% and 30% can be achieved, in reference to the classical implementation [33].

Elements of adapting the decomposition model to structures of the PAL logic blocks are included in the Row Decomposition method (DW) [33, 53, 61]. The results obtained using the algorithms based on Row Decomposition were slightly worse with respect to the number of logic blocks, from the results generated by the methods based on Column Decomposition. They were however significantly better with respect to the number of logic levels.

Computational complexity of the algorithms based on the Row and Column Decomposition limits their usage for functions with large number of arguments. In such cases it is however possible to apply the methods basing on Binary Decision Diagrams. The program module dekBDD, developed as a result of the research work [58, 63], is capable of performing the whole synthesis process for a function with several tens of arguments within few seconds (eg. apex5: 117 inputs, 88 outputs – synthesis time: 6,3s).

The final goal of the research works carried out in the Institute of Electronics of the Silesian University of Technology, was to develop efficient synthesis algorithms for PAL based CPLD structures. To achieve the goal, the new methods were tested in a large number of experiments, and the results were analysed. In the experiments the commonly accepted set of test circuits [59] was used as the means to verify quality of synthesis tools. The test circuits were synthesised using the programs developed by the research team (PALDec), and the results were compared with the results obtained from other tools, used as the reference. The set of reference tools was wide, it included commercial programs (MAX+Plus, Synplicity, Abel, MACHxl, ispDesignEXPERT, Warp, Quartus), available academic tools (ASYL, PLADE) [64-65], and the classical synthesis method (Espresso). As the conclusion, synthesis strategies can be proposed, in which the synthesised circuit can be optimised for different goals, and various architecture-specific features.

The synthesis strategy proposed for PAL-based CPLD-s without tri-state buffers starts with the Column Decomposition (DK). The subcircuits obtained as the result are then optimized by one of the methods utilizing the Graph of Outputs (Z_GW). Improvement in propagation delays can be gained by the use of the TBW method (1p_TBW). If the propagation delay is a critical parameter, better results can be obtained by using the Row Decomposition method (DW). The circuit after partitioning is further optimized by the Graph of Outputs method (Z_GW). If using tri-state buffers is permissible, the best results are obtained with the 1_TBW method, which generates one-level structures. If the method is unable to find a solution, the p_TBW method can be used instead. Finally the two algorithms were combined, and the 1p_TBW strategy was developed, which produces a solution with minimal number of levels. The proposed algorithm of selecting synthesis strategies for different optimisation goals, and architecture-specific features is presented in Fig. 15.



TBW – three-state output buffer;
DK – column decomposition;
DW – row decomposition;
Z_GW – multiple-output synthesis using Graph of Outputs;
1p_TBW – strategy of term expansion using three-state buffers

Fig. 15. The proposed algorithm of selecting synthesis strategies for PAL-based CPLD-s

The research works are continued, and now the main objective is to develop strategies including methods based on BDD partitioning. The dekBDD module [58] is going to be incorporated into the PALDec system. Some early results are presented in [63].

6. INTERFACE TO VENDOR TOOLS AND EXPERIMENTAL RESULTS

If thousands of experiments are to be carried out, interfacing prototype software to tools supplied by PLD vendors becomes an important issue. Software tools developed by companies or institutions independent from PLD vendors are capable of performing only the logic synthesis stage. Then the design has to be transferred to a vendor-specific system for completing the implementation stage. This regards also academic software, developed by research teams.

The main problem in porting a design to a vendor-specific system is to find an appropriate intermediate format for the design data exchange. Commercial vendor-independent systems (eg. Synplify, Leonardo Spectrum, Precision RTL) use low level netlists for this purpose. This approach is secure, because there is a little chance, that the low level structure will be interfered by implementation tools. The method is however not universal, because low level netlists contain much vendor-specific, and architecture-specific information. Using this approach requires thus to equip the synthesis software with procedures or plugins responsible for converting formats, and preparing data specific for the implementation tools. This is acceptable for commercial companies, but difficult for academic research teams, as it requires much "scientifically worthless" extra job.

It was thus desirable to find out alternative formats for the data exchange, possibly more universal, and using a higher level of abstraction. Here using a Hardware Description Language (HDL) seems to be the most obvious, and natural choice. Choosing the right abstraction level for the intermediate format is an important task, because vendor implementation software can change and "destroy" logical structures generated by synthesis tools.

Behavioral HDL description seems to be the design specification format most preferred for design entry, nowadays. Because of its high abstraction level it allows the designer to concentrate on proper description of the desired functionality. As a textual format, following the standard of the chosen language, it is universal and portable between technologies and software tools.

A number of experiments were carried out to examine various synthesis tools, and, in particular, the effects of selecting different data exchange formats, on quality of results. The tools were tested using the standard benchmarks [59]. The test circuits were implemented in CPLD structures.

It turned out that, if behavioural description was used as the entry format, quality of the solutions was not good. High abstraction level in behavioural modelling gives much freedom to the software. Logical structures can easily be "spoiled" by vendor implementation programs. During the experiments it turned out, that it is possible to propose as the intermediate format in a style of VHDL description, lying at a lower level of abstraction, than behavioural modelling, but still portable between software tools, and comprehensdible to a human. The proposed style of VHDL modelling resembles the dataflow description commonly known in the literature. More details are reported in [33, 58, 66-67].

The prototype computer program PALDec, developed during the research work, enables a convenient interaction with vendor-specific tools. The program reads input data in the Berkeley format (pla). Output data are generated as an appropriate description in a hardware description language.

Results of numerous experiments, and precision comparison of particular methods, are reported in [28-36, 38-39, 49-51, 53-55, 58, 60-62]. In this paper only sample results, for the popular benchmark rd84, will be presented. The results were obtained for the simplest PAL-based CPLD structure available – the "Classic" family from Altera. The basic building block of the Altera "Classic" devices is a macrocell containing an 8-terms PAL structure with a tri-state output buffer controlled by an extra term. Choosing those structures, which are quite old, is justified by the concern to preserve the results from being blurred by extra CPLD functionality, like XOR gates, programmable expanders etc. As a consequence, it was necessary to use an outdated synthesis tool - MAX+PLUS II. It is worth to note that for CPLD-s Quartus II gives results quite similar to those obtained in MAX+ PLUS II.

The synthesis of the rd84 benchmark was carried out using the following four methods:

- description in the pla format, synthesis and implementation in the vendor tool (**M_KL**),
- description in the pla format, synthesis in PALDec (**DK+Z_GW** strategy), implementation in the vendor tool (**DK+Z_GW**),
- description in the pla format, synthesis in PALDec (**DW+Z_GW** strategy), implementation in the vendor tool (**DW+Z_GW**),
- description in the pla format, synthesis in PALDec (**1pTBW**), implementation in the vendor tool (**1pTBW**).

The symbols accompanying the methods listed above (**M_KL**, **DK+Z_GW**, etc.) correspond to labels in the charts presented in Fig. 16. The figure presents in a synthetic form a comparison of the results.

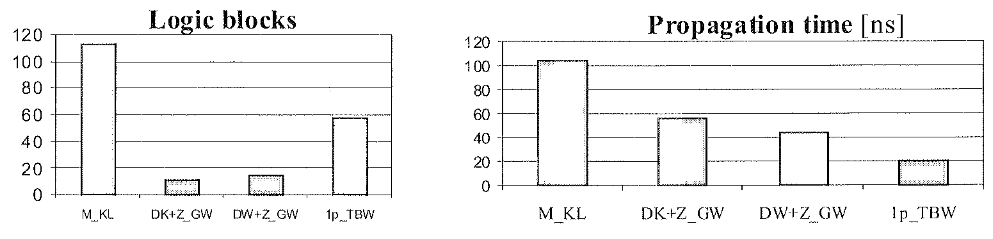


Fig. 16. A comparison of synthesis results obtained for the rd84 benchmark, and different algorithms

The comparison was presented both for the resources used, and propagation delays. As the charts show, in all of the cases synthesis in PALDec gives significantly better results, than the commercial tool. With respect to the number of logic blocks used, the Column Decomposition combined with elements of the optimization based on the Graph of Output (DK+Z_GW) is the most efficient strategy.

The timing delays were determined by the static timing analysis module of MAX+ PLUS II. In respect of propagation time, the unrivalled strategy is 1p_TBW, which uses tri-state buffers.

Similar results were obtained also for other vendor-specific tools, both for simple and complex CPLD-s [33].

7. CONCLUSIONS

The paper presents several different synthesis methods dedicated for PAL-based CPLD-s. The proposed methods are an alternative to the classical approach, based on two-level minimization of individual single-output functions. Subsequent steps of the synthesis process are adapted to logical resources of PAL-based CPLD-s. Adjusting elements of the synthesis process to logical resources characteristic for a PAL logic block allows for significant improvement of synthesis effectiveness in relation to the classical approach.

Decomposition models presented in the paper are adapted to specific requirements of PAL-based CPLD structures. The proposed partitioning of the design into PAL-based logic blocks, allows for general improvement of synthesis methods for commonly available CPLD structures. The presented approach is not limited by final optimization, which takes into account specific features of a target structure.

Authors do not claim, that the presented methods can instantly be used by commercially available design systems. On the other hand, the results obtained show possible ways of improving synthesis quality. Significant area reduction can be expected after embedding in synthesis algorithms decomposition methods adjusted for CPLD-characteristic logic resources. Unfortunately decomposition methods are very computationally complex, and time consuming. Faster algorithms can hopefully be developed for example by using BDD-s.

Results of the experiments presented in the paper prove, that the synthesis methods based on decomposition are especially attractive for CPLD structures consisting of small PAL-based blocks. In this case the solutions were minimal with respect to the number of PAL-based blocks used. If the synthesis process is to be optimized for speed, the methods utilizing three-state buffers should be applied. Tri-state buffers are commonly available in modern CPLD devices, and the synthesis algorithms are relatively simple and robust.

Results of research work conducted for many years show without doubt, that it is possible to improve significantly quality of commercial synthesis tools. It is often possible to obtain structures occupying much less resources, or much faster operating.

8. REFERENCES

1. S. Chang, M. Marek-Sadowska, T. Hwang: *Technology Mapping for TLU FPGA's Based on Decomposition of Binary Decision Diagrams*. IEEE Transactions on Computer-Aided Design, Vol.15, No.10, October 1996, pp. 1226-1235
2. A. Dzikowski, E. Hrynkiewicz: *Złożona dekompozycja obszarowa zespołu funkcji logicznych z wykorzystaniem diagramów ROBDD*. Krajowa Konferencja Elektroniki KKE'03, Kołobrzeg (Poland), Czerwiec 2003, tom II, ss. 393 398
3. Y. Lai, K. R. Pan, M. Pedram: *OBDD-Based Function Decomposition: Algorithms and Implementation*. IEEE Transactions on Computer-Aided Design, Vol.15, No.8, August 1996, pp. 977-990
4. M. Rawski, H. Selvaraj, T. Łuba, P. Szotkowski: *Application of symbolic functional decomposition concept in FSM implementation targeting FPGA devices*. Sixth International Conference on Computational Intelligence and Multimedia Applications, 2005, Aug. 2005, pp. 153-158.
5. C. Scholl: *Functional Decomposition with Application to FPGA Synthesis*. Kluwer Academic Publishers, Boston, 2001
6. H. Selveraj, T. Łuba, M. Nowicka, B. Bignall: *Multiple-valued decomposition and its applications in data compression and technology mapping*. Proceedings of ICCIMA'97, Gold Coast (Australia), 1997, pp. 42-48
7. C. Yang, M. Ciesielski: *BDS: A BDD-Based Logic Optimization System*. IEEE Transactions on CAD of Integrated Circuits and Systems, Vol.21, No.7, July 2002, pp. 866-876.
8. M. Adamski, A. Barkalov: *Architectural and Sequential Synthesis of Digital Devices*. University of Zielona Góra Press, 2006.
9. J. H. Anderson, S. D. Brown: *Technology mapping for large complex PLDs*. Proceedings of Design Automation Conference, DAC'98, January 1998, pp. 698-703.
10. A. Barkalov, L. Titarenko, S. Chmielewski: *Reduction in the number of PAL macrocells in the circuit of a Moore FSM*. International Journal of Applied Mathematics and Computer Science, Number 4, Volume 17, 2007.
11. S. L. Chen, T. T. Hwang, C. L. Liu: *A technology mapping algorithm for CPLD architectures*. IEEE International Conference on Field Programmable Technology, Hong Kong, December 2002, pp. 204-210.
12. S. Deniziak, K. Sapiecha: *An Efficient Algorithm of Perfect State Encoding for CPLD Based Systems*. IEEE Workshop on Design and Diagnostic of Electronic Circuits and Systems, DDECS'1998, Szczyrk (Poland) 1998, pp. 47-53.
13. J. Kim, S. Byun, H. Kim: *Development of technology mapping algorithm for CPLD under time constraint*. 6th International Conference on VLSI and CAD, ICVC '99, 1999, pp. 411-414.
14. H-S. Kim, J-J. Kim, Ch-H. Lin: *An efficient CPLD technology mapping under the time constraint*. Proceedings of the 12th International Conference on Microelectronics, ICM 2000, 2000, pp.265 -268.
15. J-J. Kim, H-S. Kim, Ch-H. Lin: *A new technology mapping for CPLD under the time constraint*. Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC'01, 2001, pp. 235-238.
16. A. Kaviani, S. Brown: *Technology mapping issues for an FPGA with lookup tables and PLA-like blocks*. Proceedings of the 2000 ACM/SIGDA Eighth International Symposium on Field Programmable Gate Arrays, Monterey, 2000, pp. 60-66.
17. V. Salauyou, A. Klimowicz, T. Grześ, T. Dimitrova-Grekow, I. Bulatowa: *Badania efektywności metod syntezy automatów skończonych zaimplementowanych w pakiecie ZUBR*. Miesięcznik Naukowo-Techniczny „Pomiary Automatyka Kontrola" nr 6 bis, 2006, ss. 44-46.
18. W. Sołowjew: *Synthesis of sequential circuits on programmable logic devices based on new models of finite state machines*. Proceedings of the EUROMICRO Conference on Digital Systems Design, 2001, pp. 170-173.
19. J. L. Kouloheris, A. E. Gamal: *FPGA performance versus cell granularity*. Proceedings of the IEEE Custom Integrated Circuits Conference, May 1991, pp. 6.2/1 -6.2/4

20. J. L. Kouloheris, A. E. Gamal: *PLA-based FPGA Area Versus Cell C+ Granularity*. Proceedings of the IEEE Custom Integrated Circuits Conference, May 1992, pp. 4.3.1-4.3.4.
21. K. Yan: *Logic synthesis for CPLDs and FPGAs with PLA-style logic blocks*. Fourteenth International Conference on VLSI Design, 2001, pp. 291-297.
22. K. Yan: *Practical logic synthesis for CPLDs and FPGAs with PLA-style logic blocks*. Proceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC 2001, 2001, pp. 231-234.
23. K. C. Chen, S. Muroga: *Input assignment algorithm for decoded-PLAs with multi-input decoders*. IEEE International Conference on Computer-Aided Design, ICCAD'88, Digest of Technical Papers, November 1988, pp. 474-477.
24. M. J. Ciesielski, S. Yang: *PLADE: A two-stage PLA decomposition*. IEEE Trans. on Computer--Aided Design, Vol.11, No.8, 1992, pp. 943 954.
25. S. Devadas, A. R. Wang, A. R. Newton, A. Sangiovanni-Vincentelli: *Boolean Decomposition of Programmable Logic Arrays*. IEEE Custom Integrated Circuits Conference, May 1988, pp. 2.5.1 -2.5.5.
26. S. Devadas, A. R. Wang, A. R. Newton, A. Sangiovanni-Vincentelli: *Boolean decomposition in multi-level logic optimization*. Digest of Technical Papers, IEEE International Conference on Computer-Aided Design, ICCAD-88, Nov 1988, pp. 290-293.
27. C. Yang, M. J. Ciesielski: *PLA decomposition with generalized decoders*. IEEE International Conference on Computer-Aided Design, ICCAD-89, Nov 1989, pp. 312-315.
28. D. Kania: *Two-level logic synthesis on PALs*. Electronics Letters, 1999, Vol.35, No. 11, pp. 879-880.
29. D. Kania: *Two-level logic synthesis on PAL-based CPLD and FPGA using decomposition*. Proceedings of 25-th Euromicro Conference, IEEE Computer Society Press, Milan (Italy), 1999, pp. 278-281.
30. D. Kania: *Multi-level logic synthesis on PAL-based devices with three state output buffers*. Kwartalnik Elektroniki i Telekomunikacji, 2000, 46, z.1, pp. 81-90.
31. D. Kania: *Logic synthesis on PAL-based devices containing output buffers*. Kwartalnik Elektroniki i Telekomunikacji, 2002, 48, z.1, pp. 53-66
32. D. Kania: *A p-stage Logic Synthesis for PAL-based Devices*. Kwartalnik Elektroniki i Telekomunikacji, 2004, 50, z.1, pp. 65-86.
33. D. Kania: *The Logic Synthesis for the PAL-based Complex Programmable Logic Devices*, Zeszyty Naukowe Politechniki Śląskiej, Nr 1619, Wydawnictwo Politechniki Śląskiej, Gliwice 2004.
34. D. Kania: *A New Approach to Logic Synthesis of Multi-Output Boolean Functions on PAL-based CPLDs*. Proceedings of the ACM Great Lakes Symposium on VLSI, GLSVLSI'07 Stressa – Lago Maggiore (Italy), March 2007, pp.152-155.
35. D. Kania: *Method for Efficient Implementation of Multiple-Output Function in PAL-based Devices*. Kwartalnik Elektroniki i Telekomunikacji, 1999, 45, z. 3-4, pp. 433-444.
36. D. Kania: *A technology mapping algorithm for MACH devices*. Kwartalnik Elektroniki i Telekomunikacji, 2001, 47, z. 1, pp. 65-74.
37. D. Kania: *Improved Technology Mapping for PAL-based Devices Using a New Approach to Multi-Output Boolean Functions*. DATE 02, IEEE Computer Society, Los Alamitos, 2002, p.1087.
38. D. Kania: *Logic Synthesis of Multi-Output Functions for PAL-based CPLDs*. IEEE International Conference on Field-Programmable Technology, Hong Kong, December 2002, pp. 429-432.
39. D. Kania: *An Efficient Approach to Synthesis of Multi-Output Boolean Functions on PAL-based Devices*. IEE Proceedings on Computer and Digital Techniques, Vol. 150, No. 3, May 2003, pp.143-149.
40. S. Devadas, A. R. Newton: *Exact Algorithms for Output Encoding, State Assignment, and Four-Level Boolean Minimization*. IEEE Transactions on Computer-Aided Design, Vol. 10, No. 1, January 1991.
41. D. Kania: *An Efficient Algorithm for Output Coding in PAL-based CPLDs*. International Journal of Engineering, Vol.15, No.4, November 2002, pp. 325-328.
42. G. Micheli: *Synthesis and optimization of digital circuits*. McGrew-Hill, Inc., 1994.

43. Ch. J. S h i, J. A. B r z o z o w s k i: *An Efficient Algorithm for Constrained Encoding and its Applications*. IEEE Trans. on CAD, Vol. 12, No.12, December 1993, pp.1813-1826.
44. H. A. C u r t i s: *The Design of switching Circuits. D.van Nostrand Company*, Inc., Princeton, New Jersey, Toronto, New York, 1962.
45. D. K a n i a: *Method for calculation of column multiplicity dedicated for CPLDs*. Archiwum Informatyki Teoretycznej i Stosowanej, Tom 17, z.1, 2005, pp. 65-76.
46. D. K a n i a: *Coding capacity of programmable transcoder*. Kwartalnik Elektroniki i Telekomunikacji, 1998, 44, z.2, pp. 193-204.
47. D. K a n i a: *Coding Capacity of PAL-based Programmable Transcoder with Uneven Number Terms per Output*. Kwartalnik Elektroniki i Telekomunikacji, 1999, 45, z.1, pp.73-84.
48. D. K a n i a: *Coding capacity of PAL-based logic blocks included in CPLDs and FPGAs*. IFAC Workshop on Programmable Devices and Systems, PDS 2000, Ostrava, February 2000, Published for the IFAC by PERGAMON, An Imprint of Elsevier Science, 2000, pp. 164-169.
49. D. K a n i a, A. M i l i k, J. K u l i s z: *Decomposition of Multiple-Output Functions for CPLDs*. Proceedings of Euromicro Symposium on Digital System Design, IEEE Computer Society Press, Porto, September, 2005, pp. 442-449.
50. D. K a n i a: *Logic Decomposition for CPLD Synthesis*. IFAC Workshop on Programmable Devices and Systems, PDS 2000, Ostrava, February 2000, Published for the IFAC by PERGAMON, An Imprint of Elsevier Science, 2000, pp. 49-52.
51. D. K a n i a: *Decomposition-based synthesis and its application in PAL-oriented technology mapping*. Proceedings of 26-th Euromicro Conference, IEEE Computer Society Press, Maastricht, 2000, pp. 138-145.
52. D. K a n i a, J. K u l i s z: *The row incompatibility and complement graph – a novel concept of graph for decomposition*. Programmable Devices and Embedded Systems, PDES 2006, Brno, February 2006, pp.169-173.
53. D. K a n i a, J. K u l i s z, A. M i l i k: *A novel method of two-stage decomposition dedicated for PAL-based CPLDs*. Proceedings of Euromicro Symposium on Digital System Design, IEEE Computer Society Press, Porto, September, 2005, pp.114 121.
54. D. K a n i a: *Row decomposition in logic synthesis for CPLDs*. Kwartalnik Elektroniki i Telekomunikacji, Tom 52, z.4, 2006, pp. 825-847.
55. D. K a n i a, J. K u l i s z: *A method of logic synthesis for PAL-based CPLD-s based on two-stage decomposition*. Programmable Devices and Embedded Systems, PDES 2006, Brno, February 2006, pp. 163-168.
56. T. S a s a o: *FPGA Design by Generalized Functional Decomposition in Logic Synthesis and Optimization*. Kluwer Academic Publishers, Boston/London/Dotdrecht, 1993.
57. A. M i l i k, D. K a n i a: *Application of BDD in Logic Synthesis for PAL-based Devices*. Pomiary, Automatyka, Kontrola vol. 53, nr 7, 2007, pp. 118-120.
58. A. O p a r a: *Dekompozycyjne metody syntezy układów kombinacyjnych wykorzystujące binarne diagramy decyzyjne*, Rozprawa doktorska, Politechnika Śląska, 2009.
59. *Collaborative Benchmarking Laboratory*, Department of Computer Science at North Carolina State University, http://www.cbl.ncsu/edu/
60. D. K a n i a: *A technology mapping algorithm for PAL-based devices using multi-output function graphs*. Proceedings of 26-th Euromicro Conference, IEEE Computer Society Press, Maastricht, 2000, pp. 146-153.
61. D. K a n i a, J. K u l i s z: *Logic synthesis for PAL-based CPLD-s based on two-stage decomposition*. The Journal of Systems and Software 80, 2007, pp. 1129-1141.
62. A. O p a r a, D. K a n i a: *Decomposition of multi-output function based on pseudo-MTBDD*. Pomiary, Automatyka, Kontrola vol. 54, nr 8, 2008, pp. 496-498.
63. A. O p a r a, D. K a n i a: *A Novel Non-Disjunctive Method for Decomposition of CPLDs*. Kwartalnik Elektroniki i Telekomunikacji, Vol. 55, No. 1, 2009, pp. 95-111.

64. G. Saucier, P. Sicard, L. Bouchet: *Multi-level synthesis on PAL's*. Proc. European Design Automation Conference, Glasgow, March 1990, pp. 542-546.
65. G. Saucier, P. Sicard, L. Bouchet: *Multi-level synthesis on programmable devices in the ASYL system*. Euro ASIC '90, 1990, pp. 136-141.
66. R. Czerwiński: *The FSMs state assignment for PAL-based matrix programmable structures*. PhD Thesis, Gliwice, 2006.
67. R. Czerwiński, J. Kulisz: *State machine description oriented towards effective usage of vendor-independent synthesis tools*. IFAC Workshop on Programmable Devices and Embedded Systems 2009, PDES'09, February 2009, pp. 27-32

Harc

t
I
c
c
c
c
I
t
a

Co
chine
in dev
(FPGA
used t
using
is dec
logic
Let us
maxin

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 317–333

Hardware reduction for Moore FSM implemented with CPLD

ALEXANDER BARKALOV, LARYSA TITARENKO, SŁAWOMIR CHMIELEWSKI

University of Zielona Góra, Institute of Electrical Engineering
ul. prof. Z. Szafrana 2, 65-516 Zielona Góra
A.Barkalov@iie.uz.zgora.pl; L.Titarenko@iie.uz.zgora.pl; S.Chmielewski@weit.uz.zgora.pl

Received 2009.01.08
Authorized 2009.03.27

A method of combined state assignment is proposed which targets on a decrease in the hardware amount (the number of PAL macrocells) in combinational part of Moore finite-state-machine (FSM). Some peculiarities of Moore FSM such as existence of pseudoequivalent states and dependence of output functions on states as well as a wide fan-in of PAL macrocells are used to optimize the hardware amount. It allows hardware amount decrease without decreasing in performance of the controlled digital system. An example of application of proposed method is given. Some results of experiments based on the probabilistic approach are demonstrated. It is shown that the proposed method always leads to decrease in the hardware amount in comparison with the known methods of Moore FSM synthesis.

Keywords: Moore finite-state-machine, synthesis, graph-scheme of algorithm, CPLD, PAL, macrocell, pseudoequivalent states

1. INTRODUCTION

Control unit of any digital system can be implemented as a Moore finite-state-machine (FSM) [1,2]. Recent achievements in semiconductor technology have resulted in development of such sophisticated VLSI chips as field-programmable logic arrays (FPGA) and complex programmable logic devices (CPLD) [3-6]. Very often CPLD are used to implement complex controllers [2,7]. In CPLD, logic functions are implemented using programmable array logic (PAL) macrocells [5-7]. One of the issues of the day is decrease in the number of PAL macrocells required for implementation of FSM logic circuit [2,7]. A proper state assignment [8] can be used to solve this problem. Let us point out that such characteristics of FSM as cost / area, power consumption, maximum frequency (cycle time) depend significantly on this step outcome. Because of their importance, state assignment methods are continually developed. There are effective methods based on symbolic minimization [9-11], genetic algorithms [12,13] and other heuristics [14,15]. To get a good solution, peculiarities of both FSM model and logic elements in use should be taken into account [2]. The peculiarities of Moore FSM are existence of pseudoequivalent states [14] and dependence of microoperations only on FSM internal states [1]. The peculiarity of CPLD is a wide fan-in of PAL macrocells [15]. It permits to use different sources for representation of a current state code [16,17].

In this article we propose a method of combined state assignment, which is oriented on decrease in the number of terms for both input memory functions and output functions (microoperations) of Moore FSM. The further hardware amount decrease can be reached using transformation of the pseudoequivalent states codes into codes of their classes [14].

2. BACKGROUND OF MOORE FSM

Let Moore FSM be represented by structure table [1] with the columns: a_m is a current state, $a_m \in A$, where $A = \{a_1,...,a_M\}$ is a set of internal states; $K(a_m)$ is a code of state a_m having $R = \lceil log_2 M \rceil$ bits; a_s is a state of transition; $K(a_s)$ is a code of state $a_s \in A$; X_h is a conjunction of some elements of the set of logical conditions X (or their complements) determining the transition $\langle a_m, a_s \rangle$, where $X = \{x_1,...,x_L\}$; Φ_h is a collection of input memory functions from set $\Phi = \{D_1, ..., D_R\}$ which are equal to 1 to switch the automation memory from $K(a_m)$ into $K(a_s)$; $h = 1, \dots, H$ is the number of table line. The column a_m contains collection of microoperations $Y(a_m) \subseteq Y$, which are generated in the state $a_m \in A$, where $Y = \{y_1, ..., y_N\}$. This table determines Moore FSM U_1 shown in Figure 1.

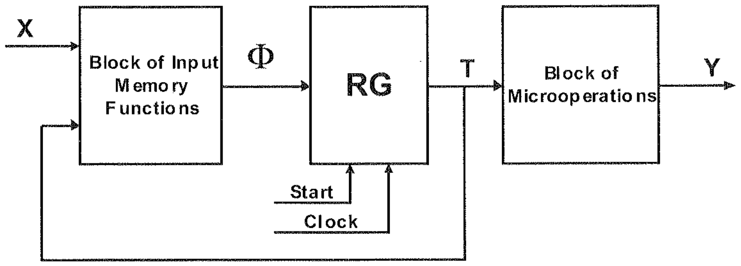


Fig. 1. Structural diagram of Moore FSM U_1

In case of U_1, block of input memory functions (BIMF) generates functions

$$\Phi = \Phi(T, X), \tag{1}$$

and block of microoperations (BMO) generates functions

$$Y = Y(T). \tag{2}$$

Here $T = \{T_1,...,T_R\}$ is a set of state variables used to encode the states $a_m \in A$. Pulse Start is used to load the code of initial state $a_1 \in A$ into register RG. Pulse Clock causes change of RG content.

The hardware amount in logic circuit of FSM U_1 can be decreased using one of the following state assignment approaches [2,14]. In the case of optimal state assignment [2], classes of pseudoequivalent states are represented by generalized intervals of R-dimensional Boolean space. It decreases the number of terms in system (1) up to the number of transitions of equivalent Mealy FSM. Let us remind that states a_s, $a_s \in A$ are pseudoequivalent states if identical inputs result in identical next states for both a_m and a_s. In the case of refined state assignment [14], each microoperation $y_n \in Y$ is represented by a generalized interval of R-dimensional Boolean space. It decreases the number of terms in system (2) up to N. In both cases, such well-known algorithm as ESPRESSO [8] can be used for state assignment. Obviously, it is impossible to apply both methods simultaneously. It means that hardware amount can be decreased either for BIMF, or for BMO. In our article we propose a method allowing hardware decrease for both combinational blocks of Moore FSM U_1.

3. MAIN IDEA OF PROPOSED METHOD

Let $\Pi_A = \{B_1, ..., B_I\}$ be a partition of the set A on the classes of pseudoequivalent states. Let the symbol U_i (Γ_j) stand for the case when a model U_i of Moore FSM is used to interpret a GSA Γ_j. Let the partition $\Pi_A = \{B_1, ..., B_7\}$ be constructed for Moore FSM U_1 (Γ_1), where $B_1 = \{a_1\}$, $B_2 = \{a_5, A_{12}\}$, $B_3 = \{a_{11}, a_{13}, a_{14}\}$, $B_4 = \{a_3, A_6\}$, $B_5 = \{a_2, a_4\}$, $B_6 = \{a_7, a_8\}$, $B_7 = \{a_9, a_{10}\}$. Let us form a system of Boolean equations

$$B_i = \bigvee_{i=1}^{I} C_{mi} A_m \, (i = 1, ..., I), \tag{3}$$

where C_{mi} is a Boolean variable equal to 1 iff $a_m \in B_i$, A_m is a conjunction of state variables corresponding to the code $K(a_m)$. In the case of FSM U_1 (Γ_1), the system (3) is the following one:

$$\begin{aligned} &B_1 = A_1; B_2 = A_5 \vee A_{12}; B_3 = A_{11} \vee A_{13} \vee A_{14}; \\ &B_4 = A_3 \vee A_6; B_5 = A_2 \vee A_4; B_6 = A_7 \vee A_8; B_7 = A_9 \vee A_{10}. \end{aligned} \tag{4}$$

Let us encode states $a_m \in A$ in such a manner that each equation from systems (2) and (3) includes a minimal possible number of terms. The well-known algorithm ESPRESSO [8] can be used to solve this problem. Let $q(B_i)$ be the number of terms in function $B_i \in \Pi_A$.

Let us represent the partition Π_A as $\Pi_A = \Pi_B \cup \Pi_C$, where $B_i \in \Pi_C$ iff $q(B_i) = 1$, otherwise $B_i \in \Pi_B$. Let us encode the classes $B_i \in \Pi_B$ by binary codes $K(B_i)$ using

$$R_1 = \lceil log_2(I_B + 1) \rceil \tag{5}$$

variables $\tau_r \in \tau$, where $I_B = |\Pi_B|$. The unit is added to number I_B (5) to reserve one code indicating that $B_i \notin \Pi_B$. If condition

$$I_B \geq 1 \tag{6}$$

takes place, then GSA Γ_j can be interpreted using the Moore FSM model U_2 (Fig. 2) proposed in this article.

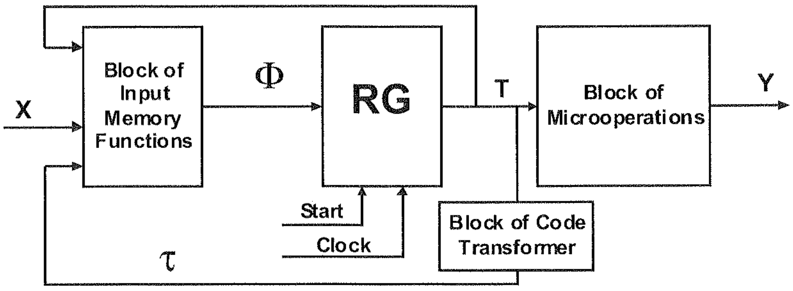


Fig. 2. Structural diagram of Moore FSM U$_2$

In FSM U_2, block BIMF implements functions

$$\Phi = \Phi(T, \tau, X), \tag{7}$$

and block of code transformer (BCT) transforms codes of pseudoequivalent states $a_m \in B_i$ into codes of the classes $K(B_i)$. To execute it, BCT generates functions

$$\tau = \tau(T). \tag{8}$$

Let us name a combined state assignment the method of state encoding, which decreases the hardware amount for blocks BIMF, BMO and BCT. In this case the total number of terms in system Φ is decreased till H_0, which is the number of transitions for equivalent Mealy FSM [14]. If condition (6) does not take place, then structural diagrams for U_1 and U_2 are the same and block BCT is absent.

The proposed method for FSM U_2 synthesis includes the following steps:

- Construction of the partition Π_A for given GSA Γ_j.
- Construction of the system of generalized formulae of transitions for classes of pseudoequivalent states $B_i \in \Pi_A$.
- Construction of systems (2) and (3).
- Combined state assignment for states $a_m \in A$.
- Construction of partitions Π_B and Π_C.
- Encoding of the classes $B_i \in \Pi_B$.

- Construction of transformed structure table.
- Construction of table for block BCT.
- Implementation of Moore FSM U_2 logic circuit using systems (2), (7), (8) and given CPLD chips.

Let us point out that application of proposed method can decrease the number of layers in FSM logic circuit and, therefore, increase the FSM performance. Obviously, if condition (6) is not satisfied, then the steps 6 and 8 of the proposed method are omitted and system (8) is absent.

4. EXAMPLE OF APPLICATION OF PROPOSED METHOD

Let us discuss the case of the Moore FSM $U_2(\Gamma_1)$ synthesis, where the partition Π_A was presented in the previous section. It is clear, that $A = \{a_1, ..., a_{14}\}$, $M = 14$, $R = \lceil log_2 14 \rceil = 4$, $T = \{T_1, ..., T_4\}$, $\Phi = \{D_1, ..., D_4\}$. Let us have the set of microoperations $Y = \{y_1, ..., y_7\}$, and let system (2) be represented as the following one:

$$\begin{aligned}
y_1 &= A_3 \vee A_5 \vee A_6 \vee A_{14};\\
y_2 &= A_2 \vee A_7 \vee A_{11} \vee A_{13};\\
y_3 &= A_2 \vee A_4 \vee A_8 \vee A_9 \vee A_{10};\\
y_4 &= A_4 \vee A_5 \vee A_8 \vee A_{10} \vee A_{12};\\
y_5 &= A_3 \vee A_6 \vee A_8 \vee A_9 \vee A_{14};\\
y_6 &= A_2 \vee A_3 \vee A_4 \vee A_6 \vee A_7;\\
y_7 &= A_2 \vee A_3 \vee A_{10} \vee A_{12} \vee A_{13}.
\end{aligned} \tag{9}$$

Generalized formula of transitions [16] describes the transitions for class $B_i \in \Pi_A$, whereas the formulae of transition [1] describes the transitions for each state $a_m \in B_i$.

Let the following system of generalized formula of transitions be constructed for the GSA Γ_1:

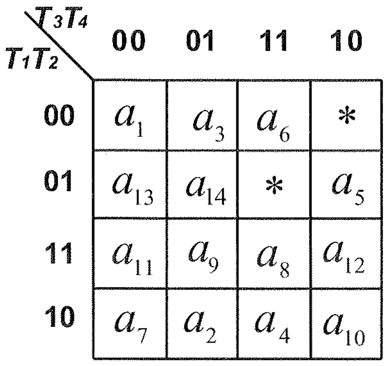
$$\begin{aligned}
&B_1 \rightarrow x_1a_2 \vee \overline{x_1}x_2a_3 \vee \overline{x_1}\,\overline{x_2}a_5;\\
&B_2 \rightarrow x_3a_1 \vee \overline{x_3}a_{10};\\
&B_3 \rightarrow x_2a_4 \vee \overline{x_2}x_3a_7 \vee \overline{x_2}\,\overline{x_3}x_4a_6 \vee \overline{x_2}\,\overline{x_3}\,\overline{x_4}a_{13};\\
&B_4 \rightarrow a_8;\\
&B_5 \rightarrow x_3a_3 \vee \overline{x_3}a_9;\\
&B_6 \rightarrow x_4a_2 \vee \overline{x_4}x_5a_{11} \vee \overline{x_4}\,\overline{x_5}x_6a_{12} \vee \overline{x_4}\,\overline{x_5}\,\overline{x_6}a_{14};\\
&B_7 \rightarrow x_3x_6a_1 \vee x_3\overline{x_6}a_3 \vee \overline{x_3}x_2a_{10} \vee \overline{x_3}\,\overline{x_2}a_{12}.
\end{aligned} \tag{10}$$

Using the algorithm ESPRESSO [8], we can get the following outcome of the combined state assignment (Figure 3).

Let us point out that equations (4) and (9) are used as the input constrains for ESPRESSO [10, 11]. Using the state codes from Karnaugh map (Figure 3), we can get the following systems of equations:

$$\begin{aligned} &y_1 = \overline{T}_1 T_4 \vee \overline{T}_1 T_3; y_2 = T_1 \overline{T}_2 \overline{T}_3 \vee T_2 \overline{T}_3 \overline{T}_4;\\ &y_3 = T_1 T_4 \vee T_1 \overline{T}_2 T_3; y_4 = T_1 T_3 \vee T_3 \overline{T}_4;\\ &y_5 = \overline{T}_1 T_4 \vee T_2 T_4; y_6 = \overline{T}_2 T_4 \vee T_1 \overline{T}_2 \overline{T}_3;\\ &y_7 = \overline{T}_2 \overline{T}_3 T_4 \vee T_1 T_3 \overline{T}_4 \vee \overline{T}_1 T_2 \overline{T}_3 \overline{T}_4. \end{aligned} \tag{11}$$

$$\begin{aligned} &B_1 = \overline{T}_1 \overline{T}_2 \overline{T}_4; B_2 = T_2 T_3 \overline{T}_4;\\ &B_3 = T_2 \overline{T}_3 \overline{T}_4 \vee \overline{T}_1 T_2 \overline{T}_3; B_4 = \overline{T}_1 \overline{T}_2 T_4;\\ &B_5 = T_1 \overline{T}_2 T_4; B_6 = T_1 \overline{T}_2 \overline{T}_3 \overline{T}_4 \vee T_2 T_3 T_4;\\ &B_7 = T_1 T_2 \overline{T}_3 T_4 \vee \overline{T}_2 T_3 \overline{T}_4. \end{aligned} \tag{12}$$



$T_1T_2 \backslash T_3T_4$	00	01	11	10
00	a_1	a_3	a_6	*
01	a_{13}	a_{14}	*	a_5
11	a_{11}	a_9	a_8	a_{12}
10	a_7	a_2	a_4	a_{10}

Fig. 3. Outcome of combined state assignment

Our analysis of the system (12) shows that $\Pi_B = \{B_3, B_6, B_7\}$, $I_B = 3$, $\Pi_C = \{B_1, B_2, B_4, B_5\}$. It means that condition (6) takes place, and FSM U_2 (Γ_1) has the structural diagram shown in Figure 2. According to (5), $R_2 = 2$, $\tau = \{\tau_1, \tau_2\}$. To minimize the number of terms in system (8), let us encode the classes $B_i \in \Pi_B$ in the following way: the more states the class B_i includes the more zeroes its code contains. In our particular case the following codes can be assigned: $K(B_3) = 10$, $K(B_6) = 10$, $K(B_7) = 11$. The code 00 indicates that $B_i \in \Pi_C$.

The transformed structure table of Moore FSM U_2 is constructed using system of generalized formulae of transitions. It includes the columns B_i, $K(B_i)$, a_s, $K(a_s)$, X_h, Φ_h, h, where $H = H_0$. Let the symbol $H_i(\Gamma_j)$ denote the number of lines for transformed structure table for Moore FSM $U_i(\Gamma_j)$. In our example, this table includes H_2 $(\Gamma_1) = 20$ lines (Table 1).

The column $K(B_i)$ contains the codes of classes $B_i \in \Pi_A$ represented as concatenations

$$K(B_i) = K(B_i)_B * K(B_i)_C, \tag{13}$$

where the part $K(B_i)_B$ is represented by variables $\tau_r \in \tau$, the subscript B means that $B_i \in \Pi_B$; the part $K(B_i)_C$ is represented by variables $T_r \in T$, the subscript C means that $B_i \in \Pi_C$; $*$ is a concatenation sign. If $\tau_1 = \tau_2 = 0$, then $B_i \in \Pi_C$, otherwise the part $K(B_i)_C$ is ignored. It is marked by signs "*" in the part $K(B_i)_C$. Codes of classes $B_i \in \Pi_C$ can be found in the following order. For example, Boolean equation for B_1 from system (12) shows that $T_1 = T_2 = T_4 = 0$ and T_3 is absent. Thus, $K(B_1)_C = 0 * 00$. Using the same approach, the following codes can be found: $K(B_2)_C = *110$, $K(B_4)_C = 00 * 1$, $K(B_5)_C = 10 * 1$. These codes are present in the column $K(B_i)$ of Table 1. The codes of states $a_s \in A$ are taken from the Karnaugh map (Figure 3).

Table 1

Transformed structure table of Moore FSM $U_2(\Gamma_1)$

B_i	$K(B_i)$		a_s	$K(a_s)$	X_h	Φ_h	h
	$\tau_1\tau_2$	$T_1T_2T_3T_4$					
B_1	00	0*00	a_2	1001	x_1	D_1D_4	1
			a_3	0001	$/x_1\ x_2$	D_4	2
			a_5	0110	$/x_1\ /x_2$	D_2D_3	3
B_2	00	*110	a_1	0000	x_3	–	4
			a_{10}	1010	$/x_3$	D_1D_3	5
B_3	01	****	a_4	1011	x_2	$D_1D_3\ D_4$	6
			a_7	1000	$/x_2x_3$	D_1	7
			a_6	0011	$/x_2/x_3x_4$	D_3D_4	8
			a_{13}	0100	$/x_2/x_3\ /x_4$	D_2	9
B_4	00	00*1	a_8	1111	1	$D_1D_2D_3D_4$	10
B_5	00	10*1	a_3	0001	x_3	D_4	11
			a_9	1101	$/x_3$	$D_1D_2D_4$	12
B_6	10	****	a_2	1001	x_4	D_1D_4	13
			a_{11}	1100	$/x_4x_5$	D_1D_2	14
			a_{12}	1110	$/x_4/x_5x_6$	$D_1D_2\ D_3$	15
			a_{14}	0101	$/x_4/x_5/x_6$	D_2D_4	16
B_7	11	****	a_1	0000	x_3x_6	–	17
			a_3	0001	x_3/x_6	D_4	18
			a_{10}	1010	$/x_3x_2$	D_1D_3	19
			a_{12}	1110	$/x_3/x_2$	$D_1D_2D_3$	20

The table of block BCT reflects the law for transformation of state codes $K(a_m)$ into class codes $K(B_i)$, where $a_m \in B_i$. It includes columns a_m, $K(a_m)$, B_i, $K(B_i)$, τ_m, m. Here the column τ_m includes variables $\tau_r \in \tau$, which are equal to 1 in code $K(B_i)$ from the m-th line of the table. In case of FSM $U_2(\Gamma_1)$, the table of block BCT has $I_M = 7$ lines (Table 2).

Table 2

Table of block BCT for Moore FSM $U_2(\Gamma_1)$

a_m	$K(a_m)$	B_i	$K(B_i)$	τ_m	m
a_{11}	1100	B_3	01	τ_2	1
a_{13}	0100				2
a_{14}	0101				3
a_7	1000	B_6	10	τ_1	4
a_8	1111				5
a_9	1101	B_7	11	$\tau_1\ \tau_2$	6
a_{10}					7

In common case, value I_M can be found as

$$I_M = \sum_{i=1}^{I} \eta_i \cdot C_i, \tag{14}$$

where $\eta_i = |B_i|$, C_i is a Boolean variable equal to 1 iff $B_i \in \Pi_B$. This table is used to derive the equations (8). In our particular case, the following system of equations can be found:

$$\begin{aligned} \tau_1 &= A_7 \vee A_8 \vee A_9 \vee A_{10} = T_1\overline{T_2T_4} \vee T_1T_2T_4; \\ \tau_2 &= A_9 \vee A_{10} \vee A_{11} \vee A_{13} \vee A_{14} = T_2\overline{T_3} \vee \overline{T_2}T_3\overline{T_4} \end{aligned} \tag{15}$$

This system was minimized using the Karnaugh map from Figure 3.

System (2) is represented by equations from the system (11). Equations from system (7) depend on terms F_h ($h = 1,\dots, H_2$ (Γ_j)) corresponding to the lines of transformed structure table. These terms are represented as:

$$F_h = \bigwedge_{r=1}^{R_1} \tau_r^{l_{rh}} \bullet \bigwedge_{r=1}^{R} T_r^{d_{rh}} \bullet X_h, \tag{16}$$

where l_{rh}, $d_{rh} \in \{0, 1, x\}$ are respectively the values of the r-th bit of codes $K(B_i)$ and $K(a_m)$ from the h-th line of the table: $\tau_r^0 = \overline{\tau}_r$, $\tau_r^1 = \tau_r$, $T_r^0 = \overline{T}_r$, $T_r^1 = T_r$, $\tau_r^x = T_r^x = 1$.

In our particular case the following equation, for example, can be derived from Table 1:

$$D_4 = F_1 \vee F_2 \vee F_6 \vee F_8 \vee F_{10} \vee F_{11} \vee F_{12} \vee F_{13} \vee F_{16} \vee F_{18} = \overline{\tau_1\tau_2}\,\overline{T_1T_3T_4}x_1 \vee \ldots \vee \tau_1\overline{\tau_2x_4x_5x_6} \vee \tau_1\tau_2x_3\overline{x_6}.$$

Let us point out that in our particular case there are no codes $K(B_i)$ with "don't care" [8] values of bits. But in common case, it is quite possible.

Implementation of Moore FSM U_2 (Γ_j) logic circuit is reduced to implementation of logic circuits for systems (2), (7) and (8) using some CPLD chips. This step is well presented in literature [7,15] and we do not discuss it in this article.

5. ANALYSIS OF PROPOSED METHOD

Let us start from analysis of our particular example. Let the symbol $H_i\ (f)$ stand for the number of terms in function $f \in \tau \cup T \cup Y$ for Moore FSM U_i. The number of PAL macrocells having q terms which is needed to implement function f can be denoted as $\eta_i(f, q)$. Using results [7], we can find that

$$\eta_i(f,q) = \left\lceil \frac{H_i(f) - q}{q - 1} \right\rceil + 1. \tag{17}$$

Let us note that subscript i determines the type of Moore FSM model. The results of our calculations are shown in Table 3.

Table 3

Characteristics of different models

	U_1		U_2		U_3		U_4	
D_1	17	8	11	5	12	6	18	9
D_2	14	7	8	4	8	4	13	6
D_3	19	9	8	4	9	4	13	6
D_4	19	9	10	5	10	5	12	6
y_1	4	2	2	1	3	1	2	1
y_2	4	2	2	1	3	1	2	1
y_3	5	2	2	1	2	1	2	1
y_4	5	2	2	1	3	1	2	1
y_5	5	2	2	1	3	1	2	1
y_6	5	2	2	1	2	1	2	1
y_7	5	2	3	1	5	2	2	1
τ_1	0	0	2	1	0	0	0	0
τ_2	0	0	2	1	0	0	0	0
BIMF	33	2	18	2	19	2	27	2
BMO	14	2	7	1	8	2	7	1
BCT	–	–	2	1	–	–	–	–
FSM	47	4	27	3	27	4	34	3

In case of Moore FSM U_1 states are encoded in an arbitrary way. Let in case of the FSM U_1 (Γ_1) we have $K(a_1) = 0000$, $K(a_2) = 0001, \ldots, K(a_{14}) = 1101$. The first column in Table 3 shows the values $H_i(f)$ and the second shows the values $\eta_i(f, q)$ for FSM $U_1(\Gamma_1)$. The first column in line BIMF shows the total number of macrocells in the logic circuit of block BIMF. The second column in this line shows the number of layers in this circuit. The same characteristics are shown for blocks BMO and BCT. The total number of macrocells and layers in logic circuit of FSM is shown in the line FSM.

In our experiments, the PAL macrocells with $q = 3$ are used. In this case the number of layers $L(f, q)$ for implementation of function f is determined as

$$L(f, q) = \lceil log_3 \eta_i(f, q) \rceil. \qquad (18)$$

The characteristics of FSM $U_2(\Gamma_1)$ are found from Table 1 and systems (11) and (15).

Using the algorithm ESPRESSO for optimal state assignments, the following codes for the FSM $U_3(\Gamma_1)$ can be obtained (Figure 4).

T_1T_2 \ T_3T_4	00	01	11	10
00	a_1	a_5	a_{12}	*
01	a_{11}	a_{13}	a_{14}	*
11	a_7	a_8	a_9	a_{10}
10	a_3	a_6	a_2	a_4

Fig. 4. Outcome of optimal state assignment

Application of ESPRESSO for refined state encoding (model U_4) produces the same results as for combined state encoding (Figure 3).

Analysis of Table 3 shows that, in case of GSA Γ_1, application of the combined state assignments produces the best solution. The logic circuit of FSM $U_2(\Gamma_1)$ has less hardware than logic circuits of $U_1(\Gamma_1)$ and $U_4(\Gamma_1)$ was well as less layers than all other logic circuits analyzed in our example.

The next step in our research was application of probabilistic approach to find an area where the model U_2 consumes less hardware than other models. There are three key points in the probabilistic approach [17,18]:

1. Use of the class of graph-schemes of algorithm instead of a particular graph-scheme of algorithm Γ. Each class is characterized by the parameter p_1, which is treated as a probability that a particular vertex of the graph-scheme of algorithm Γ is an operational one.
2. Use of matrix realization of the logic circuit of FSM [1] instead of the implementation using some standard VLSI chips. In this case we can find a hardware amount as the area of matrices for a given structure of logic circuit of FSM.
3. To study relation $S(U_i) / S(U_j)$, where $S(U_j, S(U_i)$ are the areas of the matrices for finite-state-machines U_i and U_j respectively. It is proved in [18] that such relations for the cases of matrix realization are the same as for circuits implemented with standard programmable logic devices, such as PAL, PLA or PROM.

Matrix realization of Moore finite-state-machine U_1 is shown in Figure 5.

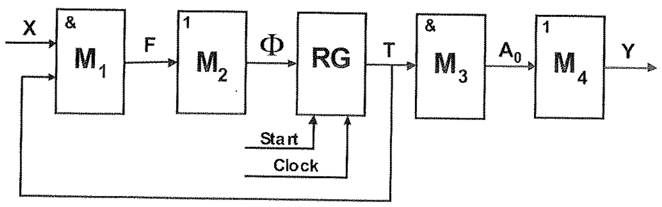


Fig. 5. Matrix realization of Moore FSM U_1

Here M_1 is a conjunctive matrix that implements the system F of terms of the system (1); M_2 is a disjunctive matrix that implements functions of the system (1); M_3 is a conjunctive matrix that implements the system A_0, where each function corresponds to conjunction $A_m (m = 1,..., M)$ respective to the code $K(a_m)$ of the state $a_m \in A$; M_4 is a disjunctive matrix that implements the functions (2). It is clear that the matrices M_1 and M_2 represent the block BIMF, and matrices M_3 and M_4 represent the block BMO. The complexity of these circuits can be expressed as

$$S(BIMF)_1 = 2(L + R) \cdot H + H \cdot R: \tag{19}$$

$$S(BMO)_1 = 2 \cdot R \cdot M + M \cdot N. \tag{20}$$

Matrix realization of Moore FSM U_2 is shown in Figure 6.

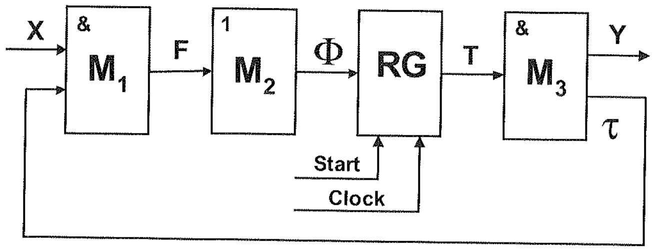


Fig. 6. Matrix realization of Moore FSM U_2

In case of U_2, let F include H_0 elements, where H_0 is the number of transitions for equivalent Mealy FSM [14]. We assume that each function of systems (2) and (5) is implemented using in average k PAL macrocells. Because of it, both BMO and BCT are represented by conjunctive matrix M_3. The complexity of these circuits can be expressed as

$$S(BIMF)_2 = 2(L + R_1) \cdot H_0 + H_0 \cdot R; \tag{21}$$

$$S(BMO)_2 = 2R(N + R_1) \cdot K. \tag{22}$$

Now we should analyze the following function:

$$f = \frac{S(BIMF)_2 + S(BMO)_2}{S(BIMF)_1 + S(BMO)_1}. \tag{23}$$

To reduce the number of variables in (23), we can use results from [16,17], where parameters L, R, H, H_0, R_1 are expressed as the following functions:

$$L = (1 - p_1) \cdot K/1,3; \tag{24}$$

$$R = \lceil log_2 p_1 \cdot K \rceil; \tag{25}$$

$$H = 17,4 + 1,7 \cdot p_1 \cdot K; \tag{26}$$

$$H_0 = 4,4 + 1,1 \cdot p_1 \cdot K; \tag{27}$$

$$R_1 = \lceil log_2(2,75 + 0,34 \cdot p_1 \cdot K) \rceil. \tag{28}$$

In expressions (24)-(28), parameter K is equal to the number of vertices in initial GSA Γ. Some results of our experiments are shown in Figure 7 – Figure 12.

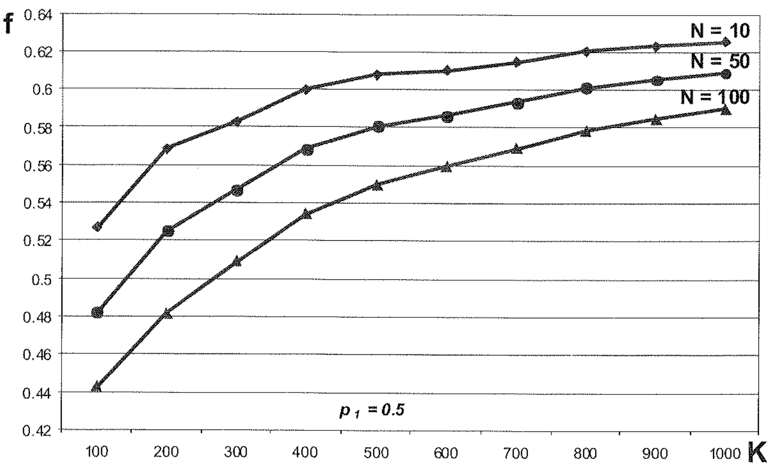


Fig. 7. Comparison of U_1 and U_2 (p_1=0,5)

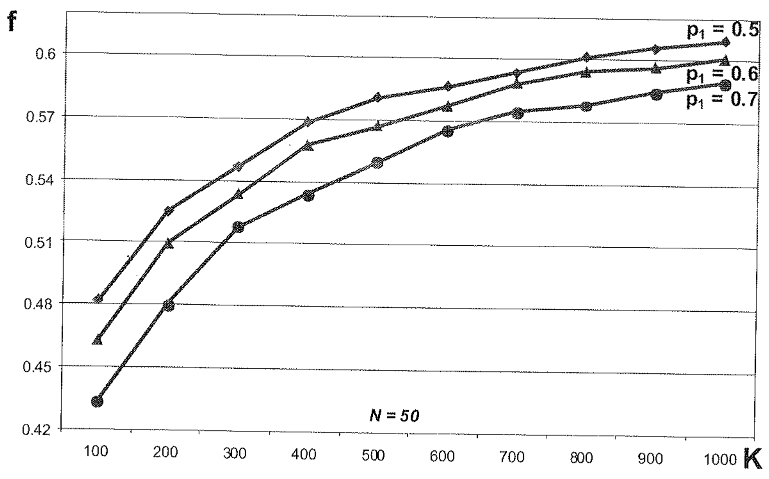


Fig. 8. Comparison of U_1 and U_2 (N=50)

As it follows from Figure 7 and 8, application of the combined state assignment always leads to decrease in hardware amount in comparison to the arbitrary state assignment (U_1). The gain is increased with decrease in the number of vertices (decrease of K), increase in the number of microoperations (increase of N) and in the number of operational vertices (increase of p_1) in interpreted GSA Γ. For example, if $p_1 = 0.7$, $N = 50$, $K = 400$, the hardware amount is decreased up to 44% in comparison with U_1. Let us point out, that small values of k ($k \leq 4$) practically do not affect the value of function f.

The matrix realization of Moore FSM U_3 is the same as shown in Figure 5. Due to optimal state assignment, the following relations are true:

$$S(BIMF)_3 = S(BIMF)_2; \tag{29}$$

$$S(BMO)_3 = S(BMO)_1. \tag{30}$$

Now we should analyze the following function:

$$f = \frac{S(BIMF)_2 + S(BMO)_2}{S(BIMF)_3 + S(BMO)_3}. \tag{31}$$

It is clear from Figure 9 and Figure 10, that application of the proposed method always gives less amount of hardware than the method of optimal state assignment. This gain is increased with decrease of the number of vertices of GSA Γ and increase of the number of operational vertices of graph-schemes of algorithm Γ (increase of parameter p_1) (Figure 9). This gain is increased with increase of the number of microoperations N in interpreted GSA Γ (Figure 10).

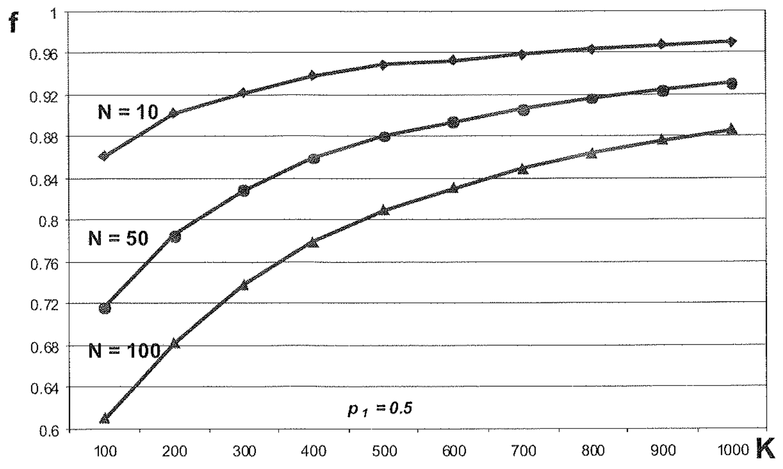


Fig. 9. Comparison of U_2 and U_3 (p_1=0,5)

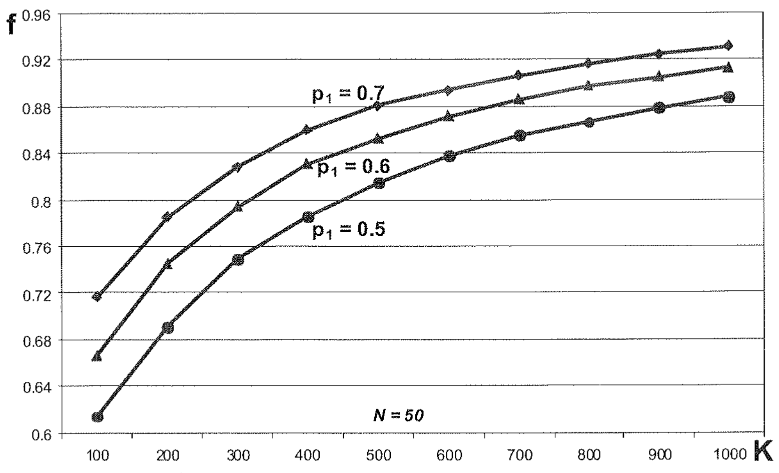


Fig. 10. Comparison of U_2 and U_3 (N=50)

The matrix realization of Moore FSM U_4 is the same as it is shown in Figure 6, but outputs τ are absent. It leads to the following relations:

$$S(BIMF)_4 = S(BIMF)_1; \tag{32}$$

$$S(BMO)_4 = 2 \cdot R \cdot N \cdot k. \tag{33}$$

Now we should analyze the following function:

$$f = \frac{S(BIMF)_2 + S(BMO)_2}{S(BIMF)_4 + S(BMO)_4}. \tag{34}$$

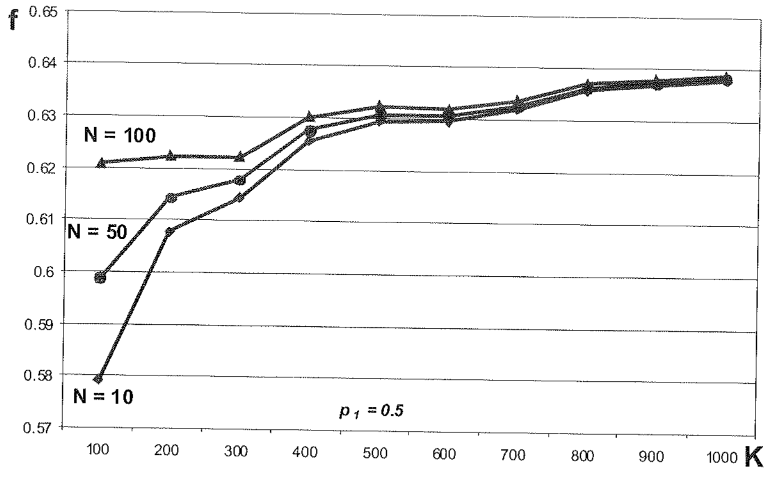


Fig. 11. Comparison of U_2 and U_4 (p_1=0,5)

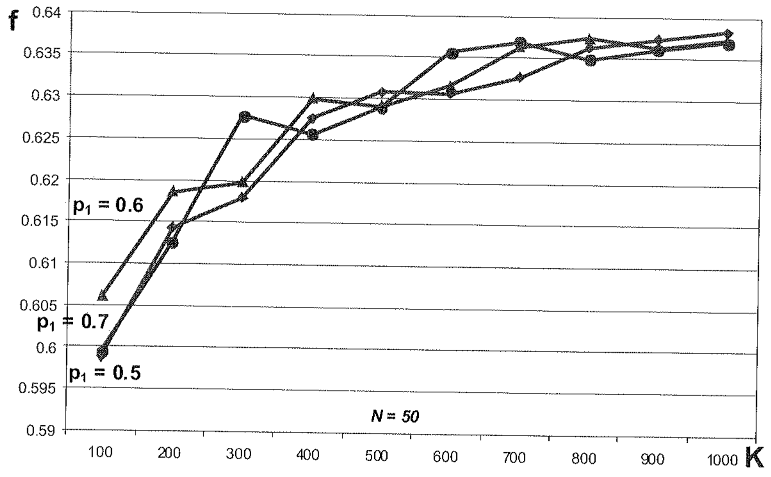


Fig. 12. Comparison of U_2 and U_4 (N=50)

igure 6,

(32)

(33)

As it follows from Figure 11 and 12, application of the combined state assignment always leads to decrease in hardware amount in comparison with the refined state assignment (U_4). The gain is increased with decrease in the number of vertices (decrease of K), decrease in the number of microoperations (decrease of N) and with increase in the number of operational vertices (increase of p_1) in interpreted GSA Γ.

To check the correctness of these experiments, we developed some original software tools oriented on industrial CPLD chips [4,5]. This software uses the standard package Web Pack of Xilinx [5] and VHDL models of Moore FSM $U_1 - U_4$. Some standard benchmarks [19] were used to conduct our experiments. The experiments conducted using this software confirm correctness of tendencies represented in Fig. 7 Fig. 12. The only difference is a slightly less gain in comparison with theoretical experiments. In all our experiments the gain was approximately 7% – 10% less than in cases of theoretical experiments.

6. CONCLUSION

The proposed method of combined state assignment is oriented on decrease in hardware amount for both blocks of input memory functions and microoperations of Moore FSM. If necessary, the codes of some classes of pseudoequivalent states are generated by the block of code transformer. In this case a PAL property such as a wide fan-in is used to operate with two sources of class codes. Our experiments show that proposed method always produces logic circuits with less amount of PAL macrocells than any known methods for Moore FSM design, in particular the methods based on refined and optimal state assignments.

Let us point out, that decrease of hardware amount is very often combined with decrease in the number of layers of resulted combinational circuit. It results in decrease of FSM cycle time and, therefore, increase of its performance as well as performance of the whole digital system. As the code transformation is executed in the same time when system data-path executes some operation, it does not lead to slowing down of digital system with Moore FSM.

Our future research is connected with exploration of possibility for the application of proposed method in the digital systems where a control unit is implemented using technology of FPGA [3,4].

7. REFERENCES

1. S. B a r a n o v: *Logic and System Design of Digital Systems*. Tallinn: TUT Press, 2008.
2. A. B a r k a l o v and M. W ę g r z y n: *Design of Control Units with Programmable Logic*. Zielona Góra: University of Zielona Góra Press, 2006.
3. C. M a x f i e l d: *The Design Warrior's Guide to FPGAs*. Amsterdam: Elsevier, 2004.
4. Z. N a v a b i: *Embedded Core Design with FPGAs*. N.Y.: McGraw Hill, 2007.
5. http://www.altera.com
6. http://www.xilinx.com
7. D. K a n i a: *Logic Synthesis Oriented on Programmable Logic Devices of the PAL type*. Gliwice: Silesian Technical University, 2004, (in Polish).
8. G. D e M i c h e l i: *Synthesis and Optimization of Digital Circuits*. N.Y.: McGraw Hill, 1994.
9. S. D e v a d a s, H.-K. M a, R. N e w t o n, A. S a n g i o v a n n i - V i n c e n t e l l i: *State Assignment of Finite State Machines Targeting Multilevel Logic Implementations*. IEEE Transactions on Computer-Aided Design, 1988, vol. 7, pp. 1290-1300.

10. T. Kam, T. Villa, R. Brayton, A. Sangiovanni-Vincentelli: *Synthesis of Finite State Machines: Functional Optimization*. Boston/London/Dordrecht: Kluwer Academic Publishers, 1998.
11. T. Villa, T. Kam, R. Brayton, A. Sangiovanni-Vincentelli: *Synthesis of Finite State Machines: Logic Optimization*. Boston/London/Dordrecht: Kluwer Academic Publishers, 1998.
12. S. Chattopadhyay: *Area Conscious State Assignment with Flip-Flop and Output Polarity Selection for Finite State Machine Synthesis: A Genetic Algorithm Approach*. The Computer Journal, 2005, vol. 48, no 4, pp. 443-450.
13. Y. Xia and A. Almaini: *Genetic algorithm based state assignment for power and area optimization*. IEEP. – Comput. Dig. T., 2002, vol. 149, pp. 128-133.
14. A. Barkalov: *Principles of Optimization of logical circuit of Moore FSM*. Cybernetics and system analysis. 1998, no 1, pp. 65-72 (in Russian).
15. V. Solovjev: *Design of Digital Systems Using the Programmable Logic Integrated Circuits*. Moscow: Hotline – Telecom, 2001, (in Russian).
16. A. Barkalov, L. Titarenko, S. Chmielewski: *Reduction in the number of PAL macrocells in the circuit of a Moore FSM*. International Journal of Applied Mathematics and Computer Science, 2007, vol. 17, no 4, pp. 565-675.
17. A. Barkalov, L. Titarenko, S. Chmielewski: *Optimization of Moore FSM on System-on--Chip*. IEEE East-West Design & Test Symposium, Kharkov, 2007, pp. 105-109.
18. A. Barkalov: *Design of Mealy finite-state-machines with the transformation of objects codes*. International Journal of Applied Mathematics and Computer Science, 2005, vol. 15, no 1, pp. 151-158.
19. S. Yang: *Logic Synthesis and Optimization Benchmarks User Guide*. Microelectronics Center of North Carolina, Research Triangle Park, North Carolina, 1991.

T
introd
mabl
encod
the d
[1, 6,
throu
that's

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 335–354

Improvements to Symbolic Functional Decomposition Algorithms for FSM Implementation in FPGA Devices

PIOTR SZOTKOWSKI, MARIUSZ RAWSKI

Institute of Telecommunications
Warsaw University of Technology, Poland
{p.szotkowski, rawski}@tele.pw.edu.pl

Received 2009.02.20
Authorized 2009.03.28

The method of symbolic functional decomposition for FSM implementation in FPGA devices yields better results than the currently widespread, two-step approaches based on state encoding and mapping of the resulting binary function. This paper presents the method using an example FSM and briefly discusses the existing algorithms, along with results obtained for benchmark FSMs. The paper also proposes a heuristic algorithm for input selection as well as a new, clique-based algorithm for the construction of the crucial decomposition blankets.

Keywords: finite state machine, FPGA, FSM, symbolic functional decomposition

1. INTRODUCTION

The method of symbolic functional decomposition of finite state machines, first introduced in [5], is a novel approach to the implementation of FSMs in field programmable gate array (FPGA) devices. Contrary to the classic, two-step solutions based on encoding the machine's states and then mapping the resulting binary function into the device's LUT cells (ideally using the method of binary functional decomposition) [1, 6, 8], the symbolic method maintains the multi-valued representation of the states throughout the whole decomposition process and encodes the states partially, in a way that's optimal in the given mapping iteration.

The implementations of this method presented in [10, 11, 12] yield better results than the classic approaches. Unfortunately, while proving the high quality of the results obtained using this method, these soluions are not really viable for larger FSMs without additional algorithms implementing heuristic input selection; the algorithms constructing the crucial blankets used in the method can also be subtantially improved upon.

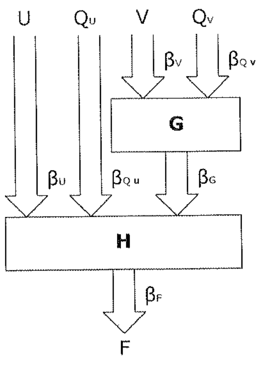


Fig. 1. Symbolic Functional Decomposition with Blankets

After a brief introduction of the existing symbolic functional decomposition algorithms, this paper proposes the algorithms for input selection heuristics and a new, clique-based algorithm for blanket construction.

2. SYMBOLIC FUNCTIONAL DECOMPOSITION

2.1. DEFINITIONS

Symbolic functional decomposition of an FSM can be described similarly to serial decomposition of a Boolean function defined using blanket algebra (see [1] and [6] for details on these two concepts). Let X be the set of primary inputs, Y be the set of primary outputs of a certain FSM specified by a state transition table. Let Q and Q' be multi-valued variables representing present and next state of this FSM. Let U and V be two subsets of X, such that $U \cup V = X$. Let Q_V and Q_U be multi-valued variables encoding variable Q. Let β_V and β_U be blankets induced by the primary input subsets V and U. Let β_{Q_V} and β_{Q_U} be blankets induced by the multi-valued variables Q_V and Q_U. Let β_Y and $\beta_{Q'}$ be blankets induced by the primary output sets and by the next state multi-valued variable Q'.

Theorem 1 *Existence of the symbolic functional decomposition [7].*

The FSM has a symbolic functional decomposition with respect to (U, Q_U, Q_V, V) *if there exists a blanket* β_G *such that* $\beta_V \bullet \beta_{Q_V} \leq \beta_G$ *and* $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$*, where* $\beta_F = \beta_Y \bullet \beta_{Q'}$.

Fig. ?? presents an outline of the symbolic functional decomposition with the abovementioned blankets.

2.2. MAIN CONCEPTS BEHIND THE METHOD

The main idea behind the symbolic decomposition method is to take a finite state machine which cannot be directly implemented in the target FPGA device's LUT cells (because after encoding the machine's states with a minimal-length encoding the sum of the encoding's width and the number of binary inputs is greater than the number of inputs of the device's LUT cells – otherwise the FSM would be directly implementable, requiring at most a trivial parallel decomposition) and decompose it into a binary function G and a smaller finite state machine H. Ideally, the G function has no more inputs than the widest LUT cells (and thus is directly implementable in them, requring at most a trivial parallel decomposition); the same goal applies to the desired H FSM – if the number of its inputs added to the number of bits in a minimal-length encoding of its state variable is not greater than the width of the widest LUT cells, H is directly implementable (again, requiring at most a trivial parallel decomposition). If G and/or H does not fulfill these goals, the given block undergoes another iteration of the decomposition process (the "classic", binary serial decomposition for the G function and the symbolic functional decomposition for the H finite state machine).

As the finite state machine undergoes the symbolic functional decomposition process, every iteration partially encodes the initial FSM's state variable: every iteration represents the state variable using the Q_U and Q_V variables, of which Q_U becomes the state variable of the H state machine and Q_V is binary encoded to create inputs to the G function. This concept makes the whole iterative process of decomposition maintain the (partial) symbolic representation of the initial FSM's state variable, while at the same time encoding it gradually in a way that is optimal for the given iteration of the decomposition process.

2.3. ADVANTAGES OF THE METHOD

As mentioned in the introduction, the advantages of the symbolic functional decomposition method (when applied to implementation of finite state machines in FGPA devices) are the ability to retain symbolic representation of the machine's states throughout the multi-level decomposition process and the partial encoding of the machine's states, optimal for a given level of decomposition.

All of the currently widespread methods of implementing FSMs in FPGA devices, such as the ones described in [2, 4, 13], are based on a two-step approach. First, the

FSM's states are encoded (in various ways, depending on the method used) to binary representation; next, the binary function (a result of replacing the states and next-states with their encoded representations) is mapped to the FPGA device's logic cells – with the (classic, binary) functional decomposition regarded as the most effective method implementing this mapping.

The main disadvantage of the two-step approach comes from the multi-level nature of the synthesis process. It's very hard to project the impact of a given encoding on more than the first level of decomposition; all of the approaches which encode the states beforehand are based on some amount of guessing about what encoding strategy would result in a good overall decomposition.

The symbolic functional decomposition method addresses this problem by skipping the encoding step and maintaining the multi-value representation of the states, effectively partially encoding the states on each step of the decomposition process in a way that's optimized for this particular iteration.

The other disdvantage of the current solutions is the choice of the number of bits used to encode the FSM's states. Contrary to naïve assumptions, minimal-bit approaches (such as simple sequential encoding of the states, an encoding based on the Gray code or a random, minimal-lenght solution) do not neccessarily yield good results. This observation led to creation of the one-hot encoding, which goes to the other extreme and uses n bits to encode n states (with one of the bits set to 1 and all the others set to 0). For some FSMs this approach yields better results than minimal-lenght encodings, but in most cases the optimal encoding length lies somewhere between the two extremes.

This characteristic of FSM implementation in FPGA devices is also addressed by the symbolic functional decomposition method. In this method the states of the machine are partially encoded on every step of the mapping process without assuming any particular number of bits, so the final encoding lenght is not determined beforehand, nor it is directly related to the number of states; instead, the encoding is adapted to (and depends on) the needs of particular decomposition iterations encountered during the mapping process, and so leads to better results than methods which must assume a particular encoding length before the mapping process even begins.

2.4. EXAMPLE FINITE STATE MACHINE

Along with the description of each of the steps of an example graph-based algorithm, an illustration based on an example finite state machine is presented. The example FSM's state transition table is given in Table 1.

The example finite state machine has ten states (init0, init1, init2, init4, IOwait, RMACK, WMACK, read0, read1 and write0). The previous, two-step approaches of implementing this state machine would either encode the states with the (seemingly) minimal number of bits – four (the Jedi and Nova methods) – or would use ten bits, one for each state (the one-hot method) [2, 4, 13].

binary
t-states
– with
method

l nature
ding on
ode the
strategy

y skip-
e states,
ocess in

nber of
imal-bit
ased on
ld good
s to the
d all the
l-lenght
een the

dressed
s of the
ssuming
rehand,
apted to
l during
assume

ased al-
ed. The

IOwait,
aches of
mingly)
ten bits,

The graph-based symbolic functional decomposition algorithm generates two symbols for the β_{Q_U} blanket and four for the β_{Q_V} blanket, effectively encoding the machine's states on three bits (one for the Q_U set and two for the Q_V set).

Table 1

State Transition Table of the Example FSM

	x_1	x_2	x_3	x_4	Q	Q'	y_1	y_2
1	-	-	0	0	init0	init1	0	0
2	0	1	0	0	init1	init1	0	0
3	-	-	1	-	init1	init2	1	0
4	1	-	1	0	init2	init4	1	0
5	-	1	1	1	init4	init4	1	0
6	-	-	0	1	init4	IOwait	0	1
7	0	0	0	-	IOwait	IOwait	0	1
8	1	0	0	-	IOwait	init1	0	1
9	0	1	1	0	IOwait	read0	0	0
10	1	1	0	0	IOwait	write0	1	1
11	0	1	1	1	IOwait	RMACK	1	1
12	1	1	0	1	IOwait	WMACK	0	0
13	-	0	1	-	IOwait	init2	0	1
14	0	0	1	0	RMACK	RMACK	1	1
15	0	1	1	1	RMACK	read0	0	0
16	1	1	0	0	WMACK	WMACK	0	0
17	1	0	0	1	WMACK	write0	0	1
18	0	0	0	1	read0	read1	1	1
19	0	0	1	0	read1	IOwait	0	1
20	0	1	0	0	write0	IOwait	0	1

3. EXAMPLE ALGORITHM

3.1. DVERVIEW OF THE ALGORITHM

The algorithm operates on blankets induced by input, output and intermediate variables (presented on Fig. ??). Some of the blankets can be easily computed based on the state machine's definition and the choice of the U and V subsets (which are defined for a single run of the algorithm), the rest is constructed by creating incompatibilty graphs, in which the vertices represent the blankets' blocks, while the edges connect the blocks that can't (or shouldn't) be merged.

The algorithm consists of four steps:

1. Computation of the blankets induced by the finite state machine (in particular, β_Q and β_F) and the direct and indirect input variable subsets (β_U and β_V).
2. Construction of β_{Q_U}, the blanket representing the direct subset of the state variable encoding.
3. Construction of β_G, the blanket representing the separations provided by the G block.
4. Construction of β_{Q_V}, the blanket representing the indirect subset of the state variable encoding.

In the final implementation, steps 2-4 are repeated so that different possible β_{Q_U}, β_G and β_{Q_V} blankets are constructed, while the whole algorithm is repeated for different selections of the U and V sets. Then, the best (for the given target architecture) decomposition is selected and the G and H tables are built; if the H table is too big to be implemented in the target architecure directly, it undergoes the whole process again, acting itself as the FSM to be decomposed.

3.2. BLANKETS INDUCED BY THE FINITE STATE MACHINEE AND BY THE DIRECT AND INDIRECT VARIABLE INPUT SETS

First step of the algorithm consists of computing the blankets induced by the finite state machine's transition table. The β_Q and β_F blankets, as well as the β_U and β_V blankets (for defined U and V sets) are used further in the algorithm as the basis for construction of the three blankets that define a particular decomposition – β_{Q_U}, β_G and β_{Q_V}.

In this step, the algorithm computes the required blankets based on the blankets induced by individual columns of the FSM's transition table. In the case of the example FSM and the chosen $U = \{x_2, x_4\}$ and $V = \{x_1, x_3\}$ sets, the following relations hold true and allow the computation of the "base" blankets: $\beta_F = \beta_{Q'} \bullet \beta_{y_1} \bullet \beta_{y_2}$, $\beta_U = \beta_{x_2} \bullet \beta_{x_4}$ and $\beta_V = \beta_{x_1} \bullet \beta_{x_3}$. Thus, the β_Q, β_F, β_U and β_V blankets equal

$$\begin{aligned}
\beta_Q &= \{\overline{1}; \overline{2,3}; \overline{4}; \overline{5,6}; \overline{7,8,9,10,11,12,13}; \overline{14,15}; \overline{16,17}; \overline{18}; \overline{19}; \overline{20}\},\\
\beta_F &= \{\overline{1,2}; \overline{3}; \overline{4,5}; \overline{6,7,19,20}; \overline{8}; \overline{9,15}; \overline{10}; \overline{11,14}; \overline{12,16}; \overline{13}; \overline{17}; \overline{18}\},\\
\beta_U &= \{\overline{1,2,3,4,9,10,16,20}; \overline{1,3,4,7,8,13,14,19}; \overline{3,5,6,11,12,15}; \overline{3,6,7,8,13,17,18}\},\\
\beta_V &= \{\overline{1,2,6,7,18,20}; \overline{1,6,8,10,12,16,17}; \overline{3,4,5,13}; \overline{3,5,9,11,13,14,15,19}\}.
\end{aligned}$$

3.3. BLANKET REPRESENTING THE DIRECT SUBSTET OF THE STATE VARIABLE ENCODING

The first blanket that has to be constructed by the algorithm (as opposed to just being computed from the state machine's representation and the U and V sets) is β_{Q_U}, the blanket representing the direct subset of the state variable encoding. This blanket, being constructed by merging of the blocks of the β_Q blanket, has to satisfy the $\beta_Q \leq \beta_{Q_U}$ condition – it cannot provide any separations not already provided by β_Q.

The more "merged" this blanket is (i.e., the fewer blocks it has), the fewer binary inputs will be required to implement it in the final decomposition (if it has ten blocks, four binary inputs are required; if it's merged down to eight blocks, three binary inputs are enough). At the same time, all separations required by the F function (i.e., by the β_F blanket) and not provided by either β_U or β_{Q_U} have to be provided by β_G (the condition of $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$ has to be satisfied); the more β_{Q_U} is "merged", the more separations have to be provided by β_G, the number of β_G's blocks is larger, and, thus, the G block requires more outputs in the final implementation.

To construct the β_{Q_U} blanket, an incompatibility graph is created with vertices representing the β_Q blocks and edges connecting these of the vertices/blocks which provide separations required by the F function (except for the separations provided already by the β_U blanket). At the same time, the edges of this graph are weighted with weights representing the number of separations lost when merging the related vertices/blocks.

Given this approach, any disconnected pair of vertices can be merged at no cost (merging them lessens the number of separations provided by the β_{Q_U} blanket, but the lost separations are either not required by the F function or are already provided by the β_U blanket); once the graph is complete the algorithm starts merging the vertices on a lowest-weight-first basis. This approach leads to a significant reduction of the number of blocks of the β_{Q_U} blanket, while still providing as many separations (required by the F function) as possible.

For the example finite state machine from Table 1, the algorithm first tries to find a pair of vertices that are not connected with an edge (so the represented blocks can be merged "at no cost"); the $\overline{1}$ and $\overline{5,6}$ blocks form such a pair. Once the graph is complete, the algorithm finds a pair with the lowest edge weight and merges it.

The number of binary inputs required for implementation of any given blanket is equal to the base-two logarithm from the number of states (rounded up); thus, the inital number of binary inputs for encoding the ten-block β_{Q_U} blanket would be four. The algorithm merges the blocks until the number of the binary inputs is smaller (so, in this case, until there are at most eight blocks, and, thus, three binary inputs suffice) and then tries to construct the corresponding β_G and β_{Q_V} blankets. Once this is done, the algorithm returns to this step and makes the β_{Q_U} blanket smaller again (in this example, merges it down to four blocks), and repeats the β_G and β_{Q_V} creation. This process is repeated until a set of possible decompositions is obtained.

In the case of the example finite state machine and the selected U and V sets, the best decomposition was obtained once the β_{Q_U} blanket was merged down to two blocks, yielding a final β_{Q_U} blanket of

$$\beta_{Q_U} = \{\overbrace{1,2,3,4,5,6,18}^{u_1};\overbrace{7,8,9,10,11,12,13,14,15,16,17,19,20}^{u_2}\}.$$

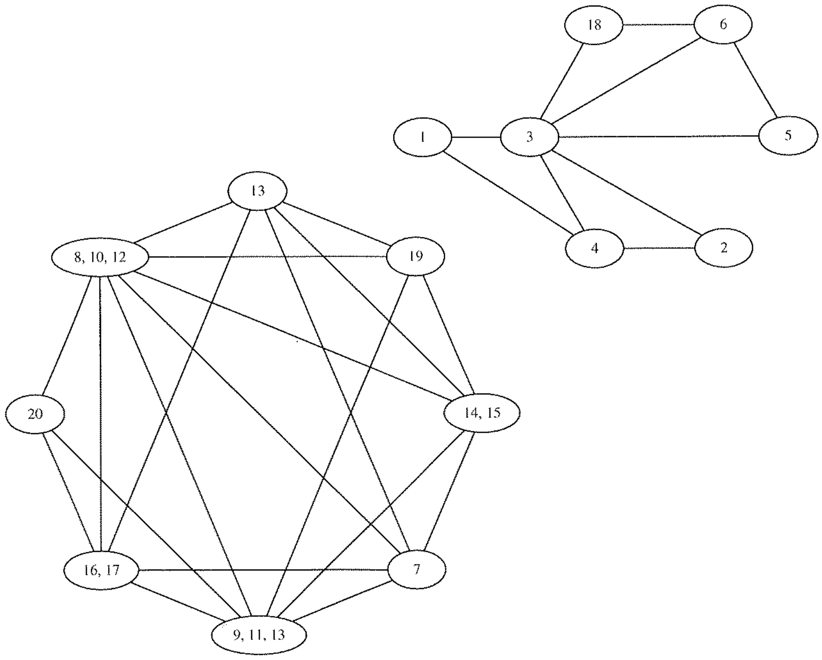


Fig. 2. The Initial Incompatibility Graph for the β_G Blanket

3.4. BLANKET REPRESENTING THE G BLOCK

Once the β_{Q_U} blanket is defined, the β_G blanket can be constructed. This blanket describes the output of the G block – it has to provide all of the separations required by the β_F blanket except for the ones already provided by β_U and β_{Q_U} – the $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$ condition has to be satisfied.

As the inputs to the G block consist of the β_V and β_Q blankets (the latter possibly merged down to form β_{Q_V} in the next step), to construct the β_G blanket a new incompatibility graph is created. The vertices of this graph consist of the blocks of the $\beta_V \bullet \beta_Q$ blanket (the β_G blanket has to fulfill the $\beta_V \bullet \beta_Q \leq \beta_G$ condition), while edges connect these of the vertices/blocks which cannot be merged (because they provide a separation required by the β_F blanket and not provided by either β_U or β_{Q_U}).

Once this graph is constructed, every disconnected pair of vertices can be merged, the number of blocks of the β_G blanket can be made smaller and thus the number of physical binary outputs from the G block can be brought down to a number that better fits the target architecture.

In the case of the example finite state machine and the β_{Q_U} blanket constructed in the previous step, the initial graph for the β_G blanket is presented in Fig. 2. Its vertices represent the blocks of the $\beta_V \bullet \beta_Q$ blanket, while edges connect these of the blocks/vertices which have to be separated (if the β_F blanket requires a separation of some of the vectors from these blocks, and that separation is not provided by either β_U or β_{Q_U}).

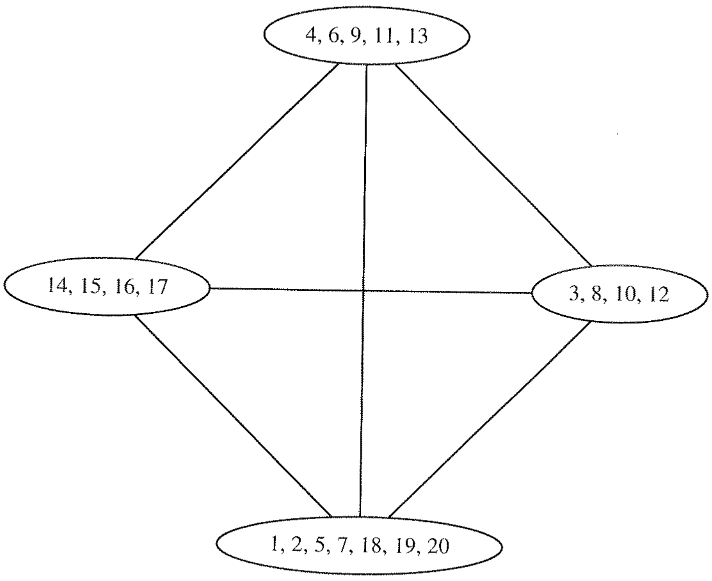


Fig. 3. The Resulting Incompatibility Graph for the β_G Blanket

Again, the number of blocks of this graph governs the number of binary outputs from the G block, and again the algorithm merges them until the number of the outputs is smaller than with the initial graph (or the graph becomes complete, as in this case no more blocks can be merged).

In the case of the example FSM, the graph is merged down to the four-vertex graph presented in Fig. 3. Thus, the resulting β_G blanket is equal

$$\beta_G = \{\overbrace{\overline{1,2,5,7,18,19,20}}^{g_1};\overbrace{\overline{3,8,10,12}}^{g_2};\overbrace{\overline{4,6,9,11,13}}^{g_3};\overbrace{\overline{14,15,16,17}}^{g_4}\}.$$

3.5. BLANKET REPRESENTING THE INDIRECT SUBSET OF THE STATE VARIABLE ENCODING

Once the β_{Q_U} and β_G blankets are constructed, the final step of the algorithm constructs the β_{Q_V} blanket. This blanket has to provide the β_G blanket with all the separations it requires and which are not provided by the β_V blanket (i.e., it has to fulfill the $\beta_V \bullet \beta_{Q_V} \leq \beta_G$ condition).

This blanket is constructed from the blocks of the β_Q blanket (to satisfy the $\beta_Q \leq \beta_{Q_V}$ condition) and, again, the idea is to merge it down so it provides all the separations that are required by β_G (and are not provided by β_V), while having as few blocks as possible.

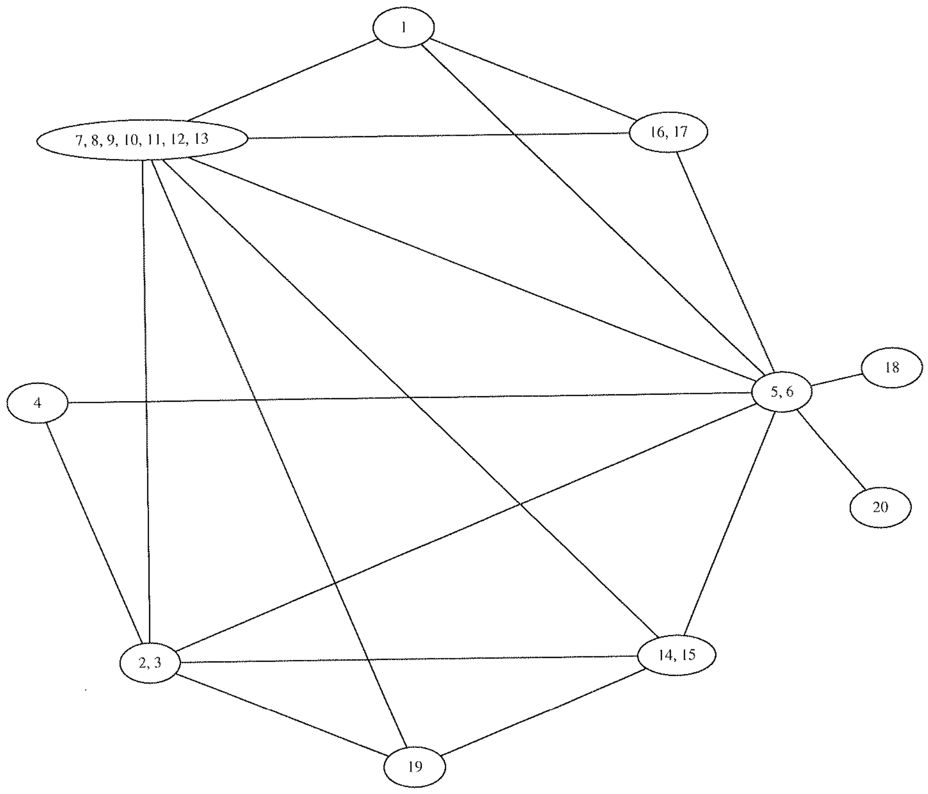


Fig. 4. The Initial Incompatibility Graph for the β_{Q_V} Blanket

To construct this blanket, again an incompatibility graph is created. The vertices of this graph represent the blocks of the β_Q blanket, while the edges connect these of the vertices/blocks which have to be separated to provide the β_G blanket with the required separations (which are not already provided by the β_V blanket). Once this blanket is constructed, the vertices that are not connected with an edge can be freely merged together (much in the same way the vertices of the β_G blanket were merged in the previous step); thus, this blanket is being merged down to lessen the number of the β_{Q_V} blocks – which also lowers the number of binary inputs to the G block.

In the case of the example FSM and the β_G blanket computed above, the inital graph for the β_{Q_V} blanket is presented in Fig. 4. Much like with the graph for the β_{Q_U} blanket, the vertices of this graph represent the blocks of the β_Q blanket, while the edges connect these of the blocks/vertices that have to be kept separated.

Again, this graph is merged down until it's implementable with the minimum number of binary inputs; in the case of the example graph from Fig. 4, the resulting graph has four vertices and is presented in Fig. 5. Thus, the resulting β_{Q_V} blanket equals

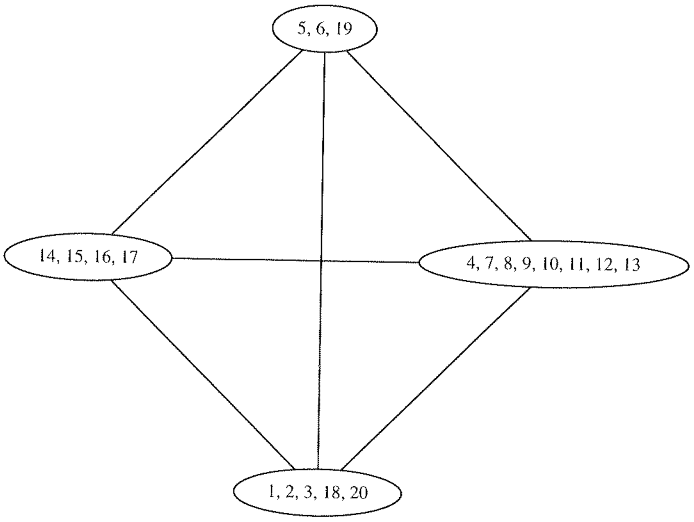


Fig. 5. The Resulting Incompatibility Graph for the β_{Q_V} Blanket

$$\beta_{Q_V} = \{\overbrace{\overline{1,2,3,18,20}}^{v_1};\overbrace{\overline{4,7,8,9,10,11,12,13}}^{v_2};\overbrace{\overline{5,6,19}}^{v_3};\overbrace{\overline{14,15,16,17}}^{v_4}\}.$$

3.6. THE FINAL DECOMPOSITION

Once the β_{Q_U}, β_G and β_{Q_V} blankets are constructed, the final decomposition is defined and the tables representing the G and H blocks can be constructed. In the case of the example finite state machine from Table 1, the final encoding of the states is presented in Table 2. As can be seen, the inital ten states are encoded to two variables, one of which has two values, while the other has four; this means that the final encoding will have three bits per state. The encoding of different states to the same set of values is not an error – in the case of the example finite state machine, the states of init0, init1 and read0 can be merged together and treated as one without any loss in functionality.

In the case of the example finite state machine and the encoding from Table 2, the final tables for the G and H blocks are presented in Table 3. The G block can be directly implemented in a four-input, two-output LUT cell and the H block can be (parallelly) implemented in five five-input, one-output LUT cells; if the target architecture does not have five-input cells, further symbolic functional decomposition of the FSM represented by the H block is required.

Table 2

State Encoding Table

Q	Q_U	Q_V
IOwait	u_2	v_2
RMACK	u_2	v_4
WMACK	u_2	v_4
init0	u_1	v_1
init1	u_1	v_1
init2	u_1	v_2
init4	u_1	v_3
read0	u_1	v_1
read1	u_2	v_3
write0	u_2	v_1

4. BLANKET CONSTRUCTION ALGORITHMS

4.1. BLANKETS INDUCED BY THE FSM

The initial blankets, induced by the finite state machine, can be easily computed based on the FSM's state transition table. The β_Q blanket, induced by the state variable, represents the separations provided by the unencoded states of the FSM. The β_F blanket, induced by the machine's binary output and the next-state variable, defines the separations that must be provided to successfully implement the finite state machine.

The β_U and β_V blankets depend on the selection of inputs for the U (free) and V (bound) subsets of the FSM's binary inputs. These of the separations required by the β_F blanket which are provied by the β_U blanket do not have to be provided by either β_{Q_U} or β_G (the $\beta_U \bullet \beta_{Q_U} \bullet \beta_G \leq \beta_F$ requirement of the symbolic functional decomposition theorem above); likewise, these of the separations required by the β_G blanket which are provided by the β_V blanket do not have to be provided by the β_{Q_V} blanket (the $\beta_V \bullet \beta_{Q_V} \leq \beta_G$ requirement).

4.2. CONSTRUCTED BLANKETS

Detailed algorithms for the construction of the β_{Q_U}, β_G and β_{Q_V} blankets were proposed in [10, 11, 12].

Table 3

Final Decomposition of the FSM from Table 1
(a) the H Block, (b) the G Block

(a)

	x_2	x_4	G	Q_U	Q'_U	Q'_V	y_1	y_2
1	-	0	g_1	u_1	u_1	v_1	0	0
2	1	0	g_1	u_1	u_1	v_1	0	0
3	-	-	g_2	u_1	u_1	v_2	1	0
4	-	0	g_3	u_1	u_1	v_3	1	0
5	1	1	g_1	u_1	u_1	v_3	1	0
6	-	1	g_3	u_1	u_2	v_2	0	1
7	0	-	g_1	u_2	u_2	v_2	0	1
8	0	-	g_2	u_2	u_1	v_1	0	1
9	1	0	g_3	u_2	u_1	v_1	0	0
10	1	0	g_2	u_2	u_2	v_1	1	1
11	1	1	g_3	u_2	u_2	v_4	1	1
12	1	1	g_2	u_2	u_2	v_4	0	0
13	0	-	g_3	u_2	u_1	v_2	0	1
14	0	0	g_4	u_2	u_2	v_4	1	1
15	1	1	g_4	u_2	u_1	v_1	0	0
16	1	0	g_4	u_2	u_2	v_4	0	0
17	0	1	g_4	u_2	u_2	v_1	0	1
18	0	1	g_1	u_1	u_2	v_3	1	1
19	0	0	g_1	u_2	u_2	v_2	0	1
20	1	0	g_1	u_2	u_2	v_2	0	1

(b)

x_1	x_3	Q_V	G
0	0	v_1	g_1
0	0	v_2	g_1
0	0	v_3	g_3
0	0	v_4	g_1
0	1	v_1	g_2
0	1	v_2	g_3
0	1	v_3	g_1
0	1	v_4	g_4
1	0	v_1	g_1
1	0	v_2	g_2
1	0	v_3	g_3
1	0	v_4	g_4
1	1	v_1	g_2
1	1	v_2	g_3
1	1	v_3	g_1
1	1	v_4	g_1

Table 4

Experimental Results for an Architecture with 5/1 LCs

FSM	art décomp	Secode	Gray	Jedi	bin
bbara	10	12	11	15	15
bbtas	5	5	5	5	5
beecnt	8	6	8	9	10
dk15	7	7	7	7	7
dk17	6	6	6	6	6
dk27	5	5	5	5	5
lion	3	3	3	3	3
s8	1	1	6	5	5
$\sum$	45	45	51	55	56

The construction of the β_{Q_U} blanket, which *de facto* introduces a partial encoding of the state variable, is crucial for the whole decomposition process. If the blanket has too many blocks, the resulting H function will represent an FSM with many states, hard to implement in the subsequent decomposition iteration; at the same time, if β_{Q_U} does not introduce enough separations required by β_F (and not provided by β_U), the missing separations will have to be provided by the β_G blanket, which will have too many blocks to be efficiently implemented in the target FPGA architecture.

[10] presents an algorithm for β_{Q_U} construction based on the concept of r-admissibility, which represents the lower bound of the number of binary inputs required by the subsequently-constructed β_G blanket. This implementation is quite fast, but the algorithm is not simple and the resulting β_G blanket more often than not crosses the lower r bound. [11] proposes a simple graph based algorithm for β_{Q_U} construction based on merging of the β_Q blocks represented as vertices in an incompatibility graph; this solution gives better results and is simpler, but at the same time is slower than the r-admissibility one.

As for the β_G and β_{Q_V} construction, [10] proposes a simple and fast uniform method, based on coloring of incompatibility graphs corresponding to the requirements for both of these blankets. [11] proposes another method, based on vertex merging of these graphs, the same it uses for β_{Q_U} construction.

The most sophisticated method for β_G and β_{Q_V} construction is proposed in [12]. This method builds both blankets concurrently, using an innovative bipainting algorithm which partially side-steps the dependencies between β_G and β_{Q_V} creation.

5. EXPERIMENTAL RESULTS

The experimental results obtained with a prototypical program *art décomp* confirm that the symbolic functional decomposition method yields better results than the common two-step approach. Tables 4 and 5 present a comparison of results for standard benchmark finite state machines obtained using the state assignment method described in [3] and [9] (the *Secode* column) and different encoding methods proposed in [2, 4, 13].

Table 5

Experimental Results for an Architecture with 5/1 and 4/2 LCs

FSM	art décomp	Secode	Jedi	Nova -i	Nova -io	one-hot
bbara	9	7	11	13	14	15
bbtas	4	4	5	3	4	6
beecnt	7	6	9	8	9	12
dk15	7	12	11	13	13	17
dk17	6	10	11	9	11	17
dk27	3	3	3	4	3	6
lion	2	2	2	3	2	3
s8	1	1	1	1	1	9
$\sum$	39	45	53	54	57	85

The tables contain the number of LUT cells used to implement the given FSM in two different FPGA devices; one with only five-input, one-output cells, the other having also four-input, two-ouput cells. As can be seen, the symbolic functional decomposition method and the algorithms proposed in this paper yield the best overall results.

6. PROBLEMS WITH THE EXISITING ALGORITHMS

There are two main problems with the existing algorithms: the lack of input selection heuristic and the problem of independent blanket construction.

The first problem is the exhausitve nature of the general algorithm. The approach described above assumes all of the possible U and V set combinations are tested; this is a feasible solution for small FSMs (and proves that the method yields better results than others), but the process of symbolic decomposition of larger FSMs takes much more time than the classic, two-step solutions. The computational complexity of blanket construction grows with the number of states (the actual growth factor depending on the algorithm); at the same time, the number of possible U and V set combiantions grows with the factorial of the number of inputs (as well as the factorial of the FPGA architecture's inputs), which makes the algorithms unsuitable for medium and large FSMs.

The second problem is that the construction possibilities (especially regarding the number of required pins) of the β_{Q_V} blanket depend heavily on the construction of the β_G blanket, which, in turn, depends heavily on the construction of the β_{Q_U} blanket. The β_{Q_V}/β_G dependency is (at least to some extent) addressed by the bipainting algorithm mentioned above; still, the possibilities of the construction of both blankets depend heavily on β_{Q_U}, and β_{Q_U} is not being constructed in a way that would optimize β_G and β_{Q_V}.

The following sections of this paper address these two problems.

7. INPUT SELECTION HEURISTICS

As mentioned in the previous section, the current algorithms simply run the decomposition process on all possible combinations of the U and V sets. The number of these combinations grows with the factorial of the number of binary FSM inputs (as well as with the factorial of the width of the FPGA architecture); thus, a heuristic for input selection is needed to consider the method feasible for larger FSMs.

7.1. BLANKET SEPARATIONS

Both algorithms described below depend on the common idiom of separations – on one hand, required by the β_F blanket; on the other, provided by the input blankets.

The β_F blanket can be interpreted as a requirement for the existence of certain separations between the FSM's transition table's rows. For example, $\beta_F = \{\overline{1,2}; \overline{3}; \overline{4,5}\}$ requires the separation of 1 from both 4 and 5, but does not require the 4|5 separation.

At the same time, the input blankets can be interpreted as *providing* certain separations. If $\beta_{x_1} = \{\overline{1,2}; \overline{3,4,5}\}$ and $\beta_{x_2} = \{\overline{1,2,3,4}; \overline{5}\}$, then β_{x_1} provides both 1|4 and 1|5 separations (and does not provide the unnecessary 4|5 separation), while β_{x_2} provides the 4|5 separation, but does not separate 1 from 4.

In general, the above β_F requires the following eight separations: 1|3, 1|4, 1|5, 2|3, 2|4, 2|5, 3|4 and 3|5. β_{x_1} provides six of them (1|3, 1|4, 1|5, 2|3, 2|4 and 2|5), while β_{x_2} only two (1|5 and 3|5); the other two separations provided by β_{x_2} – 2|5 and 4|5 – are not required by β_F. Thus, all of β_{x_1}'s separations (and only half β_{x_2}'s) are *substantial.*

Clearly, in this simple example, x_1 is a better choice for the U set (as this means β_{Q_U} and β_G will only have to provide 3|4 and 3|5). On the other hand, if $U = \{x_2\}$, then β_{Q_U} and β_G will have to provide the six separations required by β_F and not provided by β_{x_2}; what's more, $U = \{x_1, x_2\}$ wouldn't be much better than $U = \{x_1\}$, because β_{x_1} already provides half of β_{x_2}'s substantial separations.

7.2. SIMPLE ALGORITHM

The simple algorithm for heuristic input selection first computes the separations required by β_F and then computes the ones provided by each of the inputs. Next, every input is assigned the number of substantial separations (i.e., the number of separations that are also required by β_F). Once these values are computed for each of the inputs, the general usefulness of every input can be estimated.

Thanks to the above metric, instead of testing every possible combination of U and V sets, the general decomposition algorithm can now start testing the most promising combinations first by putting the "most useful" inputs in the U set and the "least useful" (hopefully also "most compressible") inputs in the V set.

With this metric, the x_1 input (with β_{x_1} providing six substantial separations) would be roughly three times more useful when put in the U set than the x_2 input (with β_{x_2} providing only two substantial separations).

7.3. ADVANCED ALGORITHM

A more through (at the cost of higher computational complexity) algorithm can go a step further when creating the input metric.

As shown above, x_1 is roughly three times more useful than x_2 when being the sole element of the U set – but if x_1 is already chosen to be a part of U, then adding x_2 there brings only a single additional substantial separation, as 1|5 is already provided by x_1. Thus, a better metric for the inputs would be one that computes the usefulness of a given input based on the ones already selected for the U set.

This metric can be computed in two ways. An iterative way would first compute the general metric for all of the inputs and select the best one of them into the U set; next, it would re-compute the metric for the rest of the inputs, disregarding the substantial separations already provided by the U set.

Another approach would be to compute all of the substantial separations provided by the input blankets, and then try to select a group of inputs which, *as a whole*, provides the largest number of substantial separations.

7.4. REVERSE APPROACHES

Note that both algorithms can be reversed, and the *least* useful inputs can be chosen first for the V set. This could be a better solution for cases when the number of inputs of the target FPGA architecture (which bounds the size of the V set) is much smaller than the number of inputs of the FSM.

For the advanced algorithm, this approach would additionally simply remove any inputs that do not provide unique substantial separations.

7.5. SIMILARITY METRIC

Another metric that should yield a good input selection heuristic would be one based on the similarity between sets of substantial separations provided by inputs.

If two inputs provide similar sets of substantial separations, then at least one of the inputs might be a good candidate for the V set. If the number of substantial separations is small, then neither input is very useful and both should go into the V set; if the number of substantial separations is significant, it should suffice if only one of the inputs is an element of the U set – the other input, being similar, doesn't provide many *additional* significant separations.

8. LOOK-AHEAD BLANKET CONSTRUCTION

The other problem with the existing algorithms is the fact that the β_{Q_U} blanket construction algorithms are based solely on the separations required by the β_F blanket (and not provided by the β_U blanket), and does not take into account the impact a given β_{Q_U} blanket's final form has on the subsequent construction of the β_G and β_{Q_V} blankets. (A similar dependency between the β_G and β_{Q_V} blanket construction is at least partially addressed by the bipainter algorithm proposed in [12].)

In most situations there are several forms of the β_{Q_U} blanket providing a given set of substantial separations and implementable on a given number of binary pins, but yielding very different optimization possibilities for the β_G (and, subsequently, β_{Q_V}) blanket optimization. The current algorithms take the first generated β_{Q_U} blanket, and discard the whole decomposition if its counterpart β_G and β_{Q_V} blankets do not yield a sensible result.

This problem can be addressed by a new algorithm for the β_{Q_U} blanket, one which takes into account β_{Q_U}'s impact on the construction possibilities of the β_G blanket (the same algorithm could be also used to construct the β_G blanket in a way that optimises β_{Q_V} construction).

8.1. CLIQUE-BASED BLANKET CONSTRUCTION

The general idea behind the β_{Q_U}, β_G and β_{Q_V} construction is to create blankets that provide as many substantial separations as required (or possible), while at the same time being implementable on as few binary pins as possible. The latter basically means keeping the number of blocks smaller or equal to a power of two, as the number of binary pins required to implement a blanket equals base-two lograrithm of the number of its blocks (rounded up).

After examining the incompatibility graphs used for β_G and β_{Q_V} construction, it can be seen that the the number of blocks in these blankets cannot be smaller than the clique number of the graphs: only disjoint vertices can be merged (or colored with the same color) to form a single block, so all of the vertices of the largest clique in the graph have to belong to separate blocks in the generated blanket.

Because every block merge of the initial β_Q blanket (during the β_{Q_U} construction) creates additional edges in the β_G's incompatibility graph, two β_{Q_U} merges yielding the same substantial separations outcome (and, of course, the same block count) can yield two very different clique outcomes for the β_G incompatibility graph. If the β_{Q_U} graph was constructed not only based on the substantial separations it provides and the number of its blocks, but also based on the size of the largest clique in the subsequent β_G incompatibility graph, a given decomposition could be implementable on fever blocks; in some situations, a sensible decomposition could be obtained where the current algorithms do not yield one.

The β_{Q_U} construction algorithms proposed previously start with β_{Q_U} equal to β_Q and iteratively merge β_Q's blocks to produce smaller blankets while losing as few substantial separations as possible. Contrarily, the clique-based algorithm starts with a single-block (i.e., fully merged) β_{Q_U} blanket, which corresponds to the densest β_G incompatibility graph. The algorithm then iteratively *introduces* new separations by splitting the blanket's blocks (initially, the single block) in a way that both makes it provide the most substantial separations and makes it cut the largest cliques in the corresponding β_G icompatibility graph; this reduces the lower bound for the number of blocks in the final version of β_G. As the number of binary pins required by a blanket is equals the rounded-up, base-two logarithm of the number of its blocks, the new separations can be introduces in increasing quantities: first the initial single block can be split into two, but then another two blocks can be introduced in one iteration, and the next one can introduce further four blocks.

9. CONCLUSIONS

The results obtained with the existing algorithms implementing the symbolic functional decomposition method prove that the method yields better results than the currently widespread, two-step approaches to finite state machine implementations in FPGA devices. Unfortunately, these algorithms lack any input selection heuristic, which makes them unsuitable for use with larger FSMs. At the same time, the β_{Q_U} blanket construction algorithms do not yield blankets optimised for the subsequent construction of β_G and β_{Q_V}, which is supposed to further improve the results.

This paper addressed both of these issues by describing various algorithms for the input selection heuristic, as well as an algorithm for β_{Q_U} construction targeted at minimising the clique number of the β_G blanket's incompatibility graph. The implementation of these algorithms should enable the application of the symbolic decomposition method to arbitrarily-sized finite state machines and yield even better results.

10. ACKNOWLEDGEMENTS

This paper was supported by the Ministry of Science and Higher Education of Poland – research grant no. N N517 030735 for 2008-2009.

11. REFERENCES

1. J. A. Brzozowski, T. Łuba: *Decomposition of boolean functions specified by cubes.* Journal of Multiple-Valued Logic and Soft Computing, 9:277–417, 2003.
2. G. de Micheli, R. K. Brayton, A. Sangiovanni-Vincentelli: *Optimal state assignment for finite state machines.* IEEE Trans. on CAD, pp. 269–284.
3. L. Jóźwiak, A. Ślusarczyk: *A new state assignment method targeting FPGA implementations.* Proc. EUROMICRO Symposium on Digital System Design DSD 2000, pp. 50–59.
4. B. Lin, A. R. Newton: *Synthesis of multiple level logic from symbolic high-level description languages.* Proc. of IFIP Int. Conf. on VLSI, pp. 187–196.
5. M. Rawski: *The novel approach to FSM synthesis targeted FPGA architectures.* Proceedings of IFAC Workshop on Programmable Devices and Systems PDS 2004, pp. 169–174, 2004.
6. M. Rawski, L. Jóźwiak, T. Łuba: *Functional decomposition with an efficient input support selection for sub-functions based on information relationship measures.* Journal of Systems Architecture, (47):137–155, 2001.
7. M. Rawski, H. Selvaraj, T. Łuba, P. Szotkowski: *Multilevel synthesis of finite state machines based on symbolic functional decomposition.* International Journal of Computational Intelligence and Applications, 6(2):257–271, 2007.
8. C. Scholl: *Functional Decomposition with Application to FPGA Synthesis.* Kluwer Academic Publishers, 2001.
9. A. Ślusarczyk: *Decomposition and Encoding of Finite State Machines for FPGA Implementation.* Technische Universiteit Eindhoven, 2004.
10. P. Szotkowski, M. Rawski: *Symbolic functional decomposition algorithm for FSM implementation.* Proceedings of the International Conference on Computer as a Tool EUROCON 2007, pp. 484–488, 2007.
11. P. Szotkowski, M. Rawski: *A graph-based symbolic functional decomposition algorithm for FSM implementation.* To be published in proceedings of the Conference on Human System Interaction HSI 2008, 2008.
12. P. Szotkowski, M. Rawski, H. Selvaraj: *A graph-based approach to symbolic functional decomposition of finite state machines.* To be published in proceedings of the International Conference on Systems Engineering ICSEng 2008, 2008.
13. T. Villa, A. Sangiovanni-Vincentelli: *Nova: state assignment of finite state machines for optimal two-level logic implementation.* IEEE Trans. on CAD, pp. 905–924.

New Trinomials $X^n + X + 1$ and $X^n + X^2 + 1$ Irreducible over $GF(2)$*

PIOTR BARTOSIK, ANDRZEJ PASZKIEWICZ

pbartos@elka.tele.pw.edu.pl
anpa@tele.pw.edu.pl
Institute of Telecommunications, Warsaw University of Technology

Received 2009.01.05
Authorized 2009.03.30

We extend the limit of investigations for trinomials irreducible over $GF(2)$, having the form $X^n + g(X)$, where $\deg(g(X)) = 1$ or $\deg(g(X)) = 2$ and complete the existing list of irreducible trinomials with that form by a dozen of new elements. We checked all degrees n below 500000 while searching for that polynomials. A large part of computations were performed by a new programming package developed especially for computations in finite fields with characteristic two. This package is a bit more than twice faster than Shoup's NTL package for trinomials and about six times faster than NTL in the case of pentanomials. We also complete the list of Mersenne irreducible polynomials for which a trinomial does not exist by pentanomials and irreducible polynomials which are lexicographicaly youngest.

Keywords: Finite fields, binary fields, sparse irreducible polynomials over finite fields, primitive polynomials, irreducible trinomials

1. INTRODUCTION

Irreducible trinomials with coefficients over small number fields are perfect candidates for effective implementations of algorithms using the finite field arithmetic. They are in particular willingly applied in fast algorithms of coding [2], [3] as well as in modern ciphers for example in ciphers based on elliptic curves geometry [7], [11], [14]. Also classical stream ciphers [6] have in natural way as their base the arithmetic of irreducible polynomials over the simplest two element field $GF(2)$. Recent applications in cryptography forces applying of large finite fields $GF(2^n)$ with n

* This paper was supported by the Ministry of Science and Higher Education of Poland - research grant no. N517 003 32/0583 for 2007-2010

being about ten thousand or higher, because of which many authors presented tables containing irreducible trinomials. Blake, Gao and Lambert [4] explored the existence of irreducible trinomials up to degree 5000. Seroussi [11] extended the computation of irreducible trinomials up to degree 10000. The second author generated for each n below 10000 an irreducible trinomial and pentanomial [8]. Recently the computations were extended up to degrees n not exceeding 30000 [9].

For security reasons in applications which are close to cryptography the degrees n of irreducible trinomials are chosen as a prime number. If n is the index of Mersenne prime number then each irreducible polynomial of degree n is also primitive. This is a good reason to search for irreducible Mersenne polynomials, especially with only few nonzero coefficients. Unfortunately in general case for about one half degrees n an irreducible trinomial does not exist. The second best way in that case is to choose irreducible pentanomials that is polynomials with exactly five nonzero coefficients. As it has been pointed out by the second author [10], the irreducible pentanomials always exist (at least for degrees n not exceeding 30000) and the number of such polynomials of degree n seem to be a quadratic function of n. Taking this in mind we were looking for irreducible pentanomials in such cases, as well as irreducible polynomials for which the power of the second monomial with nonzero coefficient is as small as possible. Such polynomials are called lexicographicaly youngest.

Irreducible over $GF(2)$ trinomials of the form $X^n + X + 1$ were investigated by N. Zierler [13]. He generated a table containing all 33 values of $n \leq 30000$ for which such a trinomials exist. H. Fredricksen R. Wisniewski [5] listed the 19 values of n, for which the trinomial $X^n + X^2 + 1$ is irreducible over $GF(2)$. The main result of that work is the following fact.

COROLLARY. Let $X^n + X^k + 1$ be irreducible over $GF(2)$ with $n \equiv 3$ or $n \equiv 5$. If e is the smallest positive integer for which $X^n + X^2 + 1$ divides the binomial $X^e + 1$ (e is called a period of the trinomial X^n+X^2+1 and 2^n-1 is divisible by e) then $X^{nr}+X^{2r}+1$ is irreducible for all numbers r having all of its prime factors divisors of the period and none of its prime factors divisors of the index of the trinomial $X^n + X^2 + 1$, where by index we mean the number $(2^n - 1) / e$.

Other words if $n \equiv 3$ or $n \equiv 5$, $2^n - 1 = e \cdot q_1^{\alpha_1} q_2^{\alpha_2} ... q_s^{\alpha_s}$, where e is period of the trinomial $X^n + X^2 + 1$ and $\gcd(e, q_i) = 1$, $(i = 1, 2, ..., s)$ then $X^{nr} + X^{2r} + 1$ is irreducible for all integers $r = q_1^{\beta_1} q_2^{\beta_2} ... q_s^{\beta_s}$ for nonnegative integers β_i $(i = 1.2, ..., s)$.

Our main goal was to extend existing tables of irreducible over $GF(2)$ polynomials of the form X^n+X+1 and X^n+X^2+1 taking the advantage of new computer technology and new package, developed for fast arithmetic in finite fields by the first author. The ideas underlying the construction and some applications of that package were earlier described in our joint paper [1].

d tables
xistence
putation
r each n
utations

egrees n
lersenne
This is
ith only
egrees n
choose
ents. As
s always
nomials
looking
or which
possible.

gated by
or which
es of n,
t of that

5. If e is
+ 1 (e is
$+X^{2r}+1$
e period
l, where

d of the
educible

nomials
hnology
hor. The
e earlier

2. METHOD OF APPROACH AND RESULTS

Trinomials $X^n + X + 1$ and $X^n + X^2 + 1$ can be effectively sieved by applying the following deep result due to Swan [12]. It has to be pointed out that the result of Swan was stated earlier by Stickelberger.

THEOREM. (Stickelberger-Swan) Let $0 < k < n$. The trinomial $X^n + X^k + 1$ has an even number of factors over $GF(2)$ in each of the following cases

a. n is even and k is odd, $n \neq 2k$ and $(nk/2) \equiv 0$ or $1 \pmod 4$;

b. n is odd, k is even, $k \nmid 2n$ and $n \equiv \pm 3 \pmod 8$, or

c. n is odd, k is even $k|2n$ and $n \equiv \pm 1 \pmod 8$.

Simple conclusions which can be deduced from the Stickelberger-Swan theorem are as follows:

- For n divisible by 8 a trinomial irreducible over $GF(2)$ does not exist;
- There are no irreducible trinomials $X^n + X + 1$ for $n \equiv 2 \pmod 8$;
- $X^n + X^k + 1$ with $n \equiv \pm 3 \pmod 8$ and k even can be irreducible only for $k|2n$. If k is odd we use $n - k$ instead of k;
- For n a prime number, $n = 13 \pmod{24}$ or $n = 19 \pmod{24}$ there are no irreducible trinomials of degree n.

By detailed study the discriminant of other polynomials types one can derive similar to the above Stickelberger-Swan criterions describing parity of the number of irreducible factors.

Table 1

Integers n <500000 such that the trinomial $X^n + X + 1$ is irreducible over $GF(2)$

2	3	4	6	7
9	15	22	28	30
46	60	63	127	153
172	303	471	532	865
900	1366	2380	3310	4495
6321	7447	10198	11425	21846
24369	27286	28713	**32767**	**34353**
46383	**53484**	**62481**	**83406**	**87382**
103468	**198958**	**248833**		

Table 2

Integers n <500000 such that the trinomial $X^nX^2 + 1$ is irreducible over $GF(2)$

5	11	21	29	35
93	123	333	845	4125
10437	10469	14211	20307	34115
47283	50621	57341	**70331**	**80141**

Detailed analysis of tables 1 and 2 show us that the rate of both types of trinomials still decreases. In our previous paper [1] where we analyzed irreducible over $GF(2)$ polynomials $f(X) = X^n + g(X)$, such that degree of $g(X)$ is as small as possible positive integer. These polynomials were called lexicographicaly young and the degree of polynomial $g(X)$ we called internal degree. For increasing values of n the probability that one can find irreducible polynomial of degree n and small internal degree is a monotone falling function. The situation can be illustrated by the following Figure 1.

One can suppose that for a given integer m the probability that there exist an irreducible polynomial with degree n and internal degree m asymptotically tends to zero as n tends to infinity. It is also visible that the number of irreducible over $GF(2)$ polynomials with a given lowest internal degree and degrees n in for consecutive disjoint intervals of degrees has an escaping maximum. For example in the interval [1,10000] the maximum is for internal degree equal to 12 (see curve S1), in the interval [10001,20000] the maximum is for 13 (curve S2), and in the interval [20001,30000] (curve S3) the maximum is located at the point 14. The general situation is illustrated by the curve S4 which is related to the whole interval [1,30000].

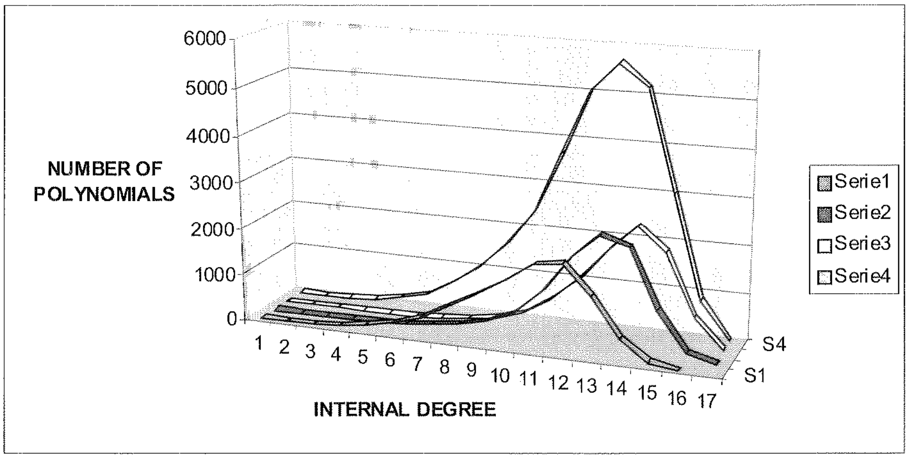


Fig. 1. The number of irreducible over $GF(2)$ polynomials with a given lowest internal degree and degrees n in the interval [1, 30000]. The curve S1 illustrates the interval [1,10000], the curve S2 – interval [10001,20000], the curve S3 – interval [20001,30000], the curve S4 – the interval [1,30000]

As it has been remarked at the beginning there is a special interest to consider irreducible trinomials of degrees being exponents of Mersenne prime numbers. Not for every Mersenne prime exponent n an irreducible trinomial exist. For that reason we generated for all exponents n up to 216091 for which Mersenne number $M_n = 2^n - 1$ is prime, one irreducible pentanomial and one irreducible polynomial with lowest internal degree. The results are illustrated in the Table 3.

Table 3

Searching results of different types irreducible Mersenne polynomials

Degree n	Trinomials	Pentanomial	Young
2	1	–	1,0
3	1	–	1,0
5	2	3,2,1,0	2,0
7	1;3	3,2,1,0	1,0
13	–	4,3,1,0	4,3,1,0
17	3;5;6	3,2,1,0	3,0
19	–	5,2,1,0	5,2,1,0
31	3;6;7;13	3,2,1,0	3,0
61	–	5,2,1,0	5,2,1,0
89	38	6,5,3,0	6,5,3,0
107	–	9,7,4,0	7,5,3,2,1,0
127	1;7;15;30;63	7,3,1,0	1,0
521	32;48;158;168	9,7,2,0	9,6,5,3,1,0
607	105;147;273	12,9,7,0	9,7,6,3,1,0
1279	216;418	16,8,3,0	11,9,8,5,3,2,1,0
2203	–	14,6,5,0	11,10,6,4,1,0
2281	715;915;1029	20,18,4,0	9,8,7,6,2,0
3217	67;576	21,9,7,0	11,10,9,8,6,5,4,3,2,0
4253	–	21,12,11,0	12,10,7,5,4,0
4423	271;369;370;649;1393;1419;2098	24,5,2,0	14,12,10,9,6,5,1,0
9689	84;471;1836;2444;4178	21,19,16,0	13,11,10,8,3,2,1,0
9941	–	29,12,10,0	9,6,5,4,1,0
11213	–	19,11,10,0	10,9,7,5,4,2,1,0
19937	881;7083;9842	29,27,21,0	13,8,7,5,2,0
21701	–	33,26,2,0	14,12,10,9,6,0
23209	1530;6619;9739	55,49,48,0	13,12,9,8,7,6,2,0
44497	8575;21034	28,20,9,0	15,12,10,9,7,5,3,2,1,0
86243	–	54,39,9,0	16,14,13,12,10,9,7,4,1,0
110503	25230;53719	40,12,4,0	18,13,11,10,7,6,2,0
132049	7000;33912;41469;52549;54454	43,36,14,0	15,14,10,9,8,7,3,2,1,0
216091	–	36,28,22,0	14,13,12,11,6,4,1,0

The time devoted for finding the following irreducible polynomial with nine nonzero coefficients

$$f_{110503}(X) = X^{110503} + X^{18} + X^{13} + X^{11} + X^{10} + X^{7} + X^{6} + X^{2} + 1$$

which is lexicographicaly youngest took about 8 hours on a home computer of the first author.

3. SOME CONCLUSIONS

Irreducible over $GF(2)$ trinomials play important role in coding theory, cryptography, telecommunications, informatics and several other areas. Therefore investigating these irreducible polynomials is interesting. We extended existing tables of the simplest trinomials of the form $X^n + X + 1$ and $X^n + X^2 + 1$, finding twelve new (See Tables 1 and 2. The new elements are written in bold letters). All degrees n up to 500000 were tested with the aim to find new trinomials with internal degree equal to 1 or 2. The distribution of that polynomials does not show any peculiarities distinguishing them from standard behavior of irreducible polynomials which are lexicographicaly young (see Figure 1). Existing of an irreducible over $GF(2)$ polynomial with small internal degree and very large degree is rather rare phenomena.

There is also interesting to study irreducible Mersenne polynomials. If $M_n = 2^n - 1$ is prime for some n, then every irreducible over $GF(2)$ polynomial of degree n is also primitive. The last feature gives great advantages in representing elements of finite fields as powers of the simplest nontrivial element of a Galois Field. Table 3 collects primitive polynomials of Mersenne degrees – trinomials (if exist), pentanomials and polynomials lexicographicaly young.

The investigations can be easily extended, say to degrees 1000000 by distributed computing methods.

4. REFERENCES

1. P. Bartosik, A. Paszkiewicz: *A study on irreducible polynomials of high degrees over GF(2), tools and results*, Telecommunication Review, Telecommunication News, 12(2008), pp. 1059-1065, (in polish).
2. E. R. Berlekamp: *Algebraic coding theory*, McGraw-Hill, New York, 1968.
3. R. Blahut: *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, Reading, Massachusetts, Repr. with Correction 1984.
4. I. F. Blake, S. Gao, R. J. Lambert: *Construction and Distribution Problems for Irreducible Trinomials over Finite Fields*; in D. Gollman (ed.) Applications of Finite Fields, Clarendon Press, Oxford (1996) pp. 19-32.
5. H. Fredricksen and R. Wisniewski: *On trinomials* $x^n + x^2 + 1$ and $x^{8j\pm1} + x^k + 1$ Irreducible *over* $GF(2)$, Inform. and Control 30, 58-63 (1981).
6. S. W. Golomb: Shift Register Sequences, Holden Day, San Francisco, 1967, Reprinted by Aegean Park.

7. A. J. Menezes et al.: *Handbook of Applied Cryptography*, CRC Press, Boca Raton, New York 1997.
8. A. Paszkiewicz: *Some observations concerning irreducible trinomials and pentanomials over* Z_2, Tatra Mountains Publications 32 (2005), pp. 129-142.
9. A. Paszkiewicz: *On some properties of irreducible trinomials over small number fields*, (a paper being recently in press).
10. A. Paszkiewicz: *Irreducible pentanomials and their applications to effective implementations of arithmetic in binary fields*, (this issue).
11. G. Seroussi: *Table of Low-Weight Binary Irreducible Polynomials.* Hewlett-Packard, HPL, pp. 98-135, August 1998.
12. R. G. Swan: *Factorization of polynomials over finie fields*, Pacific J. Math. 12, 1099-1106.
13. N. Zierler: *On* $x^n + x + 1$ *over* $GF(2)$, Inform. and Control 16, 502+505 (1970).
14. NIST. *FIPS 186-2 draft, Digital Signature Standard (DSS)*, 2000;

Ir

T
and r
been

*
of Pol

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 363–375

Irreducible Pentanomials and their Applications to Effective Implementations of Arithmetic in Binary Fields*

ANDRZEJ PASZKIEWICZ

Institute of Telecommunications, Warsaw University of Technology
anpa@tele.pw.edu.pl

Received 2009.01.07
Authorized 2009.03.26

There are only few main classes of irreducible polynomials which are used for designing arithmetic in Galois Fields with characteristic two. These are: irreducible trinomials, pentanomials, all-one polynomials (AOP) and equally spaced irreducible polynomials (ESP). The most critical and time consuming arithmetical operations in Galois Fields are multiplication and modular reduction. A special structure of the modular polynomial defining the arithmetic allows significant speedup of these operations. The best class of binary irreducible polynomials are trinomials, but for about one half of degrees below 30000 an irreducible trinomial does not exist. By exhaustive computation we established that for all degrees n between 4 and 30000 an irreducible pentanomial always exists. Therefore using irreducible pentanomials for defining the arithmetic of Galois Fields have practical interest. In the paper we investigate a function describing the number of binary irreducible pentanomials of a given degree n greater than 3 and study its properties. We also analyze the complexity of a circuit (the number of XOR and AND gates) implementing multiplication in the finite field represented by general irreducible pentanomials.

Keywords: Finite fields, binary fields, sparse irreducible polynomials over finite fields, primitive polynomials, irreducible trinomials and pentanomials

1. INTRODUCTION

Trinomials and pentanomials over finite fields are polynomials with exactly three and respectively five nonzero terms. Their computational advantages have frequently been pointed out by several authors [3], [6], [7], [10]. Primitive and irreducible poly-

* The computational part of his paper was supported by the Ministry of Science and Higer Education of Poland – research grant no. N517 003 32/0583 for 2007-2010

nomials which are sparse (polynomials with only few nonzero terms) play important role in coding theory [1], [2] and cryptography [6], [8], [10], and in particular in elliptic curve cryptography. The application in cryptography should by secure and because of it, large finite fields are preferred. It is especially important in the case of binary fields $GF(2^n)$, for which fast and efficient algorithms of evaluating discrete logarithms were designed [5] and implemented [4], [9]. The reasonable solution in the case of fields $GF(2^n)$ relays on constructing very large finite fields with 2^{10000} elements or higher. It is a common practice to choose an irreducible trinomial for representing finite field, providing that one exists. The reduction operation can be very effective speeded up in these cases. If an irreducible trinomial of a given degree n does not exist, then the next best polynomials are pentanomials. It has to be pointed out that trinomials and pentanomials are recommended by IEEE(2000) – the Standard Specification for Public-Key Cryptography (Technical Report IEEE Std. 1363-2000, Institute of Electrical and Electronics Engineers, Inc., 3 Park Avenue, New York, NY 10016-5997 USA).

Important operations in finite fields are addition, multiplication, division and computing multiplicative inverse. The first operation is very simple and can be implemented by using of a simple XOR circuit. Multiplication is the most important and complicated one. Other operations such as exponentiation, division and multiplicative inverse can be performed by computing multiplication iteratively. It is evident that efficient multiplier architectures are especially crucial for the speed of operations and lots of architectures for implementing a multiplier have been proposed in the literature [8].

Several authors presented tables containing irreducible trinomials or pentanomials useful to implement arithmetic in binary fields [3], [6], [7], [10]. Blake, Gao and Lambert [3] explored the existence of irreducible trinomials up to degree 5000. Seroussi extended the computation of irreducible trinomials up to degree 10000. Important remark is that below 10000 there exist 5148 (a bit higher than one half) numbers for which an irreducible trinomial exist. For those degrees not exceeding 10000 for which we do not have an irreducible trinomial there always exist an irreducible pentanomial. Paszkiewicz [7] generated for each n below 10000 an irreducible trinomial and pentanomial. Also all irreducible trinomials up to degree 4000 have been generated. In the same paper the functions $t_5(n)$ and $T_5(n)$ were investigated where $t_5(n)$ denote the number of irreducible pentanomials of degree n, while $T_5(n)$ is the number of all irreducible pentanomials of degree not exceeding n.

Recently Paszkiewicz[1] extended the computations reported in [7] and [10] of all irreducible trinomials up to degree n not exceeding 30000. The numerical experiments revealed some computational errors in the paper [3] which remained unnoticed in the later paper [10]. It has also been generated for each n belonging to the interval [10000, 30000] one irreducible randomly chosen pentanomial. There exist

[1] A. Paszkiewicz, *On some properties of irreducible trinomials over small number fields*, (a paper being recently in press).

5123 irreducible trinomials of degree in the interval [10001,20000] and the same number of irreducible trinomials of degree in the interval [20001,30000]

Most of the architectures introduced so far in the literature are benefited from using special irreducible polynomials which greatly reduces the complexity of the multiplication. The most important types are irreducible trinomials, pentanomials, all-one polynomials (AOP) and equally spaced polynomials (ESP). We define the four classes below.

Trinomial: $f(X) = X^n + X^m + 1$;
Pentanomial: $f(X) = X^n + X^m + X^k + X^l + 1$;
AOP: $f(X) = X^n + X^{n-1} + X^{n-2} + ... + X + 1$;
ESP: $f(X) = X^n + X^{(k-1)d} ... + X^{2d} + X^d + 1$.

It is evident that AOP are a special case of ESP – polynomials. The problem with these polynomials is that they are not available for many degrees n. For example there are only 67 values of n below 1000 for which an irreducible AOP of degree n exists. By more advanced theory one can prove, that for large real number x, the number of all values of n not exceeding x for which an irreducible AOP of degree n exists is not greater than $A\frac{x}{\ln x}$ where A is called Artin's constant and is approximately equal to 0,3739558136... .

As we mentioned before irreducible trinomials and pentanomials are most popular in several applications. It is not an accident that from the five polynomials suggested by NIST for Elliptic Curve Digital Signature Algorithm [11] two are trinomials and the other three are pentanomials. Therefore the study of arithmetic based on pentanomials are of practical interest.

2. MODULAR MULTIPLICATION BASED ON PENTANOMIALS

Let $f(X) = X^n + X^m + X^k + X^l + 1$ be a general irreducible pentanomial and $A(X)$, $B(X)$ are two elements of the field $GF(2^n)$. They can be represented as two polynomials of degree at most n-1 with binary coefficients. A classical multiplication of two polynomials over a field with two elements consists of 2 steps. The first of them is an "ordinary" multiplication. We obtain a polynomial $C(X)$, of degree at most $2n$-2 where $C(X) = A(X) \cdot B(X)$ and

$$C(X) = \sum_{j=0}^{n-1} a_j X^j \sum_{k=0}^{n-1} b_k X^k = \sum_{m=0}^{2n-2} (\sum_{j=0}^{m} a_j b_{m-j}) X^m. \tag{1}$$

In the second step we reduce the polynomial $C(X)$ by the basis polynomial $f(X)$ to obtain a polynomial $D(X)$, where

$$D(X) = C(X)(\text{mod } f\ (X)). \tag{2}$$

This step is totally dependent on the choice of the modular polynomial $F(x)$. By above notations the equality (2) leads to the following equations:

$$X^{n+i}=\begin{cases} X^{i+l}+X^{i+k}+X^{i+n}, (0 \le i \le n-m); \\ X^{i}+X^{i+l}+X^{i-n+m}+X^{i-n+m+l}+X^{i-n+m+k}+X^{i-n+2m}, (n-m \le i < n-k); \\ X^{i}+X^{i+l}+X^{i-n+k}+X^{i-n+k+l}+X^{i-n+2k}+X^{i-n+m}+X^{i-n+m+l}+X^{i-n+2m}, \\ (n-k \le i \le n-l); \\ X^{i}+X^{i-n+l}+X^{i-n+2l}+X^{i-n+k}+X^{i-n+2k}+X^{i-n+m}+X^{i-n+2m}, (n-l \le i < n-2). \end{cases} \quad (3)$$

One can prove that the total number AND and XOR gates in a circuit performing the modular multiplication via the above scheme (3) are as follows:

$$\begin{aligned} \#\text{ AND GATES } &= n^2; \\ \#\text{ XOR GATES } &= (n-1)^2, \end{aligned}$$

while the

$$\text{Total Delay } = T_{AND} + ceil(\log_2 n) \cdot T_{XOR},$$

where $ceil(x)$ in the above formula is defined as the smallest integer greater than or equal to x.

The fundamental and crucial problem of the existence of at least one irreducible pentanomial of a given degree n greater than 3 over the field with two elements still remain unsolved. Trying to answer to that question we performed a vast computation for searching at least one irreducible pentanomial of degree n over $GF(2)$ up to large degrees.

3. EXHAUSTIVE SEARCH FOR BINARY IRREDUCIBLE PENTANOMIALS

Denote by $t_5(n)$ the number of irreducible pentanomials of a given degree n over $GF(2)$. Let $T_5(n)$ by the number of all irreducible binary pentanomials with degrees not exceeding n.

$$T_5(n) = \sum_{k=4}^{n} t_5(k) \quad (4)$$

Nowadays it is not known if the function $t_5(n)$ has positive values for all integers n greater than 3. A spare contribution to resolve that question are computations performed by the author. For every $n \le 800$ we determined the value of $t_5(n)$ and for all other $800 < \le 30000$ we found exactly one irreducible pentanomial. The initial part of this job was performed in my earlier paper [7] where we established that $t_5(n)$ has positive values for $n \le 10000$. All earlier and recent computations were performed in a small local computer network consisting of about 10 Pentium PC computers working on Microsoft Windows operation system. During recent as well as earlier computations we used the following simple and well known algorithm for verifying the irreducibility of a given binary polynomial $f(x)$ of degree n.

Algorithm for testing irreducibility
of the polynomial $f(X)$ over $GF(2)$

```
$A(X) \leftarrow X$
for $j \leftarrow 1$ to $n$ do
 {
   $A(X) \leftarrow A(X)^2 \bmod f(X)$
   if $GCD(A(X) + X, f(X)) \not\equiv 1$
      then return "reducible"
 }
if $A(X) = X$ then return "irreducible"
elsereturn "reducible"
```

Additionally we determined the value of the $t_5(n)$ for some isolated values n=1279 and n=2203, which as it easily can be seen are indices of two consecutive Mersenne prime numbers. The value $t_5(n)$ for n=2203 has been found by P. Bartosik while testing his package for effective operations on binary polynomials. We have $t_5(1279)$=1411790 and $t_5(2203)$=2027566 for these values of n.

4. COMPUTATIONAL RESULTS

We present results of all computations in short, compact form. On Figures 1-17 we illustrate the behavior of the function $t_5(n)$, while the Fig. 18 shows the graph of the function $T_5(n)$.

It is interesting to remark that the general graph of the function $t_5(n)$ (see Fig. 1 below) is a superposition of several independent graphs. The graph on the bottom of Fig. 1 corresponds to all arguments n divisible by 8.

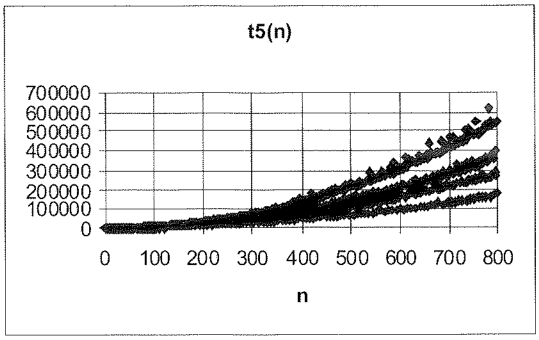


Fig. 1. Graph of the function $t_5(n)$

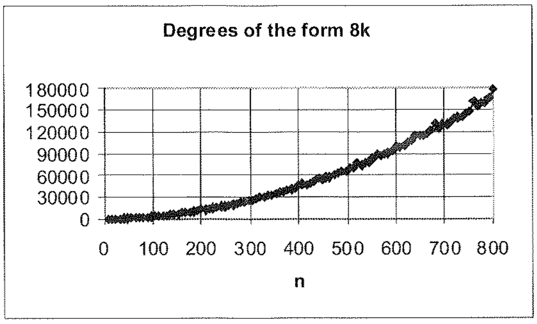


Fig. 2. Graph of the function $t_5(n)$, $n = 8k$

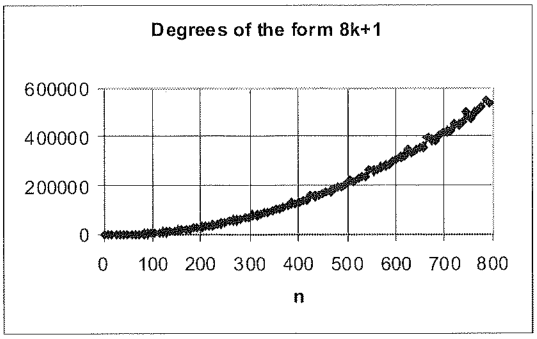


Fig. 3. Graph of the function $t_5(n)$, $n = 8k + 1$

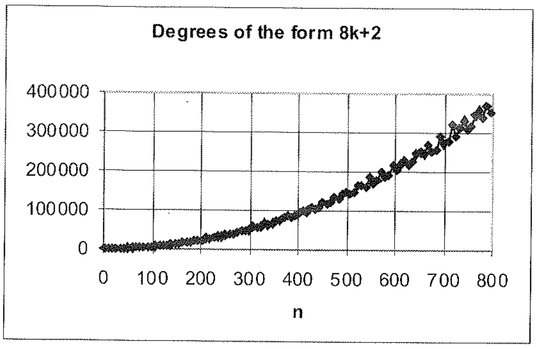


Fig. 4. Graph of the function $t_5(n)$, $n = 8k + 2$

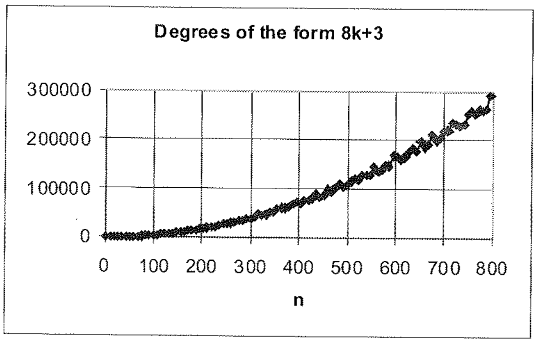


Fig. 5. Graph of the function $t_5(n)$, $n = 8k + 3$

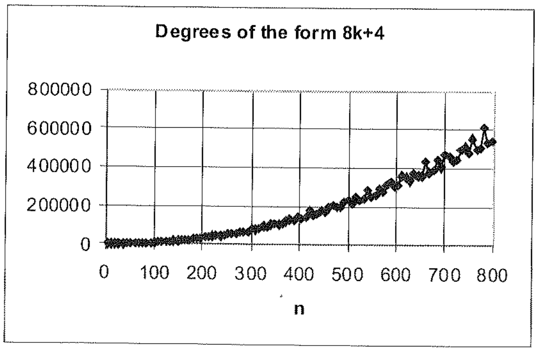


Fig. 6. Graph of the function $t_5(n)$, $n = 8k + 4$

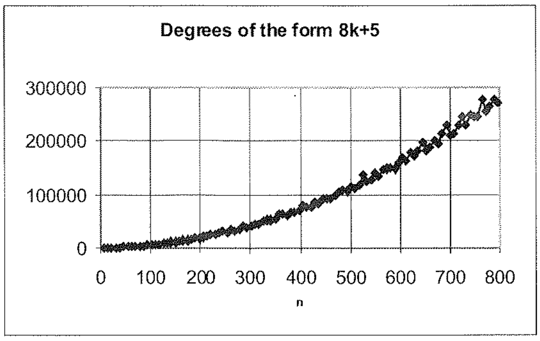


Fig. 7. Graph of the function $t_5(n)$, $n = 8k + 5$

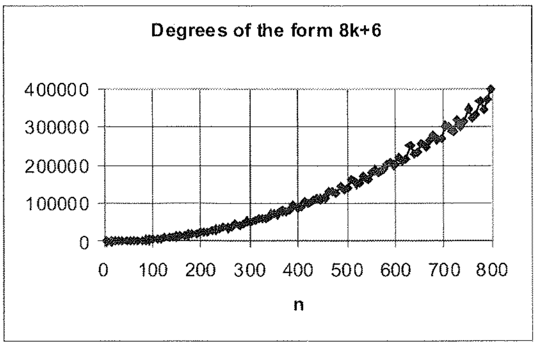


Fig. 8. Graph of the function $t_5(n)$, $n = 8k + 6$

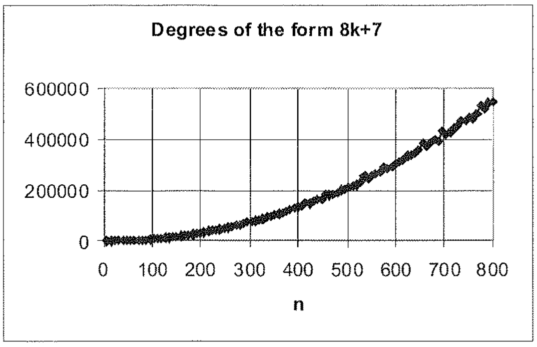


Fig. 9. Graph of the function $t_5(n)$, $n = 8k + 7$

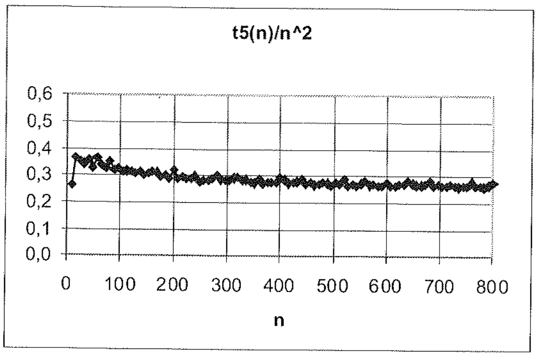


Fig. 10. Graph of the function $t_5(n)/n^2$, $n = 8k$

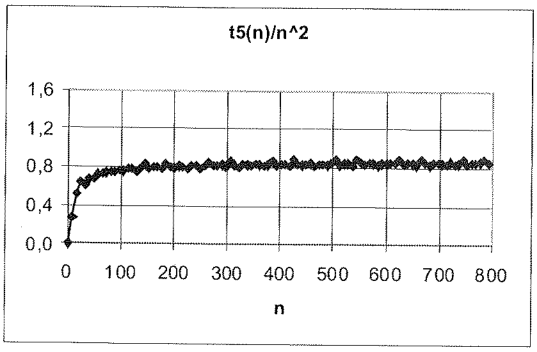


Fig. 11. Graph of the function $t_5(n)/n^2$, $n = 8k + 1$

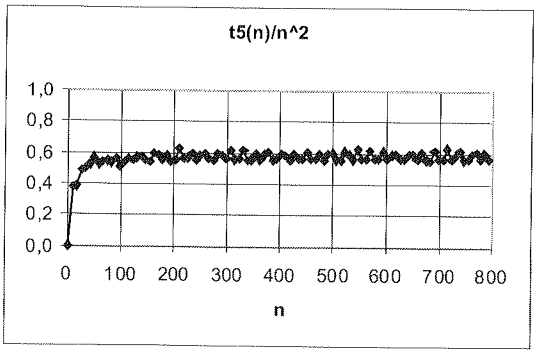


Fig. 12. Graph of the function $t_5(n)n^2$, $n = 8k + 2$

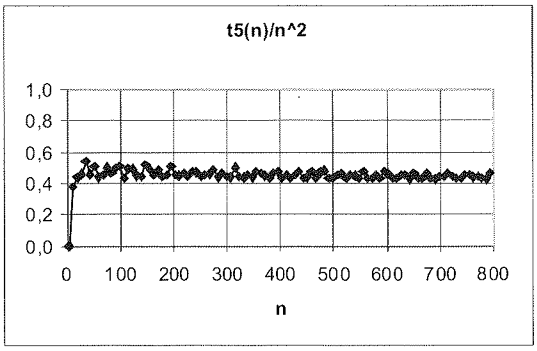


Fig. 13. Graph of the function $t_5(n)/n^2$, $n = 8k + 3$

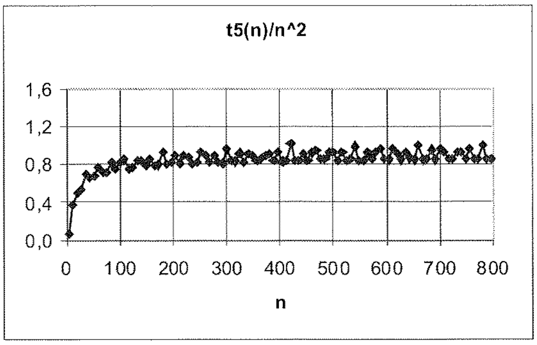


Fig. 14. Graph of the function $t_5(n)/n^2$, $n = 8k + 4$

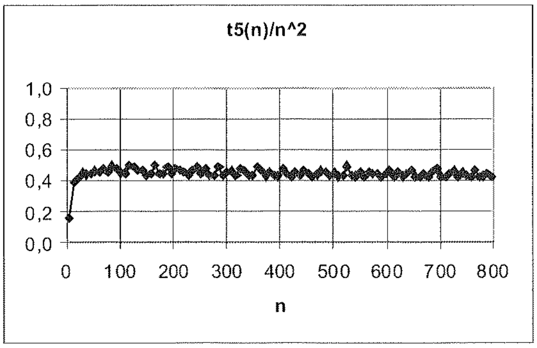


Fig. 15. Graph of the function $t_5(n)/n^2$, $n = 8k + 5$

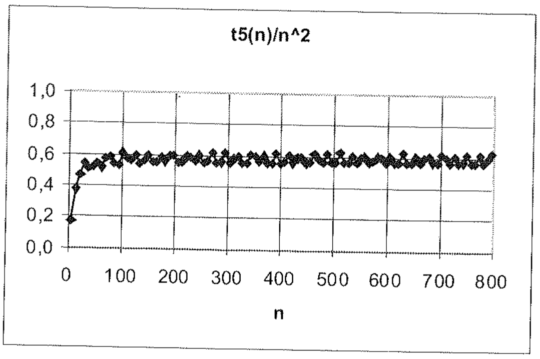


Fig. 16. Graph of the function $t_5(n)/n^2$, $n = 8k + 6$

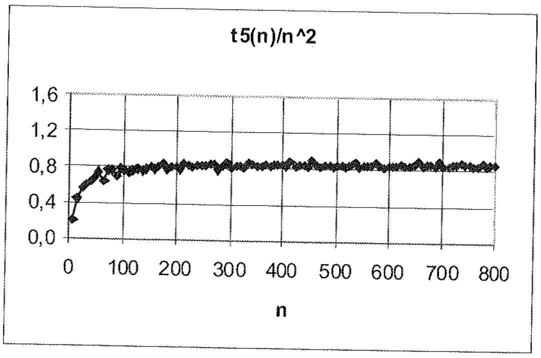


Fig. 17. Graph of the function $t_5(n)/n^2$, $n = 8k + 7$

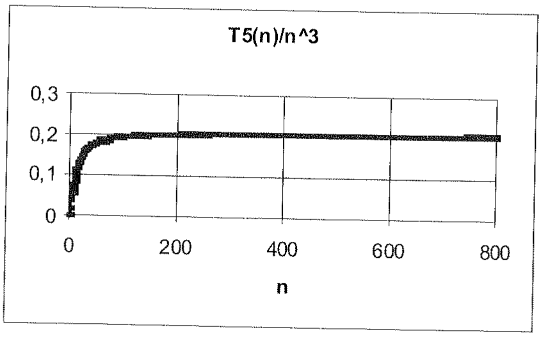


Fig. 18. Graph of the function $T_5(n)/n^3$,

5. CONCLUSIONS

Irreducible pentanomials over $GF(2)$ play important role in developing circuits for effective implementations of binary arithmetic. We established by computations the following facts:

1. For all $4 \leq n \leq 30000$ there exist at least one irreducible pentanomial of degree n over $GF(2)$;
2. Graph of the function $t_5(n)$ defining the number irreducible pentanomials of degree n over $GF(2)$ (see Fig. 1-17) splits into several independent sub-graphs;
3. The function $t_5(n)$ is a quadratic function of the variable n depending on the rest class modulo 8;
4. The function $T_5(n)$ defining the number of all irreducible pentanomials of degree not exceeding n over $GF(2)$ is similar to the cubic function of n;
5. For the values 1279, congruent to 7 modulo 8 and 2203, congruent to 3 modulo 8 we have $t_5(1279)/1279^2 = 0{,}86303$ and $t_5(2203)/2203^2 = 0{,}41777$ respectively. This remains with excellent agreement with results of our computations for small arguments of the function $t_5(n)$ (see Figure 17 and Figure 13 for comparison).

6. ACKNOWLEDGEMENTS

The author is very grateful to Piotr Bartosik for computing the value of $t_5(2203)$ within the scope of a students project. Thanks are also due to Mrs. Danuta Ojrzeńska Wójter for free access to the Divisions Laboratory during weekends.

7. REFERENCES

1. E. R. Berlekamp: *Algebraic coding theory*, McGraw-Hill, New York, 1968.
2. R. Blahut: *Theory and Practice of Error Control Codes*, Addison-Wesley Publishing Company, Reading, Massachusetts, Repr. with Correction 1984.
3. I. F. Blake, S. Gao, R. J. Lambert: *Construction and Distribution Problems for Irreducible Trinomials over Finite Fields*; in D. Gollman (ed.) Applications of Finite Fields, Clarendon Press, Oxford (1996) pp. 19-32.
4. M. Borowski: *Application of the Rivest-Chor Cipher for Securing Information in Special Telecommunication Networks*, PhD Thesis, MUT, Warszawa 1995 (in polish).
5. D. Coppersmith: Fast Evaluation of Logarithms in Fields of Characteristic Two, IEEE Trans. Inform. Theory, vol. IT-30, pp. 587-594, 1984.
6. A. J. Menezes et al.: *Handbook of Applied Cryptography*, CRC Press, Boca Raton, New York 1997.
7. A. Paszkiewicz: *Some observations concerning irreducible trinomials and pentanomials over* Z_2, Tatra Mountains Publications 32 (2005), pp. 129-142.
8. F. Rodriguez-Henriques, C. K. Koc: *Parallel Multipliers Based on Special Irreducible Polynomials*, IEEE Trans. On Computer, 52(12), Dec. 2003, pp. 1535-1542.
9. M. Sadowski: *Analysis and Implementation of the Coppersmith's Algorithm for Finding Discrete Logarithms in* $GF(2^n)$, Master Thesis, WUT, Warszawa, 2006 (in polish).

10. G. S e r o u s s i: *Table of Low-Weight Binary Irreducible Polynomials.* Hewlett-Packard, HPL, pp. 98-135, August 1998.
11. NIST. *FIPS 186-2 draft, Digital Signature Standard (DSS)*, 2000;

Op

C
resu
that
of th
radio
spec
sche
are

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 2, pp. 377–387

Optimal Transmission Time of Secondary User in an Overlay Cognitive Radio System

BABAK ABBASI BASTAMI, EBRAHIM SABERINIA

Department of Electrical and Computer Engineering
University of Nevada, Las Vegas, NV 89154 USA
abbasiba@unlv.nevada.edu
Ebrahim.Saberinia@unlv.edu

Received 2009.02.20
Authorized 2009.03.286

Optimal Opportunistic channel access of the unlicensed users has been a major problem in a cognitive radio system. In this paper, we consider an overlay cognitive radio system, where the secondary user senses the channel for an empty slot and transmits a constant power for a time period without sensing the channel again. We obtain the optimal transmission time of the secondary user to achieve the maximum data rate while keeping the interference on the primary user under a given threshold. We derive closed-form expressions for the interference at the primary receiver and the achievable data rate for the secondary user. Our analysis is based on a Markov model for the primary user active and idle times and we consider the probability of error in sensing by the secondary user. Computer simulation results of the system show the validity of our analysis.

Keywords: cognitive radio, overlay system, interference analysis, imperfect sensing

1. INTRODUCTION

Current scheme of allocating radio frequency bands for different wireless services resulted in an inefficient system. Recent measurements in the United States have shown that 70% of the allocated spectrum is not fully utilized [1]. To improve the utilization of the radio spectrum, a different allocation scheme has been proposed using cognitive radio systems [1]. In such a system, cognitive or secondary users share the licensed spectrum opportunistically with the primary users that hold the license. While this scheme has the capability to increase the overall utilization of the spectrum, there are several challenges that should be answered before it can be implemented. The main B. A. BASTAMI, E. SABERINIA E.T.Q. challenge is to control the amount of interference on the primary user (PU) caused by the secondary user (SU). Generally, two main approaches have been proposed to control the interference on the PU. In the spectrum overlay scenario, the SU access the spectrum whenever it senses that the PU is idle. The PU can transmit at any time and the cognitive user should have the ability to monitor the channel status and decide whether to transmit or not. On the other hand, in the spectrum underlay technique, the secondary user can transmit at any time, but the power spectral density (PSD) of the transmitted signal should be low enough, preferably at noise level, for small interference on the PU. However, even in the overlay scheme, channel sensing is used to increase the capacity of the SU. Using channel information, a power control scheme can be designed for the SU such that it maximizes its transmission capacity while keeping the interference on the PU below a threshold [2]. On the other hand, a perfect overlay system can have zero interference. This requires the SU to have the capability to detect the channel status without any error. Furthermore, it should have the ability to detect immediately a PU transition from idle to active and suspend its own transmission. Designing such a system is very complicated. In practical scenarios, we have to consider some possibility of sensing errors for the SU. Furthermore, we can assume that the SU transmits its signal for a limited period of time without sensing once it detects a free channel [3]. This means that there will be some interference on the PU. In this paper, we study this interference and design a system that maximizes the secondary user'fs capacity while keeping the interference below a threshold. We consider an overlay cognitive radio system where the SU may have error in sensing the channel. Also we assume that the SU does not perform any sensing when it is in transmission mode. We derive closed-form expressions for the interference on the PU and the achievable data rate of the SU. Our main objective is to maximize the data rate of the secondary receiver under the primary user interference limit constraint and derive the optimal secondary user busy time duration in terms of the primary user timing parameters and probabilities of imperfect sensing. We assume that the primary user active and idle mode durations obey an exponential distribution like Markovian models.

Some recent studies consider similar problems in overlay cognitive radio systems. In [5], the interference and capacity of the SU are analyzed assuming errorless sensing by the SU. The idle and the active durations of the PU have been modeled as exponential random variables. Extension of [5] to a general distribution for the idle and busy times of the PU is presented in [3]. In this paper, we analyze the interference on the primary user and the capacity of the secondary user considering the possibility of error in sensing. In [4], an analysis has been done for the outage capacity of the secondary user taking into account the possibility of sensing errors. However, the work in [4] does not cover the amount of interference on the PU.

The paper is organized as follows: in section 2, we introduce the system model. The formulas for the interference on the PU and the achievable data rate of the SU are obtained in section 3. We also discuss the optimization problem in this section.

In section 4, we provide the simulation results and compare them with the derived analytical results. Section 5 concludes the paper.

2. SYSTEM MODEL

We consider a wireless communication system where the primary users can be inactive for some portion of the time. The busy and idle periods of the primary channel are modeled with two random variables τ_1 and τ_2 respectively. The idle period, τ_1, is assumed to have exponential distribution. The length of the transmitted packet of the PU is usually considered as a random variable with a long tail distribution. Hence, exponential distribution would be a good choice for the busy period, τ_2, as well [3]. Therefore, the distribution function for the τ_1 and τ_2, $f(\tau_i)$, $i = 1, 2$, can be written as:

$$f(\tau_i) = \lambda_i \exp(-\lambda_i t), \tag{1}$$

where, $\lambda_1 1 = \dfrac{1}{T_{\mathrm{OFF}}}$ and $\lambda_2 = \dfrac{1}{T_{\mathrm{ON}}}$ and T_{OFF} and T_{ON} are respectively the average PU idle and active durations.

The secondary user successively senses the channel until it detects that the channel is in idle mode. Then, it transmits a packet for duration of T. We assume that during this transmission time the SU cannot sense the channel. To be more general in our analysis, we do not assume perfect sensing. The probability of incorrect sensing by the SU when the PU is idle is assumed to be P_{fa} (probability of false alarm) and the probability of the incorrect sensing when the PU is busy is assumed to be P_m (probability of miss detection). Let's denote the sensing duration of the SU with T_s. We assume that the value of T_s is small compared to T_{OFF}, T_{ON} and T.

The value of the transmission time, T, is a system design parameter. It affects two important system performance parameters. The first performance parameter is the amount of interference on the PU from the SU. Since the SU does not perform any sensing during its transmission period, it is probable that the PU starts transmitting within the transmission time of the SU. Apparently the larger the transmission time T, the higher is the probability of the interference. The second performance parameter which is affected by T is the bit rate of the SU. The longer the SU transmits once it detects an idle channel, the higher its achievable data rate is. In this paper, our goal is to find the optimal transmission time T, in the sense that it provides the highest bit rate for the SU while keeping the interference on the PU below some threshold. Our analysis of the system is based on the alignment of the time line of the SU compared to the time line of the primary user. Figures 1(a) and 1(b) show typical time lines of the primary and secondary users. The PU alternates between idle and busy periods, but the secondary user time line shows different behavior. After each sensing interval of the SU, we may have a transmission interval or another sensing interval based on the output of the sensing information. On the other hand, after any transmission interval we definitely have a sensing interval. In other words, the switching between sensing and transmission time follows a Markov model which makes analysis complicated. Simpler analysis is possible, if we change our point of view of the secondary user time line.

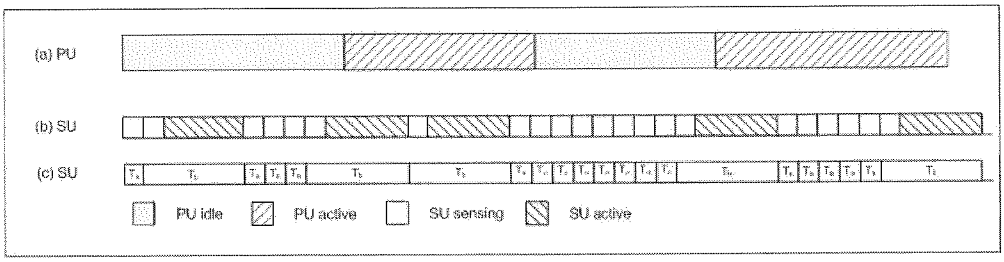


Fig. 1. Typical time lines of the primary and secondary users

Let T_a be a sensing interval that follows with another sensing interval and T_b be a combination of a sensing interval that follows with a transmission interval and the transmission interval itself. Therefore, T_a has a duration equal to T_s and T_b has a duration equal to $T + T_s$. Figure 1(c) shows this alternate point of view of the SU time line. With this point of view, the intervals T_a and T_b can follow each other independently. Meaning that at any point of time, regardless of what type of interval we had before, we can have either an interval of T_a or T_b. Suppose P_a and P_b represent the probability of the occurrence of each of the intervals type T_a and T_b respectively. During a total time of T_{Total}, the average portions of the time which are occupied with T_a and T_b types of intervals would be $\frac{P_aT_a}{P_aT_a + P_bT_b}T_{\text{Total}}$ and $\frac{P_bT_b}{P_aT_a + P_bT_b}T_{\text{Total}}$ respectively. Therefore, the average number of the types T_a and T_b intervals during T_{Total} would be:

$$n_{T_a} = \frac{P_aT_{\text{Total}}}{P_aT_a + P_bT_b} \tag{2}$$

and

$$n_{Tb} = \frac{P_bT_{\text{Total}}}{P_aT_a + P_bT_b} \tag{3}$$

respectively. We will use these equations in our analysis of interference and bit rate in the following sections.

3. SYSTEM PERFORMANCE EQUATIONS

In this section we evaluate the system performance equations based on the system model parameters described in the previous section.

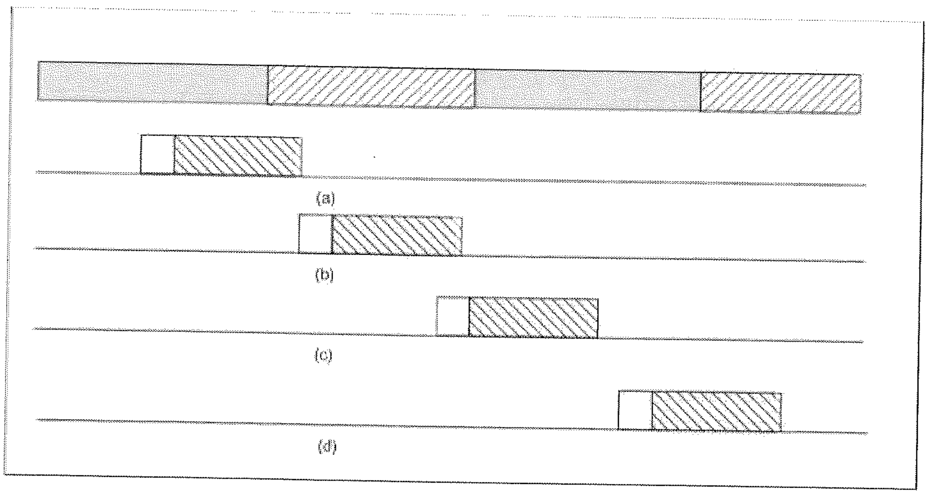


Fig. 2. Joint timing between the primary and the secondary user

3.1. INTERFERENCE ANALYSIS OF THE PRIMARY USER

The interference on the PU is proportional to the overlapping time in which both PU and SU are simultaneously transmitting. In other words we have:

$$I_p = K_1 T_{\text{ov}} \tag{4}$$

where, I_p is the expected value of the interference and T_{ov} is the expected value of the overlapping time. Constant K_1 denotes the interference per unit of the overlapping time and depends on the power spectral density of the SU transmitted signal and the distance between the primary receiver and the secondary transmitter.

In order to calculate the overlapping time we consider different scenarios of conflict in the system. Figure 2(a) shows a scenario where the PU starts transmission while the SU is in transmission period. In this case, the sensing by the SU was done when the primary user was in idle mode. Suppose t_{Ls1} is the point in time where the last sensing has ended for the SU during previous PU idle duration. Apparently, $t_{\text{Ls1}} < \tau_1 < t_{\text{Ls1}} + T$. If we define $\tau_1' = \tau_1 - t_{\text{Ls1}}$ as a random variable that represents the remaining time of the PU idle period, the probability that τ_1' is less than an arbitrary value Δ can be calculated as:

$$\begin{aligned} &P(\tau_1' < \Delta) = P(\tau_1 - \tau_{\text{Ls1}} < \Delta | \tau_{\text{Ls1}} < \tau_1 < \tau_{\text{Ls1}} + T) \\ &= \frac{1 - \exp(-\Delta/T_{\text{OFF}})}{1 - \exp(-T/T_{\text{OFF}})} \qquad 0 \le \Delta \le T \end{aligned} \tag{5}$$

Equation (5) represents the cumulative distribution function (CDF) of τ_1'. The overlapping time in this scenario can be written in terms of τ_1', τ_2 and T as follows:

$$\tau_{\text{ov1}} = \begin{cases} T - \tau_1^{'} & T - \tau_1^{'} \leq \tau_2 \\ \tau_2 & T - \tau_1^{'} > \tau_2 \end{cases} \tag{6}$$

Evaluating the expectations in (6), we obtain

$$T_{\text{ov1}} = E(\tau_{\text{ov1}}) = T_{\text{ON}} \frac{T_{\text{OFF}} \exp(-T/T_{\text{OFF}}) - T_{\text{ON}} \exp(-T/T_{\text{ON}}) + (T_{\text{ON}} - T_{\text{OFF}})}{(T_{\text{ON}} - T_{\text{OFF}})(1 - \exp(-T/T_{\text{OFF}}))} \tag{7}$$

Another scenario of overlapping active transmission times is where the SU starts transmitting when the primary channel was already busy (Figure 2(b)). This scenario only happens if there is a wrong sensing by the SU during the busy interval of the PU. Let τ_{ov2} collectively indicates the overlapping time during the busy interval of the PU other than the overlapping time in the previous scenario. We can use Equations (2) and (3) to calculate the expected value of τ_{ov2}. When the PU is busy, type T_a interval happens when the sensing by the SU is correct and type T_b interval happens when the sensing by the SU is incorrect. Therefore, $P_a = 1 - P_m$ and $P_b = P_m$. For a given τ_2, the total duration within which the secondary user may make a wrong decision causing interference is $\tau_2 - \tau_{\text{ov1}}$. Therefore, the total numbers of T_a and T_b intervals are:

$$\begin{aligned} N_a &= \frac{(1 - P_m)(\tau_2 - \tau_{\text{ov1}})}{(1 - P_m)T_a + P_m T_b} \\ N_b &= \frac{P_m(\tau_2 - \tau_{\text{ov1}})}{(1 - P_m)T_a + P_m T_b} \end{aligned} \tag{8}$$

Noting that a sensing time (Ts) exists in both T_a and T_b intervals, the total average time of sensing during the time $\tau_2 - \tau_{\text{ov1}}$ would be $(N_a + N_b)T_s$. Hence, the average overlapping time in this case would be

$$\begin{aligned} \tau_{\text{ov2}} &= \tau_2 - \tau_{\text{ov1}} - (N_a + N_b)T_s \\ &= \tau_2 - \tau_{\text{ov1}} - \frac{(\tau_2 - \tau_{\text{ov1}})}{(1 - P_m)T_a + P_m T_b} T_s. \end{aligned} \tag{9}$$

Therefore, the expected value of this overlapping time would be

$$T_{\text{ov2}} = E(\tau_{\text{ov2}}) = T_{\text{ON}} - T_{\text{ov1}} - \frac{(T_{\text{ON}} - T_{\text{ov1}})}{(1 - P_m)T_a + P_m T_b} T_s. \tag{10}$$

Considering the overlapping times in the two scenarios, the total expected value of the overlapping time would be:

$$T_{\text{ov}} = T_{\text{ov1}} + T_{\text{ov2}} = T_{\text{ON}} - \frac{\left(T_{\text{ON}} - T_{\text{ON}} \frac{T_{\text{OFF}} \exp(-T/T_{\text{OFF}}) - T_{\text{ON}} \exp(-T/T_{\text{ON}}) + (T_{\text{ON}} - T_{\text{OFF}})}{(T_{\text{ON}} - T_{\text{OFF}})(1 - \exp(-T/T_{\text{OFF}}))}\right)}{(1 - P_m)T_a + P_m T_b} T_s. \tag{11}$$

3.2. DATA RATE ANALYSIS OF THE SECONDARY USER

The data rate of the SU is proportional to the amount of time that the SU transmits without overlapping with the PU. Let $\tau_{\rm nov}$ denotes the none-overlapping time in which the secondary user is transmitting within the idle period of the PU. The data rate, C_s, of the SU is

$$C_s = K_2 T_{\rm nov}, \tag{12}$$

where $T_{\rm nov} = E(\tau_{\rm nov})$ is the expected value of the non-overlapping time. Constant K_2 denotes the data rate per unit of the non-overlapping time and depends on factors such as modulation type and symbol duration of the secondary user. In order to evaluate the data rate of the SU, first, consider the scenario shown in Figure 2(c), where the PU switches to its idle mode and the SU is still transmitting because of a bad sensing result when the PU was busy. Suppose $\tau_{\rm Ls2}$ is the last sensing end point of the SU during τ_2 and let $\tau_2' = \tau_2 - \tau_{\rm Ls2}$. Since $\tau_{\rm Ls2}$ is the end time of the last sensing interval, we should have $\tau_{\rm Ls2} < \tau_2 < \tau_{\rm Ls2} + T$. Therefore, the probability that τ_2' is less than an arbitrary value Δ can be calculated as:

$$\begin{aligned} P(\tau_2' < \Delta) &= P(\tau_2 - \tau_{\rm Ls2} < \Delta | \tau_{\rm Ls2} < \tau_2 < \tau_{\rm Ls2} + T) \\ &= \frac{1 - \exp(-\Delta/T_{\rm ON})}{1 - \exp(-T/T_{\rm ON})} \qquad 0 \le \Delta \le T \end{aligned} \tag{13}$$

Equation (13) represents the distribution of τ_2'. The non-overlapping time then depends on the τ_2', τ_1 and T as follows

$$\tau_{\rm nov1} = \begin{cases} T - \tau_2' & T - \tau_2' \le \tau_1 \\ \tau_1 & T - \tau_{21}' > \tau_1 \end{cases} \tag{14}$$

Manipulating (14), we have

$$\begin{aligned} &T_{\rm nov1} = E(\tau_{\rm nov1}) = \\ &T_{\rm OFF} \frac{T_{\rm ON} \exp(-T/T_{\rm ON}) - T_{\rm OFF} \exp(-T/T_{|rmOFF}) + (T_{\rm OFF} - T_{\rm ON})}{(T_{\rm OFF} - T_{\rm ON})(1 - \exp(-T/T_{\rm ON}))}. \end{aligned} \tag{15}$$

Now consider the scenario as shown in Figure 2(d) in which the SU senses and transmits during the idle interval of the PU. Let $\tau_{\rm nov2}$ indicates the collective non-overlapping time during the idle period of the PU other than the non-overlapping time in previous scenario. When the PU is idle, type T_b interval happens when the sensing by the SU is correct and type T_a interval happens when the sensing by the SU is incorrect. Therefore $P_b = 1 - P_{fa}$ and $P_a = P_{fa}$. Given τ_1, the total duration within which the secondary user may make a correct decision causing interference is equal to $\tau_1 - \tau_{\rm nov1}$. Therefore, the total numbers of T_a and T_b intervals are:

$$
\begin{aligned}
N'_a &= \frac{P_{fa}(\tau_1 - \tau_{\text{nov1}})}{P_{fa}T_a + (1 - P_{fa})T_b} \\
N'_b &= \frac{(I - P_{fa})(\tau_1 - \tau_{\text{nov1}})}{P_{fa}T_a + (1 - P_{fa})T_b}.
\end{aligned}
\tag{16}
$$

Since the sensing time exists in both T_a and T_b intervals, the average number of sensing during the time $\tau_1 - \tau_{\text{nov1}}$ would be $(N'_a + N'_b)T_s$. Hence, the non overlapping time is equal to:

$$
\begin{aligned}
\tau_{\text{nov2}} &= \tau_1 - \tau_{\text{nov1}} - (N'_a + N'_b)T_s \\
&= \tau_1 - \tau_{\text{nov1}} - \frac{(\tau_1 - \tau_{\text{nov1}})}{P_{fa}T_a + (1 - P_{af})T_b}T_s.
\end{aligned}
\tag{17}
$$

The expected value of this non-overlapping time is equal to

$$
T_{\text{nov2}} = E(\tau_{\text{nov2}}) = T_{\text{OFF}} - T_{\text{nov1}} - \frac{(T_{\text{OFF}} - T_{\text{nov1}})}{P_{fa}T_a + (1 - P_{fa})T_b}T_s. \tag{18}
$$

Hence, the resulting total expected value of the non-overlapping time would be

$$
\begin{aligned}
&T_{\text{nov}} = T_{\text{nov1}} + T_{\text{nov2}} = \\
&T_{\text{OFF}} - \frac{\left(T_{\text{OFF}} - T_{\text{OFF}}\frac{T_{\text{ON}}\exp(-T/T_{\text{ON}}) - T_{\text{OFF}}\exp(-T/T_{\text{OFF}}) + (T_{\text{OFF}} - T_{\text{ON}})}{(T_{\text{OFF}} - T_{\text{ON}})(1 - \exp(-T/T_{\text{ON}}))}\right)}{P_{fa}T_a + (1 - P_{fa})T_b}T_s
\end{aligned}
\tag{19}
$$

3.3. OPTIMAL TRANSMISSION TIME OF THE SECONDARY USER

Our objective is to obtain the value of the SU busy period, T, which maximizes the data rate of the SU and maintains the average interference on the PU receiver smaller than a target value. We assume that the average values of the busy and idle durations of the PU, the sensing information probabilities and the length of the sensing period of the SU are given known values. It can be easily verified that both T_{nov} and T_{ov} are increasing function respect to T. Hence, the optimum value of T which maximizes the SU data rate for a given interference threshed η is equal to $T_{\text{ov}}^{-1}(\eta)$.

4. SIMULATION RESULTS

In order to verify our analysis, we have performed a simulation of the system described in section 2 using MATLAB. In our simulations, the mean values of the PU idle and busy periods are assumed to be $T_{\text{TON}} = 1$ and $T_{\text{OFF}} = 2$ which gives the ratio of $\alpha_1 = 66\%$ for the idle period. The false alarm and miss detection probabilities of the sensing process are assumed to be 0.2. The sensing duration of the SU is set to 0.01. We run two systems simultaneously and calculate overlapping busy time of both as well as busy time of the secondary user without overlapping with the primary user.

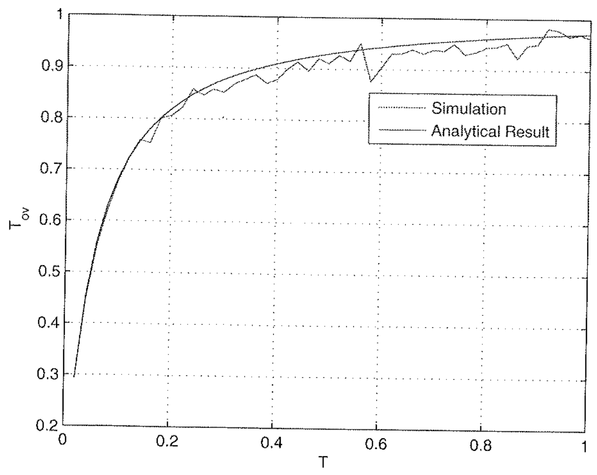


Fig. 3. The average overlapping time of the Pu and SU busy periods as an indicator of the average interference on the PU

Figure 3 shows the simulation results for the overlapping busy period as a function of the SU transmission time and compares it with the analytical results of equation (11).

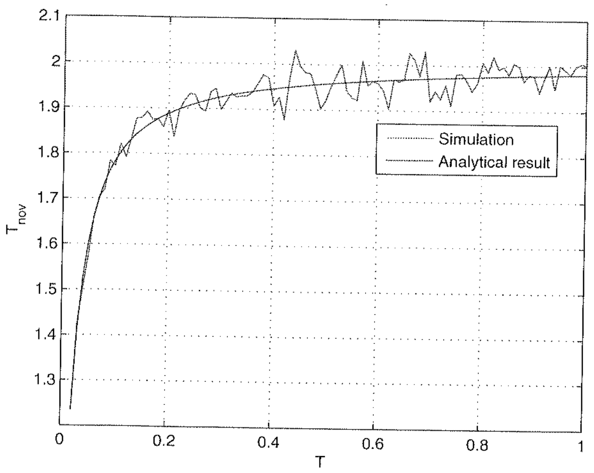


Fig. 4. The average non-overlapping time when the SU is transmitting in the idle period of the PU as an indicator of the data rate of the SU

Figure 4 shows the simulation result for the busy time of the SU without overlapping with the PU and compares it with the analytical results of equation (19). The simulation results have a very good agreement with the analytical results.

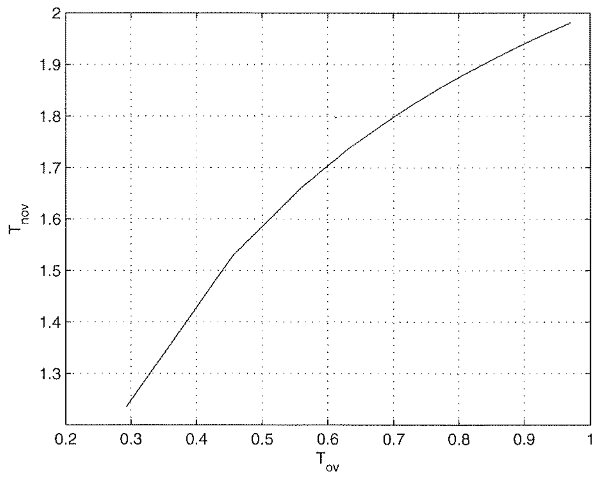


Fig. 5. The relationship between the system performance parameters

Figure 5 shows the relationship between the maximum data rate of the secondary user with the interference threshold of the primary user. For a given threshold on the interference and system parameters, we can calculate maximum overlapping time. Using Figure 5, we can find the corresponding transmission time of the SU without overlapping with the PU which can be translated to the total data rate of the secondary user.

5. CONCLUSION

We calculated the optimal transmission time of the secondary user in a practical overlay cognitive radio system. In our assumed system, the secondary user does not sense the channel when it is transmitting. There is also a non-zero probability of error in sensing. We derived closed-form expressions for the interference on the primary receiver and the achievable data rate of the secondary user. The maximum data rate of the cognitive user under the primary user interference constraint has been derived. Our obtained performance expressions are in a promising agreement with the simulation results.

6. REFERENCES

1. S. H a y k i n: *Cognitive radio: Brain-empowered wireless communications*, IEEE J. Selected. Areas of Communications, Vol. 23, no. 2, pp. 201-220, February 2005.
2. K. H a m d i, W. Z h a n g, K. B e n L e t a i e f: *Power Control in Cognitive Radio Systems Based on Spectrum Sensing Side Information*, IEEE International Conference on communications, pp. 24-28, June 2007.

3. S. Huang, X. Liu, Z. Ding: *Opportunistic spectrum access in cognitive radio networks*, Proceedings of the 27th IEEE Conference on Computer Communications, pp. 1427-1435, April 2008.
4. R. Urgaonkar, M. J. Neely: *Opportunistic Scheduling with Reliability Guarantees in Cognitive Radio Networks*, Proceedings of the 27th IEEE Conference on Computer Communications, pp. 1301-1309, April 2008.
5. Q. Yang, S. Xu, K. S. Kwak: *Outage Performance of Cognitive Radio with Multiple Receive Antennas*, IEICE Transaction on Communications, Vol. E91.B, pp. 85-94, January 2008.
6. S. Srinivasa, S. A. Jafar: *Soft Sensing and Optimal Power Control for Cognitive Radio*, IEEE Global Telecommunications Conference, pp. 1380-1384, November 2007.
7. S. Srinivasa, S. A. Jafar: *Cognitive Radio Networks: How Much Spectrum Sharing is Optimal?*, IEEE Global Telecommunications Conference, pp. 3149-3153, November 2007.
8. W. S. Jeon, D. G. Jeong: *An efficient quiet period management scheme for cognitive radio systems*, IEEE Transactions on Wireless Communications, Vol. 7, issue 2, pp. 505-509, February 2008.
9. Q. Zhao, L. Tong, A. Swami: *Decentralized cognitive Mac for Dynamic spectrum access*, First IEEE International Symposium on New Frontiers in Dynamic Spectrum Access Networks, DySPAN 2005, pp. 224-232, Nov 2005.
10. J. Hillenbrand, T. A. Weiss, F. K. Jondral: *Calculation of detection and false alarm probabilities in spectrum pooling systems*, IEEE Communications. Letters, Vol. 9, no. 4, pp. 349-351, April 2005.
11. R. Etkin, A. Parekh, D. Tse: *Spectrum sharing for unlicensed bands*, First IEEE International Symposium on Dynamic Spectrum Access Networks, DySPAN, pp. 251-258, 2005.
12. Q. Zhao, S. Geirhofer, L. Tong, B. M. Sadler: *Optimal dynamic spectrum access via periodic channel sensing*, in Proc. Wireless Communications and Networking Conference (WCNC), 2007.

XXIIIrd IEEE-SPIE Symposium on Photonics and Web Engineering 30-31 January 2009, Warsaw, FE&IT WUT

Participants of one of sessions during the 23rd IEEE-SPIE Symposium on Photonics and Web Engineering, Faculty of Electronics and Information Technologies, Warsaw University of Technology, FE&IT, WUT, 31.01.2009, in front of the prof. J. Groszkowski bust monument, patron of the Faculty. There are sitting (l to r): mgr Tomasz Czarski, dr Maciej Linczuk, prof. T. Morawski – invited speaker, prof. R. Romaniuk – symposium Chair, dr Krzysztof Poźniak, mgr Arkadiusz Kalicki.

During the weekend days of 30-31 January 2009, at the Faculty of Electronics and Information Technologies, Warsaw University of Technology, there took place the next 23rd Symposium on Advanced Applications of Photonic and Electronic Control and Measurement Systems. The Symposium was attended by over 50 young researchers, a number of them members of IEEE. The young researchers originate from WUT and collaborating institutions like DESY, CERN, Max-Planck Institute, etc. There were presented over 40 research papers. The Symposium official language is English. The Symposium is organized under the eminent patronage of domestic institutions: PSP – Photonics Society of Poland, Committee of Electronics and Telecommunications of Polish Academy of Sciences, FE&IT WUT, as well as international institutions: IEEE-R8, and SPIE-Europe. The Symposium has been organized two times a year, for 12 years. The proceedings are published in the USA in the series of Proc.SPIE, and in Poland as special issues of Elektronika Monthly - a professional journal of the Association of Polish Electrical Engineers, and Electronics and Telecommunications Quarterly – a Journal by Polish Academy of Sciences.

The subject of WILGA Symposium series are advanced applications of photonic and electronic distributed, large, control and measurement systems in high energy physics research, astrophysics of elementary particles, superconducting accelerator technology, laser technology of FEL machines, etc. The Symposium participants usually take part in various large research experiments around the globe like: LHC and CMS, E-XFEL and FLASH, ILC and CLIC, Auger Observatory and Chandrayaan satellite, ALBA, GSI, FAIR and CBM, BESSY, PITZ, and others. Two times a year they meet face to face at home in WILGA to discuss the results and work development. Usually, during the everyday work course, only the video conferences are possible, as well as e-mails. The best of young researchers, who show exceptional skill in the team work and display research creativity have big chances in doing their M.Sc and Ph.D. theses at one of the biggest experiments. Quite a number of PERG/ELHEP students spend their vacations in big European HEP research institutions like DESY in Hamburg and CERN in Geneva, also in Fermilab in Chicago. They participate in summer student programs. The experience gained there is exceptional and uncompared.

International Research Collaboration "Pi-of-the-Sky" (with participation of members from PERG/ELHEP Laboratory) discovered in March 2008 an exceptionally massive gamma ray burst (GRB) with an accompanying optical flash. The GRB was catalogued as 080319. The GRB was probably associated with formation of a super-massive black hole. The distance from the event was estimated for 7,5bln light years, i.e. half of the age of our universe. The flash was visible with a naked eye for a minute. The observation was done by a system of four-coupled wide angle super-sensitive cameras constructed by the collaboration and ELHEP Ph.D. students. The cameras were positioned in ESO's Las Campanas facilities. The discovery was published in NATURE, nr.455, 2008.

International Research Collaboration on CMS – The Compact Muon Solenoid (with participation of PERG/ELHEP members) finished in November 2008 the construction of the detector at LHC in CERN. The ELHEP Laboratory took part in building of the muon trigger. A 300 pages manual on The CMS was published by IOP/SISSA in October 2008. A number of ELHEP members and Ph.D. students are active in the works on the construction and upgrade of the LHC. The work goes on Linac 4, SPS accelerator, LHC booster, also on a new generation of the safety system (Infer Lock) for the LHC.

International research consortium organized around the Indian Moon satellite Chandrayaan-1, with participation of PERG/ELHEP Ph.D. students, and coordinated by Max Planck Institute of Solar Research, has finished work on some apparatus for the satellite.

The satellite was launched and positioned on an orbit around the Moon in December 2008 and started regular measurements. A Ph.D. student from ISE (P. Sitek) participated in construction of the SIR-1 near infrared spectrometer. SIR-1 actually collects measurement data from the Moon's surface.

ELHEP Laboratory (Photonics and Electronics for HEP Experiments) traditionally closely cooperates with a number of institutions in this country which participate in large, international research experiments. These are, among others: Sołtan Institute for Nuclear Studies (IPJ) in Świerk/Otwock, Institute of Experimental Physics at Warsaw University. The young researchers from these institutions actively participate in the WILGA Symposium. Some of the Ph.D. students of WUT are employed at IPJ in order to continue their research and to supplement for modest university Ph.D. fellowship. During their stays with the experiments they are paid per diem for short stays and they get experiment fellowships during longer stays.

The WILGA Symposia, organized in winter - smaller but international and with more focused topical range, and the spring ones, organized during the whole last week of May, much bigger, fully international, play in this country a completely unique role. During the most popular years, the May Symposia gather more than 350 young researchers from this country and from allover Europe. These are very special meetings of young researchers, indeed. The meetings are completely void of any formalities and any idealistic approach. They are devoted only to the science, research, new technologies and the conditions of research work for young scientists in different parts of Europe and IEEE-Region 8. During more than a dozen of years of WILGA Symposium activities, it has gathered a few thousand of young researchers. The results of their work were published in nearly 20 volumes of Proc.SPIE accessible via Internet data bases of the American Institute of Physics, IEEE eXplore, Scitopia, SPIE Digital Library, Amazon, Scopus and others. The young researchers which went through WILGA school may be encountered all over the world in big research experiments and advanced engineering and technology businesses like in: Spain, Italy, England, Switzerland, France, Argentina, Germany, USA, India and other places.

The XXIInd Symposium had a special invited speaker, who was professor Tadeusz Morawski, professor of radioelectronics, a prolific author of research literature, but also a very well known palindromist. The occasion was the presentation of his new book on the history of Polish palindromes. The book entitled "Kobyła ma mały bok – czyli o Polskich Palindromach" has been issued recently as vol No. 24 of the "Biblioteczka Rozrywki". This is the fifth book on palindromes by professor Morawski. During the session the author presented the book and after that signed and dedicated the book for the session participants.

The Symposium organizers express solid hope that the IEEE-SPIE WILGA Symposium cycle will be successfully continued. The next Symposium from the series is scheduled for the last week of May, 25-31.05.2009. It will be held traditionally in WILGA on the Vistula River near Warsaw in the WUT creative work resort. The organizers warmly invite B.Sc., M.Sc., Ph.D. students, young researchers and their tutors/mentors to WILGA. The Symposium is nearly cost free. There is no entrance fee. Cheap nights and cheap, but extremely good, food is offered by the summer WUT WILGA Resort staff. A unique research-gastronomical specialty of WILGA Symposium are late night topical working sessions combined with a grill sponsored by Photonics Society of Poland and IEEE. SPIE funds special awards for the best presentations during the Symposium. Full information about WILGA are accessible through the Symposium web: http://wilga.ise.pw.edu.pl.

professor Ryszard S.Romaniuk, Warsaw University of Technology, ISE
WILGA Symposium Chairman

INFORMATION FOR AUTHORS

An article published in other magazines can not be submitted for publishing in E.T.Q. The size of an article can not exceed 30 pages, 1800 character each, including figures and tables.

Basic requirements
The article should be submitted to the editorial staff as a one side, clear, black and white computer printut in two copies. The article should be prepared in English. Floppy disc with an electronic version of the article should be enclosed. Preferred wordprocessors: WORD 6 or 8.
Layout of the article:
- Title
- Author (first name and surname of author/authors)
- Workplace (institution, address and e-mail)
- Concise summary in a language article is prepared in (with keywords).
- Main text with following layout:
  - Introduction
  - Theory (if applicable)
  - Numerical results (if applicable)
  - Paragraph 1
  - Paragraph 2
  - . . . .
  - Conclusions
  - Acknowledgements (if applicable)
  - References
- Summary in additional language:
  - Author (firs name initials and surname)
  - Title (in Polish, if article was prepared in English)
  - Extensive summary, hawever not exceeching 3600 characters (along with keywords) in Polish, if artide was prepared in English). The summary should be prepared in a way allowing a reader to obtaoin essential information contained in the artide. For that reason in the summary author can place numbers of essential formulas figures and tables from the article.

Pages should have continues numbering.

Main text
Main text cannot contain formatting such as spacing, underlining, words written in capital letters (except words that are Commonly written in capital letters). Author can mark suggested formatting with pencil on the margin of the article using commonly accepted adjusting marks.

Tables
Tables with their titles should be placed on separate page at the end of the article. Titles of rows and columns should be written in small letters with double line spacing. Annotations concerning tables should be placed directly below the table. Tables should be numbered with Arabic numbers on the top of each table. Table can contain algorithm and program listings. In such cases original layout of the table will be preserved. Table should be cited in the text.

Mathematical formulas
Characters, numbers, letters and spacing of the formula should be adequate to layout of main text. Indexes should be properly lowered or raised above the basic line and clearly written. Special characters such as lines, arrows, dots should be placed exactly over symbols which they ar attributed to. Formulas should be numbered with Arabic numbers placed in brackets on the right side of the page. Units of measure, letters and graphic symbols should be printed according to requirements of IEE (International Electronical Commission) and ISO (International Organisation of Standardisation).

References
References should be placed at the and of the main text with the subtitle "References", References should be numbered (without brackets)adequately to references placed in the text. Examples of periodical [@], non-periodical [2] and book [3] references:
1. F. Valdoni: A new milimetre wave satellite. E.T.T. 1990, vol. 2, no 5, pp. 141– 148
2. K. Anderson: A resource allocation framework. XVI International Symposium (Sweden). May 1991, paper A 2.4
3. Y. P. Tvidis: Operation and modeling of the MOS transistors. New York, McGraw-Hill, 1987, p. 553

Figures
Figures should be clearly drawn on plain or milimetre paper in the format not smaller than 9×12 cm. Figures can be (editor – CorelDRAW). Photos or diapositives will be accepted in black and white format not greater than 10×15 cm. On the margin of each drawing and on the back side of each photo author name and abbreviation of the title of article Should be placed. Figure's captures should be given in two languages (first in the language the article is writes in and then in additional language). Figure's captures should be also listed on separate page. Figures should be cited in the text.