POLISH ACADEMY OF SCIENCE
COMMITTEE FOR ELECTRONICS AND TELECOMMUNICATIONS

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY

KWARTALNIK ELEKTRONIKI I TELEKOMUNIKACJI

VOLUME 55 – No 1

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. 9,25 Ark. druk. 7,5	Podpisano do druku w lutym 2009 r.
Papier offset, kl. III 80 g. B-1	Druk ukończono w lutym 2009 r.

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

Dear Professor Woliński,

Please accept our sincerest congratulations and best wishes. We wish you good health vitality and we expect more distant – round – jubilees.

Józef Modelski
Chairman of the Committee
of Electronics and Communications

Editorial Board of Electronics
and Telecommunications Quarterly

Professor Wiesław L. Woliński was born in Dąbrowa Górnicza in January 1929. During the Second World War he finished his primary school and as 14-years old boy he was forced to word hard in German wire factory in his hometown. He graduated from the Warsaw University of Technology (WUT) in 1955. He defended his doctoral and habilitation dissertations at the Faculty of Electronics of WUT in 1964 and in 1968, respectively.

He was appointed Professor in 1975 and Full Professor in 1989. He was a co-organizer of Institute of Microelectronics and Optoelectronics (IM&O) of the Faculty of Electronics and Information Technology of WUT in 1970. Until 1978 he served as a vice-director and from 1978 to 1981 as a director of that Institute. Until his retirement, he served also as a head of the Optoelectronics Division of IM&O (1970-1999).

Professor W. Woliński greatly contributed to the organization and development of scientific school on Optoelectronics and Laser Technology. Professor's scientific and organization achievements were recognized by the community of scientists, and in 1991 he was elected Corresponding Member and in 2007 Full Member of Polish Academy of Sciences (PAN). From 1993 to 1999 he was vice-chairman and from 1999 to 2007 he chaired the Committee on Electronics and Telecommunication of PAN.

Since 1991 he has been a chairman of the Polish Optoelectronics Committee of the Association of Polish Electrical Engineers (SEP). In 1987 he was co-initiator of establishing the Polish Chapter of International Society for Optical Engineers (SPIE) – transformed into Photonics Society of Poland in 2008.

Professor W. Woliński has been head or principal investigator of several scientific projects involving academic – industry partnership and technology transfer.

In 1986-1991 he was the coordinator of the National Program for Research and Development (CPBR 8.12) 'Fundamentals of Laser Technology' and in the period of 1995 -2001 he was a head of the WUT Priority Program 'Photonics Engineering'.

His principal research interests include optoelectronic devices, gas and solid state laser physics, technology and applications.

During the fifties, his scientific activities resulted in the development and construction of variety of optoelectronic devices such as infrared image converters, spectral sources of light for interferometers, etc. Since sixties his research interests have been directed toward the laser physics and laser technology. They have encompassed both fundamental and applied investigations concerning the physical processes responsible for the conditions of the laser radiation generation in the active volumes of the gases and in the doped solid-state media, and also the design, construction and development of laser systems for industrial and medical applications.

As the result of Professor's activities, numerous unique laser devices were developed and introduced into the national industry and medicine.

The most important scientific results achieved by Professor himself or under his supervision involve:

– determining the dependence between optical and photoelectrical properties of semitransparent Ag-O-Cs photocathodes,

– explanation and description of the delay effect between the excitation and laser pulse in a He-Ne pulse laser,

– development of a new method that was patented in 1967, for excitation of molecular lasers (CO2 -N2 -He) by applying the transverse electric discharge – the patent is probably the earliest patent on this subject,

- explanation and description of the gas pressure changes in the anode and cathode regions in working argon-ion lasers; development of method for increasing the laser output power by applying segmented permanent magnets (patented),

– defining the influence of multi-ion and multi-photon processes on the optical transitions and specifying conditions for up-conversion processes and lasing on specific wavelengths in various types of monocrystals and laser glass doped with lanthanide ions,

- theoretical description of a planar distributed feedback laser.

He has authored or co-authored over 200 scientific papers published in prestigious Polish and foreign scientific journals as well as numerous conference contributions – both national and international. He is also an author or co-author of 15 patents. He is co-editor of 13 volumes of *SPIE Proceedings* of the most important conference on *Laser Technology* in Poland. From 1982 to 2004 he was the chairman of Program Board of a journal *Elektronika* (SEP). In 1986-2008 he was Editor-in-Chief of the journal *Electronics and Telecommunications Quarterly*. Since 2005 he has served as the Branch Editor of journal *Optoelectronics and Photonics – Bulletin of the Polish Academy of Sciences, Technical Sciences*. Up to 2006 he was a Chair of The International Editorial Advisory Board of the journal *Optoelectronics Review*.

Professor W. Woliński was actively involved in didactic activities for over 50 years. The first laser laboratories for students, seminars and lectures in optoelectronics and laser technology were organized by Professor in 1965. From the beginning of Professor's career, he was engaged actively in teaching undergraduate and graduate courses including Laser Physics, Integrated Optics, Fundamentals of Photonics and Laser Techniques and others. For over ten years, he was head of Postgraduate Studies Department at WUT. He has supervised numerous master's theses and doctoral dissertations. Some of his students have become full or associate professors. Some of them have became leaders in both industry and science.

Prof. W. Woliński has been rewarded with many high rank national distinctions and awards for scientific achievements and service to the technical community.

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

El

m
of
or
de
op

as

cr
br
th
ar

cl
T
S

c
M

d
th
n

a
p
ir

e

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

N

J
A

A
S
A
I

CONTENTS

MI

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 9–30

Scheduling Architectures for DiffServ Networks with Input Queuing Switches

MEI YANG[†], HENRY SELVARAJ[†], ENYUE LU[‡], JIANPING WANG[⋆], S. Q. ZHENG[*], YINGTAO JIANG[†]

[†] *Department of Electrical and Computer Engineering*
University of Nevada, Las Vegas, Las Vegas, NV 89154
[‡] *Dept. of Mathematics and Computer Science*
Salisbury University, Salisbury, MD 21801
[⋆] *Department of Computer Science*
City University of Hong Kong, Hong Kong
[*] *Department of Computer Science*
The University of Texas at Dallas, Richardson, TX 75080
[†]*meiyang@egr.unlv.edu, selvaraj@unlv.nevada.edu, yingtao@egr.unlv.edu,*
[‡]*ealu@salisbury.edu,* [⋆]*jianwang@cityu.edu.hk,* [*]*sizheng@utdallas.edu*

Received 2009.01.20
Authorized 2009.03.21

Due to its simplicity and scalability, the differentiated services (DiffServ) model is expected to be widely deployed across wired and wireless networks. Though DiffServ supporting scheduling algorithms for output-queuing (OQ) switches have been widely studied, there are few DiffServ scheduling algorithms for input-queuing (IQ) switches in the literature. In this paper, we propose two DiffServ scheduling algorithms for DiffServ networks with IQ switches: the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm. The basic idea of DDS and HDS is to schedule EF and AF traffic according to their minimum service rates with the reserved bandwidth and schedule AF and BE traffic fairly with the excess bandwidth. Both DDS and HDS find a maximal weight matching but in different ways. DDS employs a centralized scheduling scheme. HDS features a hierarchical scheduling scheme that consists of two levels of schedulers: the central scheduler and port schedulers. Using such a hierarchical scheme, the implementation complexity and the amount of information needs to be transmitted between input ports and the central scheduler for HDS are dramatically reduced compared with DDS. Through simulations, we show that both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic as well as fair bandwidth allocation for BE traffic. The delay and jitter performance of DDS is close to that of PQWRR, an existing DiffServ supporting scheduling algorithm for OQ switches. The tradeoff of the simpler implementation scheme of HDS is its slightly worse delay performance compared with DDS.

Keywords: Quality of service, DiffServ, scheduling, input-queuing switches

1. INTRODUCTION

The rapid growth of the Internet and wireless communications has driven the demand for wired/wireless broadband Internet access with quality of service (QoS) support. The two main approaches to provide QoS are: Integrated Services (IntServ) [4] and Differentiated Services (DiffServ) [3]. Fine-grained QoS guarantees can be achieved by IntServ. However, the scalability of the IntServ model is limited due to the per-flow reservation and heavy signaling overhead [12]. The DiffServ model is proposed to meet different QoS requirements for various types of clients and network applications. It addresses scalability by a coarse-grain differentiation model.

The DiffServ model [3] is orientated toward edge-to-edge service across a single domain. Traffic is classified into a limited number of service classes according to the service level agreement (SLA) with the network provider. The flow-based traffic classification and conditioning is pushed to edge routers of the domain. Core routers of the domain do not need to maintain per-flow state information, but only need to forward packets according to the per hop behavior (PHB) associated with each service class, which is identified by the DiffServ code point (DSCP) field in the header of each packet. The DiffServ model matches the heterogeneous feature of the Internet and it is capable of providing end-to-end QoS guarantees by bilateral agreements between neighboring domain owners [5]. Due to its simplicity and scalability, DiffServ is expected to be widely deployed across wired and wireless networks [2, 12].

Currently, the IETF defines a set of PHBs which include Expedited Forwarding (EF) PHB, Assured Forwarding (AF) PHB group, and Best Effort (BE) PHB. The EF PHB provides low loss, low delay, low jitter, assured bandwidth, and end-to-end service through the DiffServ domain. The EF PHB is ideally suitable for voice over IP (VoIP), audio-, video- streaming, and other real-time applications. The AF PHB group provides services with minimum rate guarantee and low loss rate [9]. Four AF classes (AF1, AF2, AF3, and AF4) are defined and each class has three levels of drop precedence [1, 9, 22]. The level of forwarding assurance of an IP packet belonging to an AF class depends on the amount of resources allocated to the AF class, the current load of the AF class, and the drop precedence of the packet. AF PHBs are suitable for network management protocols, such as Telnet, SMTP, FTP, HTTP. All data packets belonging to the BE class are not policed and are forwarded with the best effort.

The implementation of PHBs relies much on the scheduling and queuing schemes used in DiffServ compliant switches and routers. In order to provide premium service to EF traffic, packets belonging to EF class should be served prior to packets belonging to other classes. Meanwhile, to prevent the influence of damaging EF traffic to other traffic, the service rate (bandwidth) for EF traffic should be limited to its peak information rate (PIR). For each AF class, a minimum service rate, referred as committed information rate (CIR), should be guaranteed. On the other hand, to avoid starvation of BE traffic, backlogged BE queues should be served if excess bandwidth is available. In practice, we desire those scheduling and queuing schemes which are efficient in providing differentiated services for different traffic classes, with high throughput, and simple in implementation.

Existing DiffServ supporting scheduling schemes for output-queuing (OQ) switches include priority queuing (PQ), weighted round-robin (WRR), PQWRR [19, 25], and class-based queuing (CBQ) [11, 18]. CBQ ensures explicit rate control for each traffic class by the rate control mechanisms functioned at two schedulers: the general scheduler and the link-sharing scheduler [8]. Compared with PQ and WRR, PQWRR delivers the minimum delay and jitter for EF traffic and provides better bandwidth allocation for AF traffic and BE traffic by priority scheduling of EF traffic and non-EF traffic, and weighted round-robin scheduling of AF traffic and BE traffic. In terms of implementation, PQWRR is simple and more practical than CBQ. Nevertheless, these schemes all assume OQ switch architectures which are not scalable for high line rates and/or large numbers of ports due to the speed limitation of the switching fabric and memories.

Compared with OQ switches, input queuing (IQ) switches are more scalable and practical since they only need the switching fabric and memories to run at the line rate. We hence focus our study on DiffServ supporting scheduling algorithms for IQ switches. Many QoS supporting scheduling algorithms have been proposed for IQ switches. Most of them are maximal weight matching (MWM) based algorithms with different definitions of the weight, such as algorithms with the weight defined as a function of queue length (e.g. the successive incremental matching over multiple ports (SIMP) algorithm [23], the longest normalized queue first (LNQF) algorithm [16], the worst-case longest port first (LPF), and prioritized LPF algorithms [24]), algorithms with the weight defined as credits of bandwidth [13], and algorithms with the weight defined as time difference [6]. Another noticeable QoS scheduling algorithm is the hierarchical scheduling algorithm [15], which combines a dynamic algorithm which is used to determine input-output matchings and a static algorithm which is used to select a request in the granted input port. However, due to the lack of bandwidth reservation schemes, all these algorithms do not provide bandwidth or delay guarantee for each traffic class. Although the distributed mutlilayered scheduler (DMS) [7] for multistage switches can provide delay bounds for EF flows and guaranteed bandwidth for AF flows, the complex structure of DMS and maintenance of per-flow queues prevent its practical use. In [10], the Adaptive Weighted Fair Queueing with Priority (AWFQP) scheduler attempts to provide QoS guarantees to EF, AF, and BE classes with two levels of schedulers: the Priority Queueing Scheduler and the Fair Queueing Scheduler in the first level, and the Adaptive Queueing Scheduler in the second level.

In this paper, we propose two DiffServ scheduling algorithms for IQ switches: the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm, to provide dynamic bandwidth allocation for DiffServ classes. The basic idea of DDS and HDS is to schedule EF and AF traffic according to their minimum service rates with the reserved bandwidth and schedule AF and BE traffic fairly with the excess bandwidth. Both DDS and HDS find a maximal weight matching but in different ways. DDS employs a centralized scheduling scheme. HDS features a hierarchical scheduling scheme that consists of two levels of schedulers: the central scheduler and port schedulers. Using such a hierarchical scheme, the implementation complexity and the amount of information needs to be transmitted between input ports and the central scheduler for HDS are dramatically reduced compared with DDS. Through simulations, we evaluate the performance of DDS and HDS under bursty arrivals and compare them with PQWRR. We show that both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic as well as fair bandwidth allocation for BE traffic. DDS also achieves the delay and jitter performance for EF traffic close to that of PQWRR and the delay performance for AF traffic better than that of PQWRR at high loads.

The rest of the paper is organized as follows. Section 2 introduces the IQ switch architecture. Section 3 describes the preliminaries for both algorithms. Section 4 presents the DDS algorithm. Section 5 presents the HDS algorithm. Section 6 discusses the simulation results and comparison with PQWRR. Section 7 concludes the paper.

2. IQ SWITCH ARCHITECTURE

Figure 1 shows an $N \times N$ IQ switch architecture. We assume that all data packets arriving at the switch are segmented into fixed-size cells, transmitted through the switching fabric, and reassembled back into original data packets before they leave the switch. We also assume that time is slotted such that one cell slot is equal to the transmission time of one cell on the input/output line. To remove head-of-line (HOL) blocking, each input port maintains N groups of virtual output queues (VOQs), and each group of VOQs is used to buffer cells destined for an output port.

A VOQ group is composed of K VOQs, each dedicated to buffering cells of a DiffServ class. Figure 2 shows the queuing scheme used at input port I_i, $1 \leq i \leq N$, in which a separate FIFO queue $Q_{i,j,k}$ is used to buffer cells belonging to traffic class k, $1 \leq k \leq K$, and destined for output port O_j, $1 \leq j \leq N$. For the DiffServ model, we have $K = 6$ with $k = 1$ to 6 representing the classes of EF, AF1, AF2, AF3, AF4, and BE respectively. When a cell arrives at an input (port), it is classified based on its DSCP field and output port address, and buffered in the VOQ corresponding to its traffic class and output (port).

In each cell slot, a scheduling algorithm is needed to determine which N cells in the N^2K VOQs to be transmitted through the switching fabric. In the following, we assume that scheduling in the current cell slot is based on the VOQ status of the previous cell slot, and switching in the current cell slot is based on the scheduling decision made by the previous cell slot.

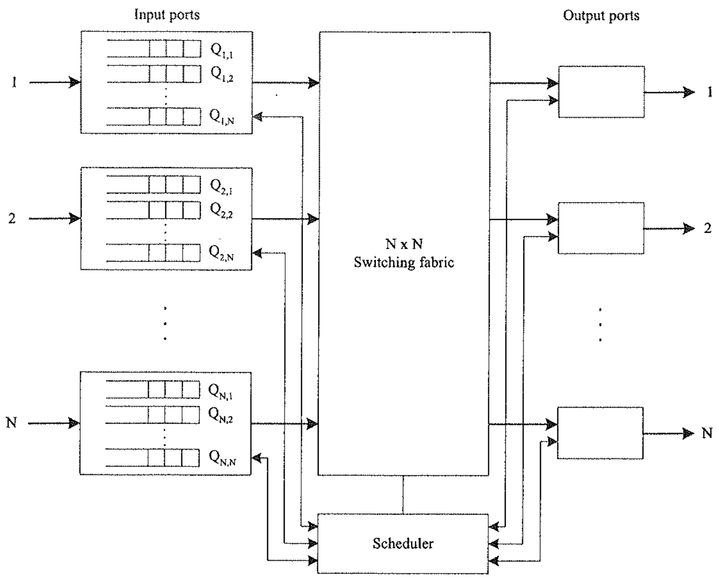


Fig. 1. The IQ switch architecture

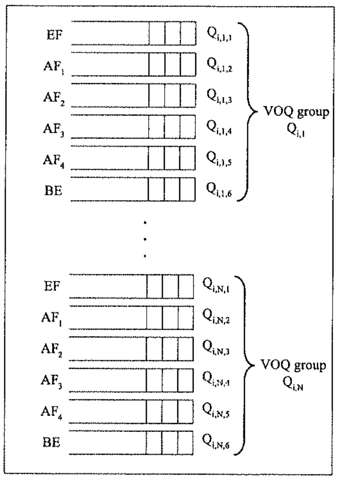


Fig. 2. Queuing scheme at input port I_i

3. PRELIMINARIES

Three factors need to be considered when designing a DiffServ supporting scheduling algorithm for IQ switches. First, to provide minimum bandwidth guarantees for EF and AF classes, the scheduling algorithm needs to consider the PIR for EF class and CIRs for four AF classes. Meanwhile, to avoid starvation of BE class, backlogged queues should be served if the excess bandwidth is available. Hence, class differentiation and bandwidth reservation and measurement schemes need to be introduced in the scheduling algorithm. Second, the switch throughput should be kept as much as possible. Third, the scheduling algorithm should be simple in implementation.

In the next two sections, we present the dynamic DiffServ scheduling algorithm and the hierarchical DiffServ scheduling algorithm. The service discipline of DDS and HDS is the same: If the reserved bandwidth is available, it serves EF or AF traffic first so that the PIR for EF class and the CIR for each AF class are guaranteed; otherwise, it serves non-EF traffic fairly so that BE traffic is not starved. The difference between DDS and HDS is the scheduling scheme used to find a maximal weight matching. DDS employs a centralized scheme, while HDS features a hierarchical scheme. Before we present each algorithm, we first introduce the bandwidth reservation and measurement schemes at each output port.

We use L to denote the bandwidth of each output link, which is divided into two categories, reserved bandwidth and excess bandwidth (e.g., 90% as the reserved bandwidth and 10% as the excess bandwidth). The reserved bandwidth is further divided into five parts, each corresponding to the guaranteed bandwidth for a non-BE DiffServ class. To provide bandwidth guarantees for AF classes in a finer granularity and enforce smooth AF traffic, we introduce the time unit of *frame*, which is composed of T time slots. Each output port O_j, $1 \leq j \leq N$, maintains the following variables.

- $R_{j,k}$ denotes the reserved (guaranteed) bandwidth for class k, where $1 \leq k \leq K-1$. $R_{j,1}$ = PIR for EF class, $R_{j,k}$ = CIR for AF($k-1$) class, $2 \leq k \leq K-1$, and $\sum_{k=1}^{K-1} R_{j,k} \leq 1$.
- $C_{j,k}$ denotes the cell counter for class k. $C_{j,1}$ counts the number of EF cells up to the current slot, and $C_{j,k}$, $2 \leq k \leq K-1$, counts the number of AF($k-1$) cells transmitted in the current frame. We set $C_{j,1} = 0$ at cell slot $t = 0$, and $C_{j,k} = 0$ at cell slot $t \bmod T = 0$ for $2 \leq k \leq K-1$.
- $S_{j,k}$ denotes the bandwidth utilization status for class k. $S_{j,k} = 1$ if $C_{j,1}/t < R_{j,1}$ for EF class or $C_{j,k}/T < R_{j,k}$ for AF($k-1$) class, $2 \leq k \leq K-1$; $S_{j,k} = 0$ otherwise.

At the beginning of each cell slot, each output port O_j, $1 \leq j \leq N$, sends S_j to the central scheduler. Each input port I_i, $1 \leq i \leq N$, collects the waiting time of the HOL cell of each non-empty VOQ $Q_{i,j,k}$ as $w_{i,j,k} = t - t'_{i,j,k}$, where $t'_{i,j,k}$ is the entering time slot of the HOL cell. We use a mapping function to map the weight value into the range of 0 to $2^{b_k} - 1$, where b_k is the number of bits used to represent the weight range of traffic class k. In this paper, we use a saturation function which is defined as follows.

$$f(w_{i,j,k}) = \begin{cases} w_{i,j,k} & \text{if } 0 \leq w_{i,j,k} < 2^{b_k}, \\ 2^{b_k} - 1 & \text{otherwise.} \end{cases} \tag{1}$$

4. THE DDS ALGORITHM

The DDS algorithm finds a maximal weight matching in a centralized way. At the start of each cell slot, each input port I_i sends a weighted vector with NK values to the scheduler. For each VOQ group $Q_{i,j}$, a weighted request vector $V_{i,j}$ is constructed as $(f(w_{i,j,1}), f(w_{i,j,2}), \cdots, f(w_{i,j,K}))$.

4.1. THE DDS ALGORITHM

The DDS algorithm works iteratively, with each iteration consisting of the following three steps.

Step 1: **Request.** Each unmatched input I_i sends request vectors $V_{i,j}$'s to their corresponding outputs.

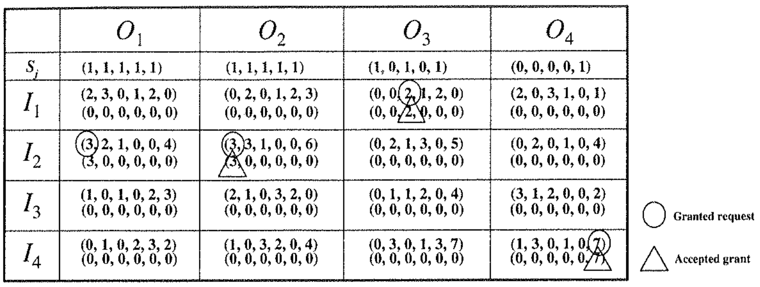
Step 2: **Grant.** For each unmatched output O_j, once it receives at least one non-zero request vector, it grants one input as follows.

- If $S_{j,1} = 1$ or $S_{j,k} = 1$ for $2 \leq k \leq K - 1$, it grants the input with $\max\{f(w_{i,j,k}) | f(w_{i,j,k}) > 0, 1 \leq i \leq N\}$ starting from $k = 1$ to $K - 1$; otherwise, it grants the input with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 1 \leq i \leq N, 2 \leq k \leq K\}$.
- If $f(w_{i',j,k'}) > 0$ is selected for some traffic class k' of input $I_{i'}$, it sends $I_{i'}$ a grant vector with the k'-th entry equal to $f(w_{i',j,k'})$ and other entries equal to '0', and other inputs zero grant vectors (all entries of the vector are set as '0').

Step 3: **Accept.** For each input I_i that receives at least one non-zero grant vector, it selects the output with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 1 \leq j \leq N\}$ starting from $k = 1$ to K. The accepted output is notified of the acceptance.

As described in the grant step, if the reserved bandwidth is available, the DDS algorithm allocates the reserved bandwidth to EF and AF traffic by serving the request with the highest weight value of the highest priority class; otherwise, it allocates the excess bandwidth to AF and BE traffic fairly by serving the request with the highest weight value among AF classes and BE class. Additionally, the DDS algorithm is starvation-free since the weight is generated based on the waiting time of the HOL cell and the excess bandwidth is shared by AF and BE traffic fairly.

Note that in grant and accept steps, there might be ties, i.e. requests with equal weights. Ties may exist among different traffic classes, or among different inputs/outputs. To ensure fairness, we break ties by making selections desynchornizedly. We set the selection starting position of each output or input in the static round-robin way. For example, at cell slot t, O_j starts its selection of inputs from $(j + t) \bmod N$ and its selection of classes from $(t \bmod (K-1))+1$, and I_i starts its selection of outputs from $(i+t) \bmod N$. Compared with breaking ties randomly [20], static round-robin is much easier to implement.



	O_1	O_2	O_3	O_4
S_j	(1, 1, 1, 1, 1)	(1, 1, 1, 1, 1)	(1, 0, 1, 0, 1)	(0, 0, 0, 0, 1)
I_1	(2, 3, 0, 1, 2, 0) (0, 0, 0, 0, 0, 0)	(0, 2, 0, 1, 2, 3) (0, 0, 0, 0, 0, 0)	(0, 0, 2, 1, 2, 0) (0, 0, 2, 0, 0, 0)	(2, 0, 3, 1, 0, 1) (0, 0, 0, 0, 0, 0)
I_2	(3, 2, 1, 0, 0, 4) (3, 0, 0, 0, 0, 0)	(3, 3, 1, 0, 0, 6) (3, 0, 0, 0, 0, 0)	(0, 2, 1, 3, 0, 5) (0, 0, 0, 0, 0, 0)	(0, 2, 0, 1, 0, 4) (0, 0, 0, 0, 0, 0)
I_3	(1, 0, 1, 0, 2, 3) (0, 0, 0, 0, 0, 0)	(2, 1, 0, 3, 2, 0) (0, 0, 0, 0, 0, 0)	(0, 1, 1, 2, 0, 4) (0, 0, 0, 0, 0, 0)	(3, 1, 2, 0, 0, 2) (0, 0, 0, 0, 0, 0)
I_4	(0, 1, 0, 2, 3, 2) (0, 0, 0, 0, 0, 0)	(1, 0, 3, 2, 0, 4) (0, 0, 0, 0, 0, 0)	(0, 3, 0, 1, 3, 7) (0, 0, 0, 0, 0, 0)	(1, 3, 0, 1, 0, 7) (0, 0, 0, 0, 0, 7)

○ Granted request
△ Accepted grant

Fig. 3. An example of the DDS algorithm

Figure 3 shows an example of the DDS algorithm for a 4×4 switch. In the current cell slot, the bandwidth utilization vector S_j for each output O_j is given in the second row. In the request step, each input I_i sends a request vector to each output O_j as shown in the first vector of each cell. In the grant step, O_1 grants the EF request from I_2 since the reserved bandwidth for EF class is still available and I_2 has the largest EF request among all inputs. For the same reason, O_2 grants the EF request from I_2. O_3 grants the EF request from I_1 since there is no EF request to O_3, the reserved bandwidth for AF1 class is used up, and I_1 has the largest AF2 request among all inputs. O_4 grants the BE request from I_4 since the reserved bandwidths for all non-BE classes are used up and the BE request from I_4 is the largest among all non-EF requests from all inputs. The grant received at each input is shown as the second vector in each cell. In the accept step, I_1 accepts the grant from O_3. Having two grants with the same value, I_2 accepts one according to tie-breaking scheme, for instance, the grant from O_2. I_4 accepts the grant from O_4. In the first iteration, three pairs of inputs and outputs are matched. More iterations can be conducted to enlarge the number of matched inputs and outputs.

The core of the DDS algorithm is a maximal weight matching algorithm. The number of iterations needed to converge is at most N. Through simulations, we find that on average $\log N$ iterations are adequate to achieve satisfying performance.

4.2. HARDWARE IMPLEMENTATION SCHEME OF DDS

To implement the DDS algorithm, one can use the scheduler architecture shown in Figure 4 (a), in which each input/output is associated with an arbitration component. As shown in Figure 4 (b) and (c), each arbitration component can be constructed by K copies N-input comparator-trees [20], each being used to find the maximum weight value for a class k, $1 \leq k \leq K$. One more comparator-tree is needed for each grant arbitration component to choose the maximum weight value of all classes. Each grant or accept arbitration component has $O(\log N \log b)$-gate delay, where $b = \max\{b_k \mid 1 \leq k \leq K\}$. Such an implementation of the DDS algorithm has $O(\log^2 N \log b)$-gate delay. Each arbitration component consumes $O(NKb)$ gates since each comparator tree is composed of $O(Kb)$ gates. The whole DDS scheduler consumes $O(N^2Kb)$ gates.

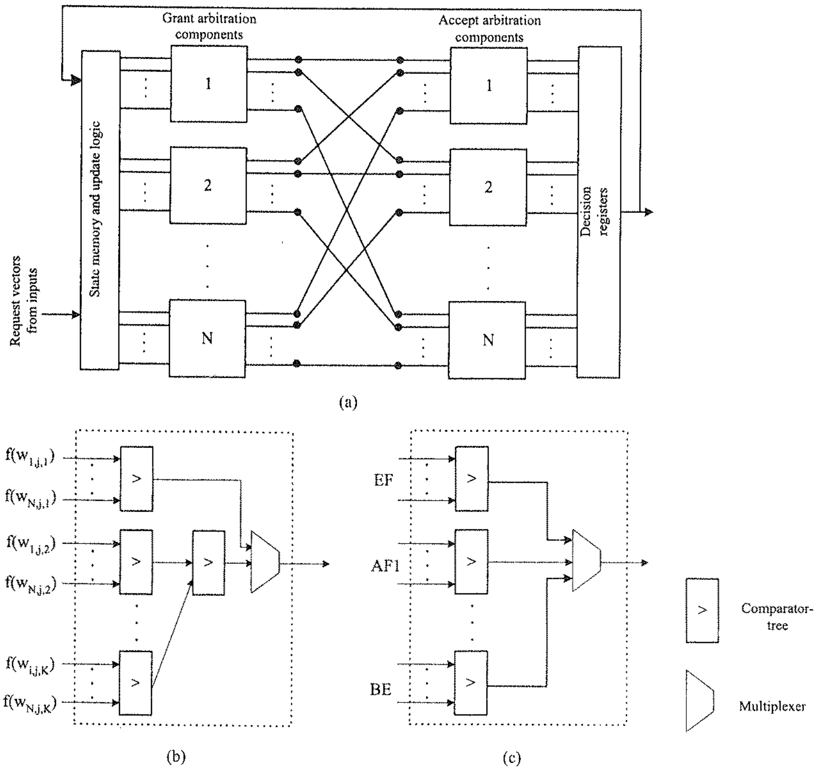


Fig. 4. (a) Block diagram of a DDS scheduler. (b) The grant arbitration component for output O_j. (c) The accept arbitration component for input I_i

5. THE HDS ALGORITHM

As we can see from the previous section, the construction of the DDS scheduler is complex. In order to reduce the implementation complexity of the scheduler, we extend the idea of hierarchical scheduling [15] and propose the hierarchical DiffServ scheduling algorithm. The HDS algorithm separates the tasks of providing differentiated services and maximizing switch throughput by employing two levels of schedulers. One level is the central scheduler which is designed to maximize the switch throughput by computing a maximal size matching (MSM) between input ports and output ports. The other level is formed by input port schedulers which provide differentiated services by serving cells belonging to different classes dynamically. In light of the idea of exhaustive matching [17], the central scheduler employs a three-phase exhaustive MSM algorithm. At the granted input port, the service policy changes according to the bandwidth utilization at the destined output port such that minimum bandwidth guarantees for EF and AF classes and fair bandwidth allocation for BE class are provided.

In the HDS algorithm, at the start of each cell slot, each input port I_i only needs to send a $2N$-bit vector P_i to the central scheduler, where $P_{i,j} = 2$ if I_i has more than one EF cells in VOQ group $Q_{i,j}$, $P_{i,j} = 1$ if I_i has at least one cell in VOQ group $Q_{i,j}$, and $P_{i,j} = 0$ otherwise.

5.1. THE HDS ALGORITHM

The HDS algorithm works in two stages.

Stage I: The central scheduler finds a maximal size matching in a three-phase exhaustive scheme iteratively. We assume that each input port I_i has an accept pointer a_i indicating the accept starting position, and each output port O_j has a grant pointer g_j indicating the grant starting position. Each iteration of stage I consists of the following three steps.

Step 1: **Request.** Each I_i sends a request to every O_j for which it has a queued cell.

Step 2: **Grant.** If an unmatched O_j receives any request, it selects one request to grant starting from the input port that g_j points to in a round-robin manner. For *the first iteration*, if $P_{i,j} = 2$ for some I_i, g_j is updated to i, otherwise, g_j is updated to one beyond the granted input port.

Step 3: **Accept.** If an unmatched I_i receives any grant, it selects one grant to accept starting from the output port that a_i points to in a round-robin manner. a_i is updated to the accepted output port.

After Stage I finishes, the central scheduler will send to each input port I_i an N-bit grant vector G_i, and S_j if there exists $G_{i,j} = 1$ for some j.

Stage II: For each input I_i that receives a non-zero grant vector (assuming that $G_{i,j} = 1$), if $\sum_{k=1}^{K-1} S_{j,k} f(w_{i,j,k}) \neq 0$, then it will select $Q_{i,j,k}$ such that $S_{j,k} = 1$ starting from $k = 1$ to $K - 1$; otherwise, it will select $Q_{i,j,k}$ with $\max\{f(w_{i,j,k}) \mid f(w_{i,j,k}) > 0, 2 \leq k \leq K\}$.

Figure 5 illustrates an example of the exhaustive scheduling algorithm used at stage I for a 4×4 switch. At the beginning of the cell slot, grant pointers are set as $g_1 = 1$, $g_2 = 3$, $g_3 = 3$, and $g_4 = 2$, and accept pointers are set as $a_1 = 2$, $a_2 = 4$, $a_3 = 3$, and $a_4 = 1$. Given the request matrix P, in the request step, each input port I_i sends a request to each output O_j with $P_{i,j} > 0$ for $1 \leq i, j \leq 4$ as shown in Fig. 5 (a). As shown in Fig. 5 (b), in the grant step, each output grants one request starting from its grant pointer and updates its grant pointer accordingly. Notice that O_3 grants the request from I_3 and let g_3 stay at I_3 since $P_{3,3} = 2$. In the accept step, each input port accepts one grant starting from its accept pointer and updates its accept pointer to the accepted output port as shown in Fig. 5 (c). The generated grant matrix G is shown in the figure. Using such a pointer updating scheme, in the next cell slot, request from VOQ group $Q_{3,3}$ will continue to be favored, thereby serving EF traffic with the highest priority.

$$P = \begin{bmatrix} 1 & 2 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 \\ 1 & 2 & 1 & 1 \end{bmatrix} \qquad G = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

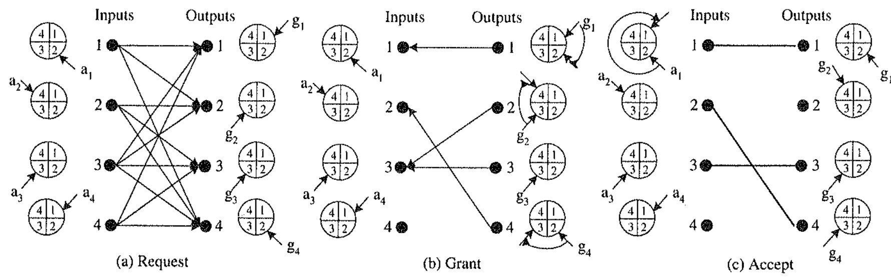


Fig. 5. An example of the exhaustive scheduling algorithm used at the central scheduler

Similar to the DDS algorithm, the HDS algorithm also finds a maximal weight matching. However, different from the DDS algorithm, the HDS algorithm distributes the selection of the highest weight request to each input port, hence simplifies the operation at the central scheduler. In each cell slot, the central scheduler only needs to find a maximal size matching. As one can understand, the tradeoff of the two-level scheduling is that the maximal weight matching found by the HDS algorithm may not be as good as the one found by the DDS algorithm in terms of the total weight.

5.2. HARDWARE IMPLEMENTATION SCHEME OF HDS

To implement the central scheduler, one can use the scheduler architecture shown in Figure 6 (a), in which each input/output is associated with an arbiter, which is responsible for selecting one out of N requests. Each arbiter can be implemented by the parallel round-robin arbiter (PRRA) proposed in [26], which has $O(\log N)$-gate delay and consumes $O(N)$ gates. We find through simulations that on average $\log N$ iterations are adequate to achieve satisfying performance. Hence, the first stage of the HDS algorithm can be implemented in $O(\log^2 N)$-gate delay and $O(N^2)$ gates.

As shown in Figure 6 (b), each port scheduler majorly consists of K N-input multiplexers, one K-input multiplexer, and one K-input comparator-tree, which is responsible for selecting the maximum weight value among all traffic classes of the same VOQ group. Each port scheduler has $O(\log N + \log K \log b)$-gate delay, where $b = \max\{b_k \mid 1 \leq k \leq K\}$, and consumes $O(NK + Kb)$ gates.

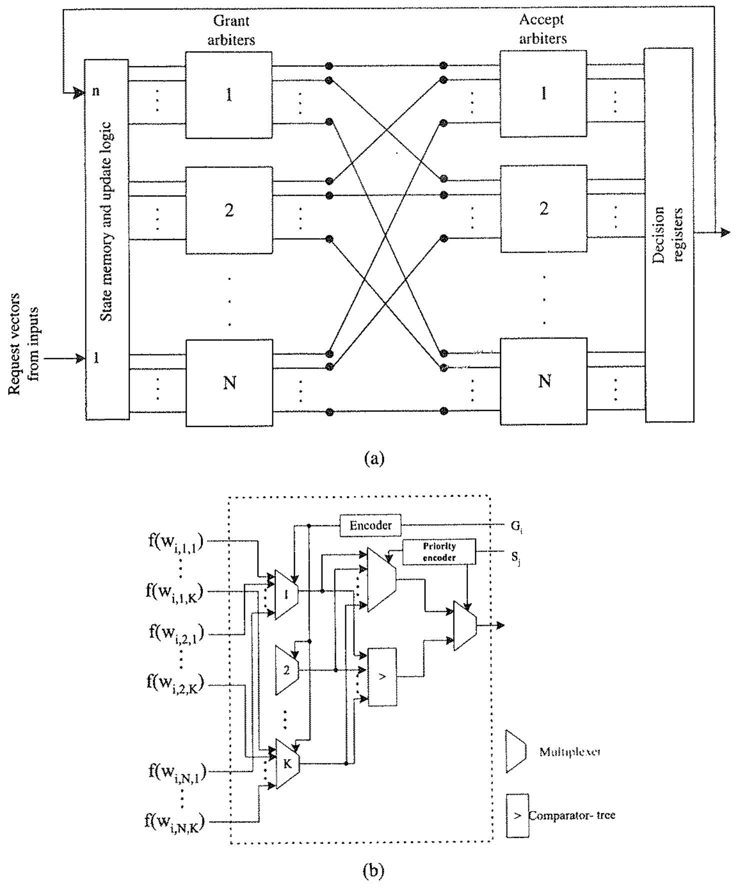


Fig. 6. (a) Block diagram of the central scheduler. (b) Block diagram of a port scheduler

The total delay of such an implementation of the HDS algorithm is $O(\log^2 N + \log K \log b)$-gate delay, which is faster than the implementation of the DDS algorithm, $O(\log^2 N \log b)$-gate delay. The total number of gates needed for the HDS scheduler is $O(N^2K + NKb)$, which is also smaller than that of the DDS scheduler, $O(N^2Kb)$ gates.

In addition, the amount of information to be transmitted between each input port and the central scheduler in the HDS algorithm is much less than in the DDS algorithm. In each cell slot, in the HDS algorithm, each input port only needs to send $2N$ bits to the central scheduler and the central scheduler only needs to send $N + K$ bits back to each input port, while in the DDS algorithm, each input port needs to send NKb bits to the scheduler and the scheduler needs to send back NK bits to each input port. Table 1 summarizes the difference of implementation complexity between HDS and DDS.

Table 1

Comparison of the implementation complexity of HDS and DDS

Algorithm	**Time (gate delay)**	**Area (number of gates)**	**Bits sent from each input**	**Bits sent back to each input**
HDS	$O(\log^2 N + \log K \log b)$	$O(N^2K + NKb)$	$2N$	$N + K$
DDS	$O(\log^2 N \log b)$	$O(N^2Kb)$	NKb	NK

6. PERFORMANCE EVALUATION

We evaluate the performance of the DDS and HDS algorithms in two aspects: fairness and efficiency, where fairness is measured by the received bandwidth and efficiency is measured by the average cell delay and delay jitter. The cell delay is the queuing delay that a cell encounters in the switch. For EF traffic, we also consider its delay jitter performance, which is defined as the difference between the cell delays of two consecutive cells. To validate our evaluation, we compare the performance of the DDS and HDS algorithms with that of the PQWRR algorithm for OQ switches.

A cell-based simulator is developed and simulations have been conducted assuming that all queue sizes are infinite. In our simulations, we consider bursty traffic arrivals using 2-state modulated Markov-chain sources [21]. Each source alternately generates a burst of full cells (all with the same destination) followed by an idle period of empty cells. The number of cells in each burst or idle period is geometrically distributed. Let $E(B)$ and $E(D)$ be the average burst length and the average idle length in terms of the number of cells respectively. Then, we have $E(D) = E(B)(1-\rho)/\rho$, where ρ is the load of each input source. We assume that the destination of each burst is uniformly distributed.

In all the simulations, we assume that the average cell arrival rates of EF class and AF classes to each output link are 18%, 24%, 20%, 16%, and 12% respectively by default. To ensure guaranteed service to EF traffic, we set its PIR a little more than its arrival rate [11], e.g. $R_{j,1} = 18\% \times 1.1 = 19.8\%$. The CIRs for AF1 through AF4 to each output port are 24%, 20%, 16%, and 12% respectively. In the following simulations, we assume the frame size is 1000 and $b_k = 4$ for all $1 \le k \le K$.

6.1. BANDWIDTH ALLOCATION

First, we evaluate the effectiveness of the DDS and HDS algorithms supporting dynamic bandwidth allocation when a link is overloaded. We assume a 4×4 switch, the average burst length $E(B) = 32$, and the number of iterations allowed for DDS and the Stage I of HDS is 4. We assume that output link 1 is the overloaded link and we vary the load to each VOQ group destined for output link 1 from 0.1 to 1.0.

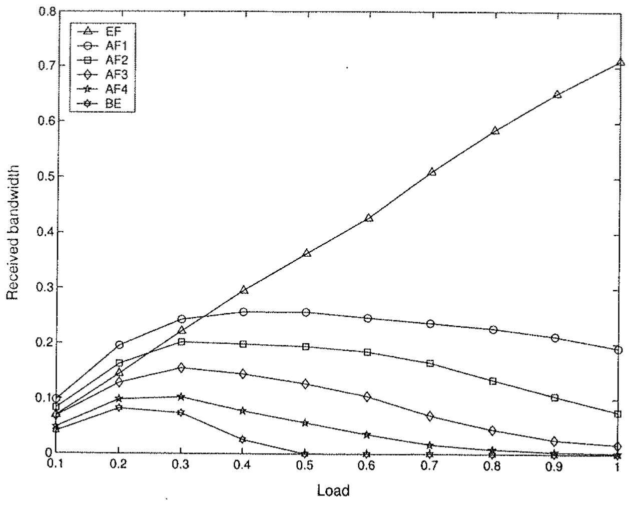


Fig. 7. Received bandwidth using PQWRR

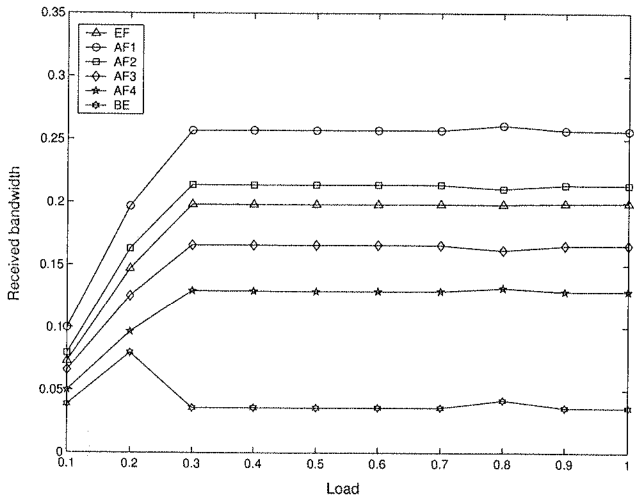


Fig. 8. Received bandwidth using DDS

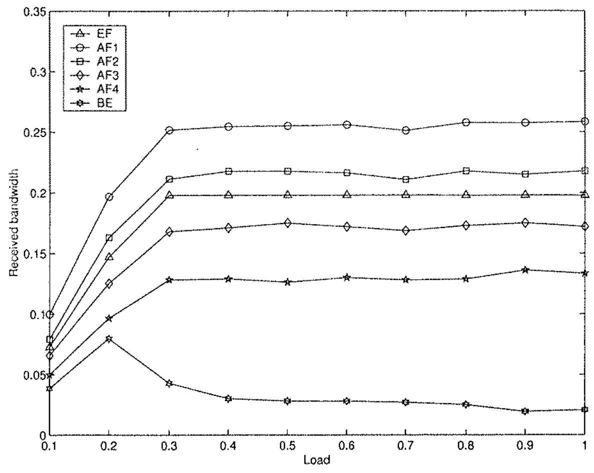


Fig. 9. Received bandwidth using HDS

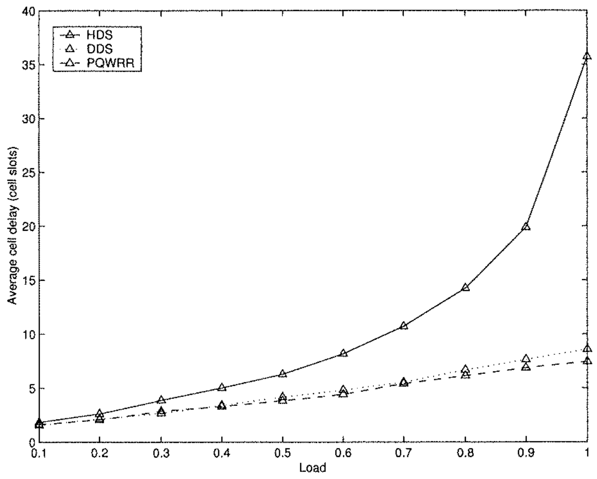


Fig. 10. Delay performance of EF traffic

Figures 7 to 9 show the received bandwidth of each traffic class for PQWRR, DDS, and HDS respectively. For a load below 0.25, the received bandwidth of each traffic class is able to keep up with its arrival rate for three schemes. However, for a load beyond 0.25, the received bandwidth of EF traffic by PQWRR still follows the arrival rate without regarding to the limitation of its PIR. For a load beyond 0.30, due to the influence of damaging EF traffic, the received bandwidth of AF traffic by PQWRR is degrading dramatically, and BE traffic cannot get any service at all.

On the other hand, DDS and HDS guarantee but limit the received bandwidth of EF traffic to its PIR, 19.8%, assure the CIR for each AF traffic, and avoid the starvation of BE traffic when the load is greater than 0.25. For example, when the load is at 0.40, the bandwidth received by EF, AF1, AF2, AF3, AF4, and BE traffic for DDS is 19.8%, 25.70%, 21.37%, 16.60%, 12.92%, and 3.6% respectively, while for HDS is 19.8%, 25.45%, 21.76%, 17.10%, 12.89%, and 3.0% respectively. Such bandwidth distributions conform to the design goal of DDS and HDS, which is to provide minimum bandwidth guarantees for non-BE classes and fair bandwidth allocation for BE class.

6.2. DELAY PERFORMANCE

Next, we examine the delay performance of DDS and HDS using simulations of a 16×16 switch under bursty arrivals assuming $E(B) = 32$ and the destination of each burst uniformly distributed. The number of iterations allowed for DDS and HDS is set as 4. Figure 10 shows the average cell delay vs. load of EF traffic for DDS, HDS, and PQWRR. The average cell delay of EF traffic using DDS is very close to that using PQWRR. The average cell delay of EF traffic using HDS is not as good as that using DDS and PQWRR. Figure 11 shows the jitter distribution of EF traffic at load 0.90 for DDS, HDS, and PQWRR. For DDS and HDS, over 90% EF traffic has jitter less than 1 cell slot, which is comparable to PQWRR.

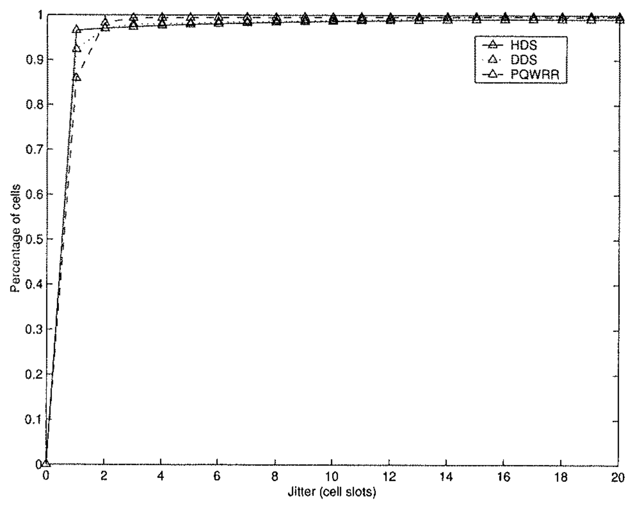


Fig. 11. Ef jitter distribution

Figure 12 shows the average cell delay vs. load of AF1 and AF2 traffic for DDS, HDS, and PQWRR. Figure 13 shows the average cell delay vs. load of AF3 and AF4 traffic for DDS, HDS, and PQWRR. The average cell delay of each AF class using DDS is close to that using PQWRR for loads below 0.95. For loads over 0.95, DDS performs even better than PQWRR. The reason is that DDS uses a function of the waiting time as the weight but PQWRR uses the queue length as the weight. In Figure 13, for loads lower than 0.60, HDS performs close to PQWRR. With loads going up, the performance of HDS is degrading. Figure 14 shows the average cell delay vs. load of BE traffic for DDS, HDS, and PQWRR. For loads lower than 0.90, HDS performs better than DDS and PQWRR. In general, DDS outperforms HDS in delay performance. This is consistent with our intuition that using a centralized scheme DDS tends to find a larger weight maximal weight matching than HDS.

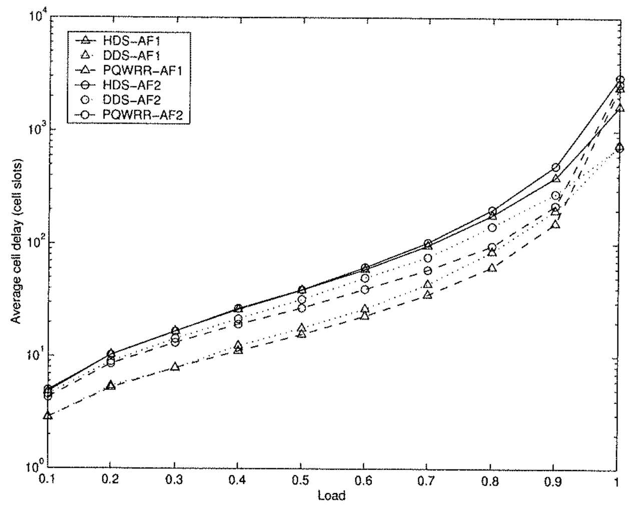


Fig. 12. Delay performance of AFI and AF2 traffic

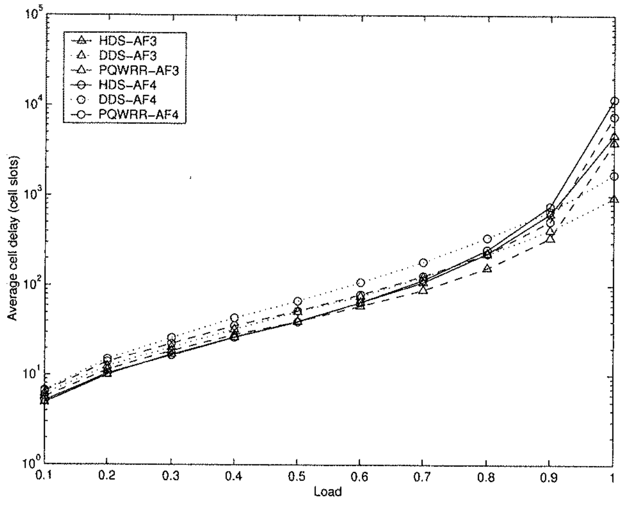


Fig. 13. Delay performance of AF3 and AF4 traffic

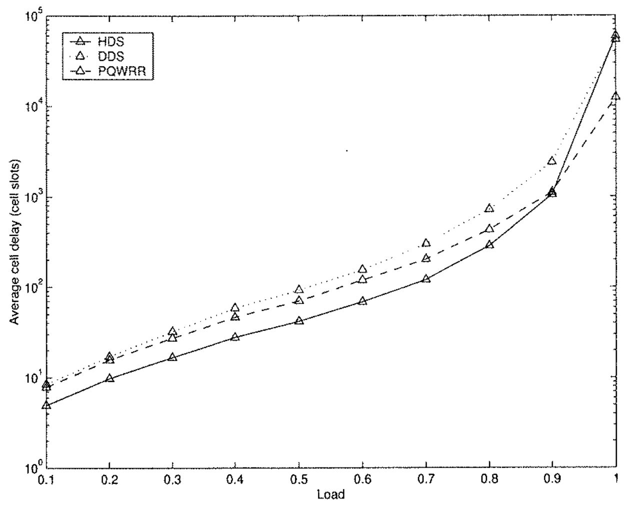


Fig. 14. Delay performance of BE traffic

In the worst case, N iterations are needed for DDS to find a maximal weight matching. Similarly, at most N iterations are needed for the central scheduler of HDS to find a maximal size matching. However, the number of iterations allowed in one cell slot is limited in reality. Figures 15 and 16 show the effect of the number of iterations allowed on the average cell delay of AF1 traffic using DDS and HDS respectively. We can see that DDS or HDS with 2 iterations achieves significant performance improvement over DDS or HDS with 1 iteration. The performance of DDS or HDS with 4 iterations is very close to the performance of DDS or HDS with 16 iterations. That is why we set the number of iterations allowed as 4 for previous simulations on 16×16 switches.

The purpose of using frame is to smooth bandwidth sharing of AF traffic in a finer way. As we can understand, the smaller the frame size, the finer bandwidth sharing. However, smaller frame size may introduce longer cell delay. Figure 17 and Figure 18 show the influence of different frame sizes to the average cell delay of AF classes for DDS and HDS respectively. It shows that the performance of classes AF1 and AF2 improves, while the performance of classes AF3 and AF4 degrades as the frame size increasing from 1000 to 10000. In the previous simulations, we set the frame size at 1000.

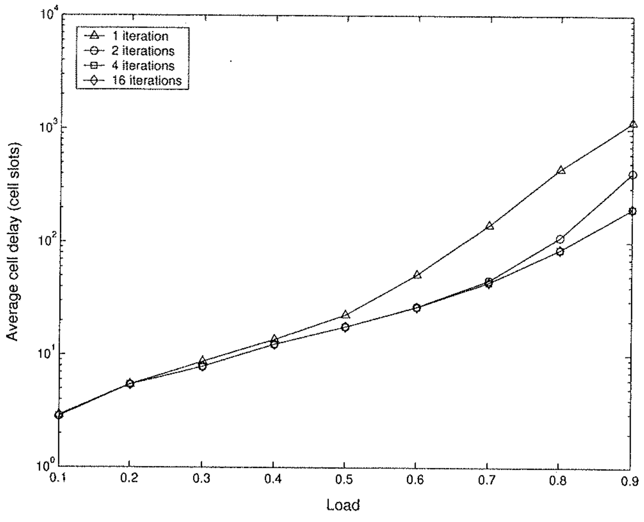


Fig. 15. Delay performance of AF1 traffic with different number of iterations allowed using DDS

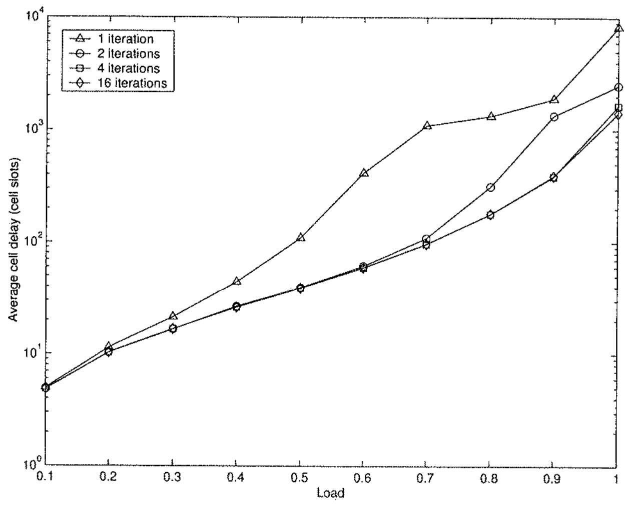


Fig. 16. Delay performance of AF1 traffic with different number of iterations allowed using HDS

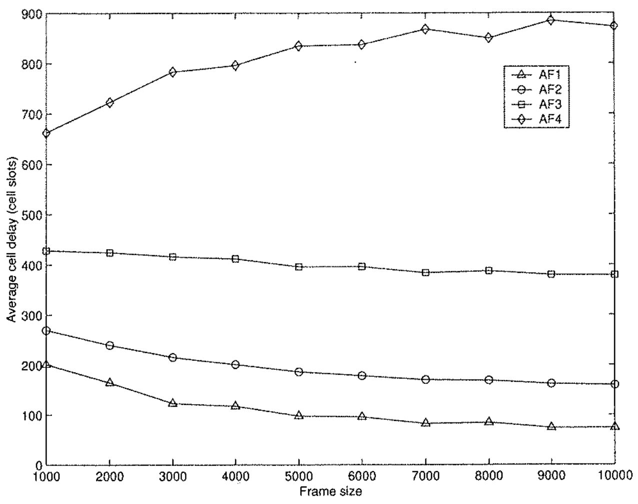


Fig. 17. Delay performance of AF1 traffic vs. different frame sizes using DDS

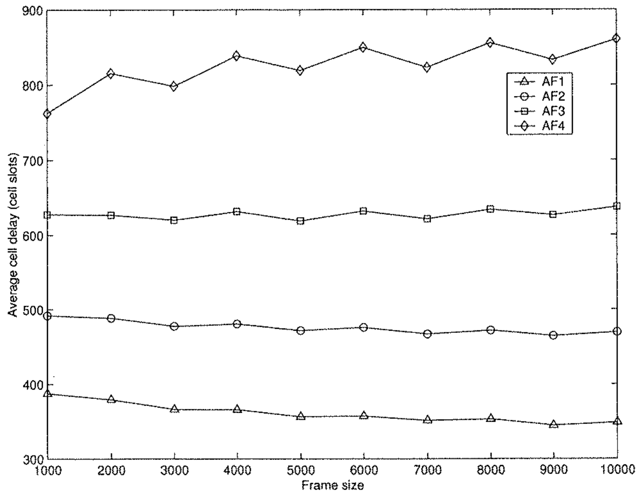


Fig. 18. Delay performance of AF1 traffic vs. different frame sizes using HDS

the l
alloc
at ou
and
BE
gene
queu
com
port
trade
in th
and
mod

1. D
p
p
2. A
w
C
3. S
fo
4. R
o
5. E
2
6. C
s
7. F
I
8. S
I
9. J
I
10. I
s
11. V
2
12. H
w

7. CONCLUSION

In this paper, we proposed the dynamic DiffServ scheduling (DDS) algorithm and the hierarchical DiffServ scheduling (HDS) algorithm, to support dynamic bandwidth allocation for DiffServ classes on IQ switches. With bandwidth measurement scheme at output ports, both DDS and HDS provide minimum bandwidth guarantees for EF and AF traffic with the reserved bandwidth as well as fair bandwidth allocation for BE traffic with the excess bandwidth. We show that DDS is starvation-free since it generates the weight based on the waiting time of the head-of-line cell instead of the queue length. Compared with DDS, the advantage of HDS is that the implementation complexity and the amount of information needs to be transmitted between each input port and the central scheduler are much reduced by using a hierarchical scheme. The tradeoff of HDS is its slightly worse delay performance compared with DDS, as shown in the simulation results. Since IQ switches are more scalable than OQ switches, HDS and DDS are very useful to implement DiffServ model and other differentiated service models, such as the Olympic service [9].

8. REFERENCES

1. D. Adami, S. Giordano, M. Pagano, R. Secchi: *Optimization of scheduling algorithms parameters in a DiffServ environment*, Symposium on Applications and the Internet Workshops, 2005, pp. 276–279.
2. A. Bader, G. Karagiannis, L. Westberg, et. al.: *QoS signaling across heterogeneous wired/wireless networks: resource managment in DiffServ using the NSIS protocol suite*, International Conference on Quality of Service in Heterogeneous Wired/Wireless Networks 2005, pp. 51–56.
3. S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Weiss: *An architecture for differentiated services*, IETF RFC 2475, Dec. 1998.
4. R. Braden, D. Clark, S. Shenker: *Integrated services in the Internet architecture: an overview*, IETF RFC 1633, 1994.
5. B. Carpenter, K. Nichols: *Differentiated services in the Internet*, Proceedings of the IEEE. 2002, vol. 90, no. 9, pp. 1479–1494.
6. C. Chen, M. Komatsu: *An adaptive scheduler to provide QoS guarantees in an input-buffered switch*, International Conference on Communications, 2002, vol. 2, pp. 1118–1122.
7. F. Chiussi, A. Francini: *A distributed scheduling architecture for scalable packet switches*, IEEE Journal of Selected Areas in Communications 2000, vol. 18, no. 12, pp. 2665–2683.
8. S. Floyd, V. Jacobson: *Link-sharing and resource management models for packet switches*, IEEE/ACM Transactions on Networking 1995, vol. 3, no. 4, pp. 365–386.
9. J. Heinanen, F. Baker, W. Weiss, J. Wroclawski: *Assured forwarding PHB group*, IETF RFC 2597, 1999.
10. I. S. Hwang, B. J. Hwang, C. S. Ding: *Adaptive weighted fair queueing with priority (AWFQP) scheduler for DiffServ networks*, Journal of Informatics & Electronics 2008, vol. 2, no. 2, pp. 15–19.
11. V. Jacobson, K. Nichols, K. Poduri: *An expedited forwarding PHB group*, IETF RFC 2598, 1999.
12. H. Jiang, W. Zhuang, X. Shen, A. Abdrabou, P. Wang: *Differentiated services for wireless mesh backbone*, IEEE Communications Magazine 2006, vol. 44, no. 7, pp. 113–119.

13. A. Kam, K. Sui: *Linear complexity algorithms for QoS support in input-queued switches with no speedup*, IEEE Journal of Selected Areas in Communications 1999, vol. 17, no. 6, pp. 1040–1056.
14. N. D. Kiameso, H. Hassanein, H. T. Mouftah: *Analysis of prioritized scheduling of assured forwarding in DiffServ Architectures*, IEEE International Conference on Local Computer Networks, 2003, pp. 614.
15. H. Kim, K. Kim, Y. Lee: *Hierachical scheduling algorithm for QoS guarantee in MIQ switches*, IEEE Electronic Letters 2000, vol. 36, no. 18, pp. 1594–1595.
16. S. Li, N. Ansari: *Provisioning QoS features for input-queued ATM switches*, Electronics Letters 1998, vol. 34, no. 19, pp. 1826–1827.
17. Y. Li, S. Panwar, H. J. Chao: *The dual round-robin matching with exhaustive service*, IEEE Workshop on High Performance Switching and Routing, 2002, pp. 58–63.
18. G. Mamais, M. Markaki, G. Politis, I. S. Venieris: *Efficient buffer management and scheduling in a combined IntServ and DiffServ architecture: a performance study*, International Conference on ATM, 1999, pp. 236–242.
19. J. Mao, W. M. Moh, B. Wei: *PQWRR scheduling algorithm in supporting of DiffServ*, International Conference on Communications, 2001, vol. 3, pp. 679–684.
20. N. Mckeown: *Scheduling algorithms for input-buffered cell switches*, Ph. D. Thesis, Univerity of California at Berkeley, 1995.
21. N. Mckeown: *The iSLIP scheduling algorithm for input-queued switches*, IEEE/ACM Transactions on Networking 1999, vol. 7., no. 2, pp. 188–201.
22. T. Minagawa, T. Kitami: *Packet size based dynamic scheduling for assured services in DiffServ network*, Electronics and Communications in Japan 2004, vol. 88, no. 1, pp. 12–20.
23. R. Schoenen, G. Post, G. Sander: *Prioritized arbitration for input-queued switches with 100% throughput*, IEEE ATM Workshop, 1999, pp. 253–258.
24. M. Song, M. Alam: *Two scheduling algorithms for input-queued switches guaranteeing voice QoS*, IEEE GLOBECOM, 2001, pp. 92–96.
25. Y. Zhang, P. G. Harrison: *Performance of a priority-weighted round robin mechanisms for differentiated service networks*, IEEE International Conference on Computer Communications and Networks, 2007, pp. 1198–1203.
26. S. Q. Zheng, M. Yang, J. Blanton, P. Golla, D. Verchere: *A simple and fast parallel round-robin arbiter for high-speed switch control and scheduling*, IEEE Midwest Symposium on Circuits and Systems, 2002, pp. 671–674.

a c
pro
cial
of s
de,
the
suc
me
con
ma
rel

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 31–46

Software versus hardware testing of microprocessors

JANUSZ SOSNOWSKI, PIOTR GAWKOWSKI

Institute of Computer Science, Warsaw University of Technology,
ul. Nowowiejska 15/19, Warsaw Poland,
Email: jss@ii.pw.edu.pl

Received 2008.12.18
Authorized 2009.01.12

The paper deals with the problem of developing built-in-self-test (BIST) in microprocessors. We outline classical approaches based on hardware implementations, show their drawbacks and present software implementations, which can increase test effectiveness. Combining these two approaches we describe possibilities of improving test observability using available on-chip mechanisms related to on-line testing and event monitoring. The presented considerations are completed with an original technique based on application driven testing.

Keywords: BIST, autotesting, test effectiveness, microprocessor testing

1. INTRODUCTION

High system dependability (reliability, availability, safety, maintainability, etc.) is a common requirement for most contemporary digital systems used in classical data processing, telecommunication, process control, microcontrollers, etc. One of the crucial points in achieving high dependability is efficient testing to check the correctness of system operation [1,18,29,31]. In general, we have off-line (performed in a test mode, e.g. for maintenance purposes) and on-line testing (performed concurrently with the running application). These two approaches are supported with special hardware such as design for test (DFT), built-in-self-test (BIST) and on-line error detection mechanisms [1,29]. All of them involve some circuit and performance overheads. In commercial circuits (e.g. microprocessors) and systems these mechanisms do not cover many functional blocks or are limited to specific fault classes. Hence, we still cannot rely only on these approaches.

Classical BIST and DFT techniques provide a special test mode, which does not cover some features of the operational mode. In systems comprising processors we have the possibility to eliminate this gap with software based self-testing (SBST). In our Institute advanced studies on SBST have been initiated in 90s [25,26] and continued later on [27-31]. This approach gained significant interest in recent literature (e.g. [13,14,16,20] and references). In practice, the problem of balancing hardware and software approaches to testing arises. We have got some experience in dealing with this concept. In particular, we have improved test observability using various on-line monitoring. Many microprocessor systems dedicated for unique applications (most embedded systems) use the hardware resources in a limited way – this creates the possibility of test simplification by application driven testing, which we have verified in some microcontrollers.

In section 2 we give an outline of hardware mechanisms improving microprocessor testability and discuss their capabilities. Software based approaches are discussed in section 3, here we also consider the problem of integrating hardware and software approaches to assure better fault coverage at a lower cost. The possibilities of simplifying tests for fixed applications are presented in section 4. Final conclusions are given in section 5.

2. HARDWARE MECHANISMS SUPPORTING MICROPROCESSOR TESTS

Various hardware mechanisms are being developed to improve circuit testability. In general, we can distinguish 3 classes of these mechanisms: improving test observablity or controllability, assuring autodiagnostics and on-line testing. The first class relates to design for test (DFT) approaches which provide a better access to functional block inputs (controllability) and outputs (observability). The most universal technique within DFT is the test scan path, which introduces a test mode with serial test interface. In the test mode the circuit flip-flops are reconfigured into a shift register. So, we can shift-in a test vector into internal flip-flops of the tested circuit, generate a single clock pulse to store the test responses in these flip-flops and then shift them out (e.g. to check by a signature monitor). This idea can be optimised by selecting only some subsets of flip-flops (partial scan path), creating multiple test paths, optimising their configuration, etc. In the literature various scan path techniques have been proposed to assure a balance between the testability and costs (circuit overhead and performance degradation) [1,22,33]. Some other ad hoc DFT solutions use multiplexers (for normal and test data, to cut signal loops, etc.) and other circuit specific adaptations (e.g. test registers embedded in hardly accessible circuit parts like address part of CPU cache memory). Typically, the required test patterns comprise a lot of don't cares, so sometimes it is reasonable to provide them to the tested circuit in a compressed form. This results in supplementary decompression circuitry embedded into DFT mechanisms [22]. Classical scan based tests assure high coverage for non delay defects. Delay fault testing needs more complex DFT circuitry or BISTs. However, native mode testing in software based self-tests facilitates to resolve this problem (compare section 3).

Natural extension of DFT techniques is autotesting circuitry (BIST – built-in-self-test). Circuit level (or structural) BISTs use various scan path techniques, employ linear feedback shift registers (LFSRs) and signature analysers (e.g. MISR – multi input signature register) for test vector generation and result compaction, respectively. They offer good test quality at the cost of area overhead and some performance degradation. Functional BISTs use special instructions for testing. Test instructions can be generated by an additional hardware (pseudorandom generator or deterministic generator). In 90s and early 2000s DFT and BIST circuitry comprised in most microprocessors covered a limited chip area ([22,29,32] and references). Most BISTs were targeted at simple test algorithms detecting single stuck-at faults. In most microprocessors BIST mechanisms are limited to some functional blocks such as cache memories, memory address translation buffer (TLB), branch target buffer (BTB) with branch prediction circuitry. In contemporary processors the microarchitecture circuitry is beyond DFT or BIST range due to unacceptable chip area overhead and possible performance degradation. For complex circuits, e.g. SoC (system on chip) or SiP (system in package), we may have separate BISTs related to various resources or common BISTs for similar functional blocks. The used BIST approaches are limited to some specific test algorithms covering a basic class of faults and cannot be adapted to new fault models revealed during exploitation ([29,31] and references). These limitations are systematically alleviated in recent microprocessors and some optimisation of test management is also introduced (lowering power consumption, reducing test time and test data volume). We will illustrate this trend in the sequel.

An example of complex processor with sophisticated DFT and BIST mechanisms is UltraSPARC T1 [32]. It is composed of 8 Sparc processor cores, each has 4 hardware supported threads per instruction pipeline. Each core comprises level one cache (L1) and has an access, via fast switch, to a common second level (L2) cache (4 ways banked). DFT features provide several test access modes and are targeted at production tests with high stuck-at fault coverage and high toggle coverage (bit flips) for burn-in tests. In RAMTEST mode a direct access interface is available to facilitate testing (externally) of embedded RAM structures (L1, L2 caches, etc.). In MBIST mode L1 and L2 caches are tested with BIST performing modified March C- algorithm (20n operations, n is the number of memory words). This is at-speed test (with internal clock frequency). For the remaining memory arrays (278 small memories) there is *Macrotest* support based on so called memory scan collars providing accesses to their inputs and outputs but at low speed. Such additional functional at-speed tests are needed for some critical memories. The processor is almost a full-scan design (only 3% flip-flops and latches are out of the scan). Good testability is assured by 32 parallel scan chains balanced to reduce test time: 16 chains within CPU cores (2 per core) and 16 within the non core circuitry. Tests are performed individually on each CPU core and additional tests check the interoperability of the multicore structure. To speed-up the tests, it is planed to develop parallel testing of all cores. Moreover, power testing is performed which involves maximum activity of cores up to 32 threads. The automatically generated test patterns (ATPG) cover 92% stuck-at faults, *Macrotest* patterns increase this to 94%. It was found that 4-8% memory arrays passed RAMTEST but failed MBIST. This confirms that dynamic defects in memories are important (detected by high speed MBIST, which assures at-speed testing).

A study of improving testability in AMD Athlon processor is presented in [22]. The main idea was to introduce modular testing based on scan chains. The initial single level design assumed scan paths with maximum 4000 flip-flops. Partitioning the chip into 10 modules scan paths were reduced to 33-921 flip-flops, at the cost of about 5% increase of flip-flops. Unfortunately, this resulted in an increase of the number of test patterns by 370% and 290% for compressed and non compressed version, respectively. However, the test time reduction due to shorter test paths was 40% and 68%, respectively. Introducing the second level of partitioning of the main modules (in total 33 submodules, 1-6 submodules per module) better results were achieved: test time reduction over 80% at the cost of 6.5% of additional flip-flops. The test data volume was similar to the basic solution. Moreover, the modular approach assured higher fault coverage.

In parallel with DFT and BIST mechanisms various on-line error detection and system monitoring circuitry are included. The on-line error detectors mostly relate to various error detection (or correction) codes in memory structures and transmission channels. They cover faults with limited multiplicity e.g. single faults. Most microprocessors generate so called exceptions in abnormal situations e.g. memory access violation (AV), illegal instruction (II), privileged instruction (PI), breakpoint (BP), stack overflow (SO), integer overflow (IO). We have analyzed the effectiveness of these mechanisms for a sample of programs executed under Windows at IBM PC platforms by injecting bit flip faults. These faults generated 16-70%, 0-15% and 13-50% exceptions, for faults located in the program code, data memory area and CPU registers, respectively. The distribution of activated exceptions was as follows (they confirm the dominating effectiveness of access violation):

AV: 56-96%, 95-100%, 98-100%; II: 3-14%, 0%, 0-1%;

SO: 0-6%, 0%, 0%; IO: 0-3%, 0%, 0%.

In contemporary processors we have more functional blocks with higher complexity. Quite often it happens that some blocks are replicated e.g. multicore processors, multiple ALUs, multithreaded control units. On one hand this creates new challenges for testing but on the other hand some possibilities to simplify testing appear. In systems with built in redundancy, e.g. based on duplication, test results can be easily compared from two modules e.g. at the level of bus transfers. Some processors adapted for fault tolerant structures comprise a built in comparator at the level of external bus. So in the slave mode they compare states on the bus with those produced internally by the processor. All these mechanisms improve test observability.

Many manufacturers have added so called performance counters to monitor the impact of various architectural mechanisms on the system performance [11,28]. These mechanists can also be used during testing to improve observability of some hardly accessed blocks e.g. related to the microarchitecture circuitry, instruction flow through the pipeline. Original studies related to this problem have been initiated in our Institute in 90s ([28] and references) and continued later on. Typically, the performance monitoring hardware (PMH) detects and counts events related to instruction decoding, memory co-operation, resource utilisation, instruction flow cache, TLB and BTB operation, etc. Some examples of monitored events are the number and type of instructions completed (loads, stores, branches, floating point). The utilisation of a considered resource can be characterised by counts of the number of cycles a given resource (e.g. floating point multiplier) is in use or busy. Appropriate configuration of monitor detectors and counters provides the requested information under various conditions for specified time intervals or program execution points, segments [11,28]. Sometimes event masks are added for better qualification of interesting events, e.g. in Pentium III detection of load accesses to the L2 cache allows specifying a mask to select accesses to lines with a specified state (modified, shared, exclusive, valid). We can similarly create a condition to mask out events qualified by the current privilege level of the processor. The counting process can be tuned to specified edge and threshold conditions. This allows counting the number of stalls or total number of stall cycles, the number of times the processor completed in a cycle more than a specified number of instructions, etc. Newly developed processors extend the number of detected events and counters. In first Pentium processors only a few counters were available. Pentium 4 provides 48 event detectors and 18 event counters. This assures more data collected simultaneously than in older processors. In the case of testing this will reduce the number of test runs. In the newer processors the interface between the monitoring hardware and software is becoming more complex, hence some supporting software is available at the operating system or user level [11] (*e.g. brink and abys*). In the case of multicore processors we can have independent counters for each core (e.g. POWER 4 implements 8 counters per CPU and they can be divided into 2 sets of 4 counters for monitoring each thread). The POWER 5 provides many events: the most critical are linked directly to the counters, others are selected and transmitted via three 32 bit busses. Beyond event monitoring some processors comprise supply current monitoring (IDDQ). Tests involving IDDQ monitoring are quite efficient, in [33] 90% stuck-at fault coverage has been achieved with 10 test patterns.

DFT and BIST mechanisms operate in a test mode, which differs from the normal operation mode. Hence, some circuitry used in the operational mode is not checked in the test mode (circuit reconfiguration). With DFT and BIST techniques we also face some power problems [36]. The achieved high test controllability results in frequent state changes of signals within the circuit during testing. This may be much more than during the normal operation, so the problem of excessive power consumption and circuit overheating may appear. This issue can be taken into account while configuring the test strategies e.g. by performing test sequentially for different functional blocks. As the nanotechnology scales down more attention is needed to delay faults by developing specially adapted DFT and BIST circuitry. Moreover, increasing circuit complexity leads to the requirements of faster tests and lower volume of test data (compression techniques). DFT and BIST mechanisms based on structural testing are not acceptable in many functional blocks (e.g. microarchitecture) due to excessive area overhead, performance degradation and power consumption.

Resuming we can state that hardware implemented testability improving mechanisms facilitate testing and increase fault coverage, but they do not resolve the testing problems completely. The outlined test gaps in DFT and BIST as well the need for functional tests checking module interactions and normal operational mode are challenging problems for software based testing.

3. SOFTWARE BASED SELF-TESTING

The drawbacks of hardware implemented testing (outlined in section 2) can be alleviated by software based testing (SBST), which uses inherent system intelligence of the incorporated CPUs. SBST techniques move the testing processes from external testers to the tested chip by using native processor language to generate test stimuli and analyse test responses. This approach assures an operation in normal conditions (at-speed tests). SBST can be used in classical computer systems, embedded SoC and SiP systems which comprise deeply integrated processors usually with no direct access from the outside.

In software based testing (SBST) the CPU applies test patterns to the functional blocks [4,8,12,13,23,29] by standard read/write instructions related to these blocks. The test data can be either stored in memory or generated by a test program running on CPU. The CPU captures test responses with read instructions. The methods of testing system functional blocks depend upon their program level visibility. From the program level perspective we can distinguish visible, partially visible, not visible and hidden components. For visible components we can control their inputs and observe outputs with single or coupled instructions. These components can be accessed via data (inputs and outputs relate to program data e.g. data processing units such as ALU, barrel shifter, memory and data transfer units) or via addresses (e.g. program counter, address calculation unit). Partially visible components generate control signals, which affect the operation of visible components (e.g. operation mode register, instruction sequencer). Hidden components are used for improving system performance (e.g. CPU microarchitecture circuitry). Hidden and nonvisible components (related to external environment co-operation) are difficult to test.

SBSTs can be based on functional or structural approaches. Functional SBSTs use high abstract level models related to instruction sets (e.g. [4,8,17,21,27]). To achieve high fault coverage they need long test sequences. Their effectiveness can be improved with additional pseudorandom instruction sequences (PSR) [19,26,27,30]. Structural SBST bases on finding test patterns targeted at specified components in relation to their low level models (e.g. gate level) and logic faults [4,6,15,34]. So, generated test patterns are transformed into appropriate instruction sequences (taking into account various restrictions). This approach can be encountered for simple processors and various test pattern generators were proposed. Many functional blocks of regular structure (e.g. ALU, barrel shifter, register file, RAM) can be tested with high accuracy without detailed knowledge of their implementation (mixed structural and functional approach [27]). In developed SBST methods the authors optimise test time, memory requirements, fault coverage, and average power consumption [30,36]. Some experience with this approach in our Institute is reported in [26-31] and references. In the sequel we comment selected results.

Classical functional test approaches assure 80-95% stuck-at fault coverage. Using structural approach it is possible to assure 95-99.8% fault coverage for path delays [15]. Pseudorandom tests of the same length assure only 40-80% [15]. However PSR tests may cover other fault models (including not known). Improved functional SBST for Plasma RISC processor assured 90-99.9% of fault coverage for visible components, 96.4% for hidden (pipelines), 63.1% for others and 95.6% total [19]. In [2] some extension of SBST into self diagnostics (SBDT) was proposed. It allows uniquely diagnose 62% of single stuck-at faults and to classify 84% in equivalent classes containing less than 10 faults for i8051 microcontroller [2].

Simple short pseudorandom tests assure relatively low fault coverage e.g. for OpenRISC1200 processor [16] basic random instruction test (about 37000 instructions) assures 68% fault coverage. Directed random test generator presented in [16] optimises test patterns generated for the processor functional model (gate level independent). This test uses some set of template instruction blocks, special test data related to control flow and corner cases. The operands in the block templates are random (several tenths of each template were instantiated). A balanced selection of templates assures over 72% fault coverage. Test cases are improved by identifying functional blocks with low fault coverage and adding more templates targeted at these blocks. In particular 300 additional templates targeted at multiplier circuitry increased fault coverage to 78% (the multiplier contained abut 42% of the processor faults). The third round of improvement (mostly for ALU – 500 additional template instances) assured about 83%. The random test for the register file was not satisfactory, so some RTL level deterministic test was added resulting in 86.5% of total CPU fault coverage. The test duration was about 27 ms, which is a small fraction for most applications. So, such tests can be embedded in periodic testing with reasonably high frequency to mimic on-line testing [31].

Developing test sequences for functional blocks with limited access we have to use special sequences of instructions to apply test stimuli and check their responses. This increases the test time and may interfere with optimal flow of test stimuli. This is more critical in testing components not visible at the software (assembler) level. In particular this holds for performance mechanisms related to speculative execution units e.g. branch prediction circuitry, instruction pipelines, prefetch units, address generation circuitry (microarchitecture circuits – [28,29] and references). Faults within this circuitry may result in performance and operational errors. The performance errors cause lower processing speed while assuring correct functional results. For example, an erroneous branch prediction may result in flushing the pipeline and correcting the misspredicted branch target during the branch completion phase. So, the only penalty is loss of some clock cycles. Operational errors propagate to the results of the functional tests due to the implied faulty execution flow of the test program.

Test controllability and observability problems in SBST approach can be alleviated with additional hardware improving testability (e.g. test registers, which provide direct access to blocks with low controllability or observability: bit cells in cache memories, BTB and TLB structures). We can also employ special test instructions dedicated to some functional blocks (instruction level design for testability). Such instructions may be interleaved with normal application related instructions to assure periodic testing ([23] and references). They can be executed during cache miss cycles or replace NOP instructions, which appear quite often due to branch delays, etc. For simple PAYEX RISC processor hardware BIST and instruction level DFT resulted in 13.1% and 5.6% area overhead with fault coverage (stuck-at faults) of 91-100% and 82-97%, respectively [23].

Test observability can be significantly improved using the on-chip counters as well as available on-line detectors. These mechanisms allow us to catch the effects, which normally escape classical software based functional tests targeted at operational errors. We use these mechanisms in testing CPU microarchitecture, in particular, circuitry involving parallel processing of instructions (e.g. decoding), speculative circuitry supported with various memories such as caches, TLB and BTB. They have significant impact on system performance, which is not visible directly at the program level.

Cache memories constitute significant part of CPUs. Unfortunately, software testing of caches is quite complex due to accesses conditioned by tag addresses (stored in cache directory) and comparison circuitry. Moreover, in instruction caches read and write operations are invoked by instruction execution. Original test procedures were presented in [25]. They can be significantly simplified using on-chip monitoring. In particular, we can count cache hits (or misses), data writes and CPU clock pulses during execution of test sequences. This simplifies and improves test observability. So, each test procedure has to be preceded with reading the states of these counters. They are verified at specified points of tests. For data caches we have reduced significantly test execution from 15n+9 to 11n+6 operations in the test of the cache control logic; and from 92n+2Del to 25n +7 + 2Del for the cache directory test based on March G algorithm (n is the number of memory words, Del is about 100ms delay). Similarly we have reduced abut 3 times the execution time of tests for instruction caches, in particular we do not need additional sequences assuring that the cache delivers the requested information (not the higher level memory) as this is resolved by monitoring cache hits, etc. More details are given in [30].

More spectacular are improvements of testing functional blocks optimising system performance. There is no direct access to individual pipe stages. However, it is possible to force appropriate stimuli to any of them using sophisticated sequences of instructions. Similarly, the reaction to the applied stimuli can be deduced by observing consecutive state changes of the whole pipe. This is quite cumbersome process which is simplified with the use of on-chip monitors (section 2). Developing such tests for Intel Pentium processors we used the following events: memory data read, data write, on-chip data or code cache misses, external cache misses, BTB hits and TLB misses, taken branches, number of mispredicted branches, instructions executed, number of cycles instruction fetch is stalled, number of cycles instruction length decoder is stalled, number of misaligned data memory references, number of instructions retired, number of simultaneously decoded *k* instructions, etc. Due to the limited number of available event counters, some tests required repetitions to collect more parameters. Moreover, to assure accurate results, some care is needed (e.g. the test measurement sequence is aligned to 16 byte boundaries and interrupts are disabled). Additionally, to eliminate cache loading and branch prediction effects, the measurement sequence is repeated 6 times (loading cache, setting 4-bit BTB history and final execution). The last run gives the stable results. For an illustration tab. 1 presents a set of performance characteristics of *Qsort* program. They were derived from activated on-chip monitoring counters. In the case of performance or operational faults the presented parameter values will be disturbed. Some other examples are given in [28,29].

Table 1

Selected results of monitoring *Qsort* program

CPU clock pulses: 14195	BTB misses: 170
Number of decoded instructions in a clock	misspredicted taken branches: 55
cycle:	branches decoded: 1740
0 – 2743, 1 – 5863, 2 – 3277, 3 – 2325	resource stalls: 2393

It is relatively easy to check the functional coverage of developed tests in relevence to the considered blocks. We did this using statistics of monitored events, executed instructions, sequences of instructions, etc. This data was also helpful in refining test capabilities. Fault coverage analysis needs special tools and RTL level models of processors. Interesting results are presented in [10]. They relate to RISC MIPS R10000 processor model for 1 and 2 bit branch predictors. The classical tests assure 76% and 79% coverage of stuck-at faults in 1 and 2 bit branch prediction circuitry, respectively. These tests enhanced with on-chip monitoring assure over 97% and 96% stuck-at fault coverage, respectively. Enhancing all functional tests with event monitoring improved fault coverage to the level of 93% and 92%, respectively. Classical CPU functional tests (with no sequences dedicated to BTB) assure 72% and 74% fault coverage of the whole processor and about 42% fault coverage for BTB. Here it is worth noting that the BTB circuitry contributes 25% and 75% of hardware for control logic (mostly susceptible to performance errors) and memory storing branch information (mostly susceptible to operational errors), respectively.

In the literature on SBSTs most authors deal with stuck-at fault models and use deterministic or random tests generated for functional of RTL models of processors. SBST is promising to cover path delay faults, however, generation of effective test programs is still an open problem. Recently some effective solutions dealing with delay faults have been also proposed e.g. [15,24]. In [15] functionally testable paths are identified from the CPU gate-level model taking into account restrictions resulting from instruction set. Next, for such paths, appropriate instruction sequences covering signal propagation are derived. Most of the proposed delay tests base on RTL circuit models and sometimes are supported with special test instructions [2]. In practice it is not possible to sensitise all possible signal propagation paths, so an important issue is to select the most representative and critical ones e.g. signal paths with delay crossing some specified threshold, like *a*% of the clock period (in [5] it is assumed that *a*= 5%). Moreover, we can eliminate functional redundant paths i.e. paths which do not determine circuit performance. In [3] tests of path delay faults were generated for 8051 microcontroller (using RTL model). The identified critical paths were grouped into coherent sets which can be sensitised simultaneously (they begin at some register output and terminate at some register inputs). The generated tests (1.3 KB of static code) cover 97% of considered paths. It is worth noting that classical functional test targeted at stuck-at faults (with 94% coverage) assures only sensitisation of 26% paths and detects 13% delay faults. This confirms the need of special dynamic tests.

In [9] the authors present a methodology of mapping dynamic tests into CPU instructions for RISC processor OR12000 basing on its Verilog RTL description. The authors generate automatically tests for robust delay faults and paths whose delays are over a specified threshold. It assures 96% fault coverage. In [5] more complex processor was considered (Sun's Niagara multiprocessor chip). The authors propose adding special set of instructions (Access Control Extension – ACE) improving test controllability and observability of internal functional blocks. This enhancement leads to 5.8% chip area increase. The software based test procedures can be performed also in a periodic way without loosing the current state of the processor. So this can mimic on-line testing as well. The software nature of tests provides the capability of upgrading tests during the processor lifetime. The fault coverage achieved is 99.2% (for individual CPU modules it ranged from 91.4 to 100%). Test lengths for a single CPU core were in the range 200K and 150K instructions for tests related to stuck-at and path delays faults, respectively. The path delay tests are limited to paths with delays within 5% of the clock period.

Delay faults involve more complex techniques and tools to find test paths and stimuli. They can be improved with evolutionary approach [16]. It is worth noting that testing faults related to delays and various data, instruction or structural (resource) dependencies is based on very specific instruction sequences which should not be disturbed by test observability requirements (e.g. embedded checking instructions). This problem can be alleviated by using on-chip monitors and on-line detectors which deliver directly some responses (with no interference with normal operation). In multicore systems arises the possibility of more accurate comparison of tests executed using different resources. This comparison can be performed at specified boundaries e.g. microatchitectural level, but this requires built-in comparators.

4. APPLICATION BASED TESTING

In many embedded microprocessor systems there is only one program executing some specified tasks. This creates a possibility of simplifying testing by restricting it only to the used resources. Practically, data processing resources are used in a large extent by most programs (e.g. ALU, register stack), so they should be tested for a representative test cases. These blocks have good test controllability and observability, so they are relatively easy to test. The most difficult part of CPU relates to instruction decoder, sequencer and control unit blocks, interrupt handling circuitry and performance enhancement microarchitecture circuitry. Complete testing of this circuitry is a challenging task, which was discussed in previous sections. In the case of a fixed program the scope of used functionality within these circuits is significantly reduced (e.g. the number of used instruction codes). Moreover, the data and instruction flow is fixed, so the problem of various instruction and data dependencies is limited to fixed patterns that can be tested completely as opposed to universal testing which needs taking into account wide range of possible dependencies within a specified time window (or instruction cycles). Similarly we can test the propagation of signals in the sensitised paths during the application execution. These remarks lead us to application dependant testing. We have checked this possibility for some real microcontrollers and here some results are given for an illustration.

Developing test procedures for the microcontroller, it is interesting to collect statistics related to resource usage: static and dynamic instruction coverage, activity of resources, register state changes, sequences of correlated instructions, distribution of memory states, etc. For this purpose we use various software tools developed in the Institute. They provide useful information, which facilitates constructing test programs. We illustrate this with some statistics of developed programs for two microcontroller applications: a car immobiliser and controller of a chemical process.

The car immobiliser was implemented using Atmel AT89C51 microcontroller. The developed program has been written in C language (800 lines) and comprises 1312 bytes of code (577 controller instructions) [34]. The program uses only 29 machine instructions from the list of 255 opcodes. Dynamic distribution of instructions is given in Fig. 1. The most frequently used instructions are MOV (26%), JB (7.8%), JNB (6.9%). There are 12 instructions within the group *Others* (SJMP 2.8% and the remaining 11 – 0.8%). The static instruction distribution is a little bit different. However, the general profile is similar to the dynamic one. For comparison purposes an equivalent program was developed for Intel 80×86 processor platform. The code length was more than twice longer (3157 bytes, which corresponds to 832 instructions). The program uses only a small percentage from the very long reach list of instructions (26 different instructions). The distribution of most frequently used instructions is as follows: *mov* – 29.9%, *and* – 18.38%, *test* – 8.3%, *jnz* – 7.3%, *shr* – 7.2%, *push* – 7,5% (lower case mnemonics distinguish these opcodes from AT89C52 ones).

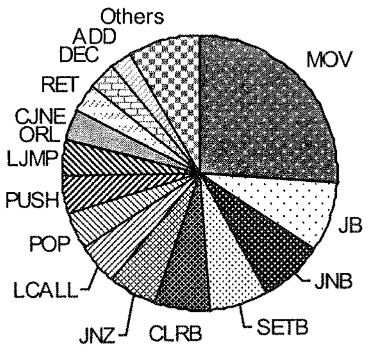


Fig. 1. Instruction statistics for a car immobiliser (AT89C51 microcontroller)

Resource usage of the controller is significantly restricted. The program uses only 1312 bytes within the controller 4kB flash memory and only 32 out of 128 bytes of data RAM (22 locations assume only a few states). The on-chip used peripherals are: timers/counters, port P0 (5 bits within 8) and port P1 – 7 bits. The arithmetic unit is used scarcely. Such statistics are useful in developing self test programs. This allows us to optimise diagnostic programs. On one hand, there is no need to check not used resources (e.g. serial transmitter in our case). On the other hand, the control programs are fixed, so it is more effective to generate self-test procedure on the base of the application then as a universal test program [29]. The developed 6 test scenarios cover all possible situations during normal operation of the immobiliser. They cover faults within instruction flow, decoding and the control unit (instruction sequencer). The data path processing blocks (ALU) and memory are tested in classical way, these blocks have good controllability and observability from the software level.

Another example of microcontroller was rectification column control based on DMC prediction algorithm [7]. The controller program implemented on Intel 80×86 platform written in C++ practically used only a small fraction of available resources. Fig. 2 shows static and dynamic distribution of processor instructions. They relate to typical control scenarios described by single step changes of input states. Checking program flow coverage validated this test scenario.

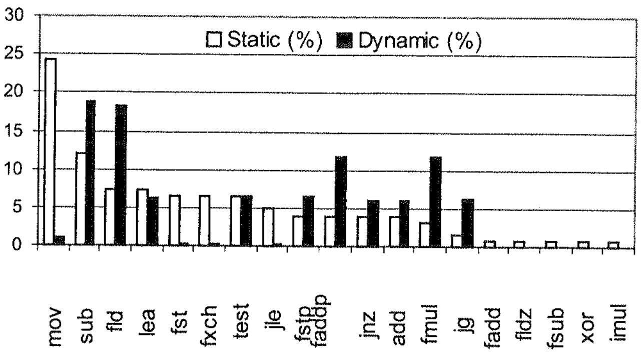


Fig. 2. Instruction distribution for the Intel 80×86 microcontroller of the rectification column

The program uses only 19 instructions including 9 floating point instructions. This is a small fraction of CPU and FPU instructions. This shows that we can reduce significantly the testing procedure for instruction decoder, sequencer, and microarchitecture pipelines. These circuits are sensitive to various data and instruction dependencies, which are very difficult for universal testing of the considered processor. For example the number of possible dependencies within any sequence of 10 instructions is enormous. In the discussed application we can take into account only a small fraction of the processor instructions (those used in the program). The dynamic distribution confirms that only 50% of used instructions have a dominant impact on appearing dependencies. All of them are verified by the application driven test.

It is also worth noting that only some subset of data processing functions (both integer and floating point) are used, e.g. no division operations, neither elementary functions nor barrel shifter. Hence, the added deterministic tests are significantly reduced. This also holds for the used addressing modes and other resources. The application based tests are checked by observing generated outputs of the DMC controller (4800 bytes for a test scenario composed of 300 iterations). Moreover, they can be easily compacted at run time into a single value to save the memory space if required. Hence, the test observation does not influence control and date flow of the application. In other words, the testing process is consistent with the normal operation conditions and the available on-line detectors can directly signal many faults. In simulation experiments with injected transient faults we have found that 59.6%, 0.7% and 52.7% of faults injected into code, data and registers generate system exceptions. This improves test observability.

5. CONCLUSIONS

The paper shows that the effectiveness of hardware implemented DFT and BIST mechanisms is limited in functional and fault space coverage due to the cost compromise. Software based testing gives new perspectives in this respect. However, some functional blocks or circuits (microarchitecture) cannot be tested efficiently due to low test controllability and observability. Hence, an important issue is to combine software testing with hardware BISTs and support it with some additional hardware mechanisms. We have proved that on-chip monitoring and on-line error detectors are useful in this process. Microprocessor circuits are quite complex, so developing universal effective tests covering all the functionality is cumbersome. In many applications this functionality is used in a limited way and this gives a chance of significant simplification of test algorithms. This was illustrated with some examples. The proposed approaches give also new possibilities in efficient testing delay faults.

A new view is needed on self-testing integrating various approaches and techniques. In particular, good interfaces with hardware implemented test mechanisms are needed. In SoC and SiP chips the proposed approach may gain higher interest due to the high complexity and limited access to internal resources. It is also interesting to use programmable logic (e.g. FPGA) which can be reconfigured during testing to improve test controllability and observability.

6. REFERENCES

1. M. Abramovici, M. Breuer, A. D. Friedman: *Digital system testing and testable design*, Computer Science Press, 1996.
2. P. Bernardi, et al.: *An effective technique for minimising the cost of processor software based diagnosis in SoCs*, IEEE DATE Conference, 2006, pp.412-417.
3. P. Bernardi, et al.: *On the automatic generation of test programs for path delay faults in microprocessor cores*, 12th IEEE European Test Symp. 2007, pp. 179-184.
4. L. Chen, S. Ravi, A. Raghunathan, S. Dey: *A scalable software based self-test methodology for programmable processors*, IEEE DAC Conference, 2003, pp. 548-553.
5. K. Constantinides, O. Multu, T. Austin, V. Beratcocco: *Software based online detection of hardware defects*, 40th IEEE/ACM Int. Symposium on Microarchitecture, 2007, pp.97-108.
6. F. Corno, E. Sánchez, M. S. Reorda, G. Squillero: *Automatic test program generation: a case study*, IEEE Design and Test of Computers 21, 2004, pp. 102-109.
7. P. Gawkowski, M. Ławryńczuk, P. Marusak, P. Tatjewski, J. Sosnowski: *Software implementation of explicit DMC algorithm with improved dependability*, In T. Sobh et al. (Eds.), Novel Algorithms and Techniques in Telecommunications, Automation and Industrial Electronics, Springer Science+Business Media B.V., 2008, pp. 214-219.
8. S. Gurumurthy, S. Vasudevan, J. A. Abraham: *Automated mapping of precomputed module level test sequences to processor instructions*, IEEE Int. Test Conference, 2005, paper 12.3.
9. S. Gurumurthy, et al.: *Automatic generation of instructions to robustly test delay defects in processors*, IEEE European Test Symposium, 2007, pp. 173-178.

10. M. Hatzmibail, M. Psarakis, D. Gizopoulos, A. Paschalis: *A methodology for detecting performance faults in microprocessors via performance monitoring hardware*, IEEE Int. Test Conference, 2007, paper 29.3.
11. L. K. John, L. Eeckhout, (editors): *Performance evaluation and benchmarking*, CRC Taylors &Francis, 2006.
12. K. Kambe, M. Inoue, H. Fujiwara: *Efficient template generation for instruction-based self-test of processor cores*, IEEE Asian Test Conference, 2004, pp.151-158.
13. N. Kranitis, et al.: *Application and analysis of RTL-level software based self-testing for embedded processor cores*, IEEE Int. Test Conference, 2003. pp. 715-785.
14. A. Krstic, W. C. Lai, L. Chen, K.-T. Cheng, S. Dey, *Embedded software based self-testing of SoC design*, IEEEE DAC Conference, 2002, pp.355-359.
15. W. Ch. Lai, A. Krstic, K-T. Cheng: *Test program synthesis for path delay faults in microprocessor cores*, IEEE Int. Test Conference, 2000, pp. 1080-1089.
16. A. Merentitis, G. Theodorou, M. Giorgaras, N. Kranitis: *Directed random SBST generation for on-line testing of pipelined processors*, IEEE Int. On-Line Testing Symposium, 2008, pp. 273-279.
17. P. Mishra, N. Dutt: *Functional coverage driven test generation for validation of pipelined processors*, IEEE DATE Conference, 2005, pp. 678-683.
18. A. D. Palma, et al.: *Automotive microcontroller end-of-line test via software based methodologies*, 8th Int. Workshop on Microprocessor Test and Verification, 2008, pp.77-82.
19. A. Paschalis, D. Gizopoulos: *Effective software based self-test strategies for on-line periodic testing of embedded processors*, IEEE Trans. on CAD, vol. 24, no. 1, 2003, pp. 88-99.
20. M. Psarakis et al.: *Systematic software based self test for pipe-lined processors*, IEEE DAC Conference, 2006, pp. 393-398.
21. H. Rizk, C. Papachristou, F. Wolff: *Designing self test programs for embedded DSP cores*, IEEE DATE Conference, 2004, pp. 816-823.
22. A. Sehgal, J. Fitzgerald, J. Rearick: *Test cost reduction for the AMD Athlon processor using test partitioning*, IEEE Int. Test Conference, 2007, paper 1.3.
23 S. Shamshiri et al.: *Instruction level test methodology for CPU core self testing,* ACM Trans. on Design Automation, vol. 10, no. 4, 2006. pp. 673-689.
24. V. Singh, M. Inoue, K. K. Saluja, H. Fujiwara, *Instruction-based delay fault self-testing of processor cores*, IEEE Int. Conf. on VLSI Design, 2004, pp. 933-938.
25. J. Sosnowski: *In system testing of cache memories*, IEEE Int. Test Conference, 1995, pp. 384-393.
26. J. Sosnowski, A. Kusmierczyk: *Pseudorandom testing of microprocessors at instruction/data flow level*, The 2nd EDCC Conference, Springer, 1996, pp.246-263.
27. J. Sosnowski, T. Bech: *Extensive testing of floating point unit*, Euromicro Conference, IEEE Comp. Society, 2000, pp. 180-187.
28. J. Sosnowski, R. Jurkiewicz, J. Nowicki: *Experimental evaluation of CPU performance features*,. Proc. of EUROMICRO DSD Symposium, IEEE Comp. Soc., 2001, pp. 194-201.
29. J. Sosnowski: *Software based self-testing of microprocessors*, Journal of System Architecture, 52, 2006, pp. 257-271.
30. J. Sosnowski: *Improving software based self-testing for cache memories*, 2nd IEEE Int. Design and Test Wokshop, 2007, pp.49-54.
31. J. Sosnowski: *Enhancing software tests for COTS systems*, IEEE East-West Desgn and Test Int. Symposium, 2007, pp. 63-68.
32. P. J. Tan et al.: *Testing UltraSPARC T1 microprocessor and its challenges*, IEEE Int. Test Conference, 2006, paper 16.1.
33. D. Wang, et al.: *The design for testability features of a general purpose microprocessor*, IEEE Int. Test Conference, 2007, paper 9.2.

34. Ch. H.-P. W e n et al.: *On a software based methodology and its application*, IEEE VLSI Test Symposium, 2005, pp.107-113.
35. A. W i l c z y ń s k i, J. S o s n o w s k i, P. G a w k o w s k i: *Flexible microcontroller simulator for testing purposes*, IFAC Workshop PDS 2004, pp. 310-315.
36. J. Z h o u, H-J. W u n d e r l i c h: *Software based self-test of processors under power constraints*, IEEE DATE Conference, 2006, pp. 430-435.

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 47–56

Dedicated to the memory of our friend Andrzej Mąkowski

On the Distribution of Numbers n Satisfying the Congruence $2^{n-k} \equiv 1 \pmod{n}$ for k=2 and k=4[*]

ANDRZEJ PASZKIEWICZ[*], ANDRZEJ ROTKIEWICZ[**]

[*] *Institute of Telecommunications, Warsaw University of Technology*
anpa@tele.pw.edu.pl
[**] *Institute of Mathematics, Polish Academy of Sciences*
rotkiewi@impan.gov.pl

Received 2008.12.02
Authorized 2009.01.08

In this paper we address an old problem concerning the existence of infinitely many solutions n of the congruence $2^{n-k} \equiv 1 \pmod{n}$ for an arbitrary positive integer k. The existence of infinitely many solutions of that congruence follows from more general but not constructive theorems, which do not give an answer about the number of solutions below a given limit x. It is well known that if $k = 1$, then our congruence hold for every prime number $n > 2$ as well as for infinitely many odd composite integers n, called pseudoprimes. If $k = 3$ then every number n of the form $3p$ (p an odd prime) is a solution of the congruence $2^{n-3} \equiv 1 \pmod{n}$. We study the distribution of consecutive solutions of our congruence in the two simplest but resistant cases $k = 2$ and $k = 4$

Keywords: prime and pseudoprime numbers, prime primitive divisors, Mersenne numbers

1. INTRODUCTION

Let a, $k > 1$ be arbitrary positive integers. The second author asked in many occasions (see [10]) about the existence of infinitely many composite integers n, such that

$$a^{n-k} \equiv 1 (\mathrm{mod}\ n) \tag{1}$$

[*] The computational part of his paper was supported by the Ministry of Science and Higher Education of Poland - research grant no. N517 003 32/0583 for 2007-2010

It is well known that the answer is affirmative in the case $k = 1$. The numbers satisfying the condition are called pseudoprime numbers to the base a. In [3] A. Mąkowski obtained a general result: for any natural number $k \geq 2$ there are infinitely many composite n such that for any positive integer a with $(a, n) = 1$ such that (1) holds. This extends an earlier result proved by D.C. Morrow for $k = 3$ (see [7]).
W.L. McDaniel proved the following general theorem.
Theorem:(McDaniel) *The congruence* $a^{n-k} \equiv b^{n-k} \pmod n$ *has infinitely many composite solutions n for all triples* (a, b, k) *with possible exception of* $(2, 1, 4)$, $(3, 2, 3)$, $(7, 3, 3)$, $(2^u + 1, 2^u - 1, 3)$ *for* $u \geq 2$ *and* $(b+1, b, 2)$ *and* $(b+3, b, 2)$ *for* $b \neq 1$.

As we can see the case $k = 4$ is not covered by the above Theorem. The second author has proved (see [9], [10]) that the above Theorem is valid in the case of $k = 2$. Fortunately technique of that proof can also, with minor changes, be applied to obtain the proof for $k = 4$. First we remark, that if $2^m \equiv 5 \pmod m$, then $n = 2^m - 1$, satisfies the congruence $2^{n-4} \equiv 1 \pmod n$. Indeed if $(2^m - 5) / m$ is a positive integer, then from the congruence $2^m \equiv 1 \pmod{2^m - 1}$ it follows that

$$\begin{gathered}(2^m)^{(2^m-5)/m} \equiv 1 (\text{mod } 2^m - 1),\\ 2^{2^m-5} \equiv 1 (\text{mod } 2^m - 1) \textit{ and}\\ 2^{n-4} \equiv 1 (\text{mod } n) \text{ for } n = 2^m - 1.\end{gathered}$$

Thus $2^{n-4} \equiv 1 \pmod n$ for $n = 2^{n_0} - 1$, where $n_0 = 19147$ (This is the least solution of the congruence $2^n \equiv 5 \pmod n$).

Suppose now that $2^{n-4} \equiv 1 \pmod n$ and $n > 10$. Let p be a primitive factor of the number $2^{n-4} - 1$ (a prime factor of $2^n - 1$ is said to be a primitive if it does not divide any of the numbers $2^m - 1$ for $m = 1, 2, \ldots, n-1$. By the theorem of K. Zsigmondy [11] such a prime factor exists for any $n > 6$ and is of the form $nt + 1$).

Now we shall show, that $n_1 = np$ is also a solution of the congruence

$$2^{n_1-4} \equiv 1 \ (\text{mod } n_1).$$

We have $p = 2(n-4)k + 1$, where k is a positive integer and $p \geq 2n - 5 > n$ and $(p, n) = 1$. Thus

$$np - 4 = n[2(n-4)k + 1] - 4 = (n-4)(2nk + 1).$$

Hence $2^{n-4} - 1 \mid 2^{np-4} - 1$, and since

$$\begin{gathered}2^{n-4} \equiv 1 (\text{mod } n),\\ 2^{n-4} \equiv 1 (\text{mod } p), (p, n) = 1,\end{gathered}$$

we have $np \mid 2^{np-4} - 1$ and $n_1 = np$ satisfies the congruence $2^{n_1-4} \equiv 1 \pmod{n_1}$.

This completes the proof of the first exception in the McDaniel's Theorem. It has to be pointed out, that general proof of Rotkiewicz's Problem concerning the congruence (1) was obtained, in rather complicated way, by Kiss and Phong [2]. It is easy to see that from every solution of the congruence $2^m \equiv k+1 \pmod{m}$, k – fixed we can get a solution of the congruence $2^{n-k} \equiv 1 \pmod{n}$, but even in the simplest case $k = 2$ we do not know if the converse is true. An old conjecture of R. L. Graham [1] asserts that for all $k \neq 1$, there are infinitely many n such that $2^n \equiv k \pmod{n}$. In many cases solutions of the congruence $2^n \equiv k \pmod{n}$ are relatively small numbers, but unfortunately for $k = 3$, there are known only 5 solutions. Four of them are listed in the Table 1 below.

Table 1

Least consecutive solutions n of the congruence $2^n \equiv 3(\text{mod } n)$

4700063497	D. H. and Emma Lehmer
3468371109448915	Max Alekseyev, Nov 2006
8365386194032363	Joe Crump, 2000
10991007971508067	Max Alekseyev, Nov 2006

The fifth solution was found by Peter Montgomery in 1999 by factoring $2^{485} - 3$ with SNFS (see [6]) and is the following:

$$n = 63130707451134435989380140059866138830623361447484274774099906755.$$

This is a good idea to find solutions $2^n = c \pmod{n}$ by other means rather than just extensive trial method. We can, for instance, factor $2^x - c$ into primes p for various x and test if $n = px$ is a solution. But for very large n (having for example about 512 bits) this approach has also limited applications. Without new theoretical ideas solving the congruential equation $2^n = c \pmod{n}$ for a fixed c seem to be hard problem. Nevertheless J. Crump, M. Alekseyev and P. Montgomery, just mentioned before, found new solutions of the congruence $2^n = c \pmod{n}$ for initial odd numbers c not exceeding 25.

For every $k < 1000$ there exists a solution of the congruence $2^n \equiv k(\text{mod } n)$ (see Joe Crump's web side www.immortaltheory.com/NumberTheory/2nmodn.htm). We list below (Table 2) only these solutions n which are greater than 10^9 and any smaller are known.

J. Crump observed some interesting facts when computing $2^n \pmod{n}$ concerning extremely not uniform distribution of these numbers. First is that $2^n \pmod{n}$ is mostly even. For every prime $n > 2$ the result is 2. Surprisingly, for no evident reason some odd numbers occur very often. The first of them is 33857 which occurs 90 times using the first 5000000 odd numbers. Next up is 233927 which occurs 84 times followed by the 125000 which occurs 76 times. However the domination seems to be temporary. At first 33857 dominated these three but quickly gave up the place to the number 233927.

Table 2

Solutions n of the congruence $2^n \equiv k \pmod{n}$ $(k < 1000, n > 10^9)$

k	n	k	n
33	2463240427	69	887817490061261
141	6782813041	185	680827363701
231	220567109627	273	137251416059
309	157458552833	311	343738144742266767
313	20703919615	399	1754831987
405	2908385595517	461	3885540629
465	164196324252985941533	510	14039809331
581	303976350129	609	7933293691
615	8495801767	619	855892952693
645	1067706307	649	3577612969291381957
651	2263272601	666	7213999231
669	455286451057	675	5808628761697
685	4677383419	741	9988731091037308003
771	128053255763	799	25455747038179
831	9741494677	849	2725497894187
861	8592082903	866	20935242067
871	930186118118867	881	9736056265781
885	100882904641861	913	2120018333
939	141691000109	969	93615868517
981	4443747872057	987	1158892877

2. NUMERICAL INVESTIGATIONS AND RESULTS

In our joint paper [8] we asked about growth rate of a function $C_2(x)$, where $C_2(x)$ is the number of solutions of the congruence $2^{n-2} \equiv 1 \pmod{n}$ with n less than a positive number x. For a given positive integer k we can in general define $C_k(x)$ as a number of solutions of the congruence $2^{n-k} \equiv 1 \pmod{n}$ below x. We called them in [8] – Fermat's congruences with delay k. In this paper we significantly extend the table of solutions of the congruence $2^{n-k} \equiv 1 \pmod{n}$ for the delay 2, previously prepared for the limit $n < 10^{10}$. We also determined all solutions of the congruence $2^{n-k} \equiv 1 \pmod{n}$ with delay $k = 4$ for $n < 10^{11}$.

Our computations were performed in a small laboratory consisting of 10 Pentium PC computers during their idle time. All computers were running under Windows XP operation system and collecting all entries of Tables 3 and 4 took less than 30 hours of work on every of the 10 computers.

Table 3

The initial solutions n_k of the congruence $2^{n_k-2} \equiv 1 \ (mod\ n_k)$ for n_k less than 10^{11}

k	n_k	k	n_k
1	20737=89 · 233	2	93527=7 · 31 · 431
3	228727=127 · 1801	4	373457=7 · 31 · 1721
5	540857=31 · 73 · 239	6	2231327=7 · 151 · 2111
7	11232137=7 · 31 · 191 · 271	8	15088847=31 · 233 · 2089
9	15235703=7 · 79 · 27551	10	24601943=79 · 239 · 1303
11	43092527=71 · 337 · 1801	12	49891487=47 · 71 · 14951
13	66171767=89 · 233 · 3191	14	71429177=31 · 1103 · 2089
15	137134727=127 · 151 · 7151	16	207426737=7 · 151 · 311 · 631
17	209402327=23 · 199 · 45751	18	269165561=7 · 79 · 233 · 2089
19	302357057=23 · 463 · 28393	20	383696711=89 · 233 · 18503
21	513013327=31 · 32575081	22	1145222057=7 · 31 · 191 · 27631
23	1198235777=31 · 2089 · 18503	24	1200963953=7 · 73 · 89 · 26407
25	1210344599=73 · 337 · 49199	26	1336271543=89 · 233 dot 64439
27	1530206537=23 · 79 · 842161	28	1654163777=7 · 151 · 431 · 3631
29	2247340097=23 · 79 · 151 · 8191	30	2383604687=71 · 73 · 199 · 2311
31	2745926897=7 · 31 · 1831 · 6911	32	3067561177=71 · 1151 · 37537
33	3444456017=73 · 89 · 151 · 3511	34	3543720833=73 · 48544121
35	3567496337=31 4177 · 27551	36	3638049527=7 · 271 · 601 · 3191
37	4135367777=23 · 127 · 337 · 4201	38	4343487407=7 · 31 · 431 · 46441
39	4404655367=7 · 31 · 3191 · 6361	40	5056376807=191 · 271 · 97687
41	5108079407=23 · 79 · 881 · 3191	42	5793119327=31 · 73 · 239 · 10711
43	6008364727=23 · 31 · 1801 · 4679	44	6629012777=127 · 631 · 82721
45	6876643727=31 · 71 · 73 · 127 · 337	46	7650778457=7 · 151 · 631 · 11471
47	8143707497=23 · 3319 · 106681	48	8369326319=23 · 607 · 599479
49	8605878287=31 · 863 · 321679	50	9039241577=151 · 487 · 122921

cd. Table 3

k	n_k	k	n_k
51	9046381577=127 · 1801 · 39551	52	10702466777=79 · 463 · 292601
53	10915386649=233 · 727 · 64439	54	12193861799=7 · 79 · 4177 · 5279
55	12299106503=23 · 73 · 103 · 71119	56	13985638127=31 · 13367 · 33751
57	17312443631=7 · 103 · 503 · 47737	58	17343716537=7 · 31 · 1721 · 46441
59	17554812167=73 · 881 · 272959	60	19414922177=7 · 89 · 199 · 156601
61	22869186617=73 · 89 · 151 · 23311	62	23155375817=7 · 31 · 73 · 79 · 18503
63	25367354777=31 · 71 · 127 · 151 · 601	64	25494650777=191 · 199 · 631 · 1063
65	26508475007=7 · 31 · 233 · 524287	66	27823040777=7 · 47 · 223 · 601 · 631
67	31109707127=337 · 751 · 122921	68	32024538119=7 · 73 · 89 · 704161
69	35802513623=47 · 10711 · 71119	70	35879030327=7 · 311 *cdot* 1801 · 9151
71	42245251127=23 · 71 · 89 · 290671	72	47909079407=7 · 73 · 89 · 991 · 1063
73	51416714351=199 · 431 · 599479	74	51563126617=127 · 1871 · 217001
75	55837893377=7 · 31 · 73 · 271 · 13007	76	63731873207=23 · 71 · 73 · 89 · 6007
77	64999529399=7 · 89 · 199 · 524287	78	72236945497=89 · 233 · 3483481
79	74605302977=337 · 1801 · 122921	80	75049066127=7 · 31 · 47 · 73 · 100801
81	78625368857=7 · 151 · 3191 · 23311	82	78943150127=271 · 337 · 751 · 1151
83	82597478537=7 · 23 · 79 · 991 · 6553	84	83072478127=127 · 22751 · 28751
85	83266672777=89 · 167 · 881 · 6359	86	90578716697=7 · 31 · 271 · 631 · 2441
87	94437919487=911 · 4447 · 23311	88	97019792537=31 · 431 · 1609 · 4513

Table 4

The initial solutions n_k of the congruence for $2^{n_k-4} \equiv 1 \ (mod\ n_k)$ for n_k less than 10^{11}

k	n_k	k	n_k
1	7=7	2	40369=7 · 73 · 79
3	673663=337 · 1999	4	990409=7 · 151 · 937
5	1697609=127 · 13367	6	2073127=7 · 73 · 4057
7	6462649=127 · 151 · 337	8	7527199=79 · 151 · 631
9	7559479=23 · 103 · 3191	10	14421169=7 · 31 · 66457

cd. Table 4

k	n_k	k	n_k
11	21484129=79 151 · 1801	12	37825753=7 · 73 · 79 · 937
13	57233047=337 · 169831	14	130647919=31 · 631 · 6679
15	141735559=7 · 73 · 79 · 3511	16	179203369=89 · 631 · 3191
17	188967289=7 · 31 · 73 · 79 · 151	18	218206489=31 · 337 · 20887
19	259195009=11119 · 23311	20	264538057=7 · 73 · 79 · 6553
21	277628449=7 · 151 · 262657	22	330662479=7 · 73 · 79 · 8191
23	398321239=79 · 151 · 33391	24	501126487=1447 · 346321
25	506958313=4177 · 121369	26	612368311=47 · 73 · 178481
27	767983759=89 · 103 · 83777	28	936337783=79 · 1447 · 8191
29	1009345159=71 · 79 · 179951	30	1043030839=103 · 151 · 199 · 337
31	1557166769=23 · 743 · 91121	32	1654549879=199 · 751 · 11071
33	1674606289=89 · 271 · 69431	34	2179652017=23 · 2927 · 32377
35	2487376129=7 · 601 · 631 · 937	36	2945176609=127 · 2647 · 8761
37	3060103279=31 · 73 · 631 · 2143	38	3073399489=31 · 79 · 151 · 8311
39	3141290569=337 · 1999 · 4663	40	3476295697=7 · 73 · 79 · 86113
41	3702682129=31 · 73 · 631 · 2593	42	3811185679=7 · 47 · 73 · 89 · 1783
43	4040908249=7 · 23 · 31 · 881 · 919	44	4085251609=47 · 487 · 178481
45	4345359529=7 · 73 · 79 · 107641	46	4774978639=31 · 127 · 1212847
47	5145413719=103 · 2143 · 23311	48	5207822239=71 · 151 · 199 · 2441
49	5434493719=31 · 4409 · 39761	50	8112440119=7 · 151 · 937 · 8191
51	8952662047=7 · 103 · 12417007	52	9509147641=7 · 727 · 1868569
53	9590474929=7 · 31 · 151 · 487 · 601	54	10425250879=79 · 8191 · 16111
55	12160941223=337 · 36085879	56	12746032159=7 · 151 · 271 · 44497
57	15351309439=337 · 4663 · 9769	58	15524707279=23 · 31 · 71 · 73 · 4201
59	16984371289=127 · 5737 · 23311	60	18023668537=7 · 73 · 79 · 446473
61	18335476849=151 · 5209 · 23311	62	19745250529=239 · 4967 · 16633
63	22442226769=31 · 127 · 607 · 9391	64	22534825009=31 · 73 · 89 · 127 · 881
65	22677320761=337 · 4663 · 14431	66	23110996279=7 · 31 · 4513 · 23599
67	26427995689=79 · 151 · 631 · 3511	68	26921830279=103 · 2593 · 100801

cd. Table 4

k	n_k	k	n_k
69	28706132137=337 · 7993 · 10657	70	28837802569=23 · 233 · 937 · 5743
71	28970538007=23 ·89 ·199 ·71119	72	30502999993=23 · 89 · 103 · 199 · 727
73	32510425369=31 · 10657 · 98407	74	33241731943=7 · 73 · 79 · 823447
75	35830141663=79 · 223 · 2033839	76	35927776399=31 · 28351 · 40879
77	37284039577=23 · 89 · 2089 · 8719	78	41344905169=151 · 2089 · 131071
79	41915535679=79 · 151 · 1801 · 1951	80	42213350953=337 · 4663 · 26863
81	43307250529=103 · 2143 · 196201	82	43624199479=7 · 47 · 1151 · 115201
83	44277688129=73 · 7591 · 79903	84	46220564737=47 · 2143 · 458897
85	46649950879=7 · 73 · 4951 · 18439	86	46726848679=7 · 151 · 44207047
87	47708928337=7 · 103 · 479 · 138143	88	47788795729=89 · 191 · 881 · 3191
89	47900002297=7 · 79 · 89 · 233 · 4177	90	51295825039=71 · 79 · 191 · 47881
91	53085400879=7 · 1951 · 3887047	92	55876862209=7 · 30391 · 262657
93	61655287009=79 · 151 · 631 · 8191	94	61920838399=7 · 73 · 79 · 1533871
95	68280792529=151 · 2239 · 201961	96	75237309679=7 · 151 · 271 · 262657
97	75958473199=7 · 23 · 31 · 89 · 271 · 631	98	76126153537=7 · 233 · 2089 · 22343
99	77652121369=631 · 1327 · 92737	100	79491041959=7 · 73 · 79 · 1969111
101	81235557319=7 · 23 · 31 · 89 · 199 · 919	102	82396696729=31 · 47 · 73 · 601 · 1289
103	83083752679=23 · 89 · 127 · 319591	104	86123562529=3511 · 4201 · 5839
105	89682628039=151 · 233 · 1103 · 2311	106	94162480129=31 · 359 · 1151 · 7351

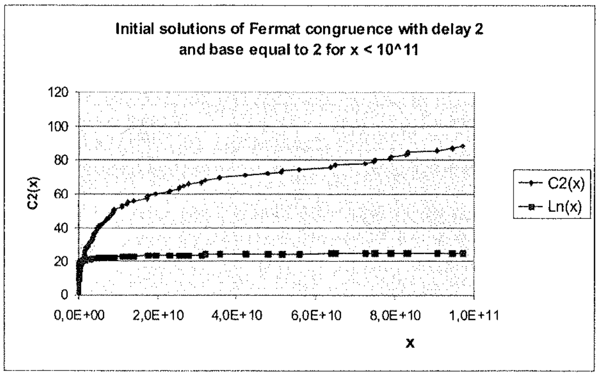


Fig. 1. The number $C_2(x)$ of solutions of the congruence $2^{n-2} \equiv 1 \ (mod\ n)$, $n < x$

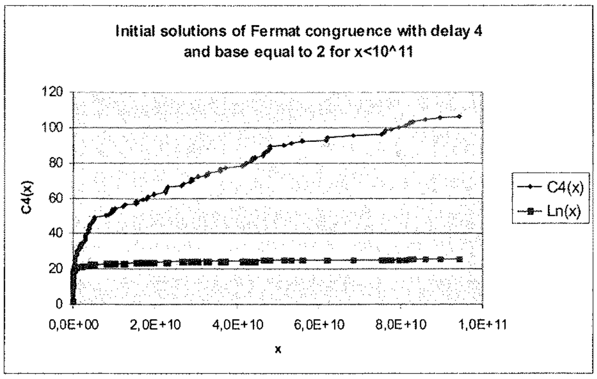


Fig. 2. The number $C_4(x)$ of solutions of the congruence $2^{n-4} \equiv 1$ $(mod\ n)$, $n < x$

The results of our computations are expressed in Table 3, Table 4 and illustrated on Fig. 1 and Fig. 2. There are some observations when analyzing these data.

1. Let S_k be the set of solutions of the congruence $2^{n-k} \equiv 1$ (mod n). The intersection of two sets related to different numbers k and j ($k > j$) is not empty with possible exception if n divides $2^{k-j} - 1$. Our sets S_2 and S_4 are disjoint;
2. All numbers in the sets S_2 and S_4 are composite squarefree integers (see Table 1 and Table 2);
3. The growth rate of the function $C_2(x)$ and $C_4(x)$ is much faster than logarithmic (see Fig. 1 and Fig. 2);
4. There are only four solutions of the congruence $2^{n-k} \equiv 1$ (mod n) for $k = 2$ which consist of two factors: $20737 = 89 \cdot 233$, $228727 = 127 \cdot 1801$, $513013327 = 31 \cdot 32575081$, $3543720833 = 73 \cdot 48544121$.
   For $k = 4$ we have five solutions of that congruence which consist of two factors: $673663 = 337 \cdot 1999$, $1697609 = 127 \cdot 13367$, $57233047 = 337 \cdot 169831$, $259195009 = 11119 \cdot 23311$, $12160941223 = 337 \cdot 36085979$;
5. Most of elements of both sets S_2 and S_4 are numbers having their last decimal digits equal to 1.

3. ACKNOWLEDGEMENTS

We are very indebted to Mrs. Danuta Ojrzeńska Wójter for kindly making access to the students laboratory in the idle time. Without this help our computation would take much more time and effort.

4. REFERENCES

1. P. Erdös, R. L. Graham: *Old and new problems in Combinatorial Number Theory, Old and new problems and results in combinatorial number theory.* Université de Geneve, L'Enseignement Mathématique, 1980;

2. P. K i s s, B u i M i n h P h o n g: On Problem of A. Rotkiewicz, Math. Comp. 48(1987), pp. 751-755;
3. A. M ą k o w s k i: *Generalization of Morrow's D numbers*, Simon Stevin, 36(1962), 71;
4. W. L. M c D a n i e l: *The generalized pseudoprime congruence* $a^{n-k} \equiv b^{n-k}$ *(mod n)*, C.R.H. Math. Rep. Acad. Sci. Canada, Vol. 9(2), 1987, pp. 143-147;
5. W. L. M c D a n i e l: *Some pseudoprimes and related numbers having special forms*. Math. Comp. 53(1989), pp. 407-408;
6. P. M o n t g o m e r y: http://www.spacefire.com/numbertheory/2nmodn.htm, at the web side of J. Crump;
7. D. C. M o r r o w: *Some properties of D numbers*, Amer. Math. Monthly 58(1951), 324-330;
8. A. P a s z k i e w i c z, A. R o t k i e w i c z: *On pseudoprimes of the form* $a^n - a$, Proceedings of the Eleventh Conference on Fibonacci Numbers, Braunschweig, 2004 (still being in press);
9. A. R o t k i e w i c z: *Pseudoprime numbers and their generalizations*. Student Association of the Faculty of Sciences, University of Novi Sad, Novi Sad 1972, pp.i.+169; M.R. 48#8373
10. A. R o t k i e w i c z: *On the congruence* $2^{n-2} \equiv 1$ *(mod n)*, Math. Comp. 43(1984), pp. 271-272; MR 85e : 1105;
11. K. Z s i g m o n d y: *Zur Theorie der Potenzreste*. Monatshefte Math. Phys. 3(1892), pp. 264-284

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 57–69

On Least Prime Primitive Roots mod $2p$ for Odd Primes p*

ANDRZEJ PASZKIEWICZ

Institute of Telecommunications, Warsaw University of Technology
anpa@tele.pw.edu.pl

Received 2008.11.27
Authorized 2009.01.09

In this paper we derive a conditional formula which allows to compute the natural density of prime numbers with a given least prime primitive root modulo $2p$ and compare theoretical results with the numerical evidence. We also illustrate graphically these densities as functions of the upper limit x for primes below x.

Keywords: Primes, primitive roots, prime primitive roots, extended Riemann hypothesis

1. INTRODUCTION

Let $g(m)$ and $G(m)$ denote the least primitive and the least prime primitive root modulo m, respectively. It is well known since C. F. Gauss that if m has a primitive root then it should be equal to 2, 4 or be of the form p^k or $2p^k$, where p is an odd prime. In [1] we proved the following two Lemmas in which the letters p, q, r are reserved for primes and $\log_2 x = \log\log x$.

Lemma 1. Let $M = \{r_1, ..., r_m\}$, *be a set of primes,*

$N_M(x) = \{p \leq x :$ *every* r *in* M *is a primitive root mod* $p\}$.

On the assumption of the Riemann hypothesis for each extension

$$Q(\sqrt[k]{1}, \sqrt[l_1]{r_1}, ..., \sqrt[l_m]{r_m})$$

where $k = l.c.m.l_i$ *is squarefree we have*

$$|N_M(x)| = A_M \cdot \mathrm{Li}x + O_M(\mathrm{Li}x(\log x)^{-1}(\log_2 x)^{2^{|M|}-1})$$

* The computational part of his paper was supported by the Ministry of Science and Higher Education of Poland - research grant no. N517 003 32/0583 for 2007-2010

where A_M is defined as follows.

Let $c(p)$ be the natural density of the set

$$\{q : q \equiv 1(\text{mod } p), \textit{at least one of } r_1, ..., r_m \textit{ is a } p\text{–th } \textit{power residue mod } r\}$$

and let $\bar{c}(p) = 1 - c(p)$. Also let $G(r_1, ..., r_m)$ denote the set of numbers of the form

$$a = r_1^{\varepsilon_1} ... r_m^{\varepsilon_m} \equiv 1(\text{mod} 4), \ \varepsilon_i = 0 \text{ or } 1, \textit{ and finally let}$$

$$f(a) = \prod_{p|a} \frac{c(p)}{1 - c(p)}.$$

Then

$$A_M = \prod_{i=1}^{\infty} \bar{c}(p_i) \sum_{a \in G(r_1, ..., r_m)} f(a). \tag{1}$$

Lemma 2. *In the notation of Lemma 1*

$$c(p) = \frac{1}{p-1}\left(1 - \left(1 - \frac{1}{p}\right)^m\right).$$

Let $G(p, p_k)$ be the least prime primitive root of p greater than p_k. It is easy to see that $G(p, 2) = G(2p)$. The main our result is the following theorem.

Theorem. *Assume that Riemann hypothesis holds for each of the fields $Q(\sqrt[k]{1}, \sqrt[l_1]{r_1}, ..., \sqrt[l_m]{r_m})$, where $k = l.c.m.l_i$ is squarefree. Then the set of primes p such that $G(p, p_k) = p_n$ has a natural density equal to*

$$E(p_k, p_n) = \sum_{m=1}^{n-k} (-1)^{m-1} \Delta_m \cdot c_{k,m,n} \tag{2}$$

where

$$\Delta_m = \prod_{i=1}^{\infty} \left(1 - \frac{1}{p_i - 1}\left(1 - \left(1 - \frac{1}{p_i}\right)^m\right)\right),$$

$$c_{k,m,n} = \frac{1}{2} \sum_{\substack{|M|=m \\ M \subset \{p_{k+1}, ..., p_n\} \\ M \ni p_n}} \left\{\prod_{p \in M} (1 + d_{m,p}) + \prod_{p \in M} (1 + (-1|p) \cdot d_{m,p})\right\}, \tag{3}$$

and

$$d_{m,p} = \frac{1}{p-1}\left(1 - \left(1 - \frac{1}{p}\right)^m\right) \cdot \left(1 - \frac{1}{p-1}\left(1 - \left(1 - \frac{1}{p}\right)^m\right)\right)^{-1}.$$

Proof. By the sieve principle the number $N(p_k, x)$ of primes $\leq x$ with $G(p, p_k) = p_n$ equals

$$\sum_{\substack{M\subset\{p_{k+1},\dots,p_n\}\\ M\ni p_n}} (-1)^{|M|-1} N_M(x),$$

hence, by Lemma 1,

$$N(p_k, x) = \sum_{M\subset\{p_{k+1},\dots,p_n\}} (-1)^{|M|-1} A_M \cdot \mathrm{Li}x + O_{k,n}\left(\mathrm{Li}x(\log x)^{-1}(\log_2 x)^{2^n-1}\right).$$

$$E(p_k, p_n) = \sum_{\substack{M\subset\{p_{k+1},\dots,p_n\}\\ M\ni p_n}} (-1)^{|M|-1} A_M. \tag{4}$$

Now if $M = \{r_1, ..., r_n\}$ we have by Lemma 2

$$\overline{c}(p_i) = 1 - \frac{1}{p_i - 1}\left(1 - \left(1 - \frac{1}{p_i}\right)^m\right)$$

$$\frac{c(p)}{1 - c(p)} = d_{m,p},$$

hence

$$\prod_{i=1}^{\infty} c(p_i) = \Delta_m. \tag{5}$$

On the other hand, if $M_\varepsilon = \{r \in M : r \equiv \varepsilon \bmod 4\}$, the condition $\prod_{\mu=1}^{m} r_\mu^{\varepsilon_\mu} \equiv 1 \pmod 4$ is equivalent to $\sum_{r_\mu\in M_{-1}} \varepsilon_\mu \equiv 0 \pmod 2$ (note that $r \in M$ implies $r > 2$).

We have

$$\begin{aligned}\sum_{a\in G(r_1,\dots,r_m)} f(a) &= \sum_{k=0}^{|M_1|} \sum_{\substack{N\subset M_1\\ |N|=k}} \prod_{r\in N} d_{m,r} \cdot \sum_{k=0}^{\left[\frac{|M_{-1}|}{2}\right]} \sum_{\substack{N\subset M_{-1}\\ |N|=2k}} \prod_{r\in N} d_{m,r} = \\ &= \prod_{r\in M_1} (1 + d_{m,r}) \frac{\prod_{r\in M_{-1}} (1 + d_{m,r}) + \prod_{r\in M_{-1}} (1 - d_{m,r})}{2} = \\ &= \frac{1}{2}\left\{\prod_{p\in M} (1 + d_{m,p}) + \prod_{p\in M} (1 + (-1|p) \cdot d_{m,p})\right\}\end{aligned} \tag{6}$$

and (2) follows from (3), (4) and (6), which finishes the proof.

2. COMPUTATIONS AND RESULTS

Let $E_2^*(p_n, x)$, $E^*(x)$ be two functions defined as follows

$$E_2^*(p_n, x) = \frac{1}{\pi(x)} \sum_{p<x:G(2p)=p_n} 1$$

$$E_2^*(x) = \frac{1}{\pi(x)} \sum_{p<x} G(2p).$$

The first function is a natural density of primes below x, with the least prime primitive root modulo $2p$ equal just to p_n and the second, the average value of the least prime primitive root modulo $2p$ extended over primes below the limit x. We tabulated the functions with the step 10^8 up to the limit $x < 10^{10}$, and $n \leq 25$. Results are illustrated on Fig. 1-17. We also calculated the initial values of the function $E(2, p_n)$ defined by (2), (see Table 2). There is a very good agreement with results of calculation and numerical evidence (column 3 and column 4 of Table 2). The average value of the least prime primitive root modulo $2p$ illustrates Figure 17. One can suppose, that there exists a finite limit of the function $E_2^*(x)$ as x tends to infinity.

All computations were performed on one powerful PC computer with Pentium processor running on Linux operation system. The experiment has finished after approximately one day of computation.

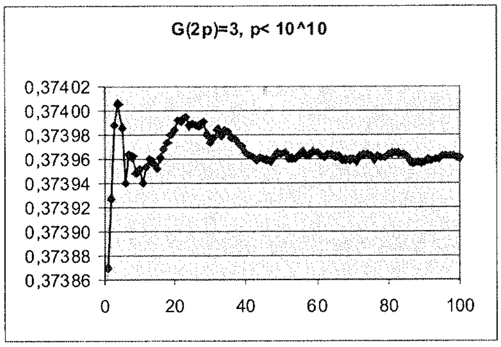


Fig. 1.

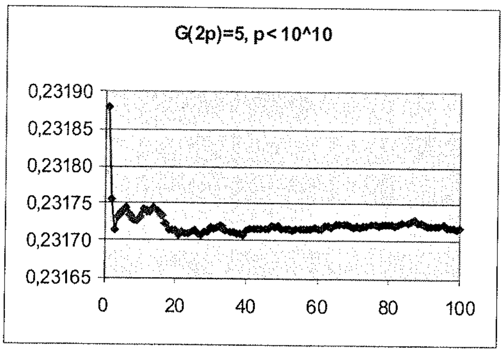


Fig. 2.

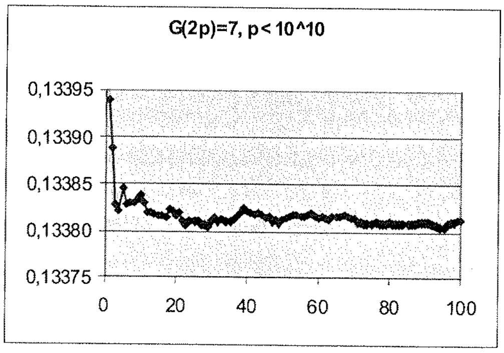


Fig. 3.

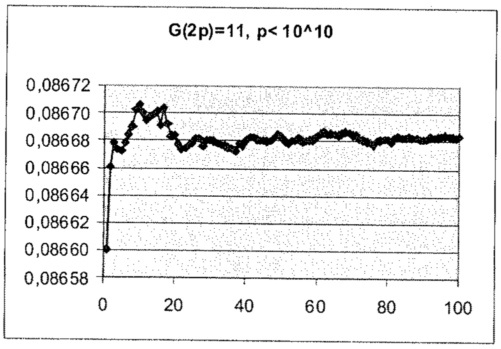


Fig. 4.

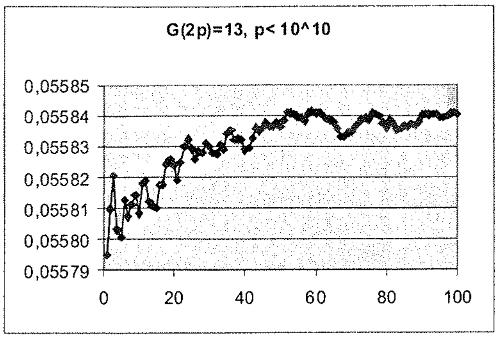


Fig. 5.

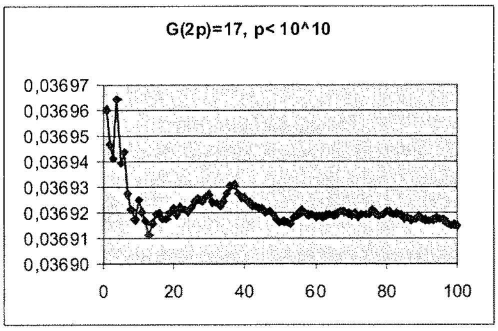


Fig. 6.

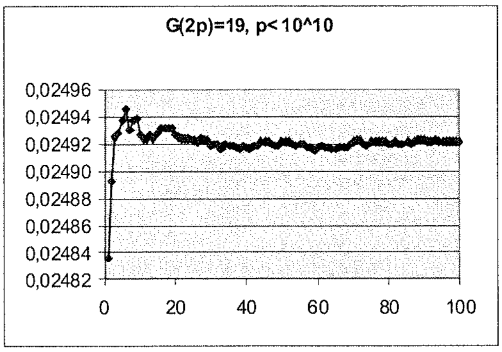


Fig. 7.

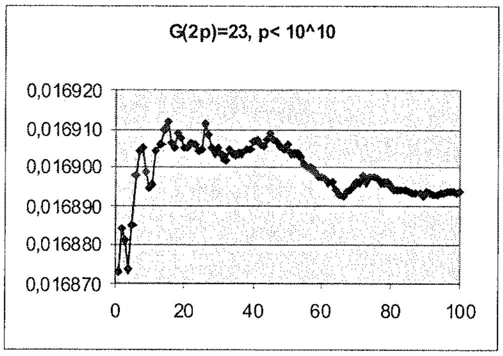


Fig. 8.

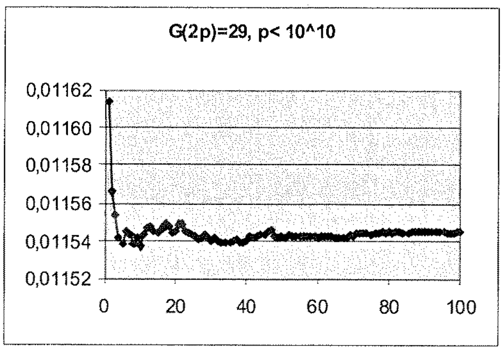


Fig. 9.

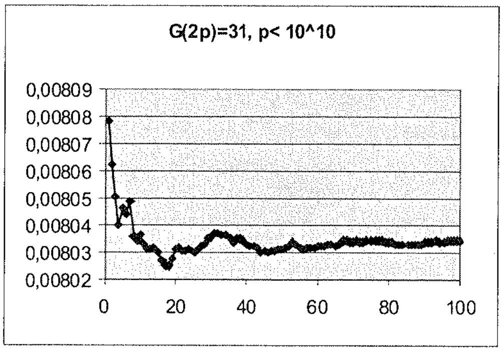


Fig. 10.

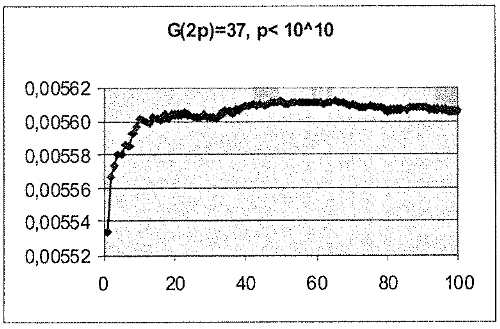


Fig. 11.

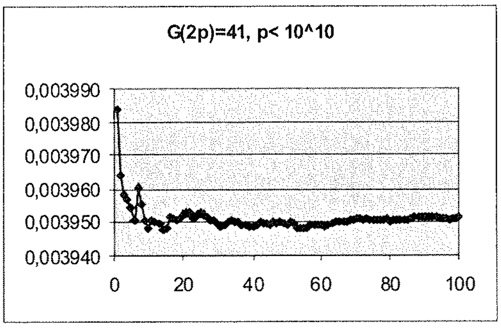


Fig. 12.

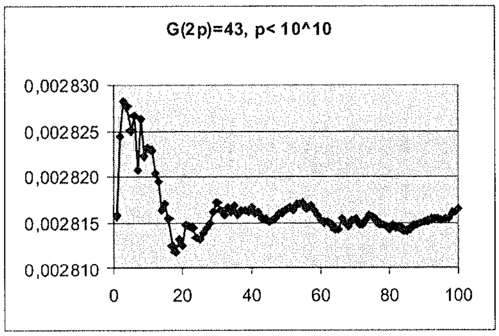


Fig. 13.

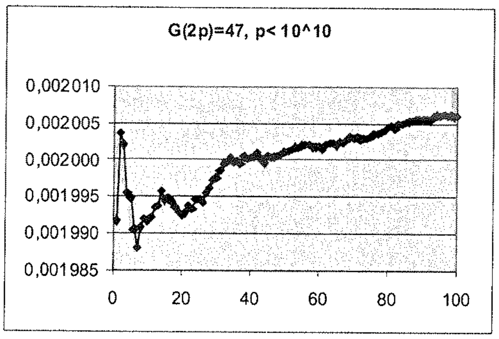


Fig. 14.

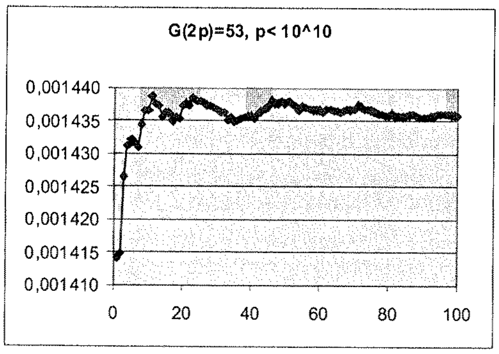


Fig. 15.

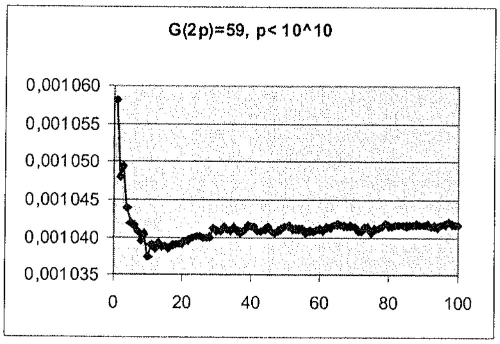


Fig. 16.

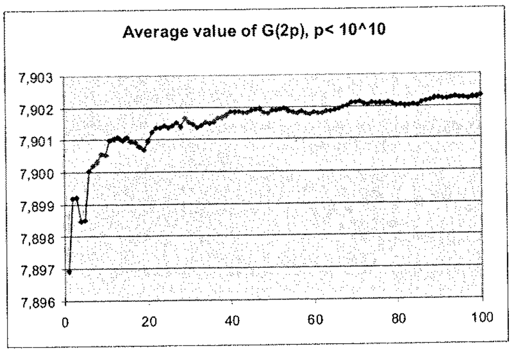


Fig. 17. Average value of the least prime primitive root mod $2p$

Table 1

Least primes p with a given least prime primitive roots modulo $2p$

$G(2p)$	$g(2p)$	$G(p)$	$g(p)$	p	$p-1$
5	5	5	5	23	$2 \cdot 11$
7	7	2	2	11	$2 \cdot 5$
11	11	2	2	59	$2 \cdot 29$
13	13	13	13	457	$2^3 \cdot 3 \cdot 19$
17	17	2	2	131	$2 \cdot 5 \cdot 13$
19	19	19	19	191	$2 \cdot 5 \cdot 19$
23	21	2	2	181	$2^2 \cdot 3^2 \cdot 5$
29	21	29	21	409	$2^3 \cdot 3 \cdot 17$
31	31	31	10	1021	$2^2 \cdot 3 \cdot 5 \cdot 17$
37	21	37	14	1031	$2 \cdot 5 \cdot 103$
41	41	41	6	1811	$2 \cdot 5 \cdot 181$
43	15	43	6	271	$2 \cdot 3^3 \cdot 5$
47	33	2	2	1531	$2 \cdot 3^2 \cdot 5 \cdot 17$
53	15	53	6	2791	$2 \cdot 3^2 \cdot 5 \cdot 31$
59	15	2	2	31531	$2 \cdot 3 \cdot 5 \cdot 1051$
61	15	2	2	28477	$2^2 \cdot 3^2 \cdot 7 \cdot 113$
67	57	2	2	33301	$2^2 \cdot 3^2 \cdot 5^2 \cdot 37$
71	35	2	2	149341	$2^2 \cdot 3 \cdot 5 \cdot 19 \cdot 131$

cd. Table 1

$G(2p)$	$g(2p)$	$G(p)$	$g(p)$	p	$p-1$
73	15	73	6	23911	$2 \cdot 3 \cdot 5 \cdot 797$
79	15	79	10	11971	$2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 19$
83	69	83	69	110881	$2^5 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11$
89	21	89	6	103091	$2 \cdot 5 \cdot 13^2 \cdot 61$
97	87	97	44	71761	$2^4 \cdot 3 \cdot 5 \cdot 13 \cdot 23$
101	33	2	2	266701	$2^2 \cdot 3 \cdot 5^2 \cdot 7 \cdot 127$
103	35	103	35	290041	$2^3 \cdot 3 \cdot 5 \cdot 2417$
107	21	107	10	31771	$2 \cdot 3^2 \cdot 5 \cdot 353$
109	39	109	14	448141	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 97$
109	39	109	14	448141	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 11 \cdot 97$
113	33	113	33	2447761	$2^4 \cdot 3 \cdot 5 \cdot 7 \cdot 31 \cdot 47$
127	35	127	6	674701	$2^2 \cdot 3 \cdot 5^2 \cdot 13 \cdot 173$
131	115	131	10	3248701	$2^2 \cdot 3 \cdot 5^2 \cdot 7^2 \cdot 13 \cdot 17$
137	55	2	2	2708581	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 6449$
139	21	139	18	690541	$2^2 \cdot 3 \cdot 5 \cdot 17 \cdot 677$
149	21	149	14	190321	$2^4 \cdot 3 \cdot 5 \cdot 13 \cdot 61$
151	35	151	6	2080597	$2^2 \cdot 3 \cdot 7 \cdot 17 \cdot 31 \cdot 47$
157	33	157	33	4076641	$2^5 \cdot 3^2 \cdot 5 \cdot 19 \cdot 149$
163	21	163	14	3545281	$2^6 \cdot 3^2 \cdot 5 \cdot 1231$
167	33	167	33	11643607	$2 \cdot 3^2 \cdot 13 \cdot 17 \cdot 2927$
173	133	173	18	16135981	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 103 \cdot 373$
179	141	179	94	5109721	$2^3 \cdot 3 \cdot 5 \cdot 7^2 \cdot 11 \cdot 79$
181	15	181	15	9633751	$2 \cdot 3 \cdot 5^4 \cdot 7 \cdot 367$
191	21	2	2	2697301	$2^2 \cdot 3^6 \cdot 5^2 \cdot 37$
193	15	193	15	25738831	$2 \cdot 3^3 \cdot 5 \cdot 13 \cdot 7333$
197	21	2	2	63577141	$2^2 \cdot 3 \cdot 5 \cdot 11 \cdot 96329$
199	15	199	6	37565431	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 41 \cdot 4363$
211	39	211	6	4022911	$2 \cdot 3^2 \cdot 5 \cdot 44699$
223	21	2	2	24694141	$2^2 \cdot 3 \cdot 5 \cdot 411569$
227	15	227	6	298155271	$2 \cdot 3 \cdot 5 \cdot 7 \cdot 71 \cdot 19997$

cd. Table 1

$G(2p)$	$g(2p)$	$G(p)$	$g(p)$	p	$p-1$
229	91	2	2	194948461	$2^2 \cdot 3^2 \cdot 5 \cdot 7^2 \cdot 23 \cdot 31^2$
233	69	233	6	453507991	$2 \cdot 3 \cdot 5 \cdot 13 \cdot 31 \cdot 37511$
239	15	239	12	187155691	$2 \cdot 3 \cdot 5 \cdot 1223 \cdot 5101$
241	35	241	14	449032321	$2^7 \cdot 3^2 \cdot 5 \cdot 11 \cdot 19 \cdot 373$
251	35	251	22	672618871	$2 \cdot 3^3 \cdot 5 \cdot 7 \cdot 11 \cdot 32353$
257	117	257	10	794932741	$2^2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 630899$
263	65	263	14	137568061	$2^2 \cdot 3^2 \cdot 5 \cdot 7 \cdot 23 \cdot 47 \cdot 101$
269	95	2	2	3495804181	$2^2 \cdot 3 \cdot 5 \cdot 11 \cdot 17 \cdot 311569$
271	69	2	2	8742210541	$2^2 \cdot 3 \cdot 5 \cdot 7^2 \cdot 59 \cdot 101 \cdot 499$
277	57	277	57	443571241	$2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 29 \cdot 131 \cdot 139$
281	57	281	34	8493717961	$2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13 \cdot 259271$
283	69	283	22	1095701881	$2^3 \cdot 3 \cdot 5 \cdot 7 \cdot 13 \cdot 19 \cdot 5281$
307	39	307	12	565822531	$2 \cdot 3^3 \cdot 5 \cdot 7 \cdot 13 \cdot 23029$
347	15	347	15	1160260711	$2 \cdot 3 \cdot 5 \cdot 72 \cdot 17 \cdot 29 \cdot 1601$
349	39	349	6	1622723341	$2^2 \cdot 3 \cdot 5 \cdot 7 \cdot 1511 \cdot 2557$

Table 2

Results of theoretical and numerical computations

n	p_n	$E(2, p_n)$	$E_2^*(p_n, x), x < 10^{10}$
2	3	0,3739	0,37396
3	5	0,231	0,231718
4	7	0,133	0,133812
5	11	0,086	0,086684
6	13	0,055	0,055841
7	17	0,036	0,036915
8	19	0,024	0,024921
9	23	0,016	0,016894
10	29	0,011	0,011545
11	31	0,008	0,008034
12	37	0,005	0,005606
13	41	0,003	0,003952
14	43	0,0028	0,002817
15	47	0,0020	0,002006
16	53	0,0014	0,001436

3. ACKNOWLEDGEMENTS

The author is very grateful to Professor A. Schinzel from the Institute of Mathematics of Polish Academy of Sciences for sketching the proof of Theorem.

4. REFERENCES

1. A. Paszkiewicz, A. Schinzel: *On the least prime primitive root modulo a prime*, Math. Comp., vol. 71, No. 239, pp. 1307-1321.

S

sup
rang
twe
whe
syst
and
four
moc
add

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 71–93

SystemC-based Codesign of Distributed Embedded Systems

STANISŁAW DENIZIAK, RADOSŁAW CZARNECKI

Cracow University of Technology, Warszawska 24, 31-155 Cracow, POLAND
E-mail: pedenizi@cyf-kr.edu.pl, czarnecki@pk.edu.pl

Received 2008.12.02
Authorized 2009.01.20

Most of existing co-synthesis methods for embedded systems requires a task graph model of a system. This work presents a codesign methodology for embedded systems specified using SystemC language. For each system specification, developed according to this methodology, it is possible to automatically generate the task graph or the conditional task graph corresponding to this specification. To simplify the codesign process and to reduce the time required to develop the specification, a framework in the form of a library built on top of the SystemC language core was created. This library contains definitions of communication channels, interfaces, ports and macros implementing a model of computation corresponding to the task graph semantics. Benefits of the presented methodology were demonstrated by comparing synthesis results of the same system, represented by different SystemC models and using our co-synthesis methods for SOC, SOPC and dynamically reconfigurable SOPC systems.

Keywords: SystemC, codesign, system synthesis

1. INTRODUCTION

SystemC [1] is the most widely accepted system-level specification language. It supports a powerful generic model of computation, which enables defining a wide range of different customizable methods of communication and synchronization between processes. SystemC has proven to be suitable for design of embedded systems, where the ability to develop heterogeneous specifications is very important. Electronic system level design environments use specific models of computation, abstraction levels and design methodologies. To make SystemC more general, only a basic modelling foundation has been added to the language. It is assumed that design libraries and modelling guidelines needed to support these specific design methodologies will be added on the top of the language core.

To perform system-level optimizations, like static task scheduling, task allocation, hardware/software partitioning, co-synthesis methods require models providing information about static data/control flow. The most widely used models of dataflow-oriented embedded systems are: a task graph [2-7] and a conditional task graph [8-12]. The task graph represents information about static data/control flow in a graphical form, it is not an executable specification required for modelling. Though it is possible to extract task graph from C program [13], it works only on sequential programs and is not applicable to system-level specifications given in SystemC. Default SystemC model of computation does not enable to determine data/control flow using static code analysis. Therefore, it is not possible to create an efficient task graph model. To avoid this problem an embedded system should be specified using the model defined by the task graph semantics.

In this work we present codesign method for embedded systems specified using SystemC language. Our approach is based on the methodology of creating SystemC specifications semantically equivalent to the task graph or the conditional task graph. The methodology consists of a library of C++ classes built on top of the SystemC language and a set of rules restricting use of SystemC constructs to the model of computations consistent with the task graph semantics. For each specification developed according to the methodology the corresponding task graph can be generated automatically, reducing the amount of development time and a potential for mistakes to occur. For the task graph generated from the specification given in SystemC, a co-synthesis algorithm can be applied to generate target architecture of the system. In this paper we present 3 co-synthesis methods for distributed embedded systems, each of them optimizes the system assuming SOC (System on Chip), SOPC (System on Programmable Chip) or DRSOPC (Dynamically Reconfigurable System on Programmable Chip) implementation. The target architecture of the system consists of general purpose processor cores and dedicated hardware modules. Algorithms start with an initial solution (architecture with highest speed or architecture occupying smallest chip area). Next, they produce new solutions using iterative improvement methods. In our approach the same executable SystemC specification is used for system modelling and for codesign. In this way we may avoid divergence between co-verification and co-synthesis. This is the main advantage of our methodology.

The rest of the paper is organized as follows. The next section reviews related work concerning methods for the co-synthesis of distributed embedded systems. Existing environments defining different models of computation in SystemC language will also be presented. Next, the main system-level features of SystemC are outlined. The overall specification methodology is presented in section 4. Section 5 contains a short description of our co-synthesis methods. Next, experimental results are given. The paper ends with conclusions.

2. RELATED WORK

The co-synthesis of distributed embedded systems consists of the following tasks:

- allocation; determines the quality and quantity of resources (processing elements – PEs and communication links – CLs), to be used,
- assignment; determines tasks to be executed on each PE, and a CL for each transmission,
- task scheduling; determines time at which each task and each transmission will start.

Allocation, assignment and scheduling are each NP-complete, thus the co-synthesis is computationally a very hard problem.

Due to the complexity of co-synthesis, the algorithms giving best solutions (e.g. mixed integer linear programming [14] or exhaustive exploration) are limited to small systems, only. Other approaches are based on constructive or iterative refinement heuristics. Some probabilistic optimization methods e.g. simulating annealing [12] or genetic algorithms [15] have been applied to the co-synthesis problem as well.

Constructive algorithms [16] build a system allocating incrementally new components. Since such approach is capable of inspecting only local effects of changes, different performance estimation methods were used to predict the global impact of these changes. The methods, usually based on the best- and worst-case analysis, prefer PEs with the highest speed or with the lowest cost and disregard remaining PEs. Although constructive algorithms are fast and are capable of producing high quality results, they are prone to becoming trapped in local minima. Iterative improvement algorithms [17] start with a sub-optimal solution and try to improve the system quality by making local changes to the system. Existing iterative algorithms also tend to be trapped in local minima. The main reason is that iterative improvement methods consider only local changes driven by immediate gain. Probabilistic optimization algorithms are capable of escaping local minima. However, performance of these methods strongly depends on selected values of parameters controlling the algorithm. For example, in the MOGAC genetic algorithm [15] each task graph has a different random seed for which the algorithm finds the best solution most rapidly.

Since, certain system features that are to be optimized, like cost, speed or power consumption, strongly depend on the implementation method, different co-synthesis methods were proposed for SOCs [18], SOPC, dynamically reconfigurable SOPCs [19-21], systems with dynamic voltage scalable processors [22], etc. But almost all of them use the task graph representation of the system.

The semantics of the task graph defines a certain model of computation. Therefore, SystemC specification will be synthesizable using the above methods, if the specification is created according to this model, or if it is possible to automatically convert this specification to conform to this model. Several design environments were proposed for SystemC. The OSCI TLM [23] defines a library of channels and interfaces which meet the Transaction Level Modelling standard requirements [24]. HetSC library [25] [26] supports some of the well-known models of computation: SR, KPN, PN, CSP, and SDF. SysteMoC library [27] [28] also permits the development of specifications under the well-known models of computation, for the purpose of the model based system design. None of the above models of computation is consistent with the task graph. Moreover, all the above approaches are frameworks for modelling, verification, design space exploration and embedded software generation. There is no any codesign method which may be applied to the proposed models.

3. SYSTEMC SEMANTICS

SystemC is a library of C++ classes and macros for system and hardware design. System-level specification is a set of modules, communicating using channels. Channels are connected to module ports. Each module contains at least one process. Processes are activated according to a sensitivity list, defined statically or dynamically. SystemC provides the following system-level features: events, channels and interfaces. These constructions define the model of computation used in the specification

3.1. EVENTS

The event is a low-level primitive which is used to construct different forms of synchronization. Wait/notify model is used. Processes waiting for events are suspended. Process resumes its execution when any event (or set of events) from the specified sensitivity list is generated. The sensitivity list for each process is defined statically or dynamically. The static sensitivity list is defined in the module constructor, separately for each process and can not be changed during execution. The dynamic sensitivity list is specified inside the wait operation and temporarily overwrite static list.

Three kinds of process are possible: thread process, method process or clocked thread process. Each thread process has its own thread of execution and can be suspended and resumed. Method process is executed to its end each time after activation. Clocked thread process is a thread process which can be activated only by positive or negative clock edge event.

3.2. INTERFACES

The interface specifies a set of methods to be implemented within a channel. Only ports matching given interface type may be used with channels implementing this interface. The main role of interfaces is to separate transmissions from computations, i.e. interfaces provide for modules an access to communication methods implemented in channels.

3.3. CHANNELS

The channel implements one or more interfaces and may be primitive or hierarchical. Primitive channels do not contain processes and can not access other channels. Hierarchical channels are modules, they may contain processes and other modules, and may access other channels. In SystemC only two types of communication primitive channels are predefined: signal and fifo.

The signal is a simplest communication channel. It corresponds to a hardware line. Transmission is without blocking and buffering, any change of signal state generates an event. Another kind of signal is a buffer. The only difference is that event is generated after every write, even if it does not change a buffer state.

The fifo enables transmissions with buffering and with blocking or without blocking of communicating processes. Operations for reading and writing as well as operations for checking the fifo state are provided.

3.4. MODEL OF COMPUTATION

The semantics of SystemC corresponds to the Discrete Event (DE) model of computation. Each process may generate any number of events, for each event a time at which it will occur is specified. All pending events are ordered according to their timestamp. When the given event occurs all processes waiting for this event are activated.

Synchronization capabilities provided by events and the wait operation allow a broad range of different communication methods to be implemented without having to change the simulation engine. Designer may construct specific channels defining rules for communication between processes and for process activation. For example communication methods implemented in the fifo channel specify the synchronization mechanism used in dataflow process networks: the *read()* method suspends the calling process when the buffer is empty, while the *write()* method generates the event which resumes the suspended reading process.

4. SYSTEMC SPECIFICATION METHODOLOGY

The specification methodology defines a set of requirements which should be fulfilled by any specification developed for the purpose of the co-synthesis. These requirements restrict the communication and synchronization methods in such a way that specifications are consistent with a model of computation corresponding to the task graph semantics. To reduce the time required for the specification development, whenever it was possible requirements were expressed with C++ classes and macros. First, the methodology of the task graph specification will be presented. Next, it will be supplemented to support the conditional task graph.

4.1. TASK GRAPH

Task graph is a directed, acyclic graph $G =< V, E >$, where: V is a set of vertices corresponding to tasks, E is a set of edges corresponding to communication between tasks. Labels associated with edges represent a size of single transmission. Only one node vs has no predecessors, it will be called the initial node. System specification may consist of more than one task graph, each executed with different rate. A sample task graph is presented on Fig.1.

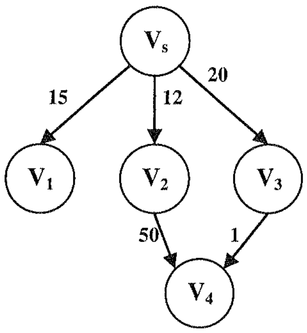


Fig. 1. A sample task graph

Let u_i be any node in the task graph and $v_0, \ldots, v_n$ be all its predecessors.
Definition 1 [Task graph model of computation]

Task graph model of computation (TGMoC) is defined as follows:
1. *Task v_s is self-activated with the rate f_R ($f_R \geq 0$).*
2. *Any active task is executed to its end without suspending.*
3. *Task ui is activated every time after finishing the execution of all tasks $v_0, \ldots, v_n$.*
4. *Transmissions from v_j (j=0,...,n) to u_i are buffered. The size of a buffer is defined by the label of the corresponding edge.*

Above definition specifies the only synchronization and communication methods which are permitted in the TGMoC. In the presented methodology they are supported by a library containing the following components:
- TG_NODE(*proc, inp_1,..., inp_n*) – the macro for process declaration,
- *tg_chan* – the communication channel,
- *tg_in_if*, *tg_out_if* – interfaces declaring operations implemented in *tg_chan*,
- *tg_in*, *tg_out* – ports matching *tg_in_if* and *tg_out_if* interfaces,
- *fire_event* – an event activating process.

TG_NODE defines a method process proc activated by a *fire_event* instantiated in this macro. This event is generated each time after finishing the execution of all processes connected to $inp_1, \ldots, inp_n$ ports. Channel *tg_chan* contains a fifo buffer with a size equal to the label of the corresponding edge. This channel does not generate an event after each write but only after end of transmission. This is the main difference between the *tg_chan* and the standard SystemC fifo channel.

Components described above implement all rules given in Def.1. Therefore, any SystemC specification is semantically consistent with TGMoC when it fulfils the following requirements:

R1. All processes are declared using TG_NODE macro.
R2. Communication is specified using only *tg_chan* channels.
R3. There are no static sensitivity declarations.

The first two conditions exclude synchronization other than using *tg_chan*. The last condition ensures that processes will be activated only according to rules implemented in the TG_NODE macro. Fig.2 presents a structure of the specification given in SystemC, corresponding to the task graph from Fig.1.

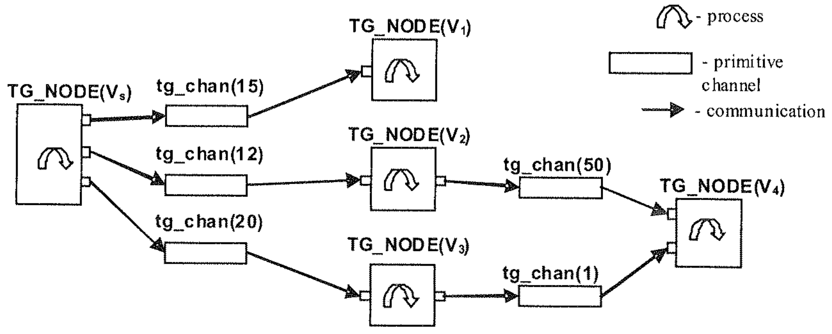


Fig. 2. Structure of a TGMoC specification

4.2. CONDITIONAL TASK GRAPH

The conditional task graph is also a directed, acyclic graph $G =< V, E >$, but additionally it contains conditional edges and two types of nodes: fork nodes and join nodes. The fork node is a node with conditional output edges. All mutually exclusive conditional edges for the same fork node begin conditional paths which meet in the join node. Fig. 3 shows a sample conditional task graph with one fork node v_s and one join node v_4.

Let w_i be any fork node, $e_1, \ldots, e_n$ be a set of all mutually exclusive output conditional edges and x_k be the join node ending conditional paths coming out from node w_i.

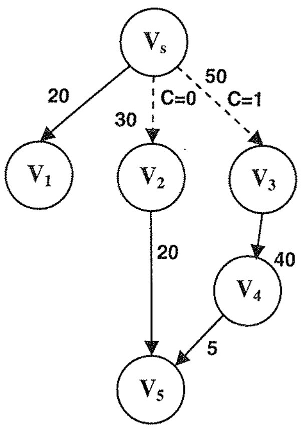


Fig. 3. A sample conditional task graph

Definition 2 [Conditional task graph model of computation]

Conditional task graph model of computation (CTGMoC) is defined as follows:

5. *After finishing the execution of task w_i only one task (ending edge e_j, for which the condition equals true) is activated.*
6. *Task x_k is activated every time after finishing execution of one ending task, for each set of incoming mutually exclusive conditional paths.*

For all other nodes and edges rules 1-4 from Def.1 are applied.

Def. 2 specifies semantics for the fork task and the join task. Thus, the library was supplemented with the following components:

- CTG_JOIN($chan_0, chan_1, \ldots, chan_n$): the macro joining channels $chan_1, \ldots, chan_n$ corresponding to edges ending mutually exclusive paths, into one channel $chan_0$,
- *ctg_chan*: the communication channel corresponding to a conditional edge,
- *ctg_out_if*: the interface defining operations implemented in *ctg_chan* class,
- *ctg_out*: the port matching the *ctg_out_if* interface.

Methods of activation of fork tasks are the same as in Def.1, thus these tasks may be declared using TG_NODE macro. Conditions associated with conditional edges are specified in declarations of the corresponding *cfg_chan*. Channel *cfg_chan* generates an event activating the next process only when the condition evaluates to true. The main problem in determining activation of the join task is to detect mutually exclusive paths. CTG_JOIN macro joins all specified channels into one channel, which should be specified as an input channel in the join task declaration. In this way all mutually exclusive input channels are visible as one channel. This solution is similar to that used in [9] and it can be also applied for graphs with nested conditions.

To assure consistency with CTGMoC, any SystemC specification has to meet requirements R1-R3 and the following ones:

R4. All channels corresponding to conditional edges are specified as *ctg_chan.*

R5. All mutually exclusive input channels for each join task are declared using CTG_JOIN macro.

Structure of the specification given in SystemC corresponding to the task graph from Fig.3 is presented on Fig.4.

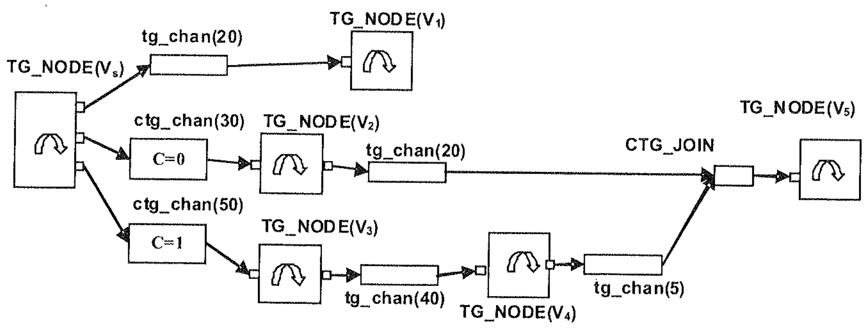


Fig. 4. Structure of a CTGMoC specification

5. THE CO-SYNTHESIS

HW/SW co-synthesis is the process of partitioning system specification into hardware and software processing elements connected by communication links. The goal of the co-synthesis is to find the best target architecture satisfying given constraints e.g. maximal cost or minimal speed.

In this section three co-synthesis methods will be presented. Each of them optimizes different system implementation: SOC, SOPC and Dynamically Reconfigurable SOPC. The algorithms are based on iterative improvement heuristics, taking into consideration sophisticated modifications and possibilities of further improvements. Starting from the initial solution, at each step some changes to the actual solution are considered and then the solution giving the best gain is selected. Architecture of the system is iteratively modified until it achieves the best architecture that satisfies the goal of the co-synthesis (the lowest cost or the highest performance). The main components of the iterative improvement algorithm are:

- the initial solution,
- the metric of the gain,
- system refinement methods.

The above components have to be defined in such a way that the algorithm will be capable of escaping local minima. The draft of the iterative improvement co-synthesis algorithm looks as follows:

```
Generate initial solution $A^{Cur}$;
Repeat {
$A^{best}=A^{Cur}$;
gain=0;
while ((A'=Modifications($A^{Cur}$)) $\neq$ 0) do{
        gain(A')=Quality(A')-Quality($A^{Cur}$);
        if(gain(A')>0) then $A^{Cur}$=A';
        }
}until (gain(A')>0);
```

Refinement process is controlled by the gain that describes a quality of the improvement. The gain is the difference between two compared solutions. Quality of solution is usually characterized by a few parameters (e.g. cost, performance). Since the number of possible changes in the system is very large, therefore only a few of these changes should be taken into consideration. Otherwise, the algorithm would be not suitable for large systems due to high computational complexity. In our co-synthesis algorithms two basic methods of refinement are considered: an allocation and a removing of single resource. It is possible to perform both kinds of changes in the same step, in this way movement of a task from one *PE* to other ones can be done. Such system modifications allow global changes of the system architecture. Moreover, simple modifications are still possible. For example, allocating one *PE*, assigning one task to it and then removing this *PE* and moving this task to some other *PE*, moves task from one *PE* to another.

5.1. CO-SYNTHESIS OF SOC SYSTEMS

System on Chip is one of possible implementations of distributed embedded systems. It is assumed that the target architecture of SOC includes processor cores (*GPPs*) and dedicated hardware components, also called hardware virtual components (*VCs*). A *GPP* executes all of assigned tasks sequentially. Each *VC* executes exactly one task. With each PE_i (*GPP* or *VC*) the following parameters are associated:
– $C_i(v_j)$ – implementation cost of each task v_j,
– $t_i(v_j)$ – execution time of each task v_j.

Values of $C_i(v_j)$ and $t_i(v_j)$ are known for IP modules. For other tasks they can be computed using performance and hardware effort estimation methods [29]. Moreover, with each GPP_i the unit cost CU_i, is associated. CU_i is independent of the number of tasks allocated to GPP_i.

Communication between processing elements is established using communication links (*CLs*). Sharing of communications links is allowed. Communication links are treated similarly as *GPP*. During synthesis link allocation and scheduling of transmissions are performed. Each type of communication link CL_i has the following parameters:

– cost of the link CC_i,
– bandwidth b_i (Bytes/s).

The time $t_k(v_i, v_j)$ required for data transfers between tasks v_i and v_j using communication link CL_k is evaluated using the following rule:

$$t_k(v_i, v_j) = \begin{cases} \left\lceil \dfrac{d_{ij}}{b_k} \right\rceil & \text{– if tasks are assigned to different PEs,} \\ 0 & \text{– otherwise.} \end{cases} \tag{1}$$

It is assumed that transmissions do not interfere with computations. Such model of communication is most commonly used, and may be implemented using dual-port buffers between PEs and buses or as communication using shared memory.

Assuming that a cost is defined by the total ASIC area, the total cost of a system may be specified using the following equation:

$$C = \sum_{i=1}^{r} CU_i + \sum_{j=1}^{p} C(v_j) + \sum_{i=1}^{c} CC_i \tag{2}$$

where r is the number of $GPPs$, p is the number of tasks, and c is the number of communications links.

The goal of the co-synthesis algorithm for SOC systems, used in our approach (EWA [30]), is to find the system architecture occupying the smallest chip area and satisfying given time constraints. The fastest architecture of the system is always selected as the initial solution. In this solution, the PE with the fastest execution time is allocated to each task. Since the goal of refinement is to reduce the cost of the system, so this cost should be the main factor influencing the gain. However, greedy algorithms, taking into consideration only cost, are quickly trapped into local minima. The main idea of the algorithm is to define the gain in such a way that it will take into consideration the global impact of the considered improvement. In the approach presented here the possibilities of modifying system architecture in the subsequent steps of the algorithm are defined using the following parameter:

$$\Omega = \sum_{i=1}^{n} (L_i - S_i) \tag{3}$$

where:
S_i – is the earliest time to start the execution of the i-th task,
L_i – is the latest time to start the execution of the i-th task, ensuring satisfaction of all time constraints.

S_i and L_i are evaluated using ASAP (As Soon As Possible) and ALAP (As Late As Possible) algorithms for the current architecture. If for any of the tasks we have $L_i < S_i$ then the current solution violates time constraints. This condition is verified for each solution. Bigger $L_i - S_i$ usually means more possibilities of allocating the i-th task. During system refinement task assignments and scheduling are changed, and so L_i

and S_i should be computed after each step. The global impact of any modification is defined as the increase of Ω caused by the modification:

$$\Delta\Omega = \Omega_{new} - \Omega_{old} \tag{4}$$

Finally, the gain ΔE taking into consideration cost reduction and the global impact of the system refinement is defined as follows:

$$\Delta E = \begin{cases} \dfrac{-\Delta K_s}{-\Delta\Omega}, for \Delta\Omega < 0 \\ -\Delta K_S, for \Delta\Omega = 0 \\ -\Delta K_S \cdot \Delta\Omega, for \Delta\Omega > 0 \end{cases} \tag{5}$$

where ΔK_S denotes the cost increase. Gain is defined only for modifications decreasing the cost of a system (otherwise the modification is not taken into consideration).

The following system changes are considered during refinement:

1. Allocation of one PE and assigning to it as many tasks as possible to achieve the highest gain.
2. Removing one PE with all tasks, which were assigned to it, being moved to other PEs. All task movements are done according to the highest gain principle.

More details about EWA method are presented in [30].

5.2. CO-SYNTHESIS OF SOPC SYSTEMS

Modern FPGAs enable the integration of a complex system on one device, also called System On Programmable Chip (SOPC). The goal of the co-synthesis is to find the architecture of SOPC system with the highest performance, which does not exceed the size of a target FPGA. Each task is characterized by the same time and cost (area) parameters as for SOC systems. The time of the transmission is also computed in the same way as for SOC systems (equations 1). The total area of the system implemented in FPGA is defined as in equation (2). Execution time of all tasks in the system is defined as follows:

$$T = max(max(t_k(GPP_1), \ldots, t_k(GPP_p)), max(t_k(VC_{p+1}), \ldots, t_k(VC_r))) \tag{6}$$

where: $t_k(PE_j)$ is the finish time of a task scheduled as the last one on PE_j (VC_j or GPP_j). Therefore, performance of the SOPC system is the following:

$$\lambda = 1/T \tag{7}$$

The initial solution in co-synthesis of SOPC system [31] is the architecture where all tasks are assigned to one GPP occupying the smallest area. Such solution leaves most space in FPGA available for new resources. Two methods of the modification are used: adding one *GPP* or *VC*, and removing one *GPP* or *VC*. Both modifications can be made in one step of the algorithm.

The list scheduling method, where priorities are assigned to each task, is applied in the algorithm. For each task v_i, times of execution of tasks, on all paths starting from v_i are computed. The longest time is taken as the priority of the task. Tasks with higher priorities are scheduled first. Task scheduling, allocation of *CLs* and communication scheduling are done simultaneously.

To increase the probability of getting out of local maxima of performance, the possibility of optimization in the next steps is also taken into consideration in the calculation of the gain. Let the parameter α define the available space in the FPGA:

$$\alpha = C_{max} - C_{cur} \tag{8}$$

where: C_{max} is the area of the target FPGA device and C_{cur} is the area of the current solution. The best solution is a solution with the highest λ. But from the other side, for solutions with the highest α, there is a greater probability of optimization in the next steps of the refinement. Thus, the gain ΔE, that describes the quality of the improvement, is defined as follows:

$$\Delta E = \begin{cases} \Delta\alpha * \Delta\lambda & when\ \Delta\alpha > 0 \\ \Delta\lambda & when\ \Delta\alpha = 0\ and\ \Delta\lambda > 0 \\ -\Delta\lambda/\Delta\alpha & when\ \Delta\alpha < 0 \\ 0 & when\ \Delta\lambda \le 0 \end{cases} \tag{9}$$

where: $\Delta\alpha = \alpha(\mathrm{A}^{\mathrm{cur}}) - \alpha(\mathrm{A}^{\mathrm{prev}})$ is the increase in the available area caused by the modification, $\Delta\lambda = \lambda(\mathrm{A}^{\mathrm{cur}}) - \lambda(\mathrm{A}^{\mathrm{prev}})$ is the increase in the speed, $\alpha(\mathrm{A}^{\mathrm{cur}})$ and $\lambda(\mathrm{A}^{\mathrm{cur}})$ are parameters of the current solution, $\alpha(\mathrm{A}^{\mathrm{prev}})$ and $\lambda(\mathrm{A}^{\mathrm{prev}})$ are parameters of the best solution, found in the previous step. More details about co-synthesis algorithm of SOPC systems are presented in [31].

5.3. CO-SYNTHESIS OF DYNAMICALLY RECONFIGURABLE SOPC SYSTEMS

Many FPGA devices support a partial and dynamic reconfigurability [32]. This feature enables a larger part of the application to be accelerated in hardware. In partially reconfigurable FPGAs only a part of the configuration can be modified. In this way, computations may overlap with reconfigurations, reducing the reconfiguration time overhead.

Many assumptions presented in the previous section are the same for dynamically reconfigurable SOPC systems (e.g. library of components, parameters of tasks). Thus in this section only differences between the co-synthesis of SOPC (described in p.5.2) and DRSOPC systems (COSEDYRES [33]) will be pointed out. It is assumed that the target architecture includes *GPPs*, *CLs* and dynamically reconfigurable sectors

RSs. Hardware components (*VCs*) are placed in sectors *RSs*. One or more *VCs* can be assigned to one sector for the same time frame, thus *RS* can execute exactly one selected subset of tasks. Tasks assigned to one *RS* may run in parallel. After all tasks assigned to the same sector have finished their execution the sector is reconfigured to allocate new *VC* modules.

Areas of all available *RSs* are calculated as the sum of areas of some *VCs* from the library, next they are justified according to the requirements of a module based reconfiguration. The area occupied by the sector RS_i is defined as C_{RSi}. Reconfiguration time for each *RS* is calculated on the basis of the reconfiguration time of one logical cell (t_r), which is taken from the datasheet of a target FPGA. It is assumed that the target architecture includes an additional embedded processor *GPPr*, which controls reconfiguration process. Let the architecture of a system be composed of p processor cores GPP_i ($i = 1,\dots, p$), one processor that controls reconfiguration GPP_r occupying the area Cu_r, r sectors RS_i ($i = p + 1,\dots, p + r$) and c communication links CL_j ($j = 1,\dots, c$). The total area of the system is defined as follows:

$$C = \sum_{i=1}^{p} Cu_i + \sum_{i=p+1}^{p+r} C_{RSi} + \sum_{j=1}^{c} Cc_j + Cu_r \tag{10}$$

During the initialization all available sizes of *RS* are being calculated. Next, the initial solution is created in the same way as for SOPC systems. The best sector size is the size that can contain as many as possible different groups of tasks. To achieve greater flexibility a few different sizes of *RS* should be available. First, sums of areas of all possible *VC* groups are being computed. Next, sums which are most frequent or sums which values are the most similar (values are in a given range) are being chosen as available sector sizes. If an embedded system is represented by the conditional task graph, then all mutually exclusive tasks (MET) should be determined during the initialization step. For this purpose, the algorithm of a CTG labelling is used.

Two methods of the modification are also used: adding or removing one *GPP* or *RS*, moreover modifications that remove or add so called "time context" of *RS*, are also considered. "Time context" is a time between consecutive reconfigurations of *RS* (only limited number of *VCs* can be executed in such context of *RS*,) and it is treated as resource. The gain ΔE that describes the quality of the improvement is defined according to equation (9).

In the COSEDYRES, the physical constraints on placement of reconfigurable modules are taken into account [21]. Reconfigurable sectors are always strict linearly placed, i.e. each *RS* occupies a contiguous set of CLB columns. Such restrictions eliminate some possible optimal solutions, but physically unrealizable because of placement infeasibility.

If DRSOPC system is specified using CTG graph, the algorithm prefers to assign MET tasks to the same RS. In this way the area of a system is decreased and then more tasks may be assigned to hardware, thus the system performance may be higher.

When MET tasks are assigned to the same *GPP*, they can be scheduled in the same time frame; therefore the scheduling algorithm takes them into account, too.

It is assumed that the basic reconfigurable module in FPGA is a frame, spanning the height of an FPGA (as in Xilinx FPGAs). Each *RS* consists of multiple adjacent frames. Hence, *RS* placement should be strict linear. Another constraint concerns the communication scheduling: the sector reconfiguration and any transmission through this sector are not allowed in the same time. Therefore, transmissions can not overlap in time with reconfiguration. To find the best sector placement, all possible layouts of *RSs* in an FPGA are evaluated. For each *RSs* placement, task scheduling and communication scheduling, satisfying reconfiguration requirements, are performed. The fastest implementation is selected as the A^{best}.

Architecture generated using COSEDYRES is supplemented with the *GPPr* processor that controls the reconfiguration of sectors. *GPPr* is a general purpose processor or a special IP module. Part of an FPGA is reserved for *GPPr*, before the algorithm starts. In this way the whole system is implemented in one FPGA and there are no other external modules to control reconfiguration process. Details of COSEDYRES algorithm are presented in [33].

6. EXPERIMENTAL RESULTS

In this section the benefits of using our methodology for developing the system specification using SystemC language, to achieve high quality results during the co-synthesis, will be presented. First it will be showed that if designer does not consider the methodology presented in this work, results of the synthesis may be far from optimal. Next, some experimental results of the co-synthesis of SOC, SOPC and DRSOPC systems that are specified as task graphs using SystemC will be presented. The benefits of the dynamic reconfigurability also will be demonstrated. For this purpose, the same example will be synthesized and implemented as DRSOPC and SOPC systems [31, 33]. Next, by comparing results of the co-synthesis of systems represented by *TG* and *CTG* graphs, benefits of considering mutually exclusive tasks during the system optimization will be presented.

If we do not consider methodology of creating SystemC specifications semantically equivalent to the task graph, the static dependencies between tasks will not be known before the co-synthesis (DE model of computations). Thus, the co-synthesis process should assume the worst case, where all tasks should be executed during the required time period. Let the library of available resources given in Tables 1 and 2 specifies task parameters for the system specified in Fig. 3. Table 3 presents experimental results. First, the system was synthesized as a SOC, using the method described in p.5.1. First part of the table contains results obtained for the system specified using DE and TG models of computation, the following rows correspond to different time constraints. Benefits of our methodology are especially visible for hard constraints, where TG model enables higher area minimization. Moreover, for the first case without our methodology it was not possible to find solution satisfying given constraint. The second part of Table 3 presents results obtained for SOPC systems. Here we may observe opposite dependency: results of optimization are lower for hard area constraints. When the available FPGA area is small, it is occupied mainly by GPP and only a few tasks may be allocated in hardware. Thus, most tasks are executed sequentially. For larger FPGAs it is possible to allocate more GPPs or VCs, therefore more tasks may be executed in parallel. Specifying MET tasks using CTG model of computation gives more possibility of optimization in all cases, especially for DRSOPC systems.

Table 1

Library of software and hardware components

tr=0.86 μs / CLB				
Task	GPP (Cu=180 CLBs)		VC	
	$t_i(v_j)$ [μs]	$C_i(v_j)$	$t_i(v_j)$ [μs]	$C_i(v_j)$ [CLB]
V_0	202	66	61	152
V_1	220	32	94	109
V_2	264	24	30	184
V_3	136	29	21	224
V_4	190	32	26	167
V_5	90	8	40	181

Table 2

Parameters of the communication link

Cl	CC [CLB]	b	availability
B1	10	9kB/μs	GPP, all VC (HW)

Table 3

Results of co-synthesis for SOC, SOPC and DRSOPC systems

T max	SOC-DE		SOC-TG		FPGA area	SOPC-DE		SOPC-TG		DRSOPC-TG		DRSOPC-CTG	
	time	area	time	area		time	area	time	area	time	area	time	area
250	–	–	155	1017	900	449	802	313	802	313	802	263	900
400	272	1017	313	829	700	577	693	454	650	374	690	345	676
600	578	743	545	637	550	714	541	618	483	477	523	374	523
800	742	608	741	534	400	871	374	838	374	567	358	507	358
					Smax = 1027, tmin = 155			Smin = 180, tmax = 1102					

Table 4 shows the comparison of co-synthesis result for random task graphs (from 10 to 70 nodes) for SOC and SOPC systems specified using DE and TG models. It is visible that the difference between optimized system parameters, for both models, increases according to the growth of the number of tasks. For larger systems there are more tasks belonging to parallel paths in task graphs, thus using TG model it is possible to achieve higher level of parallelism. In this way it is possible to find an architecture with lower cost (for SOC system) or with higher speed (for SOPC system).

Table 4

Comparison of results for co-synthesis for DE and TG models

		SOC-DE		SOC-TG			SOPC – DE		SOPC – TG	
N	T max	time	area	time	area	FPGA area	time	area	time	area
10	1500	1171	3813	1498	1979	1500	2903	1345	1772	1274
30	6000	5983	5606	5989	3559	2000	14993	1893	6292	1894
50	8000	8000	9424	7951	4331	2500	24865	2423	8046	2488
70	9000	8975	13024	8981	5819	3000	33024	2882	9543	2952

CTG model enables to specify mutually exclusive tasks. Information about these tasks may be used for area optimization, especially during co-synthesis of DRSOPC systems. The results of the synthesis of systems specified by random CTG graphs were compared with results obtained for the same systems specified by TG graphs (Table 5). The experiments showed that considering mutual exclusive tasks specified by CTG, enables to increase performance of a DRSOPC system using the COSEDYRES method (up to 30%). If MET tasks are assigned to the same reconfigurable sector, more tasks may be executed in hardware. METs are allocated only when the proper condition is satisfied (after reconfiguration of a sector). The performance increase depends on the number of MET tasks in CTG. If the number of METs in CTG is small then the performance increase may not be such significant.

Table 5

Comparison of co-synthesis results of DRSOPC specified by TG and CTG

		DRSOPC – TG			DRSOPC – CTG				
Nodes [N]	FPGA area	time	area	HW	time	area	HW	number of MET	performance increase [%]
10	1500	941	1498	4	804	1490	5	1	15
30	2000	4642	1863	13	3448	1986	15	2	26
50	2500	7054	2303	6	6531	2373	13	3	8
70	3000	9174	2935	21	8396	2964	35	4	9

6.1. EXAMPLE: EMBEDDED WEB SERVER

Today, many devices are controlled through Internet using HTTP protocol. Therefore, many embedded systems implement some network functionalities. Our real-life example is the module implementing a simple web server. Function of this server may be specified by the conditional task graph presented on Fig. 5. It consists of the following tasks:

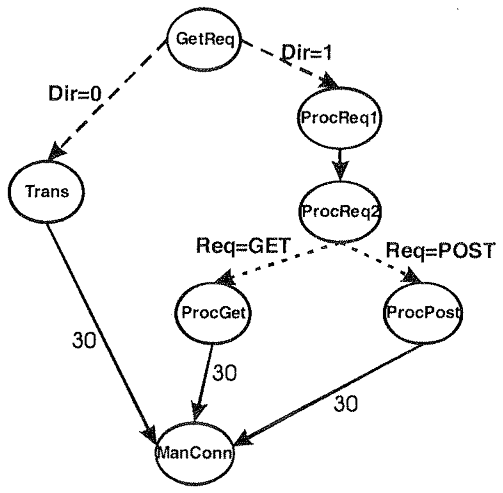


Fig. 5. Ctg specification of the embedded web server

- **GetReq:** process that waits on notifications on port 80. All requests are send to *ProcReq* and after emptying transceiver buffer the information is send to *Trans*.
- **ProcReq:** process that is activated after receiving information from *GetReq*. It reads data packages into buffer. After completing a full HTTP request the information is send to *ProcGet* or *ProcPost*, depending on the request type. Since this process is the most complex, it is decomposed into *ProcReq1* and *ProcReq2*.
- **ProcGet:** process the GET requests.
- **ProcPost:** process the POST requests.
- **Trans:** sends consecutive parts of HTML files.
- **ManCon:** manages the connection status.

Tasks *Trans* and *ProcReq* (*ProcReq1, ProcReq2*) are never executed in the same time, but depending on the *Dir* condition. If *Dir=1* then after finishing *ProcReq*, task *ProcGet* or *ProcPost* will be executed, depending on the *Req* condition. Thus, in this CTG hierarchical conditions exist.

Assume that there is only one *GPP* module available, with an area equal to 200 CLBs, and that the available communication channel transmits 30 B during 75 ns. Table 6 presents the library of hardware and software components. Synthesis results are given in Table 7. In the TG model only 2 tasks may be executed in parallel with others (*Trans* and *ProcGet* or *ProcPost*) , thus the difference between costs (in SOC implementation) for DE and TG models is not such significant. But, for the hardest constraint it is not possible to obtain solution using DE model. Similarly, for hard area constraint almost all tasks are executed by one GPP, thus it is not possible to achieve high speed, even for DRSOPC system. When the large FPGA area is available, most tasks are allocated in hardware, and reconfiguration is not necessary. Fig. 6 presents the Gantt chart for the fastest SOPC architecture containing one *GPP* executing 3 tasks and four *VCs*. Task schedule obtained for the same FPGA area constraint, but assuming DRSOPC architecture, is presented on Fig. 7. Increase in performance, in comparison to the SOPC implementation, equals 28%. Reconfigurations were partially performed in parallel with computations, thus the impact of the reconfiguration time on the decrease in the system performance was reduced.

Table 6

Parameters of tasks from the embedded web server specification

Task	SW		HW	
	t[µs]	C[100B]	t[µs]	C[CLB]
GetReq	2000	6	400	250
ProcReq1	1200	15	650	300
ProcReq2	2800	5	850	200
ProcGet	2500	13	1000	300
ProcPost	1500	12	300	50
Trans	500	4	150	150
ManConn	600	5	200	200

Table 7

Comparison of results for co-synthesis of SOPC and DRSOPC for different available area of FPGA for Embedded Web Server

T max	SOC-DE		SOC-TG		FPGA area	SOPC-DE		SOPC-TG		DRSOPC-TG		DRSOPC-CTG	
	time[µs]	area	time[µs]	area		time[µs]	area	time[µs]	area	time[µs]	area	time[µs]	area
9000	7950	493	8650	455	400	9900	284	9600	284	9100	384	7668	336
6000	6000	883	5550	737	750	6350	734	6050	734	5092	734	4792	736
4500	4500	1170	4050	1024	1000	6050	984	5150	968	3694	850	3402	964
3500	–	–	3500	1309	1300	4450	1234	3650	1268	3100	1270	3100	1220
					Cmax = 1220CLB, tmin = 3100µs			(Cmin = 200CLB, tmax = 11100 µs					

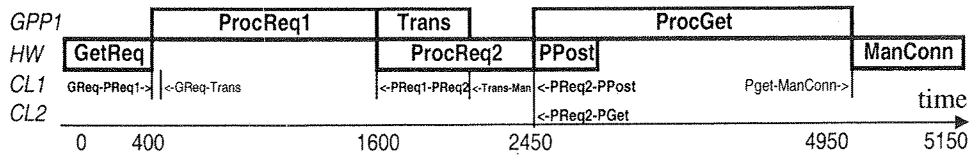


Fig. 6. Gantt chart of the web server implemented as SOPC

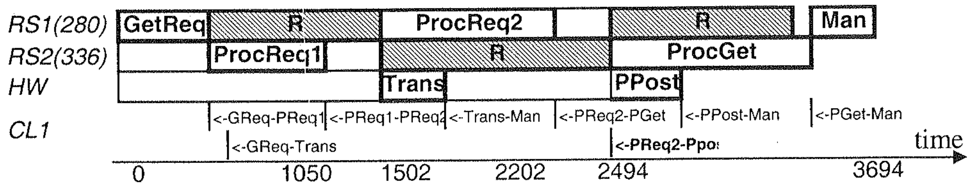


Fig. 7. Gantt chart of the web server represented by TG and implemented as DRSOPC

Finally, the system specified by the conditional task graph was synthesized. A few pairs of tasks are mutually exclusive: {(*Trans, ProcReq1*), (*Trans, ProcReq2*), (*ProcGet, ProcPost*), (*Trans, ProcGet*), (*Trans, ProcPost*)}. COSEDYRES algorithm found the architecture with two reconfigurable sectors, with areas equal to 280 and 336 CLBs. Fig. 8 illustrates scheduling of tasks for DRSOPC system specified by the CTG graph. Tasks *ProcReq2* and Trans were assigned to the same *RS1* sector and scheduled parallel (they are executed depending on the *Dir* condition). Such tasks need not be allocated in the same time, but can be allocated by dynamic reconfiguration after the condition is evaluated. Performance was increased by 8% in comparison with the previous solution.

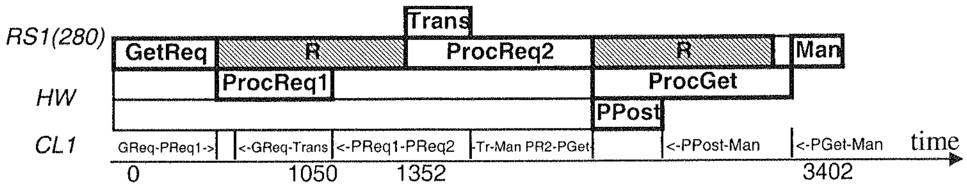


Fig. 8. Gantt chart of Web Server represented by CTG and implemented as DRSOPC

Presented examples showed that designing using task graph semantics in SystemC gives more information about dependencies between tasks and thus let for better optimization of designed systems by co-synthesis methods. The dynamic reconfiguration gives faster systems when reconfiguration tasks are properly scheduled and executed in parallel with computations. Moreover, considering mutually exclusive tasks in a system model brings additional possibilities of optimizations, in order to achieve faster dynamically reconfigurable systems (when MET tasks are allocated in the same sector).

7. CONCLUSIONS

This paper presented the methodology of developing synthesizable SystemC specifications of embedded systems and co-synthesis of such systems. System models created according to this methodology are semantically consistent with the model of computation represented by task graphs. In this way the same SystemC code may be used for modelling and for system synthesis, eliminating the necessity for manually creating task graphs. Moreover, the same model used for synthesis and for validation prevents from design errors.

To simplify the design process and reduce the time required for specification development the library of SystemC definitions was created. Communications channels and macros for process declaration restrict the synchronization and communication methods to be consistent with a task graph, a conditional task graph or a control-dataflow task graph.

Experimental results obtained for random generated systems and for real-life example confirmed, that our methodology forces designers to specify systems using model of computation, which enables obtaining very efficient architectures, using existing co-synthesis methods.

8. REFERENCES

1. *IEEE Standard SystemC Language Reference Manual*, IEEE, New York, 2006.
2. R. P. Dick, N. K. Jha: *CORDS: Hardware-Software Co-synthesis of Reconfigurable Real-time Distributed Embedded Systems*, Proc. ICCAD, 1998, pp.62-68.
3. K. B. Chehida, M. Auguin: *HW/SW Partitioning Approach for Reconfigurable System Design*, Proc. CASES 2002, 2002, pp. 247-251.
4. K. S. Chatha, R. Vemuri: *Hardware-software codesign for dynamically reconfigurable architectures*, Proc. FPL, 1999, pp.175-184.
5. S. Banerjee, E. Bozorgzadeh, N. Dutt: *Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration*, Proc. DAC, 2005, pp. 335-340.
6. Y. Qu, J.-P. Soininen, J. Nurmi: *A Parallel Configuration Model for Reducing the Run-time Reconfiguration Overhead*, Proc. DATE'06, 2006, pp. 965-969.
7. W. Wolf: *High-Performance Embedded Computing: Architectures, Applications, and Methodologies*, Morgan Kaufman, 2006.
8. A. Daboli, P. Eles: *Scheduling Under Data and Control Dependencies for Heterogeneous Architectures*, Proc. of the International Conference on Computer Design, 1998, pp. 602-608.
9. Y. Xie, W. Wolf: *Allocation and Scheduling of Conditional Task Graph in Hardware/Software Co-synthesis*, Proc. DATE, 2001, pp. 620-625.
10. D. Wu, BM Al-Hashimi, P. Eles: *Scheduling and mapping of conditional task graph for the synthesis of low power embedded systems*, IEEE Proceedings Computers and Digital Techniques, 2003, Vol. 150 Issue: 5 pp. 262-273.
11. Y. Xie, L. Li, M. Kandemir, et al.: *Reliability-aware co-synthesis for embedded systems*, Journal of VLSI Signal Processing Systems for Signal Image and Video Technology, 2007, Vol. 49, Issue: 1, pp. 87-99.

12. P. E l e s, K. K u c h c i n s k i, Z. P e n g, A. D o b o l i, P. P o p: *Scheduling of Conditional Process Graphs for the Synthesis of Embedded Systems*, Proc. of the IEEE DATE Conf., 1998, pp.132-138.
13. K. S. V a l l e r i o, N. K J h a: *Task Graph Extraction for Embedded System Synthesis*, Proc. IEEE Int. Conference on VLSI Design, 2003, pp. 480-486.
14. S. A. K h a y a m, S. A. K h a n, S. S a d i q: *A Generic Integer Programming Approach to Hardware/Software Codesign*, Proc. of IEEE International Multi Topic Conference IEEE INMIC 2001. Technology for the 21st Century, 200, pp. 6-9.
15. R. P. D i c k, N. K. J h a: *MOGAC: A multiobjective Genetic Algorithm for the Co-Synthesis of Hardware-Software Embedded Systems*, Proc. of the International Conference on Computer Aided Design, IEEE Computer Society Press, Los Alamitos, 1997, pp. 522-529.
16. B. P. D a v e, G. L a k s h m i n a r a y a n a, N. K. J h a: *COSYN: Hardware-Software Co-Synthesis of Embedded Systems*, Proc. of the 34th Design Automation Conference. ACM Press, New York, 1997, pp. 703-708.
17. T.-Y. Y e n, W. H. W o l f: *Sensitivity-Driven Co-Synthesis of Distributed Embedded Systems*, Proc. of International Symposium on System Synthesis, 1995, pp. 4-9.
18. R. P. D i c k, N. K. J h a: *MOCSYN: Multiobjective Core-Based Single-Chip System Synthesis*, Proc. of the Conference on Design Automation and Test in Europe. IEEE Computer Society Press, Los Alamitos, 1999, pp. 263-270. [19.] L. S h a n g, R. P. D i c k, N. K. J h a: *SLOPES: Hardware-Software Cosynthesis of Low-Power Real-Time Distributed Embedded Systems With Dynamically Reconfigurable FPGAs*, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2007, pp. 508-526.
20. J. O u, S. B. C h o i, V. K. P r a s a n n a: *Energy-Efficient Hardware/Software Co-synthesis for a Class of Applications on Reconfigurable SoCs, International Journal of Embedded Systems*, 2005, Vol. 1, No.1/2, pp. 91-102.
21. F. F e r r a n d i, M. D. S a n t a b r o g i o, D. S c i u t o: *A Design Methodology for Dynamic Reconfiguration: The Coronte Architecture, 19th IEEE International Parallel and Distributed Processing Symposium - Workshop 3*, 2005, pp. 163-166.
22. M. T. S c h m i t z, B. M. A l - H a s h i m i, P. E l e s: *Energy-Efficient Mapping and Scheduling for DVS Enabled Distributed Embedded Systems*, Proc. of the Conference on Design Automation and Test in Europe. IEEE Computer Society Press, Los Alamitos, 2002, pp. 514-521.
23. A. R o s s, S. S w a n, J. P i e r c e, J.-M.: *Fernandez: Transaction Level Modeling in SystemC*, www.systemc.org.
24. A. D o n l i n: *Transaction Level Modeling: Flows and use models*, Proc. CODES+ISSS'04, 2004, pp.75-80.
25. H. H e r r e r a, P. S a n c h e z, E. V i l l a r: *Modeling and Design of CSP, KPN and SR Systems with SystemC, in C. Grimm (ed.) Languages for System Specification*, Kluwer, 2004.
26. H. H e r r e r a, E. V i l l a r: *A Framework for Embedded System Specification under Different Models of Computation in SystemC*, Proc. IEEE/ACM Design Automation Conf., 2006, pp. 911-914.
27. J. F a l k, C. H a u b e l t, J. T e i c h: *Efficient Representation and Simulation of Model-Based Designs in SystemC*, Proc. Forum on Design Languages, 2006, pp. 129-134.
28. C. H a u b e l t, J. F a l k, J. K e i n e r t, T. S c h l i c h t e r, M. S t r e u b ü h r, A. D e y h l e, A. H a d e r t, J. T e i c h: *A SystemC-based Design Methodology for Digital Signal Processing Systems*, EURASIP Journal on Embedded Systems, March 2007.
29. J. H e n k e l, R. E r n s t: *High-level estimation techniques for usage in hardware/software co-design*, Proc. Asia and South Pacific Automation Conference, 1998, pp. 353-360.
30. S. D e n i z i a k: *Cost-Efficient Synthesis of Multiprocessor Heterogeneous Systems*, Control and Cybernetics, 2004, Vol.33, No.2, pp. 341-355.
31. R. C z a r n e c k i: *Kosynteza dynamicznie samorekonfigurowalnych systemów wbudowanych*, PhD Thesis, 2008, p.123, (In Polish).

32. X i l i n x I n c.: *Two flows for partial reconfiguration: module based or difference based*, Xilinx Application Note XAPP290, v.1.2, 2004.
33. R. C z a r n e c k i, S. D e n i z i a k: *Co-Synthesis of Dynamically Reconfigurable SOPCs Specified by Conditional Task Graphs*, The Open Cybernetics and Systemics Journal, 2008, Vol. 2, pp.206-218.

N
gning
PLAs
progr
mable
algor
and t

ELECTRONICS AND TELECOMMUNICATIONS QUARTERLY, 2009, 55, no 1, pp. 95–111

A Novel Non-Disjunctive Method for Decomposition of CPLDs

ADAM OPARA, DARIUSZ KANIA*

Silesian University of Technology, Department of Computer Science,
**Department of Electronics 44-100 Gliwice, ul. Akademicka 16*
Adam.Opara@polsl.pl, Dariusz.Kania@polsl.pl

Received 2008.12.15
Authorized 2009.01.20

The paper discusses the concept of a novel decomposition method dedicated for PAL-based CPLDs. The proposed approach is an alternative to the classical one, which is based on two-level minimization of separate single-output functions. The key idea of the algorithm is to search for free blocks that could be implemented in PAL-based logic blocks containing a limited number of product terms. In order to exploit better the number of product terms, a non-disjunctive decomposition is to be used. In contrast to classical methods, the functions are represented by Reduced Ordered Binary Decision Diagrams (ROBDD). The results of the experiments prove that the proposed solution is more effective in terms of the usage of programmable device resources, compared to the classical ones.

Keywords: Technology mapping, decomposition, CPLD, BDD

1. INTRODUCTION

Nowadays, programmable logic devices (PLDs) are very extensively used in designing electronic digital circuits. Simple PLDs can be divided into several kinds: PALs, PLAs, and PLEs. Based on simple PLDs, there is a group of devices called complex programmable logic devices (CPLDs). Other PLDs include FPGA (Field Programmable Gate Array) devices. Due to high complexity of these systems, efficient CAD algorithms must be developed to address their design challenges. Logic minimization and technology mapping are two important elements in this process.

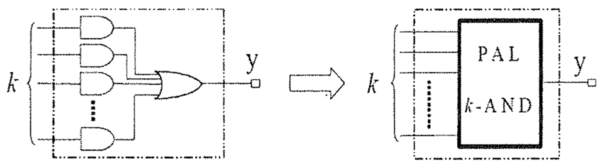


Fig. 1. Structure of a typical PAL-based CPLD block

A classical approach to the synthesis of PAL-based CPLD structures, implemented in many computer aided design tools, employs a two-level minimization of separate functions and technological fitting to the structure of programmable logic blocks [1]. The Espresso algorithm is mainly used for two-level minimization of Boolean functions [2].

The strategies of synthesis implemented in commercial CAD tools are designed mostly for small group of devices produced by single manufacturers, but they do not provide effective solutions. The synthesis methods implemented in the Hardware Description Language (HDL) compilers, such as VHDL or Verilog, use a multi-level representation of functions, while the synthesis process consists in fitting a function to the technology library patterns. These methods lead to inefficient use of the available product terms in PAL-based logic.

Recently, synthesis methods dedicated for FPGA devices based on functional decomposition have become very popular [3][4][5][6][7]. Sometimes, these methods are used for CPLD devices [8][9][10][11][12][13][14], but they cannot be directly applied. In order to use efficiently the resources of programmable structures, it is necessary to consider the characteristic features of CPLDs in the initial stage of the synthesis process. The most important feature is the number of product terms in a PAL-based logic block (Fig. 1).

The purpose of this paper is to present the decomposition methods based on binary decision diagrams (BDD) [15] dedicated for CPLDs. Only the devices with PAL-type logic blocks have been considered. The ideas presented here were inspired by a two-stage PAL-decomposition described in [16]. A novel non-disjunctive PAL-decomposition based on BDD was introduced. This decomposition model is particularly promising in case of hardly decomposable functions, and also in case of the devices incorporating PAL-based logic blocks with the number of product terms different than power of 2.

2. A DECOMPOSITION WITH PAL-BASED BLOCK FEASIBILITY

The kernel of the most popular CPLD devices is a PAL-based structure, which consists of the determined (in most cases, constant) number of terms connected to an output cell. The terms with the output cell are called PAL-type logic blocks. A two-stage PAL decomposition is presented in [16]. This decomposition offers a more efficient logic block use than the standard two level minimization and fitting. An adaptation to number of terms in PAL based logic block is a characteristic feature of a two-stage PAL decomposition. Similarly to the Ashenhurst-Curtis decomposition, a partition of a variable set into the free and bound sets has the major importance (Fig. 2). The partition is chosen so the free block can be created in one PAL based logic block.

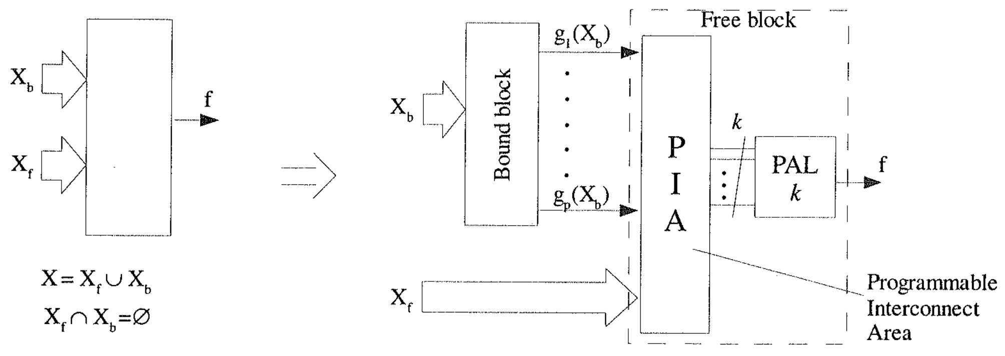


Fig. 2. Circuit partition after decomposition

$x_0x_1x_2$ \ $x_3x_4x_5$	000	001	011	010	110	111	101	100	
000	0	1	0	1	0	1	0	1	**a**
001	0	0	1	0	0	1	1	0	**b**
011	0	1	0	1	0	1	0	1	**a**
010	1	0	1	0	1	0	1	0	**a**
110	0	1	0	1	0	1	0	1	**a**
111	0	0	0	0	0	0	0	0	**0**
101	1	1	1	1	1	1	1	1	**1**
100	0	0	1	0	0	1	1	0	**b**

f

$X_f = \{x_0, x_1, x_2\}, \quad X_b = \{x_3, x_4, x_5\}$

$\mathbf{a} \rightarrow g_1(x_3, x_4, x_5) = \overline{x_3}\,\overline{x_4}\,x_5 + \overline{x_3}\,x_4\,\overline{x_5} + x_3 x_4 x_5 + x_3\,\overline{x_4}\,\overline{x_5}$

$\mathbf{b} \rightarrow g_2(x_3, x_4, x_5) = x_4 x_5 + x_3 x_5$

Fig. 3. Karnaugh map with distinguished row patterns

The two-stage PAL decomposition algorithm uses a Karnaugh map as a representation of the logic function. The rows of the map are described by the values of bound variables, whilst the columns are denoted by the values of free variables. In this map, row patterns can be determined. For function $f(X), X = X_f \cup X_b, X_f \cup X_b = \emptyset$, a **row pattern** is a row described by the function

$g(X_b)$, which returns the value 1 for all cubes associated with the columns, for which the function $f(X)$ value is 1. In case of the Karnaugh map depicted in Fig. 3, the first row denoted by letter **a** is described by the function $f_{x0=0,x1=0,x2=0} = g_1(x_3, x_4, x_5) = \bar{x}_3\bar{x}_4x_5 + \bar{x}_3x_4\bar{x}_5 + x_3x_4x_5 + x_3\bar{x}_4\bar{x}_5$. There are here three rows described by this pattern $f_{x0=0,x1=0,x2=0} = f_{x0=0,x2=1,x2=1} = f_{x0=1,x1=1,x2=0} = g_1(x_3, x_4, x_5)$. If the row pattern a is described by the function $g_1(X_b)$, the row pattern described by $\overline{g_1(X_b)}$, denoted by **ā**, is called the **row pattern complement**. In case of the Karnaugh map illustrated in Fig. 3, there were determined two other special cases of row patterns: a **full row** denoted by **1** and an **empty row** denoted by **0**.

The rows in the Karnaugh map can be broken down into a few groups:
– empty rows,
– full rows,
– rows associated with the same row pattern or its complement.

A **row multiplicity** of a partition matrix denoted by $\mu(X_f|X_b)$ is defined as a number of different row groups, except the groups containing empty and full rows. It is defined by the expression (1).

$$\begin{aligned}
&f(X, X = \{x_0, ..., x_{n-1}\}, X_b = \{x_q, ..., x_{n-1}\}_{,q<n},\\
&X_b \bigcup X_f = X, X_bX_f = \emptyset\\
&A = \{g(x_q, ..., x_{n-1}):\\
&\quad g = f_{x_0=\beta_0,...,x_{q-1}=\beta_{q-1}}, \beta_j \in \{0,1\}, j = 0, ..., q-1\}\\
&A - \text{a row patterns s et}, B - \text{a row groups s et}\\
&\mu(X_f \mid X_b) \overset{\text{def}}{=}\\
&\mid B : \quad \forall\\
&\quad {}_{g_i,g_j \in B, g_i \neq g_j} g_i, g_j \in A, g_i \neq \overline{g}_j, g_i \neq 0, g_i \neq 1 \mid
\end{aligned} \tag{1}$$

For function presented in Fig. 3, the row multiplicity is $\mu(X_f|X_b)$=2; the first group of rows contains rows **a** and **ā**, the second – rows **b**.

Each group of rows can be assigned a function defined as follows:

$$h_0(X_f) = \begin{cases} 1, & \text{for full rows} \\ 0, & \text{for all other rows} \end{cases}$$

$$h_i(X_f) = \begin{cases} 1, & \text{for rows, for which the row pattern is described by the function } g_i(X_b) \\ 0, & \text{for all other rows} \end{cases}$$

$$h'_i(X_f) = \begin{cases} 1, & \text{for rows, for which the row pattern is described by the function } \overline{g_i}(X_b) \\ 0, & \text{for all other rows} \end{cases}$$

$$\text{where } i = 1, ..., \mu(X_f \mid X_b) \tag{2}$$

Using these functions, and assuming that the row multiplicity is p, the following equation is derived:

$$\mu(X_f \mid X_b) = p \implies f(X) = h_0(X_f + \sum_{i=1}^{p} \left[h_i(X_f) \cdot g_i(X_b) + h_i'(X_f) \cdot \overline{g_1(X_b)}\right] \tag{3}$$

In case of the discussed example of a function with variables partition $X_f = \{x_0, x_1, x_2\}$, $X_b = \{x_3, x_4, x_5\}$, the functions h are expressed as follows:

$$h_0(X_f) = x_0\bar{x}_1x_2,$$

$$h_1(X_f) = \bar{x}_0\bar{x}_1\bar{x}_2 + \bar{x}_0x_1x_2 + x_0x_1\bar{x}_2, \quad h_1'(X_f) = \bar{x}_0x_1\bar{x}_2$$

$$h_2(X_f) = \bar{x}_0\bar{x}_1x_2, \qquad h_2'(X_f) = 0$$

and finally, function f will be of the form:

$$\begin{aligned} f(X) &= x_0\bar{x}_1x_2 \\ &+(\bar{x}_0\bar{x}_1\bar{x}_2 + \bar{x}_0x_1x_2 + x_0x_1\bar{x}_2)g_1(x_3, x_4, x_5) \\ &+\bar{x}_0x_1\bar{x}_2\overline{g_1(x_3, x_4, x_5)} + \bar{x}_0\bar{x}_1x_2g_2(x_3, x_4, x_5) \end{aligned} \tag{4}$$

Fig. 4. depicts the implementation of the function under consideration into a CPLD device with 6 product terms in one PAL-based block. As it can be seen, 3 PAL-based logic blocks were employed here.

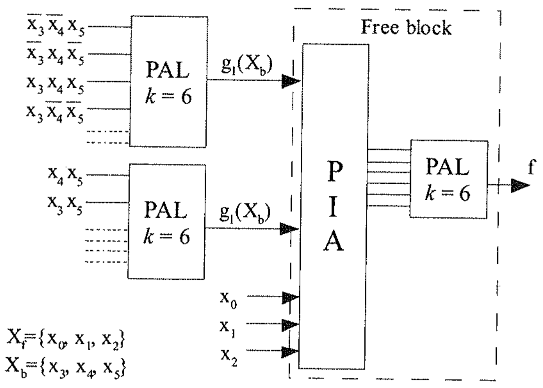


Fig. 4. Circuit after decomposition

To compare the obtained result with the classical one, a new formula must be introduced. With k-input PAL-based configurable logic blocks and functions represented by a sum of Δ_f products, the required number of logic blocks δ_f utilized in a classical approach is determined by the expression (5).

$$\delta_f = \left\lceil \frac{\Delta_f - k}{k - 1} \right\rceil + 1 \tag{5}$$

After a two-level minimization is done with the use of the Espresso algorithm, the function under consideration (Fig. 3) can be represented as a sum of 21 products. A classical approach requires 4 PAL-based logic blocks with 6 product terms i.e. one more block compared to the decomposition-based approach.

The main idea of the two-stage PAL decomposition is to search such variable partitions which could provide (i) a free block realisation in one PAL-based logic block and (ii) the smallest number of the bound block outputs. The number of the bound block outputs is equal to the row multiplicity. An effective computation of the row multiplicity for a given variable partition is the major problem in two-stage PAL decomposition algorithms. The detailed description of the decomposition algorithm can be found in [16].

3. BINARY DECISION DIAGRAMS

The Binary Decision Diagram is a graph-based structure used for a memory-efficient representation of logic functions. The BDDs were first proposed by Akers [17], and popularized by Bryant [15] and Brace et al [18]. Due to their implicit power to represent Boolean functions, BDDs are considered the most efficient Boolean representation known so far.

A BDD is a directed acyclic graph (a tree) with each node associated with a function variable. All nodes (except terminal ones) have two outgoing edges pointing two children nodes, one for variable value 0 and 1. This binary tree contains two terminal nodes termed 0-node and 1-node. The analysis of the paths connecting to the BDD terminal nodes determines the value of the function according to the values of the variables.

Only the Reduced Ordered Binary Decision Diagrams (ROBDD) have a practical meaning. In an Ordered BDD, the variables in all paths has the same variable order, and they are presented at most once on every path. The Reduced Ordered BDDs have a minimal number of nodes for the given variable order and are canonical forms of function representation. The reduced form is obtained from an OBDD by applying the reduction of the same sub-graphs and through removing all redundant nodes.

There are some ROBDDs with special attributes added to the edges for efficient memory use and faster computations [19]. A complement is one of the most known attributes. If the edge is complemented, it means that the sub-diagram pointed by this edge must be interpreted as a negation of the formula represented by the sub-diagram.

4. ROBDD APPLICATION IN DECOMPOSITION

A classical two-stage PAL decomposition employs a partition matrix as a representation of the logic function. There is a possibility to develop an algorithm using a Reduced Ordered Binary Decision Diagrams (ROBDD) as an effective representation, followed by a non-disjunctive decomposition, whilst the application of the negation attribute can additionally increase the algorithm's efficiency.

4.1. COUNTING THE NUMBER OF PATHS

As far as the synthesis of digital circuits in programmable structures with PAL-based blocks is concern, the key problem is to determine the minimal number of products in the product representation sum. In a classical approach, the Espresso algorithm may be used for this purpose. When the ROBDD is used for the logic function representation, another concept can be exploited. Each path in the diagram obtained from a root to a leaf 1 corresponds to one product. The total number of paths can vary with different variable orderings in the diagram. Changing the variable order is a way to minimize the path number. Often, the smallest number of paths is greater than the number of products after minimization, although the decomposition with path counting can provide better results than the classical approach with two level Espresso minimization.

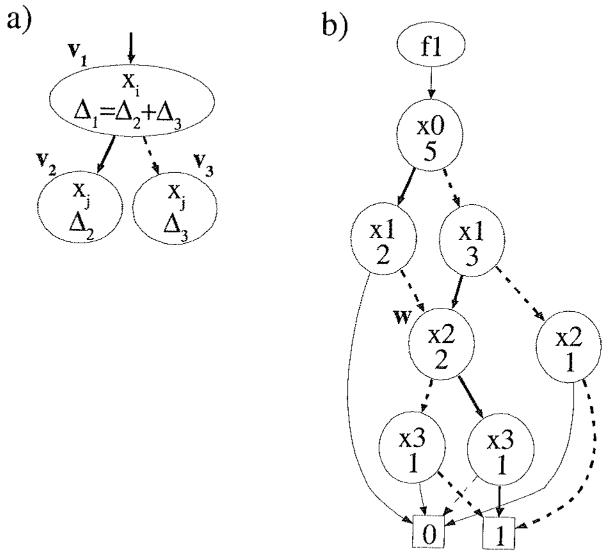


Fig. 5. Counting the number of paths

The main advantage of the method used to determine the number of products through counting the paths is the low computation complexity. The number of paths can be counted by a recursive procedure. The number of paths Δ_1 connecting the given node $\mathbf{v}_1$ to the leaf node **1** is equal to the sum of the number of paths connecting the children node (*high*($\mathbf{v}_1$), *low*($\mathbf{v}_1$)) to the leaf node **1** (Fig. 5a). Similarly to the standard procedure bdd_apply() [15], a computed table is used to store the intermediate and final results of each algorithm iteration. A result in this context means the number of paths for a given node, which is the root of a sub-graph representing a function. Due to the use of cached intermediate results, a path counting procedure will be performed only once for each node. For instance (see Fig. 5b) for a node denoted by **w**, the number of paths will be computed only once, although two edges point to this node and during a depth first traversal across the diagram, this node will be visited twice. The computation complexity of the procedure counting number of paths is $O(n)$, where n is the number of nodes in the diagram.

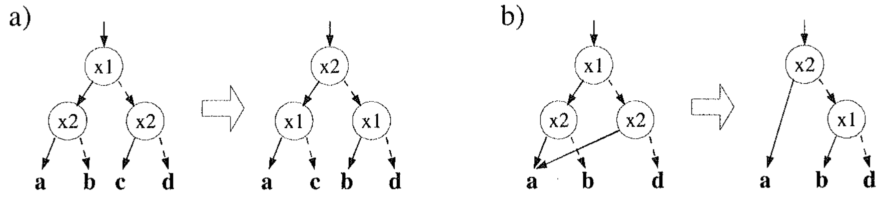


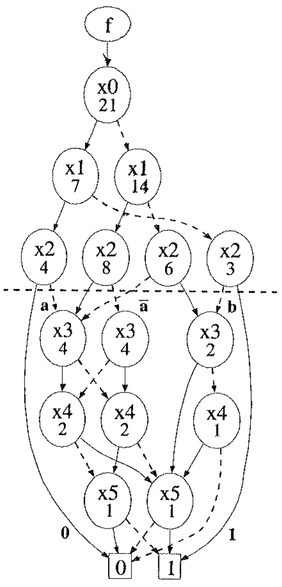
Fig. 6. An example of variable swapping in BDD ordering

The number of paths in the diagram highly depends on the variable order. It is possible to use heuristic algorithms similar to the algorithms aiming at the minimizing the number of nodes for the number of paths minimization [20]. For this purpose, a sifting algorithm [21] can be used but the optimality criterion must be changed. Each variable is moved up and down in the variable order and the position that produces the smallest OBDD size is maintained. At each position, the resulting ROBDD size is recorded and finally, the variable is moved to the best position. The ordering change is performed by swaps of variables which are adjacent in the variable ordering. The variable swapping affects the BDD structure of only two levels involved in the swap, whilst the whole part of ROBDD above and below these levels remain unchanged. All modifications have a local scope and concern two levels of nodes. This local-level of the swap operation is responsible for the efficiency of the sifting algorithm. Fig. 6 illustrates some portions of the ROBDD diagrams before and after the swap operation on two levels assigned variables x_1 and x_2. The number of nodes and paths after swapping is unchanged (see Fig. 6a) and changed (see Fig. 6b), respectively. In the second case, there is a need to recompute the total number of nodes in the diagram. Since only two levels are altered, only the number of nodes in two levels must be recounted, and the difference between the nodes number before and after swapping is added to the previous total number of nodes in the diagram. In order to compute the new number of paths in the diagram, also the results of the previous calculation can be employed. Exchanging two adjacent levels has no influence on the number of paths below these levels. The number of paths for all nodes in the upper part of the diagram must be updated. In this case, only the time of processing of the lower part of the diagram is saved.

4.2. PAL-ORIENTED BDD-BASED DECOMPOSITION

The core of the PAL-oriented decomposition is to search for a such partition of function variables to assure the free block implementation in one PAL-based block with a constrained number of the product terms. Furthermore, the partition found must provide a structure with the smallest possible outputs number of the bound block. The partitioning of the variables in a partition matrix is equivalent to the cut in the ROBDD diagram representing the logic function. The variables associated with the nodes above the cut line form a free set X_f, and below the cut line – a bound set X_b (contrary to the Ashenhurst-Curtis decomposition using the ROBDD representation).

Fig. 7 depicts the ROBDD corresponding to the Karnaugh map of the function under consideration in Fig. 3. All nodes pointed by edges crossed by the cut line will be termed the cut nodes. As it can be seen, each cut node is associated with one row pattern. The row multiplicity $\mu(X_f|X_b)$ is the number of row groups, being one row group formed by a row pattern or its complement. All nodes in a ROBDD with edge complement attributes corresponds to one row group. The row multiplicity can be efficiently computed in ROBDD with edge complement attributes by counting the number of cut nodes. Different partitions are obtained by changing the variable ordering in ROBDD and fixing the level of the cut line diagram.



$$\mathbf{a} \rightarrow g_1(x_3, x_4, x_5)$$
$$\mathbf{b} \rightarrow g_2(x_3, x_4, x_5)$$

$$\begin{aligned} f_h = {} & x_0 \overline{x_1} x_2 \\ & + \overline{x_0}\,\overline{x_1}\,\overline{x_2}\, g_1 \\ & + \overline{x_0} x_1 x_2\, g_1 \\ & + x_0 x_1 \overline{x_2}\, g_1 \\ & + \overline{x_0} x_1 \overline{x_2}\, \overline{g_1} \\ & + \overline{x_0}\,\overline{x_1} x_2\, g_2 \end{aligned}$$

Fig. 7. A diagram with the number of paths of the function under consideration

The decomposition algorithm consists of some phases. During each phase, there is established the number of free set variables which corresponds to the cut level. A variable partition is searched, which allows to obtain a free block in one PAL-based logic block and the smallest number of the bound block outputs. If in a given phase the solution is found, the cut level is incremented.

```
void decBDD(bdd f,  //decomposed function diagram
  int &levels,          //returned number of levels
  int &blocks,          //  and PAL blocks
  int K,                //product term # in PAL block
  int prlesp){      //function first time called
                    //with products # of espresso Dso
 int pr0=0, pr1=0, min_pr01;     //products #
 //path # counting - reorder BDD, return minimal # of
 //paths (products) to leaf "0" and "1"
 prod_bdd(&f, &pr0, &pr1);

 if(prlesp!=0)
   min_pr01= prlesp;//first call only products to "1"
 else
   min_pr01= min(pr0, pr1)
 //for comparison classical algorithm results
 int blocks_class= blocks_classical(min_pr01,K);
 int levels_class= levels_classical(min_pr01,K);

 if(min_pr01 < 2*K){
 //---------better classical realisation-----------
    blocks= blocks_class;
    levels= levels_class;
 }else{
 //---------decomposition--------------------------
    int cut_level= floor_log2(K) + 1;
    int li_wez_odc_best_dw;
    tbdd_set cut_nodes_set= Ø;
    bool found,found_K1;
    //searching for best partition with fixed cut lev
    found_K1= find_part(&f, &cut_nodes_set,
                        cut_level, K);
    if(found_K1){
        do{ //searching for better solutions
            found= find_part(&f, &cut_nodes_set,
                          cut_level, K);
            cut_level++;
        }while(found);
        cut_level--;
    }else{
        //search for cut level = log2 K
        cut_level--;
        found= find_part(&f, &cut_nodes_set,
              cut_level, K);
    }
    if(blocks_class <= number_of_cut_nodes+1){
        //-------better classical realisation--------
        blocks= blocks_class;
        levels= levels_class;
    }else{
       //-------cut nodes decomposition-------------
       int tmp_level, tmp_blocks;  //temporary var.
       n_lev=0, n_blocks=0;        //returned values

       for(ftmp ∈ cut_nodes_set){
           decBDD(ftmp, &tmp_level, &tmp_blocks,
                         K,0);
           blocks += tmp_blocks;
           levels = max(levels, tmp_levels);
       }

        levels+=1; blocks+=1;       //free block
        if(blocks >= blocks_class){
            //better classical realisation----------

            blocks= blocks_class;
            levels= levels_class;
        }
    }
 }//end decomposition--------------------------------
}
```

Fig. 8. The algorithm of PAL decomposition with BDD application

A detailed listing of the decomposition algorithm is presented in Fig. 8. The minimal number of products in the sum of products form is determined by the path counting (line 7, Fig. 8) in the ROBDD. The number of paths to the leaf **1** and **0** has been counted. Since the relation of the row complementing is symmetrical, there is possibility to assign a function or its complement to the rows, and to obtain the solution with the lower number of paths (products). In the case of using PAL-based logic blocks without the possibility to program the output polarities first time the decomposition procedure is employed, only positive polarisation is accepted (lines 8-11).

Additionally the number of levels and blocks in a classical approach (formula 5) is computed in the lines 13-14.

The algorithm contains a few improvements just eliminating certain situations which otherwise would be further processed, e.g. if a function after minimization is described by less than 2k implicants (line 15) (where k is the number of product terms in PAL-based logic block) then the decomposition will not reduce the number of blocks because one PAL-based logic block is needed for the free block and at least one block for the bound block, respectively. After this condition is met, a partition is searched (line 23).

After a free block is ensured in one PAL-based logic block for all partitions for which holds: $2^{|X_f|} < k$, $| X_f | \leq \lfloor \log_2 k \rfloor$, the search process will start with $| X_f | = \lfloor \log_2 k \rfloor + 1$ (lines 19 and 23).

After a partition has been found, a check will be done if the partition could reduce the number of PAL-based logic blocks compared to the classical approach (line 35). The minimal number of PAL-based logic blocks required to implement a circuit after partitioning is equal to $\mu(X_f|X_b)+1$, so the condition necessary to eliminate some partitions from further processing is given as:

$$\mu(X_f \mid X_b) < \delta_f = \left\lceil \frac{\Delta_f - k}{k-1} \right\rceil + 1 \qquad (6)$$

If the above condition is not true, further processing is to be continued and the functions represented by cut nodes will be decomposed (lines 41-45). At the end, the last check is made (lines 46-50) if the decomposed function truly gives lower number of blocks than the classical approach.

4.3. ALGORITHM REFINEMENTS

The number of paths in a ROBDD diagram connecting the root to the leaf **1** in some cases can significantly differ than the number of product terms of two level minimized logic function, e. g. a function with two products $f_0 = x_0x_1x_2 + x_3x_4x_5$, has four paths with variable ordering $x_0, x_1, x_2, x_3, x_4, x_5$ (Fig. 9). Using these paths, this function can be represented as a sum of four products $f_0 = x_0x_1x_2 + \bar{x}_0x_3x_4x_5 + x_0\bar{x}_1x_3x_4x_5 + x_0x_1\bar{x}_2x_3x_4x_5$. Although this representation is not optimal (as the experiments on benchmarks prove), the path counting decomposition gives good results

(Table 1). For further enhancement of the algorithm, the Espresso algorithm has been used instead of path counting. Looking at the algorithm in Fig. 8 the only difference is in line 7, where a two level minimization algorithm is employed as an alternative.

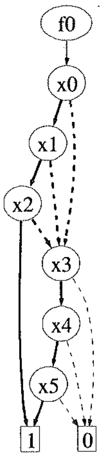


Fig. 9. A diagram with bold paths to leaf "1"

4.4. A NON-DISJUNCTIVE PAL DECOMPOSITION

In order to reduce the number of blocks level a non-disjunctive partitions can be employed. A non-disjunctive decomposition is that of the function under consideration (Fig. 10) implemented with PAL-based blocks containing 3 product terms. The first stage of the non-disjunctive decomposition is to find a good disjunctive partition. For a given variable, the ordering only x_0, can be included into the free set. In this case, a free block described by formula $f = x_0 \cdot g_2\ (x_1, x_2, x_3, x_4) + \bar{x}_0 \cdot g_1\ (x_1, x_2, x_3, x_4)$, is implemented by two product terms. Function g_2 describes a diagram rooted by node $\mathbf{v}_2$, and g_1 by $\mathbf{v}_1$. Due to the inclusion of one more variable (x_1) to the disjunctive free set, 4 product terms are needed, so limit of 3 terms in a PAL block is exceed. Function g_1 is created in one PAL block and g_2 in two blocks, respectively. Finally, using a disjunctive decomposition, a circuit can be implemented with 4 blocks situated in 3 levels.

Through the introduction of a non-disjunctive decomposition, the variable x_1 is included into the free and bound set. The free block is described by the formula $f = x_0 \cdot x_1 \cdot g_0 + x_0 \cdot \bar{x}_1 \cdot \bar{g}_0 + \bar{x}_0 \cdot g_1$, and utilizes 3 product terms. The whole circuit is built of 3 PAL-based logic blocks in 2 levels (Fig. 11).

The algorithm presented in Fig. 8 is modified so after a proper disjunctive partition is found (lines 23-34), a procedure is employed to try to add one child of a cut node to the cut node set. In the considered 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 one PAL-based logic block.

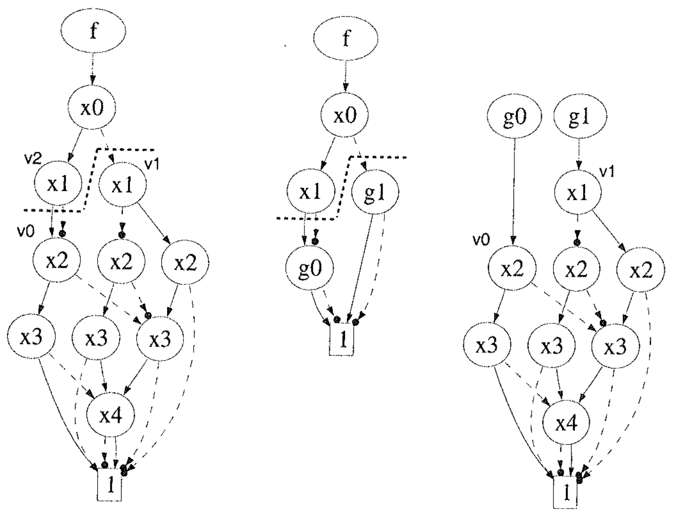


Fig. 10. A diagram cut corresponding to a non-disjoint decomposition

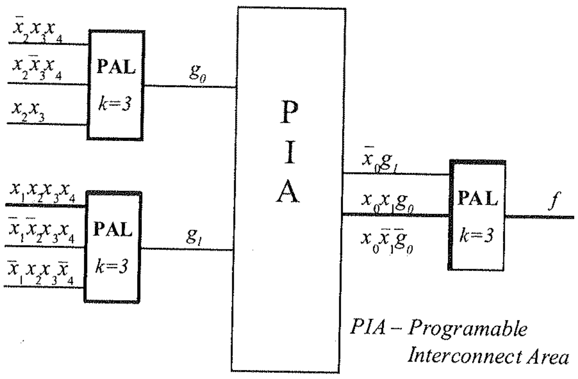


Fig. 11. A circuit structure after non-disjunctive decomposition

5. NUMERICAL RESULTS

A method of implementing a function in PAL-based structures incorporating BDD presented in this paper – decBDD – has been compared to a classical approach with respect to the number of logic blocks used, and the number of logic levels. The comparison was made for an algorithm in two versions: a simple one, denoted by decBDD, and enhanced one, denoted by decBDD+E in Table 1. For multi-output benchmarks,

Table 2

Results of experiments compared to the classical method of function implementation

			Classic																DecBDD																Gain	DecBDD+E																Gain
			k=3		k=4		k=5		k=6		k=7		k=8		k=12		k=16		k=3		k=4		k=5		k=6		k=7		k=8		k=12		k=16			k=3		k=4		k=5		k=6		k=7		k=8		k=12		k=16		
	Esp	Bdd	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L		B	L	B	L	B	L	B	L	B	L	B	L	B	L	B	L	
f2:	18	19	9	3	6	3	5	2	4	2	3	2	3	2	2	2	2	2	6	4	5	3	3	2	4	2	3	2	3	2	2	2	2	2		5	3	3	2	3	2	4	2	3	2	3	2	2	2	2	2	
f3:	14	15	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	4	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1		4	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1	
f4:	10	10	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	3	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1		3	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	
5xp1.pla			35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	27	4	23	3	19	2	18	2	16	2	15	2	12	2	11	2	14	26	3	21	2	19	2	18	2	16	2	15	2	12	2	11	2	1
9sym.pla	86	148	43	5	29	4	22	3	17	3	15	3	13	3	8	2	6	2	37	5	20	4	17	3	14	3	12	3	7	3	7	3	5	2	34	37	5	20	4	17	3	14	3	12	3	7	3	7	3	5	2	3
f0:	21	22	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	9	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2		9	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	
f2:	42	54	21	4	14	3	11	3	9	3	7	2	6	2	4	2	3	2	21	4	13	3	11	3	9	3	7	2	6	2	4	2	3	2		21	4	13	3	11	3	9	3	7	2	6	2	4	2	3	2	
f3:	34	43	17	4	11	3	9	3	7	2	6	2	5	2	3	2	3	2	17	4	9	3	9	3	7	2	6	2	5	2	3	2	3	2		15	4	9	3	9	3	7	2	6	2	5	2	3	2	3	2	
f4:	20	21	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	6	3	5	3	5	2	4	2	4	2	3	2	2	2	2	2		6	3	5	3	5	2	4	2	4	2	3	2	2	2	2	2	
clip.pla			73	4	49	3	38	3	30	3	26	2	22	2	14	2	12	2	68	4	44	3	38	3	30	3	26	2	22	2	14	2	12	2	10	66	4	44	3	38	3	30	3	26	2	22	2	14	2	12	2	1
f0:	23	23	11	3	8	3	6	2	5	2	4	2	4	2	2	2	2	2	11	3	5	3	4	3	3	2	3	2	4	2	2	2	2	2		11	3	5	3	4	3	3	2	3	2	4	2	2	2	2	2	
f1:	18	19	9	3	6	3	5	2	4	2	3	2	3	2	2	2	2	2	9	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2		9	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2	
f2:	14	15	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	7	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1		7	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1	
f51m.pla			37	3	27	3	22	2	18	2	16	2	15	2	11	2	10	2	37	3	19	3	17	3	15	2	15	2	15	2	11	2	10	2	17	37	3	19	3	17	3	15	2	15	2	15	2	11	2	10	2	1
f1:	16	16	8	3	5	2	4	2	3	2	3	2	3	2	2	2	1	1	4	4	2	2	2	2	2	2	2	2	2	2	2	2	1	1		4	4	2	2	2	2	2	2	2	2	2	2	2	2	1	1	
f2:	10	14	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1		5	3	3	2	3	2	2	2	2	2	2	2	1	1	1	1	
rd53.pla			15	3	10	2	8	2	6	2	6	2	6	2	4	2	3	1	11	4	7	2	6	2	5	2	5	2	5	2	4	2	3	1	12	11	4	7	2	6	2	5	2	5	2	5	2	4	2	3	1	1
f0:	42	48	21	4	14	3	11	3	9	3	7	2	6	2	4	2	3	2	21	4	7	3	7	3	6	3	5	3	3	2	3	2	3	2		21	4	7	3	7	3	5	3	5	3	3	2	3	2	3	2	
f1:	64	64	32	4	21	3	16	3	13	3	11	3	9	2	6	2	5	2	6	6	3	3	3	3	3	3	3	3	2	2	2	2	2	2		6	6	3	3	3	3	3	3	3	3	2	2	2	2	2	2	
f2:	35	35	17	4	12	3	9	3	7	2	6	2	5	2	4	2	3	2	15	4	8	3	6	3	5	3	5	3	5	2	4	2	3	2		15	4	8	3	6	3	5	3	5	3	5	2	4	2	3	2	
rd73.pla			70	4	47	3	36	3	29	3	24	3	20	2	14	2	11	2	42	6	18	3	16	3	14	3	13	3	10	2	9	2	8	2	121	42	6	18	3	16	3	13	3	13	3	10	2	9	2	8	2	12
f0:	84	92	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	42	5	11	4	7	3	7	3	7	3	5	3	3	2	3	2		40	5	11	4	7	3	7	3	7	3	5	3	3	2	3	2	
f1:	128	128	64	5	43	4	32	4	26	3	22	3	19	3	12	2	9	2	7	7	4	4	4	4	4	4	4	4	3	3	3	3	2	2		7	7	4	4	4	4	4	4	4	4	3	3	3	3	2	2	
f3:	70	73	35	4	23	4	18	3	14	3	12	3	10	3	7	2	5	2	27	5	11	3	11	3	9	3	8	3	6	3	5	2	5	2		27	5	11	3	11	3	9	3	8	3	6	3	5	2	5	2	
rd84.pla			142	5	95	4	72	4	58	3	49	3	42	3	28	2	21	2	77	7	27	4	23	4	21	4	20	4	15	3	12	3	11	2	301	75	7	27	4	23	4	21	4	20	4	15	3	12	3	11	2	30
f2:	22	31	11	3	7	3	6	2	5	2	4	2	3	2	2	2	2	2	8	4	6	3	5	2	5	2	4	2	3	2	2	2	2	2		6	3	4	3	5	2	5	2	4	2	3	2	2	2	2	2	
f3:	21	33	10	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2	6	3	7	3	5	2	4	2	4	2	3	2	2	2	2	2		5	3	6	3	5	2	4	2	4	2	3	2	2	2	2	2	
sao2.pla			36	3	24	3	19	2	15	2	14	2	11	2	7	2	7	2	29	4	23	3	18	2	15	2	14	2	11	2	7	2	7	2	9	26	3	20	3	18	2	15	2	14	2	11	2	7	2	7	2	1
xor5.pla	16	16	8	3	5	2	4	2	3	2	3	2	3	2	2	2	1	1	4	4	2	2	2	2	2	2	2	2	2	2	2	2	1	1	12	4	4	2	2	2	2	2	2	2	2	2	2	2	2	1	1	1
f3:	18	19	9	3	6	3	5	2	4	2	3	2	3	2	2	2	2	2	9	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2		9	3	3	2	3	2	3	2	3	2	3	2	2	2	2	2	
f4:	14	15	7	3	5	2	4	2	3	2	3	2	2	2	2	2	1	1	7	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1		7	3	3	2	3	2	3	2	3	2	2	2	2	2	1	1	
z5xp1.pla			35	3	26	3	22	2	18	2	16	2	15	2	12	2	11	2	35	3	21	2	19	2	17	2	16	2	15	2	12	2	11	2	9	35	3	21	2	19	2	17	2	16	2	15	2	12	2	11	2	
z9sym.pla	86	148	43	5	29	4	22	3	17	3	15	3	13	3	8	2	6	2	37	5	20	4	17	3	14	3	12	3	7	3	7	3	5	2	34	37	5	20	4	17	3	14	3	12	3	7	3	7	3	5	2	3
f3:	67	83	33	4	22	4	17	3	14	3	11	3	10	3	6	2	5	2	33	4	22	4	17	3	14	3	11	3	10	3	6	2	5	2		32	4	22	4	17	3	14	3	11	3	10	3	6	2	5	2	
f4:	65	82	32	4	22	4	16	3	13	3	11	3	10	3	6	2	5	2	32	4	22	4	16	3	13	3	11	3	10	3	6	2	5	2		29	4	21	4	16	3	13	3	11	3	10	3	6	2	5	2	
f5:	73	89	36	4	24	4	18	3	15	3	12	3	11	3	7	2	5	2	36	4	24	4	18	3	15	3	12	3	11	3	7	2	5	2		36	4	24	4	18	3	15	3	12	3	11	3	7	2	5	2	
f6:	77	92	38	4	26	4	19	3	16	3	13	3	11	3	7	2	6	2	38	4	26	4	19	3	16	3	13	3	11	3	7	2	6	2		38	4	26	4	19	3	16	3	13	3	11	3	7	2	6	2	
f7:	85	102	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	42	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2		41	5	28	4	21	3	17	3	14	3	12	3	8	2	6	2	
apex3.pla			312	5	222	4	175	3	151	3	128	3	119	3	89	2	79	2	312	5	222	4	175	3	151	3	128	3	119	3	89	2	79	2	0	307	5	221	4	175	3	151	3	128	3	119	3	89	2	79	2	
f0:	278	2648	139	6	93	5	70	4	56	4	47	3	40	3	26	3	19	3	139	6	93	5	70	4	56	4	47	3	40	3	26	3	19	3		139	6	93	5	70	4	56	4	47	3	40	3	21	3	19	3	
f1:	264	1227	132	6	88	5	66	4	53	4	44	3	38	3	24	3	18	3	132	6	88	5	66	4	53	4	44	3	38	3	24	3	18	3		119	6	77	5	66	4	53	4	30	4	25	3	24	3	18	3	
f2:	523	1763	261	6	174	5	131	4	105	4	87	4	75	4	48	3	35	3	261	6	174	5	131	4	105	4	87	4	75	4	48	3	35	3		131	6	74	5	57	4	46	4	38	4	23	4	22	3	17	3	
apex2.pla			532	6	355	5	267	4	214	4	178	4	153	4	98	3	72	3	532	6	355	5	267	4	214	4	178	4	153	4	98	3	72	3	0	389	6	244	5	193	4	155	4	115	4	88	4	67	3	54	3	56
f6:	36	48	18	4	12	3	9	3	7	2	6	2	5	2	4	2	3	2	18	4	12	3	8	3	7	2	6	2	5	2	4	2	3	2		18	4	12	3	8	3	7	2	6	2	5	2	4	2	3	2	
f7:	199	297	99	5	66	4	50	4	40	3	33	3	29	3	18	3	14	2	99	5	66	4	50	4	40	3	33	3	29	3	18	3	14	2		55	5	46	4	34	4	28	4	24	3	28	3	18	3	14	2	
alu4.pla			323	5	216	4	163	4	131	3	109	3	94	3	62	3	46	2	323	5	216	4	162	4	131	3	109	3	94	3	62	3	46	2	1	279	5	196	4	146	4	119	4	100	3	93	3	62	3	46	2	10
f0:	143	6809	71	5	48	4	36	4	29	3	24	3	21	3	13	2	10	2	71	5	48	4	36	4	29	3	24	3	21	3	10	5	8	4		71	5	48	4	33	5	29	3	17	3	20	4	6	3	4	3	
f1:	771	3563	385	7	257	5	193	5	154	4	129	4	110	4	70	3	52	3	385	7	198	8	119	8	34	7	30	7	48	6	32	5	14	5		132	7	94	5	69	5	22	6	22	6	32	5	20	4	11	4	
cordic.p			456	7	305	5	229	5	183	4	153	4	131	4	83	3	62	3	456	7	246	8	155	8	63	7	54	7	69	6	42	5	22	5	495	203	7	142	5	102	5	51	6	39	6	52	5	26	4	15	4	97
f1:	8	8	4	2	3	2	2	2	2	2	2	2	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1		2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	
f64:	27	30	13	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	3	9	3	7	3	5	3	5	2	4	2	3	2	2	2		11	5	8	4	6	3	5	3	4	2	4	2	3	2	2	2	
f65:	27	30	13	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	3	9	3	7	3	5	3	5	2	4	2	3	2	2	2		11	5	8	4	6	3	5	3	4	2	4	2	3	2	2	2	
f68:	27	30	13	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	3	9	3	7	3	5	3	5	2	4	2	3	2	2	2		11	5	8	4	6	3	5	3	4	2	4	2	3	2	2	2	
f69:	27	30	13	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	3	9	3	7	3	5	3	5	2	4	2	3	2	2	2		11	5	8	4	6	3	5	3	4	2	4	2	3	2	2	2	
f70:	27	30	13	3	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	3	9	3	7	3	5	3	5	2	4	2	3	2	2	2		11	5	8	4	6	3	5	3	4	2	4	2	3	2	2	2	
f72:	23	23	11	3	8	3	6	2	5	2	4	2	4	2	2	2	2	2	8	3	4	3	3	2	3	2	3	2	3	2	2	2	2	2		8	3	4	3	3	2	3	2	3	2	3	2	2	2	2	2	
f76:	28	29	14	4	9	3	7	3	6	2	5	2	4	2	3	2	2	2	13	4	9	3	7	3	6	2	5	2	4	2	3	2	2	2		11	4	9	3	7	3	6	2	5	2	4	2	3	2	2	2	
f80:	32	39	16	4	11	3	8	3	7	2	6	2	5	2	3	2	3	2	12	6	11	3	8	3	6	4	6	2	5	2	3	2	3	2		12	6	11	3	8	3	6	4	6	2	5	2	3	2	3	2	
f84:	36	37	18	4	12	3	9	3	7	2	6	2	5	2	4	2	3	2	12	6	9	5	9	3	7	2	6	2	5	2	4	2	3	2		12	6	9	5	9	3	7	2	6	2	5	2	4	2	3	2	
apex5.pla			578	4	391	3	308	3	253	2	230	2	215	2	138	2	110	2	562	6	383	5	305	3	245	4	229	2	214	2	138	2	110	2	37	550	6	378	5	300	3	245	4	224	2	214	2	138	2	110	2	6
t481.pla	481	841	240	6	160	5	120	4	96	4	80	4	69	3	44	3	32	3	45	8	41	7	32	7	49	5	47	5	46	5	21	4	14	3	546	43	8	39	7	31	7	47	5	45	5	45	5	21	4	14	3	55
f0:	87	99	43	5	29	4	22	3	18	3	15	3	13	3	8	2	6	2	43	5	27	4	22	3	15	4	15	3	13	3	8	2	5	2		15	4	14	3	13	3	11	3	10	3	10	3	8	2	5	2	
f1:	102	120	51	5	34	4	26	3	21	3	17	3	15	3	10	2	7	2	51	5	28	5	26	3	21	3	17	3	15	3	10	2	6	2		25	4	16	3	15	3	17	3	14	3	12	3	8	2	6	2	
f2:	120	169	60	5	40	4	30	3	24	3	20	3	17	3	11	2	8	2	60	5	39	5	30	3	23	4	20	3	17	3	11	2	8	2		18	4	17	3	16	3	12	3	12	3	9	3	11	2	8	2	
f3:	132	193	66	5	44	4	33	4	27	3	22	3	19	3	12	2	9	2	66	5	44	4	33	4	27	3	22	3	19	3	12	2	9	2		20	4	15	3	12	3	14	3	13	3	12	3	12	2	9	2	
f4:	111	173	55	5	37	4	28	3	22	3	19	3	16	3	10	2	8	2	55	5	37	4	28	3	21	4	17	4	16	3	10	2	8	2		22	4	23	4	17	3	11	3	11	3	8	3	10	2	8	2	
f5:	78	102	39	4	26	4	20	3	16	3	13	3	11	3	7	2	6	2	39	4	26	4	18	3	12	3	10	3	11	3	7	2	6	2		20	4	14	3	13	3	10	3	9	3	8	3	7	2	6	2	
f6:	111	173	55	5	37	4	28	3	22	3	19	3	16	3	10	2	8	2	55	5	37	4	28	3	21	4	19	3	16	3	10	2	5	2		24	4	26	4	20	3	12	3	11	3	8	3	10	2	5	2	
f7:	141	198	70	5	47	4	35	4	28	3	24	3	20	3	13	2	10	2	70	5	47	4	35	4	28	3	24	3	20	3	13	2	10	2		17	4	16	3	14	3	13	3	11	3	10	3	11	2	10	2	
f8:	70	99	35	4	23	4	18	3	14	3	12	3	10	3	7	2	5	2	35	4	23	4	14	4	12	4	10	3	10	3	4	2	4	2		14	4	10	3	11	3	8	3	8	3	8	3	4	2	4	2	
f9:	113	159	56	5	38	4	28	3	23	3	19	3	16	3	11	2	8	2	56	5	38	4	28	3	23	3	19	3	16	3	11	2	8	2		29	5	21	4	16	4	15	3	12	3	16	3	10	3	8	2	
misex3.pla			612	5	410	4	309	4	249	3	208	3	178	3	117	2	87	2	612	5	401	5	303	4	237	4	201	4	178	3	114	2	81	2	43	286	5	227	4	188	4	157	3	139	3	126	3	109	3	81	2	85
			3590		2426		1858		1506		1286		1134		751		587		3246		2088		1591		1255		1097		997		661		508			2453		1666		1327		1089		941		861		609		483		
				75		75		61		52		48		47		45		38		85		85		66		59		54		53		49		44			83		83		61		56		53		51		48		44	

this algorithm was applied separately to the outputs. The left part of the table shows the results of the synthesis performed on the benchmarks using the classical approach. The column marked with "Esp" lists the number of function products after Espresso minimization, "Bdd" lists the number of paths in the ROBDD diagram representing the function, the letter "B" list the numbers of k-product PAL-based blocks, and the columns marked with the letter "L" – lists the numbers of logic levels. The second part of the table, denoted with the heading "decBDD", contains the results obtained using the new method. In the set of 1730 cases compared, the proposed decBDD algorithm allowed to find 153 solutions, whilst the decBDD+E algorithm 254 allowed to find solutions, requiring a smaller number of logic blocks than that in the classical method. For some benchmarks, the reduction of the logic block count was significant, e.g. for rd84_f1, rd73_f1, misex_f7, cordic_f1, apex2_f2, 5xp1_f2. Significant differences can be noticed not only for small values of *k*. Unfortunately, the number of logic levels does not follow the reduction of the number of logic blocks. Among the examined benchmarks, only a few percent solutions demanded a smaller number of logic levels. The results of experiments are presented in a synthetic way in Figs. 12 and 13. The values represented on the axis of ordinates in Fig. 12 were calculated from the rational formula shown in the graph. Σblocks classical and Σblocks decBDD denote the relevant total sums of block counts obtained using the corresponding synthesis methods and presented in Table 1. The values represented in Fig. 13 were calculated in a similar manner. Analysis of the benchmarks allows us to state, that in most cases, reduction of logic block counts by using this new algorithm is obtained at the expense of a certain expansion of logic levels. The proposed method is particularly efficient, if $k = 5$ a significant reduction of block counts was observed, while preserving a comparable number of logic levels.

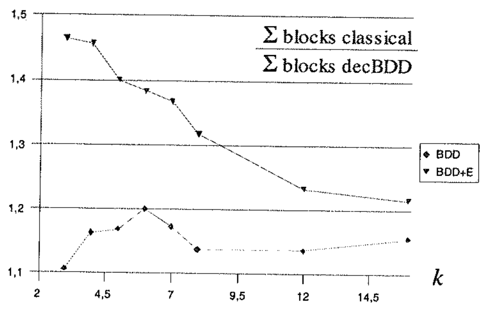


Fig. 12. A comparison of two algorithms with the classical method with respect to the number of logic blocks (see text)

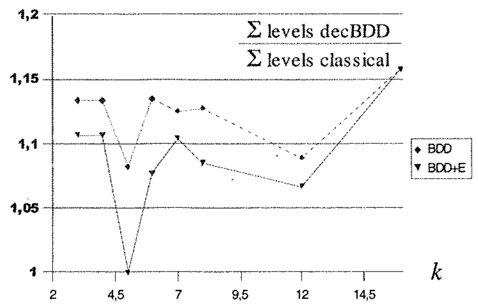


Fig. 13. A comparison of two algorithms with the classical method with respect to the number of logic levels (see text)

6. CONCLUSIONS

The paper presents a BDD-based PAL-oriented decomposition method (two variants) dedicated for PAL-based CPLDs. The essence of the methods is to incorporate a decomposition into the synthesis process dedicated for CPLD structures. The algorithm consists in sequential search for a decomposition providing feasibility of implementation of the free block in one PAL-based logic block containing a predefined number of product terms. The proposed method is an alternative to the classical approach based on two-level minimization of individual single-output functions.

The proposed methods were practically proved. The results of experiments presented in the paper for two synthesis methods, closes to each other with growing k. Conclusion is that for large k, it is better to use decBDD which works faster and gives comparable results.

Through the adjustment of the decomposition elements to the logical resources characteristic for a PAL-based logic block, significant improvement of the synthesis effectiveness in relation to the classical approach could be obtained.

7. REFERENCES

1. M. Bolton: *Digital Systems Design with Programmable Logic.* Addison-Wesley Publishing Company, 1990
2. R.K. Brayton, G.D. Hachtel, C. McMullen, A.L. Sangiovanni-Vincentelli: *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, 1984.
3. 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, 1996, pp. 1226-1235.
4. J-D. Huang, J-Y. Jou, W-Z. Shen: ALTO: *An Iterative Area/Performance Tradeoff Algorithm for LUT-Based FPGA Technology Mapping*, IEEE Transactions on Very Large Integration (VLSI) Systems, Vol. 8, No.4, pp. 392-400.
5. Y. Lai, K. Pan, M. Pedram: *FPGA synthesis using function decomposition*, Proceedings of the IEEE International Conference on Computer Design, Cambridge, 1994, pp. 30-35.

1. 6. M.-T. L a i, K.-R. R. P a n, M. P e d r a m: *OBDD-Based Function Decomposition: Algorithms and Implementation*, IEEE Transactions on Computer-Aided Ddesign of Integrated Circuits and Systems, 1996, Vol. 15, No. 8, pp. 977-990.
7. C. Y a n g, M. C i e s i e l s k i: *BDS: A BBD-Based Logic Optimization System, IEEE Transactions on CAD of Integrated circuits and systems*, Vol.21, No.7, 2002, pp. 866-876.
8. J. H. A n d e r s o n, S. D. B r o w n: *Technology mapping for large complex PLDs*, Proceedings of Design Automation Conference, DAC'98, 1998, pp. 698 -703.
9. S h i h - L i a n g C h e n, T i n g T i n g H w a n g, C. L. L i u: *A technology mapping algorithm for CPLD architectures*, IEEE International Conference on Field Programmable Technology, Hong Kong, 2002, pp. 204-210.
10. J a e j i n K i m, S a n g z o o n B y u n, H i s e o k K i m: *Development of technology mapping algorithm for CPLD under time constraint,* 6th International Conference on VLSI and CAD, ICVC '99, 1999, pp. 411-414.
11. H i - S e o k K i m, J a e - J i n K i m, C h i - H o L i n: *An efficient CPLD technology mapping under the time constraint*, Proceedings of the 12th International Conference on Microelectronics, ICM 2000, 2000, pp. 265 -268.
12. J. L. K o u l o h e r i s, A. E l G a m a l: *PLA-based FPGA Area Versus Cell C+ Granularity*, Proceedings of the IEEE Custom Integrated Circuits Conference, 1992, pp. 4.3.1-4.3.4.
13. K. Y a n: *Logic synthesis for CPLDs and FPGAs with PLA-style logic blocks*, Fourteenth In-ternational Conference on VLSI Design, 2001, pp. 291-297.
14. K. Y a n: *Practical logic synthesis for CPLDs and FPGAs with PLA-style logic blocks*, Pro-ceedings of the Asia and South Pacific Design Automation Conference, ASP-DAC 2001, 2001, pp. 231-234.
15. R. E. B r y a n t: *Graph Based Algorithms for Boolean Function Manipulation*, IEEE Transac-tions on Computers, Vol.C-35, No.8, 1986, pp. 677-691.
16. 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, Digital System Design*, Proceedings. 8th Euromicro Conference, 2005, pp. 114-121.
17. S. B. A k e r s: *Functional Testing with Binary Decision Diagrams*, Eighth Annual Conference on Fault-Tolerant Computing, 1978, pp. 75-82.
18. K. S. B r a c e, R. L. R u d e l l, R. E. B r y a n t: *Efficient implementation of a BDD package*, 27th ACM/IEEE Design Automation Conference, 1990, pp. 40-45.
19. S. M i n a t o: *Binary Decision Diagrams and Applications for VLSI CAD*. Kluwer Academic Publi-shers, 1996.
20. R. E b e n d, G. F e y, R. D r e c h s l e r: *Advanced BDD Optimization*. Springer, Dordrecht, 2005.
21. R. R u d e l l: *Dynamic variable ordering for ordered decision diagram*, IEEE International Con-ference on Computer-Aided Design, 1993, pp. 42-47.

An art
racters

Basic
The ar
Pared
cessor
Layou
• Title
• Auth
• Worl
• Conc
• Mair

Pages s

Main t
Main te
tal leter

Tables
Tables
with do
mbers o
should

Mathe
Charact
above
the basi
Formul
symbol
of Stan

Referer
Referen
rences p
1. F.
2. K.
3. Y.

Figures
Figures
tor – Co
wing an
speared

INFORMATION FOR AUTHORS

An article published in other magazines can not be submitted for publishing in E.T.Q. The size of an article 1800 characters 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 print should be prePared in English. Floppy disc or a Cd 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

Pages should have continuous numbering.

Main text
Main text cannot contain formatting such as spacing, underlining, words written in capital letters (except words in capital leters). Author can mark suggested formatting with pencil on the margin of the article using com monly marks.

Tables
Tables with their titles should be placed on a separate page at the end of the article. Titles of rows and columns letter with double line spacing. Annotations concerning tables should be placed directly below the table. Table Arabic numbers on the top each table. Table can contain algorithm and program listings. In such cases original preserved. Table should be cited in the text.

Mathematical formulas
Characters, numbers, letters and spacing of the formula should be adequate to layout of the article. Indexes reised above
the basic line and clearly written. Special characters such as lines, arrows, dots should be place they are attributed to. Formulas should be numbered with Arabic numbers placed in brackets on the right side measure, letters and graphic symbols should be printed according to requirements of IEC (International Electronical) (International Organisation of Standardisation).

References
References should be placed at themain text with the subtitle "References", References should be numbered to references placed in the text. Examples of periodical [1], 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. Andersen: A resource allocation framework. XVI International Symposium, Stockholm (Sweden),
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 (editor – CorelDRAW). Photos or diapositives will be accepted in black and white format not greater than 10×15 cm drawing and on the back of each photo author name and abbreviation of the article title should be placed. Figure's on a speared page. Figure should be cited in the text.

Additional information

On a separate page following information should be placed:

- mailing address (home or office)
- phone (home and/or office)
- e-mail

Authors in entitled to free of charge 20 copies of article. Additional copies or the whole magazine can be order expense. Author is obliged to perform the authors correction, which should be accomplished within 3 days starting from from the editorial staff. Corrected text should be returned to the editorial staff personally or by mail. Corrected the margin of copies received from the editorial staff or if needed on separate pages. In the case when the correction said time limit, correction will be performed by technical editorial staff of the publisher.

In case of changing of workplace or home address authors are asked to inform the editorial staff.

FORMULARZ ZAMÓWIENIA

Imię, nazwisko/ nazwa Firmy..

Adres..

NIP......................................

Zamawiam prenumeratę następujących tytułów, których dystrybucję prowadzi WDN PAN w Warszawie, ul. Śniadeckich 8, 00–656 Warszawa

Lp.	Tytuł	Cena 1egz. w PLN	Cena za 4 umery w 2009 roku w PLN	Zamawiam egz.
1.	Electronics and Telecomunications Quarterly 1-4/ 2009 Kwartalnik Elektroniki i Telekomunikacji 1-4/ 2009	40,00	160,00	

☐

Jestem zainteresowany zakupem następujących numerów

....

☐

Tak, jestem zainteresowany otrzymaniem wiadomości o kolejnych tytułach oraz o szczegółowej zawartości poszczególnych tytułów.
Faxem, numer..
e-mailem, adres..

Do faktury doliczamy koszty wysyłki.

Zamówienia przyjmujemy drogą elektroniczną, faxem lub pocztą na poniższy adres:
Warszawska Drukarnia Naukowa Polskiej Akademii Nauk
00–656 Warszawa ul. Śniadeckich 8,
tel/fax 022 628-76-14, 022 628-87-77
e-mail: **wdnpan@wdnpan.pl, dystrybucja@wdnpan.pl**

Zapraszamy również do naszego sklepu internetowego **www.publikacje-naukowe.pl,** gdzie mogą Państwo zapoznać się z innymi publikacjami Polskiej Akademii Nauk.

Płatność na konto:
Bank Zachodni WBK S.A. 94 1090 1883 0000 0001 0588 2816
za pobraniem TAK / NIE (odpowiednie podkreśl)