Reducing Average and Peak Temperatures of VLSI CMOS Digital Circuits by Means of Heuristic Scheduling Algorithm

Władysław Szczesniak
Faculty of Electronics, Telecommunications and Informatics
Gdansk University of Technology, 80-952 Gdansk, Poland

Abstract—This paper presents a BPD (Balanced Power Dissipation) heuristic scheduling algorithm applied to VLSI CMOS digital circuits/systems in order to reduce the global computational demand and provide balanced power dissipation of computational units of the designed digital VLSI CMOS system during the task assignment stage. It results in reduction of the average and peak temperatures of VLSI CMOS digital circuits. The elaborated algorithm is based on balanced power dissipation of local computational (processing) units and does not deteriorate the throughput of the whole VLSI CMOS digital system.

I. INTRODUCTION

This paper concerns application of heuristic scheduling algorithm to balance the load of tasks onto computation units (cu) uniformly with reduction of the total cost of the digital VLSI CMOS system (co) but without deteriorating its global computational efficiency measured e.g. by its throughput (Th). In this paper, the universal measure, named the cost (co) represents the consumption of power supply of VLSI CMOS digital system.

In the literature we can find a variety of methods concerning computational task assignment to different computation units (cu) [1], [2], [4], [5], [11]. It is especially important for reducing the supply power demand [6], [7], [8], [9]. In the paper the design objective function taken into account is the cost of the system (co). This measure has the straightforward influence on average and peak temperature of IC.

The presented BPD (Balanced Power Dissipation) algorithm can be applied to reduce global computational demand and provides balanced power dissipation of the digital VLSI CMOS system at the task assignment stage.

II. PROBLEM FORMULATION

The task to be computed is described by the tasks’ graph $G_T(V,E)$ as presented in Fig.1. An individual computational task (ct) in graph $G_T$ is represented by vertex $v_i$ in the set of vertices (e.g. in Fig. 1 ☼, ▲, ●). Each ct has to be assigned to one of the computational units of a given resource type $cu_tj \in \{cu_tj\}$ in the proper discrete time (dtu).

The value $|\{cu_tj\}|$ is equal to the number of computational units of j-type available during design process, e.g. for the graph in Fig.1 $\{cu_tj\}=\{☼,▲,●\}$. Each edge $e_{ij} \in E$ represents a data set. The process of assigning $v_i \in V$ onto $cu_tj$ is constrained by a limited set of resources $(\{cu_{tj}\} \cup \ldots \cup \{cu_{tj}\} \cup \ldots)$ and the maximal number of time slots $dtu_M$.

Each type of processing unit is capable of processing a specified type of computational tasks. The problem of assigning tasks to processors as specified here is NP complete [3].

While assigning $ct_i$ to $cu_{tj}$ in BPD algorithm, different
values of throughput ($T_h$) of each computational unit $cu_t$ are taken into consideration. The function describing normalized throughput ($Thn = T_h / \min_j T_h$) of different computational units versus their costs is shown in Fig. 2, where $0 < co_1 < co_2 < co_3 < co_4$. The appropriate cost of each computational unit characterized by the given throughput varies from $co_{low}$ to $co_{high}$. The number of different levels of $Thn$ results from the set of the computational units available for the designed system.

The main aim of the elaborated BPD algorithm is fulfilling the condition of providing the balanced power dissipation of computational units ($\cup cu_t \cup ... \cup cu_t \cup ...$) leading to reduction of average and peak temperatures of the digital VLSI CMOS system without deteriorating its throughput. In this process the total cost of the system is also decreased. The assumption is that less effective computational units are cheaper, so that replacing the chosen $cu_t$ with other, less effective, results in decreasing the total system cost.

### IV. ALGORITHM DESCRIPTION

The elaborated BPD heuristic algorithm is partially based on the research results presented in papers [9] and [10] and concerned with reducing the power consumption of digital CMOS circuits. The algorithm consists of two stages described below. At the first stage of the algorithm the computational tasks represented by graph $G_T$ have to be assigned to the elements of resources with the lowest number of discrete time units taken into account. During this stage, either ASAP or ALAP [8] algorithm is executed.

Scheduled computational task for the example in Fig.1 after the first stage of BPD algorithm is shown in Fig.3.

The second stage of BPD algorithm (Tab.I) consists of delaying the initial $SG$ created in the first stage. The $C_{su}$ set includes all the rows suitable for delaying, and $C_{ms}$ indicates the most suitable row selected from this set. The chosen $C_{ms}$ row actually undergoes the process of delaying.

#### TABLE I

**PSEUDOCODE OF THE SECOND STAGE OF BPD ALGORITHM**

| Line | Description |
|------|-------------|
| 1.   | $C_{ms} = SG$ |
| 2.   | **while** ($|C_{ms}| > 0$) |
| 3.   | **foreach** ($C_k \in C_{su}$) |
| 4.   |  if (! fsc_fulfilled($C_k$)) |
| 5.   | $C_{su} = C_{su} \setminus C_k$ |
| 6.   |  if ($|C_{su}| == 0$) |
| 7.   | **break** |
| 8.   |  if ($|C_{su}| > 1$) |
| 9.   | **foreach** ($C_k \in C_{su}$) |
| 10.  |  if (there_is_same_cu($C_k$)) |
| 11.  | $C_k = row_of_the_same_cu(C_k)$ |
| 12.  | **foreach** ($C_k \in C_{su}$) |
| 13.  |  if (fvi($v_i$) > fvi($v_j$)) |
| 14.  |    interchange($v_i$, $v_j$) |
| 15.  | $C_{ms} = \arg\max_{C_k \in C_{su}} fck(C_k)$ |
| 16.  |  else |
| 17.  | $C_{ms} = HEAD(C_{su})$ |
| 18.  | $SG_{backup} = SG$ |
| 19.  | **foreach** ($v_i \in C_{ms}$) |
| 20.  |    insert_idle_tasks($v_i$) |
| 21.  |  if (! delay_all_successors($v_i$)) |
| 22.  |    $SG = SG_{backup}$ |
| 23.  |    $C_{su} = C_{su} \setminus C_{ms}$ |
| 24.  | **break** |

#### fsc_fulfilled($C_k$) (line 4)

This function performs the check for the free space condition ($fsc$), defined by the following formula:

$$ n_i \cdot l_i \leq r_p $$

where $n_i$ is the number of $dtu$, $l_i$ is the number of tasks assigned to $C_k$ computational unit row, $r_p$ is the number of free task slots after the first occurrence of a task in $C_k$ computational unit row.

Formula (1) checks if the number of free $dtu$ slots is sufficient for the idle tasks to introduce longer processing time of a $cu_t$. Every $C_k$ selected for the increased number of $dtu$ has to fulfill condition (1). Despite the fact, that cascading tasks from the other $C_k$‘s are not taken into consideration while calculating $fsc$, condition (1) is sufficient to pre-reject quickly some $C_k$ from the $C_{su}$ set before starting the time-consuming delaying process.

#### there_is_same_cu($C_k$) (line 10)

This function simply indicates whether there is another $cu_t$ of exactly the same type as the one assigned to $C_k$, i.e. being capable of performing the same type of task in the same time.

#### row_of_the_same_cu($C_k$) (line 11)
This function returns a row containing the tasks of exactly the same type as \( C_k \).

\[ f_{vi}(v_i) \] (line 16)

This function calculates \( fvi \) factor for the \( v_i \) task according to the following formula:

\[
f_{vi} = \frac{i_{vi} + a_{vi} + (f_{vi} - s_{vi} \cdot n_i)}{p_i + 1}
\]  

where \( i_{vi} \) is the number of independent inputs of task \( v_i \), \( a_{vi} \) is the number of system outputs of task \( v_i \), \( f_{vi} = dtu_{m} - (c_{si} + n_i) \), \( dtu_{m} \) is the maximal number of \( dtu \) admissible, \( c_{si} \) is the number of \( dtu \) which \( v_i \) is assigned to, \( s_{vi} \) is the number of tasks of the same type as task \( v_i \) in the path of \( G_T \) below task \( v_i \), \( n_i \) is the number of \( dtu \) needed to perform the task, \( p_i \) is the \( p \) - label of task \( v_i \), the minimal \( p \) - label of a task equals 0, hence addition of \( 1 \) in the denominator is necessary to avoid dividing by 0.

\[ \text{interchange}(v_i, v_j) \] (line 14)

This function swaps the \( v_i \) and \( v_j \) tasks, so that \( v_i \) is located in task slots earlier occupied by \( v_j \) and vice versa.

\[ f_{ck}(C_k) \] (line 15)

The value of the function is given by:

\[
f_{ck} = \frac{P_{d_{ck}} \cdot l_{ck}}{n_dC_{k}P} \cdot l_{C_{k}}
\]  

where \( P_{d_{ck}} \) is the normalised computational load of \( C_k \) computational unit row (\( \text{cu-assigned to } C_k \) row, \( P_{d_{ck}} \) is normalised to \( cu_{ti} \), having the lowest value of \( P_d \)), \( l_{ck} \) is a number of tasks in \( C_k \) computational unit row.

The \( f_{ck} \) function is responsible for selecting the most suitable row \( (C_{ms}) \) for inserting idle tasks, from the \( C_{su} \) set. It chooses the row assigned to \( cu_{ti} \) that has the highest throughput demand, hence it gives the highest throughput demand reduction when the processing element \( p_{y}^{*} \) assigned to row \( C_k \) is slowed down.

\[ \text{insert_idle_tasks}(v_i) \] (line 20)

This function simply adds new task slots with idle tasks after the \( v_i \) task. If there is an empty task slot after the last \( dtu \) occupied by \( v_i \), then an idle task is added there. However, when there is no empty room for a new idle task, then the next task in the row of \( v_i \) is delayed. Next the data interconnections between \( v_i \) and its successor tasks must be checked. This is done by the \( \text{delay_all_successors} \) function described below.

\[ \text{delay_all_successors}(v_i) \] (line 21)

This function checks if all the data needed to perform successor task \( (s_i) \) of \( v_i \) is available on time, by checking the condition:

\[
\text{end}_dtu(v_i) \leq \text{start}_step(s_i)
\]  

If it is not fulfilled, then the successor is delayed as many \( dtu \) as needed, so that \( \text{start}_step(s_i) = \text{end}_dtu(v_i) \).

Such a delay implies the need for checking all the sets of data and interconnections between the successors of \( s_i \). If the delay is not possible due to the \( dtu_{m} \) constraint, increasing the number of \( dtu \) of \( v_i \) (and the computational unit it is assigned to) fails.

In such a case the row containing \( v_i \), i.e. \( C_{ms} \) is removed from the \( C_{su} \) set \( F \), and the process starts from the beginning with the decreased \( |C_{su}| \).

For a simple benchmark shown in Fig. 1, the results are presented in details. The obtained results are shown in Fig. 3 in a form of scheduling graphs \( SG \) for the first stage of the
algorithm, while the second stage is given in Fig. 4.

There in Fig. 4 lowering the cost of the appropriate computational units is represented by inserting the symbol ◊. It means that its throughput can be twice as low without deteriorating the efficiency of the whole computational system. Therefore our example for computational units cu_a_2 and cu_c show that their throughput can be lowered twice resulting in cost reduction of the designed computational system. Moreover, the value of the throughput obtained earlier does not deteriorate.

IV. EXPERIMENTAL RESULTS

This section presents the results obtained by applying the BPD algorithm on selected benchmarks [12].

Cost reduction is calculated for each computational task based on the number of computational units of each type before and after application of BPD algorithm. The cost of each computational unit type assumed for cost calculation is directly proportional to its throughput. To simplify the comparison of computational efficiency we assume that $T_{hu}$ can be lowered by the factor 0.5. Table II presents the assumed normalized cost due to the throughput of each computational unit type.

| cu_a | cu_b | cu_c | cu_d | cu_e |
|------|------|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 | 1 |
| $c_{298}$ | 17 | 5 | 1 | 10 |
| $c_{5315}$ | 158 | 56 | 1 | 68 |
| $c_{382}$ | 4 | 9 | 1 | 7 |
| $c_{444}$ | 4 | 17 | 1 | 7 |
| $c_{526}$ | 32 | 9 | 1 | 14 |
| $c_{5378}$ | 3 | 11 | 3 | 11 | 4 |

Tables III and IV and Fig. 5 show number of computational units of each type before and after applying BPD algorithm, respectively.

| cu_a | cu_b | cu_c | cu_d | cu_e |
|------|------|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 | 1 |
| $s_{298}$ | 3 | 13 | 3 | 2 |
| $s_{5315}$ | 100 | 12 | 38 | 53 |
| $s_{382}$ | 3 | 1 | 1 | 7 |
| $s_{444}$ | 2 | 3 | 1 | 7 |
| $s_{526}$ | 6 | 18 | 8 | 3 |
| $s_{5378}$ | 93 | 1 | 4 | 21 |

| cu_d | cu_e |
|------|------|
| $s_{298}$ | 23 | 2 |
| $s_{5315}$ | 130 | 14 |
| $s_{382}$ | 15 | 9 |
| $s_{444}$ | 8 | 18 |
| $s_{526}$ | 7 | 6 |
| $s_{5378}$ | 285 | 20 |

TABLE II

THE ASSUMED NORMALIZED COST DUE TO THE THROUGHPUT OF EACH COMPUTATIONAL UNIT TYPE

| cu_a | cu_b | cu_c | cu_d | cu_e |
|------|------|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 | 1 |
| $c_{298}$ | 17 | 5 | 1 | 10 |
| $c_{5315}$ | 158 | 56 | 1 | 68 |
| $c_{382}$ | 4 | 9 | 1 | 7 |
| $c_{444}$ | 4 | 17 | 1 | 7 |

TABLE III

THE ASSUMED NORMALIZED POWER DISSIPATION DUE TO THE THROUGHPUT OF EACH COMPUTATIONAL UNIT TYPE

| cu_a | cu_b | cu_c | cu_d | cu_e |
|------|------|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 | 1 |
| $s_{298}$ | 7 | 31 |
| $s_{5315}$ | 7 | 169 |
| $s_{382}$ | 6 | 30 |
| $s_{444}$ | 8 | 30 |
| $s_{526}$ | 14 | 38 |
| $s_{5378}$ | 119 | 333 |

TABLE IV

NUMBER OF COMPUTATIONAL UNITS OF EACH TYPE BEFORE APPLYING BPD ALGORITHM

| cu_a | cu_b | cu_c |
|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 |
| $s_{298}$ | 17 | 5 |
| $c_{5315}$ | 158 | 56 |
| $s_{382}$ | 4 |
| $s_{444}$ | 4 |
| $s_{526}$ | 32 |
| $s_{5378}$ | 3 |

TABLE V

NUMBER OF COMPUTATIONAL UNITS OF EACH TYPE AFTER APPLYING BPD ALGORITHM

| cu_a | cu_b | cu_c |
|------|------|------|
| $T_{hu}$ | 8 | 4 | 2 |
| $s_{298}$ | 23 | 2 |
| $c_{5315}$ | 130 | 14 |
| $s_{382}$ | 15 | 9 |
| $s_{444}$ | 8 |
| $s_{526}$ | 7 |
| $s_{5378}$ | 93 | 1 | 4 | 21 | 285 | 20 | 4 | 24 |
Fig. 5. Percentage of computational units of each type after applying BPD algorithm for s298 (a), c5315 (b), s382 (c), s444 (d), s526 (e), s5378 (f).
TABLE VI

| Benchmark name | Number of graph vertices | Cost reduction % |
|----------------|--------------------------|-----------------|
| s298           | 119                      | 31.25           |
| c5315          | 1994                     | 17.66           |
| s382           | 158                      | 23.44           |
| s444           | 181                      | 26.52           |
| s526           | 193                      | 42.87           |
| s5378          | 2779                     | 13.15           |

The resulting cost reduction together with the number of the $G_T$ graph vertices of each benchmark computational task is reported in Table VI.

V. CONCLUSIONS

In this paper the BPD heuristic scheduling algorithm for load balanced power dissipation resulting in reduction of average and peak temperatures of the digital VLSI CMOS digital circuits/systems was presented. The objective function introduced is measured by cost reduction of VLSI CMOS digital circuits which directly depends on power dissipated in IC. The main idea of BPD algorithm is based on decreasing the cost of chosen computational units by adjusting their efficiency to real needs without deterioration of the computational efficiency of the whole system.

The applied BPD algorithm has been verified for the chosen set of benchmarks. Experimental results proved 13 to 43 per cent cost reduction of the computing system achieved without deterioration of the system throughput with the assumed cost to throughput dependency. This reduction has a straightforward influence on decreasing the average and peak temperatures of VLSI CMOS system and results in increasing its reliability.

ACKNOWLEDGMENT

The work was partially supported by the State Committee for Scientific Research (MNiI, Poland) through the grant 3 T11B 015 27.

REFERENCES

[1] L. Benini, A. Boglio, and G. De Micheli, “A survey of design techniques for system-level dynamic power management,” IEEE Transactions on Very Large Scale of Intergartion (VLSI) Systems, 8, 2000, pp. 287-298.

[2] S. Borkar, “Low power design challenges for the decade,” Proc. Design Automation Conf. ASP-DAC, Asia and South Pacific, 2001, pp. 293-296.

[3] J. Blaziewicz, K. Ecker, E. Pesch, G. Schmidt, and J. Weglarz, Scheduling computer and manufacturing processes, Springer, Heidelberg, 1996.

[4] P. Capello, and D. Mourlouks, “A scalable, robust network for parallel computing,” Proceedings of the 2001 joint ACM-ISCOPE conference on Java Grande, pp. 78-86, June 2001.

[5] V. Karamcheti, and A. A. Chien, “A hierarchical load-balancing framework for dynamic multithreaded computations,” Proceedings of the 1998 ACM/IEEE conference on Supercomputing, 1998, (CDROM).

[6] L. Kruse, et al., “Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs,” IEEE Trans. on Very Large Scale Integration Systems, Vol. 9, 1, 2001, pp. 3-14.

[7] W-Ch. Kwon, and T. Kim, “Low-power embedded system design: Optimal voltage allocation techniques for dynamically variable voltage processors,” Proceedings of the 40th conference on Design automation, pp.125-130, June 2003.

[8] P. Michael, U. Lautcher, and P. Duzy, The synthesis approach to digital system design, Kluwer Academic Publisher, Dordrecht, 1992.

[9] W. Szczęśniak, B. Voss, M. Theisen, J Becker, and M. Glesner, “Influence of high - level synthesis on average and peak temperatures of CMOS circuits,” Microelectronics Journal 32 2001 pp.855-862.

[10] W. Szczęśniak, and P. Szczęśniak, “Low lower digital CMOS VLSI circuit design with different heuristic algorithms,” Proceedings of 2nd IEEE Int. conference on Circuits and Systems for Communications, ICCSC 2004 (CDROM).

[11] Y.-J. Yeh, S.-Y. Kuo, and J.-Y. Jou, “Converter-free multiple-voltage scaling techniques for low-power CMOS digital design,” IEEE Trans. CAD Int. Circuits Syst., 20, 2001, pp.172-176.

[12] Collaborative Benchmarking Laboratory, ftp.cbl.ncsu.edu