LOW POWER DIGITAL CMOS VLSI CIRCUITS DESIGN WITH DIFFERENT HEURISTIC ALGORITHMS

Wladyslaw Szczesniak,
Faculty of Electronics, Telecommunications, and Informatics, Gdansk University of Technology, Gdansk, Poland
wlad@ue.eti.pg.gda.pl

Piotr Szczesniak,
Faculty of Electronics, Telecommunications, and Informatics, Gdansk University of Technology, Gdansk, Poland;
R&D Marine Technology Centre, Gdynia, Poland
piotr@ue.eti.pg.gda.pl

DOI: 10.36724/2664-066X-2021-7-4-30-34

ABSTRACT

The growing demand for portable computing devices leads to new electronic systems fulfilling the requirements for the low power dissipation in the chip. Although, reduction of supply voltage is one of the most effective techniques of decreasing the power consumption in digital CMOS VLSI circuits it results in chip throughput degradation. This paper presents three versions of the Inserting Idle Operation with Interchanging heuristic algorithm, namely simple IIOI, MAximal RELativity (MAREL) and UNIform LOad (UNILO). They are applied to the high-level synthesis of CMOS VLSI circuits with power reduction. Comparison of the obtained results for the chosen set of benchmarks show the different levels of power reduction obtained by different algorithms applied. For different benchmarks the power reduction reaches up to 15%, and up to 68% with extending latency by 50%.

KEYWORDS: *Low power design, VLSI digital circuits, high level synthesis and low power design, heuristic algorithms.*

The article is reworked from unpublished 2nd IEEE International Conference on Circuits and Systems for Communications (ICCSC) materials.

INTRODUCTION

The growing demand for portable computing devices leads to new electronic systems fulfilling the requirements for the low power dissipation in the chip. Although, reduction of supply voltage is one of the most effective techniques of decreasing the power consumption in digital CMOS VLSI circuits it results in chip throughput degradation [1], [2], [3].

The main part of the power dissipated in the CMOS circuit (single functional unit fu_i) is dynamic power represented by:

$$P_{d_i} = \frac{a_i}{2} C_{Li} V_{dd_i}^2 f_{clk} \tag{1}$$

where ai is the activity factor (i.e. the probability of a power consuming transition) of the functional unit fu_i, C_{Li} is the load capacitance, V_{ddi} is the supply voltage, and f_{clk} is the clock frequency.

By decreasing the voltage swing for a given load capacitance of the chosen functional units, we can reduce power consumption quadratically as in (1) but their toggling time (T_{ti}) will be longer and is defined as [1], [3]:

$$T_{ti} \cong \frac{C_L V_{ddi}}{K\left(V_{ddi} - V_t\right)^{\alpha}} \tag{2}$$

where K is a technology constant, V_t is a threshold voltage and α is a constant depending on device technology ($1<\alpha< 2$). The toggling time (T_{ti}) defines the delay time (tdi) of the functional unit f_{ui}.

Figure 1. shows the dependency of supply voltage (V_{dd}) on normalized delay times for an inverter as a functional unit.

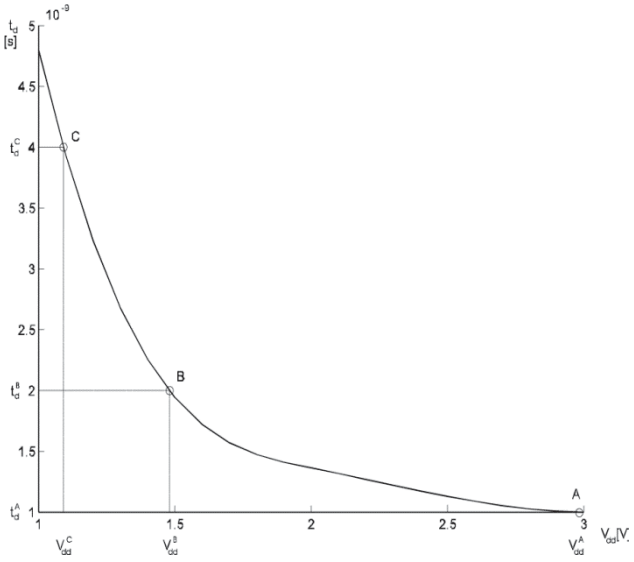

Fig. 1. Influence of the supply voltage (V_{dd}) on the nominal delay (t_d) of CMOS inverter with a load of 0.1pF for the Alcatel Mietec 0.35μm technology (V_t = 0.66V) [3]

For this example the relation between the nominal delays t_d (see Fig. 1) for points A $(V_{dd}^A = 3.0V)$ and B $(V_{dd}^B = 1,48V)$ is given by:

$$t_d^A = \frac{1}{2} t_d^B \tag{3}$$

In fact, to reduce the power dissipated in an electronic CMOS circuit, the supply (V_{dd}) and the threshold (V_t) voltages should be tuned according to the system activity requirements [1], so the maximal toggling time of the circuit is expressed by (2).

It leads to a lower value of the supply voltage Vdd that results in reducing the dynamic power consumption ($P_{di} \sim V^2_{ddi}$).

This paper presents the comparison of the results of three versions of Inserting Idle Operations with Interchanging (IIOI) heuristic algorithm, namely raw IIOI, MAximal RELativity (MAREL) and UNIform LOad (UNILO) applied to the high-level synthesis (HLS) of CMOS VLSI circuits with power reduction.

All of them utilize as soon as possible (ASAP) algorithm. All three presented algorithms lead to decreasing the supply voltage of the chosen functional units (from the constrained set of resources) without degradation of the chip throughput.

PROBLEM FORMULATION

Our task is to minimise the dynamic power (P_d) of CMOS circuits/systems during the HLS using three heuristic algorithms.

In HLS the behavioural specifications of digital systems are mapped to structural designs at the register transfer level (RTL). The behavioural specifications consist of algorithms which describe the behaviour of circuit outputs in terms of circuit inputs and timing constraints [1], [2], [3].

A computational task to be scheduled on the set of functional units $\{fu_i\}$ (resources) can be modelled by a data flow graph DFG(V,E) where a set of vertices V ($|V| = n$) represents operations of the program and a set of edges E represents the data (variables and constants) [2], [4], [6].

For the given DFG the result of the HLS can be represented by the scheduling graph (SG) in which each vertex of DFG is assigned to the appropriate element of the set of resources $\{fu_i\}$ and a control step(s) (CS) in such a way that all data dependencies are satisfied. Each column of SG corresponds to one element fu_i of the set of resources and each row to the successive control step CS.

The HLS task is constrained by a limited set of resources ({fui}) e.g. multipliers (MULT), adders (ADD), subtractors (SUB), gates, etc., and an acceptable throughput (described by the maximal number of control steps CSMAX) [3].

If it is possible to introduce slacks for chosen resources without decreasing the throughput of the whole system, then they can be supplied with a lower V_{dd}, which results in reducing power consumption. Note that in some applications, especially for systems that have different working modes, CSMAX can be extended. In the experimental part of the paper we have considered CSMAX extension by 10, 20 and 50 percent.

According to the above discussion, we can formulate the problem to be solved as follows: for given DFG assign the operations to functional units, introduce the slacks for chosen resources and construct the scheduling graph so that the dynamic power consumption of CMOS circuit/system in question is minimised without deteriorating its throughput (measured by the number of control steps CS).

The slack of the functional unit is introduced by reduction of its supply voltage *Vdd*. This results in increasing the delay of the unit and appropriate reduction of the power consumption as explained in the introduction (in this paper we consider delay by 2, 4 or 8 times). The problem of assigning the operations and distributing the functional unit slacks is solved in two stages. First, the ASAP-like base algorithm is used to construct the non-delayed scheduling graph. Then it is modified with the IIOI [3] scheduling algorithm, and its two extensions, namely MAREL and UNILO described in the next section.

SCHEDULING ALGORITHMS DESCRIPTION

Three scheduling algorithms, namely IIOI, MAREL and UNILO, used for power reduction in digital CMOS circuits during high-level synthesis are presented. All the algorithms are based on the same two-stage core, described below.

The First Stage of the Algorithms

The first stage of the algorithms is a modified ASAP scheduling process. Modifications include resource constraints and multi-cycle operations.

The pseudo-code of the ASAP stage of the algorithms is presented below, with function explanations following.

assign_p_labels ( ) (line 1)

The function assigns the p-labels to each operation.

P-label of an operation is equal to the number of the control steps needed to perform all child operations in the DFG. Operations without any child-operations (i.e. having system outputs only) have p-label equal 0;

V_r = find_ready_operations ( ) (line 5)

```
1  assign_p_labels( )
2  cstep ← 0
3  WHILE ( V ≠ ∅ )
4  {
5  Vr ←find_ready_operations(cstep)
6     WHILE ( Vr ≠ ∅ )
7     {
8    vs ←vi ∈ Vr: p_label( vs )=min
9        fus ← find_fu( vs )
10       IF ( fus ≠ NULL )
11       {
12          assign( vs , fus )
13          V ← V \ vs
14       }
15       Vr ← Vr \ vs
16     }
17    cstep ← cstep + 1
18 }
```

Fig. 2. The first (ASAP) stage of the IIOI/MAREL/UNILO algorithms

This function returns the Vr set of operations ready to be scheduled, i.e. assigned to the chosen functional unit, in the current control step. Operation is ready to be scheduled if all input values required are calculated, i.e. all parent-operations are finished.

f_{us} ← find_fu(v_s) (line 9)

This operation finds the functional unit for the chosen operation that is ready to be scheduled at the current control step. If there are no free resources at the moment, function returns NULL, and selected operation scheduling is delayed.

If there is only one functional unit of the proper type available, then it is returned.

However the differences between IIOI, MAREL and UNILO algorithms depict the situation when there is more than one functional unit of the proper type available.

The IIOI algorithm, simply chooses the first available functional unit.

The MAREL algorithm calculates the relation distance for all available functional units.

The UNILO algorithm selects the functional unit that have the smallest number of operation assigned. This ensures that load is uniformly balanced over all of the functional units.

assign(v_s , f_{us}) (line 12)

The functions assigns the operation to the chosen functional unit at the current control step. If the operation being assigned is a multi-step one then the appropriate number of following control steps is also reserved.

The Second Stage of the Algorithms

The second stage (Fig. 3.) is the same for all the algorithms. It consists of delaying of the initial SG created in the first stage. The C_{su} set includes all columns suitable for delaying, and C_{ms} indicates the most suitable column selected out of this set. The chosen C_{ms} column actually undergoes the process of delaying.

Functions used in the pseudo-code are explained below.

```
1  Csu ← SG
2  WHILE ( |Csu| > 0 )
3  {
4   FOREACH ( Ck ∈ Csu )
5  IF ( NOT fsc_fulfiled( Ck ) )
6     Csu ← Csu \ Ck
7   IF ( |Csu| == 0 )
8    GOTO WHILE_END
9   IF ( |Csu| > 1 )
10  {
11   FOREACH ( Ck ∈ Csu )
12  IF ( there_is_same_fu( Ck ))
13     {
14    Cl←column_of_the_same_fu(Ck)
15      FOREACH ( vi ∈ Ck )
16       IF (fvi(vj)>fvi( vi ) )
17        interchange( vi , vj )
18     }
19  Cms ← Ck ∈ Csu: fck( Ck )=max
20  } ELSE
21  Cms = HEAD( Csu )
22  SGbackup ← SG
23  FOREACH ( vi ∈ Cms )
24  {
25   insert_idle_operations(vi)
26  IF(NOT delay_all_successors(vi)
27   {
28    SG ← SGbackup
29    Csu ← Csu \ Cms
30    GOTO WHILE_END
31   }
32  }
33  :WHILE_END
34 }
```

Fig. 3. The second stage of the IIOI/MAREL/UNILO algorithms

fsc_fulfiled(C_k) (line 5)

This function performs the check of the free space condition (f_{sc}), defined by the formula:

$$n_i \cdot l_k \le r_o \tag{4}$$

where n_i is the number of cycles needed to perform the operation, l_k is the number of operations assigned to the C_k functional unit column, r_o is the number of free operation slots after the first occurrence of an operation in the C_k functional unit column.

The condition (4) checks if there are enough free *csteps* for the idle operations identifying the longer processing time of a fu_i. Every Ck selected for the frequency lowering has to fulfil the condition (4). Despite the fact, that cascading operations from the other C_k's are not taken into consideration while calculating f_{sc} , it is sufficient for quick pre-rejection of some C_k from the C_{su} set, before starting the time consuming delaying process.

there_is_same_fu(C_k) (line 12)

This functions simply indicates, whether there is another *fu* of exactly the same type as the one assigned to C_k, i.e. being capable of performing the same type of operation in the same time.

C_l =column_of_the_same_fu(C_k) (line 14)

This function seeks for a column containing the same operations as Ck and sets the Cl pointer to it.

fvi(v_i) (line 16)

The $fv_i(v_i)$ function calculates fv_i factor for the v_i operation with the following formula:

$$f_{vi} = \frac{i_{vi} + o_{vi} + (f_{avi} - s_{vi} \cdot n_i)}{p_i + 1} \tag{5}$$

where i_{vi} is the number of independent inputs of the operation v_i, o_{vi} is the number of system outputs of the operation v_i , $f_{avi} = cs_M - (cs_e + n_i)$, cs_M is the maximal number of csteps admissible, and cs_e is the number of *cstep*, which v_i is assigned to, s_{vi} is the number of operations of the same type as operation v_i in the path of DFG below the operation v_i, n_i is the number of cycles needed to perform the operation, p_i is the p – label of the operation v_i, the minimal p label of an operation equals 0, hence addition of 1 in the denominator is necessary to avoid dividing by 0.

The *fvi* value of an operation indicates its suitability for being slowed down. It is used when there is more than one column of the same type, in order to create least interconnected column by interchanging operations.

interchange(v_i , v_j) (line 17)

This function swaps the v_i and v_j operations, so that the v_i is located in operation slots formerly occupied by v_j and vice versa.

fck(C_k) (line 19)

The value of the function is given by:

$$f_{Ck} = P^n_{dC_k} \cdot l_{Ck} \tag{6}$$

Table I

Power reduction obtained by IIOI (I), MAREL (M) and UNILO (U) for HLS for chosen DFG benchmarks

Benchmarks		CS_{MAX} extension (ΔCS)											
circuit	gates	0%			10%			20%			50%		
		I	M	U	I	M	U	I	M	U	I	M	U
c1355	514	4.1	3.5	7.7	12.6	12.9	13.1	12.6	13.1	13.1	13.1	13.2	13.2
s208	104	14.5	14.5	14.5	22.0	22.0	22.8	22.0	31.8	34.2	31.8	46.5	42.9
s5378	2779	8.2	6.6	15.3	17.0	20.4	44.0	17.0	44.8	51.8	41.0	47.1	66.8
s9234	5597	6.4	6.7	3.6	32.2	36.1	51.6	33.0	32.3	65.4	60.7	66.3	67.6

Where P^n_{dCk} is the normalised dynamic power dissipated in C_k functional unit column (*fu*-assigned to the C_k column, P^n_{dCk} is normalised to the fu_i having the lowest value of P_{di}), l_{Ck} is a number of operations in the C_k functional unit column.

The *fck* function is responsible for selecting the most suitable column (C_{ms}) for inserting idle operations, from the C_{su} set. It chooses the column assigned to the fu_i that has the highest power demand, hence gives the highest power demand reduction when slowed down.

insert_idle_operations(vi) (line 25)

This function simply adds new operation slots with idle operations after the v_i operation. If there is an empty operation slot after the last cstep occupied by v_i, then an idle operation is added there. However, when there is no empty room for a new idle operation, then the next operation in the column of v_i is delayed. Next the data interconnections between v_i and its successor operations must be checked. This is done by the delay_all_successors function described below.

delay_all_successors(v_i) (line 26)

This function checks if all the data needed to perform successor operation (si) of v_i are available on time, by checking the condition:

END_CSTEP(VI) ≤ START_STEP(SI) (7)

If it is not fulfilled, then the successor is delayed as many cycles as needed (so that start_step(s_i)=end_cstep(v_i)). Such delay implies the need for checking all the data interconnections between the successors of s_i. If the delay is not possible due to the cs_M constraint, the frequency lowering of v_i (and the functional unit it is assigned to) fails. In such a case the column containing v_i, i.e. C_{ms} is removed from the C_{su} set F, and the process starts from the beginning.

EXPERIMENTAL RESULTS

In our experiments, we also have considered CS_{MAX} extension by 10, 20 and 50 percents. The results obtained for chosen benchmarks [4] are presented in Table 1.

CONCLUSIONS

The level of power reduction obtained by IIOI, MAREL and UNILO algorithms are very promising, yet they strongly depend on the benchmark structure. The main advantage of the presented algorithms is their simplicity which makes them very robust. Further comparison of the results obtained by IIOI, MAREL and UNILO algorithms with the evolutionary one [2] show that for some benchmarks heuristic algorithms give better results.

Moreover, the heuristic algorithms run in significantly shorter time than evolutionary ones and in many cases lead to the acceptable results.

REFERENCES

[1] L. Benini, A. Bogliolo, and G. De Micheli. A survey of design techniques for system-level dynamic power management, *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 8 3, pp. 299-316, 2000.

[2] S. Koziel, W. Szczesniak. Application of adaptive evolutionary algorithm for low power design of CMOS digital circuits, *Proc. of Ninth IEEE ICECS'2002*, Dubrovnik, pp. 685-688.

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

[4] Collaborative BenchmarkingLaboratory, ftp.cbl.ncsu.edu.