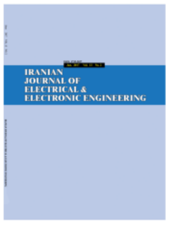
IUST Iran University of Science and Technology

Iranian Journal of Electrical and Electronic Engineering

Journal Homepage: ijeee.iust.ac.ir

Research Paper



Area Reduction of Combinational Circuits Considering Path Sensitization

S. Abolmaali*(C.A.)

Abstract: Area reduction of a circuit is a promising solution for decreasing the power consumption and the chip cost. Timing constraints should be preserved after a delay increase of resized circuit gates to guarantee proper circuit operation. Sensitization of paths should also be considered in timing analysis of circuit to prevent pessimistic resizing of circuit gates. In this work, a greedy area reduction algorithm is proposed which is path-based and benefits well from viability analysis as the sensitization method. A proper metric based on viability conditions is presented to guide the algorithm towards selecting useful circuit nodes to be resized with acceptable performance and area reduction results. Instead of using gate slacks in resizing the candidate gates, all circuit gates are down-sized first and then the sizes of circuit gates that violate the circuit timing constraint are increased. This approach leads to considerable improvement in the complexity and performance of the proposed method. Results show that area improvement of about 88% is achievable. Comparison to a pessimistic method also reveals that on average 14.2% growth in area improvement is obtained by the presented method.

Keywords: Area Reduction, Gate Resizing, Path Sensitization, Timing Analysis, Viability Analysis.

1 Introduction

IN recent years, due to the widespread use of portable electronic devices, there has been considerable research on reducing the chip area and lowering the power consumption of digital circuits. One acceptable method attained to fulfill the above requirements is reducing the size of some circuit gates. This way, in addition to the decrease in the area of the circuit, gate capacitances are also reduced which results in lower power consumption of the circuit. However, a reduction in the gate size also causes the delay of the gate to be increased, which can lead to violation of the circuit timing constraints.

To prevent this obstacle, timing analysis (path-based or block-based [1]) should be utilized during the gate resizing process to prohibit the delays of circuit paths to become greater than the circuit timing constraints. Timing analysis should be accompanied by the sensitization of paths in calculating the path delays. Path sensitization means the preparation of conditions for a transition to pass through each gate of the path, from the start to the end. Not considering the path sensitization may lead to pessimistic values for circuit delay.

In the dynamic sensitization method, considering the gate delays brings more flexibility and accuracy [2]. In this method, temporal and transitional values on the gate inputs, which cause temporal sensitization of the gates, are also considered. Two well-known dynamic sensitization conditions are viability analysis [2] and Chen-Du criterion [3]. The exactness of the Chen-Du sensitization criterion is ascertained, while the viability analysis estimates the same circuit delay as the Chen-Du method by not considering the complex gates in the circuit implementation [3, 4].

Many existing methods for gate resizing use the context of slack of gate delays to determine the amount of delay which can be added to the gate delay by shrinking the gate size, while the circuit timing constraints are not violated. The slack of a gate is

Iranian Journal of Electrical and Electronic Engineering, 2021.
Paper first received 28 November 2019, revised 11 January 2021, and accepted 15 January 2021.
* The author is with the Electrical and Computer Engineering Department, Semnan University, Semnan, Iran.
E-mail: shabolmaali@semnan.ac.ir.
Corresponding Author: S. Abolmaali.
https://doi.org/10.22068/IJEEE.17.3.1730

defined by *Maximum Arrival Time* (MAT) and *Required Time* (RT). The MAT is the delay of the latest signal which arrives at the gate after the changes in the value of circuit primary inputs at time 0. The RT is the latest time at which a signal reaches the gate without contravening the timing constraints of the circuit. For a gate *g*, the slack is defined as RT(*g*)-MAT(*g*). A positive slack means the delay of the gate can be increased by the value of the slack. Using path-based timing analysis, the slack of the gate is characterized as the difference between the timing constraint of the circuit and the delay of the longest path passing through the gate.

To prevent the generation of pessimistic results, path sensitization should be considered in the slack calculation [4]. The longest topological path traversing through a gate might be considerably longer than the longest sensitizable path (*true* path) passing through the gate, which leads to a smaller slack value for the gate.

There are plenty of works concentrating on optimizing circuit parameters like area, power, leakage, etc. using gate resizing. In [5], for a technology mapped circuit (a well-organized technology mapping procedure is used to map circuit functions to the library) under timing constraints, the power consumption is optimized by employing gate resizing. A discrete, general, restricted optimization problem model has been utilized to formulate the problem, and integer linear programming and the simplex method have been used for solving the problem by a fast algorithm. Work [6] uses the potential slack that may potentially be employed for circuit optimization. The effectiveness of potential slack as a metric for the performance of the combinational circuit is confirmed. Several approaches for approximating the circuit potential slack are explored and an optimal, polynomial-time procedure is presented. Many applications are presented for gate placement and resizing to illustrate the prediction capabilities of potential slack for circuit performance.

The authors of [7] provide a method for employing two supply voltages to improve power consumption in digital circuits using CMOS components under timing constraints. The power/delay pattern and the distribution of timing slack are first analyzed in a technology-mapped lattice. Then, by comprehensively using the slacks, timing-constrained enhancement is performed in a new paradigm. Following this paradigm, the power reduction is transformed into a maximal-weighted-independent-set problem that is solvable in polynomial time on transitive graphs. By inspecting the relationship between the node delay and slack changes, full utilization of slacks is investigated.

Work [8] utilizes a dual-threshold voltage (V_{th}) technique for optimizing overall power consumption. By employing linear programming in the simultaneous adjustment of V_{th} and resizing of the gates, it achieves considerable power improvement under delay constraints. The optimal slack value is assigned to the gates by the linear programming procedure with the aid of power-delay sensitivity. The authors of [9] employ budget management in improving the power dissipation of CMOS circuits. Budget management increases delay gradually within a circuit without disobeying timing constraints. The considered budget helps in reducing the area and power consumption of the combinational circuits. The zero-slack algorithm, that specifies the slacks in the circuit, is covered and a gate resizing procedure is introduced that utilizes budget management in circuit power optimization.

In [10] the authors state that in physical synthesis, latch placement is difficult to succeed since passing through the chip requires several cycles. They introduce RUMBLE, an enhancing method for the physical synthesis of latches that improves circuit timing using a linear timing model by synchronic replacement of several gates. RUMBLE is an incremental method that utilizes static timing analysis (for slack calculation) for optimizing the critical path's timing after gate resizing. In work [11] placement and gate resizing methods are combined to multiple-V_{th} approach by employing slack distribution management to reduce power consumption.

Authors of [12] present an efficient method for gate-version and V_{th} selection. Their algorithm first generates a perfect solution. Then, the leakage power is reduced by utilizing a Lagrangian Relaxation (LR) technique without violating timing constraints. If the gate-versions obtained by LR generate negative values for some slacks, a timing retrieval technique is employed to produce almost zero positive slack. Article [13] introduces OWARU, an incremental method for timing-driven gate allocation. Timing of the critical paths is optimized by a path smoothing algorithm that is conscious of the free space of the circuit. The location of the gates, placed on the critical paths, is changed to free spaces provided by the smoothed paths. Evaluation of the resulted delay changes is performed by an incremental static timing analysis procedure.

In [14], the authors state that block-based timing analysis methods have much less execution time in comparison to path-based ones however with less accuracy. The work is the modification of a block-based timing analysis procedure having both proper speed and perfect accuracy. The slack differences of critical paths are minimized between path-based and block-based analyses by considering a weight parameter for each circuit gate and optimizing these parameters. Although sophisticated methods are introduced in the above-mentioned works, however, none of them consider path sensitization in their works.

In article [15] area improvement is strengthened by timing analysis through the utilization of information obtained by a sensitization condition in calculating the gate slacks. The work presents a greedy heuristic for are improvement by gate sizing. It chooses from a pre-characterized library of different implementations of gates. When a gate is slowed down in this work, only the paths passing through that gate are considered and will be lengthened. However, the lengthened paths may affect the sensitization conditions of other circuit paths (as be explained in section 3) which is not taken into account. The work uses an improved Brand-Iyengar sensitization criterion [16] which is not exact and is more useful for its proposed idea in terms of reducing processing time.

In [17] for improving the power consumption of a circuit, the gates placed on noncritical paths are replaced with the smaller ones. Single and multiple gates resizing is employed for reducing the power dissipation. Gates are identified for resizing by a path-based method that considers false paths in the slack calculation. The loose sensitization criterion defined in [3] is used in this paper which is not exact in all cases. In the slack calculation process, the work slows down the earliest side-input that has the controlling value by Δd without changing the falseness of a false path. Then, it is inferred that all gates in the fan-in cone of the earliest side-input with controlling value have time slack smaller than or equal to Δd. Here, sensitization analysis is not considered for the gates in the cone, and therefore the slack values of gates are underestimated. To mitigate the execution time overhead, paths with lengths greater than a specified value are considered for computing the gate slacks. $R\times$delay, for $0 \leq R < 1$, is chosen as the threshold value. Thus, the precision of their method changes for different values for R. Also, no execution time is reported in this work.

Authors in [18] present a clustered voltage scaling procedure, which is path-based and considers the false paths in the analysis. The low supply voltage is assigned to the circuit gates with extreme slack, while the high supply voltage is considered for the gates placed on the critical paths. Original Brand-Iyengar sensitization criterion is used in this work which is not an exact method. In addition, many of the proposed methods are related to the clustering for voltage scaling, which makes their method not to be general. Moreover, the falseness of paths is not checked after voltage changes of each gate.

Authors in [19] suggest an approach at the design level to compromise reliability and supply voltage. They enlarge the extent voltage levels under acceptable timing violation. This is obtained by the methods that are employed in power-aware slack redistribution that allows an extended compromise between voltage and reliability. A design with more graceful fails is obtained that improves power consumption substantially with a negligible decrease in the application performance. Path sensitization is considered to prevent obtaining pessimistic circuit delays incurred by the existence of the false paths, which is not accurate. This is done by using a parameter α in calculating the error rate, which is acquired by vast simulations.

The work [20] is an SAT-based method using Timed Characteristic Functions (TCF) to implicitly calculate the longest true delay of a circuit. It formulates a large CNF file for the circuit and its components to find the circuit delay by decreasing the threshold time T slowly from an upper bound. Timing analysis is used to obtain the boundary. The process continues until the fulfillment of the CNF formula. Also, it uses two reduction techniques, equivalence reduction and constant reduction, for simplification of the CNF formula. It requires a lot of variables to construct the circuit CNF formula. The situation becomes worse when both rise and fall transitions are considered. Considering formulation for TCFs of the whole of the circuit is not required for obtaining the longest circuit path and is the major drawback of this work.

Authors of [21] present a TCF-based timing analysis method for path-specific timing verification which uses methods of [20] for circuit timing and logic formulation. They consider that a circuit and a specific path are given for timing analysis. They only consider the sub-circuit in the transitive fan-in cone of the primary output node of the specified path to be formulated. This way, the mentioned drawback of [20] is mitigated. For the to-be-verified paths sharing the same primary output node, they see the same sub-circuit for analysis. This is useful when multiple paths can be of interest for verification. However, in the optimization process, where paths should be analyzed one-by-one for selecting the gates for resizing, processing multiple paths is not beneficial. Besides, when a node delay is changed, this work re-evaluates its formula from the scratch and does not reuse the portions which remain unchanged.

In [22], Adjacency Criticality which is a new optimization metric is introduced. The work considers the process variation and the effect of gates lying on the fan-out cone to define the metric as the probability of placing the gates on the critical paths of the circuit. To enhance circuit timing yield, a statistical gate re-sizing technique is suggested. Statistical static timing analysis is utilized to evaluate circuit performance. However, path sensitization is not considered in this work. The research of [23] proposes a discrete gate sizer based on Lagrangian relaxation, that is very useful in decreasing the power consumption, to efficiently reduce the power and timing of the circuit. The proposed gate sizing technique is multi-threaded. Concurrent resizing of circuit gates is aided by the netlist traversal based on the directed acyclic graph. Static timing analysis is utilized in this work to update the circuit timings. Nevertheless, no path sensitization method is used in the timing analysis procedure.

All of the above-mentioned works which consider path sensitization, except the ones that use TCFs, do not employ an exact sensitization method. Methods provided in these works are general and work with any sensitization criterion. None of the researches reviewed concentrate on the properties of the exact sensitization methods and their implementation which can be useful in developing algorithms that are based on applying changes in the circuit and updating timing and sensitization attributes of the circuit multiple times.

In this article, a false-path-aware gate resizing algorithm is proposed which exploits the viability analysis [2] in timing analysis utilized for assessing whether the circuit delay meets the required timing constraints of the circuit. The algorithm uses SAT solvers for checking the satisfiability of viability conditions. The proposed method is path-based since analyzing the viability conditions requires traversing circuit paths. In the presented algorithm, no slack value is calculated for circuit gates. Instead, the size of each circuit gate is decreased at the beginning and then the sizes of circuit gates that violate the circuit timing constraint are increased. This approach prevents redundant repetitions of inner loops of the algorithm when the gate slacks are utilized. This way, the performance of the algorithm is enhanced, and scaling the size of the circuit becomes feasible.

In addition, when a circuit path is chosen by the algorithm to provide some gates to be resized, the gate selection approach has a considerable impact on the overall reduction of gate sizes and execution time of the overall algorithm. In this article, a proper metric for gate selection is proposed which is based on the viability conditions and the number of paths traversing through circuit nodes. The proposed algorithm of this work is incremental. When a gate delay is changed by resizing, the viability conditions, for those circuit paths which are analyzed previously and are not affected by this delay change, are not reprocessed. The major contributions of this work are:

1. Study of gate delay changes on altering the viability of circuit paths.
2. Proposing a complete path-based algorithm of gate resizing which utilizes viability analysis in the selection of candidate nodes.
3. Decreasing the size of all circuit gates at the beginning of the proposed method and then increasing the size of circuit gates that violate the circuit timing constraint, instead of using gate slacks.
4. Introducing an incremental algorithm for gate resizing which uses previously-obtained not-altered viabilities of circuit paths.
5. Proposing a proper metric, based on viability conditions, to select the best nodes to become a candidate for resizing.

The article organization is as follows. The next section introduces the viability analysis. Its concept and the related implementation considerations are explained. In Section 3, the impact of gate resizing on alteration of the viability of circuit paths is expressed. Section 4 describes the proposed algorithm of gate resizing which is equipped with viability. The heuristic metric which enhances the efficiency of the algorithm is presented. In Section 5, implementation considerations are pointed to.

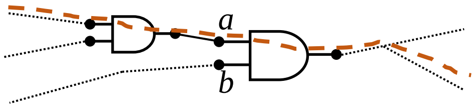


Fig. 1 A circuit portion to specify the concept of viability.

Section 6 contains the experimental results of the execution of the algorithm and discussion on them. The article is concluded in Section 7, Conclusions and Future Works.

2 Viability Analysis

This section specifies the viability analysis and important considerations of its implementation.

2.1 Viability Concept

The AND gate g in Fig. 1 with inputs a and b, along with a circuit path denoted by the dashed line, are used to illustrate the main concept of the viability analysis. Assume t_{ar} is the time that a transition through the depicted path arrives a (*online* input to g). Consider by static sensitization, b (*side* input to g) takes the stable value 0 (*i.e.*, controlling value of g at $t = \infty$). In this case, the transition on a is blocked since g is not sensitized. Now consider b has initial value 1 and at the time $t_{ar} + 1$ it gets value 0. In this situation, non-controlling value 1 is present on input b at the time the transition on a reaches g, and thus, g can temporarily be sensitized. Here, input b is considered as the *late* side input. It is stated in [24] that according to the viability concept, the sensitizing condition of a gate can be changed by a late side input to the gate.

The *viable* paths concept is introduced in [2]. Given an input vector v, they allow a gate to transmit a transition from one of its inputs. The path that the transition traverses is begun from a circuit primary input and ends at the gate's online input. Those paths which terminated at the side inputs of the gate are considered. Work of [2] states that under an input vector v, a path P containing the nodes $\{f_0, ..., f_m\}$ is viable if and only if for each gate f_i and each side input $h \neq f_{i-1}$ to f_i, either:

1. h is set to its sensitizing (not blocking) value by v, or
2. h terminates a viable path under v with a delay longer or equal than the delay of the partial path $\{f_0, \ldots, f_{i-1}\}$.

2.2 Viability Function

A Boolean function is utilized to define the generic viability analysis that checks a number of conditions [2]. For path P, the viability function ψ_P is formulated as:

$$\psi_P = \prod_{i=1}^{m} \psi_P^{f_i} \tag{1}$$

$$\psi_P^{f_i} = \sum_U ((S_U \frac{\partial f_i}{\partial f_{i-1}}) \prod_{g \in U} \psi^{g,\tau_{i-1}}) \qquad (2)$$

$$\psi^{g,t} = \sum_{P \in \mathcal{P}_{g,t}} \psi_P \qquad (3)$$

where the addition (multiplication) performs the logical OR (AND) operation. Equation (1) necessitates the viability of all path nodes for the viability of the path.

All situations that make node f_i of P viable are stated in (2). The subset of the side inputs of f_i having controlling value is denoted by U in this equation. Other side inputs are included in S_U. Term $\psi^{g,t}$ in (3) states that for making f_i viable, the list of partial paths terminating at g should contain at least one viable path whose length is greater than the length of partial path $\{f_0, \ldots, f_{i-1}\}$. In this formula, the set of terminating paths at g that have not smaller delay than t is denoted by $\mathcal{P}_{g,t}$.

In addition, $S_U \dfrac{\partial f_i}{\partial f_{i-1}}$ means that the side inputs in S_U have sensitizing (non-controlling) value, and thus, has no impact on the viability formula of the node. At last, considering S_{si} be the set of side inputs of f_i, Σ_U implies that all the subsets of S_{si} should participate in the evaluation. For input vector v, the necessary and sufficient condition for path $P = \{f_0, \ldots, f_m\}$ to be viable is $\psi_P(v) = 1$.

2.3 Dynamic Programming Approach for Implementing the Viability Function

From the available search procedures usable in obtaining the longest circuit path, the *best-first* method [25] benefits from a priority queue to store the partial sensitizable paths. The queue ordering is based on the delay of the longest partial path extension, or the *potential full length* of the partial path that is called the *esperance*. In the best-first method, the search process is ended by arriving at an output node. This is because if such a path with greater delay exists, in the best-first search procedure it should be recognized earlier. The number of paths examined by the best-first procedure is $K{\cdot}D$ in which the number of false paths is denoted by K, and D is the circuit graph diameter [25]. In contrast, an exponential number of paths are processed under the depth-first method to find the longest circuit path.

For a circuit path, all partial paths which terminate at the nodes of the path may contribute to the viability of the path [2]. This requires a recursive analysis of partial paths toward the input pins of the circuit, which may cause the same partial path to be traced multiple times in the analysis.

Besides, the best-first search method finds all side paths longer than a candidate path before the candidate path. Therefore, the recursive trace of all paths which terminate at the side inputs of g is not required. The viability function can be implemented by utilizing a dynamic programming procedure to benefit well from the advantages of the best-first method, which results in lower computations [2].

3 Gate Resizing and Changes in the Viability of Circuit Paths

When the delay of a gate is changed, some false paths may become true and some true paths may become false. First, more about sensitization conditions of viability is described. Exact algorithms, *i.e.* viability analysis and Chen-Du criterion, consider the floating mode analysis. In this mode, the initial value of each circuit node is considered as unknown. After the primary inputs receive an input vector, each node experiences a single transition to a known value that remains on the node [3]. In Fig. 2, obtained from [26], the illustration of viability conditions using floating mode analysis is presented.

The waveforms are related to the input pins of a circuit gate. The upper one is for the online input of the gate and the others are related to the side inputs. The dashed line shows the stable time of the online input. Either non-controlling (nc) or controlling (c) values can be observed on the side inputs. The shaded area points out that the signal has an unknown value and it may be unstable (varies in time). To satisfy the viability conditions, the side inputs should either have the non-controlling value or be stabilized later than the transition on the online input. It can be seen, from the two waveforms at the bottom of the figure, that the stable value of the late side inputs is not important (can be either controlling or non-controlling value).

The impact of a gate delay change on the sensitization of circuit paths is explained through the simple circuit in Fig. 3. The circuit has two primary inputs *PI1* and *PI2*, and a primary output *PO*. The name and delay of circuit gates in picoseconds (ps) are written below them. The same delay value is considered for both rise and fall

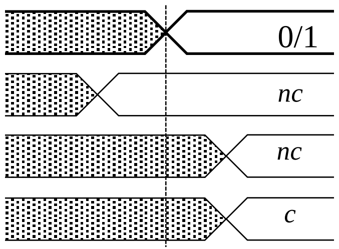


Fig. 2 Illustration of viability conditions using floating mode analysis [26].

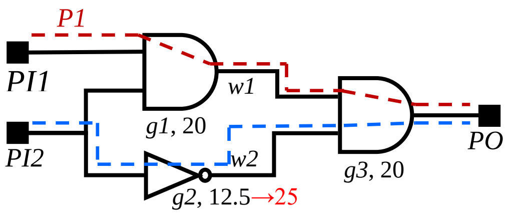


Fig. 3 Impact of a gate delay change on the sensitization of circuit paths.

transitions in the gates. The gate *g2* is the one that is supposed to be down-sized. Two paths *P1* and *P2*, denoted by dashed lines, are considered to be analyzed. First, *g*2 is considered to have the delay value 12.5 ps, and *PI1* to have a transition to 1 at time 0. In this case, for *P1* to be viable, *PI2* should have value 1 for allowing *g1* to be sensitized. Consequently, *w1* has value 1 at time 20 ps, and *w2* has value 0 at time 12.5 ps. Therefore, the transition on *w1* cannot proceed since the side input *w2* has a stable controlling value at time 20 ps. Changing *PI2* to have value 0 does not lead to sensitizing *P1*, too. Thus *P1* is a false path. However, if the delay of *g2* becomes 25 ps (by down-sizing), *w2* acts as a late side input for *w1*, and therefore *P1* is sensitized. Thus, increasing the delay of a gate can provide the condition for a line to acts as a viable late side input for a path gate, and for a false path to becoming true.

Now consider *g2* to have delay value 12.5 ps and *PI2* to have a transition to 0 at time 0. The rising transition appears on *w2* at time 12.5 ps. Since *PI2* has a falling transition, and if *PI1* has value 1, the falling transition on *w1* happens at time 20 (later than transition on *w2*). Thus, *w1* acts as a viable late side input for *w2*, and *P2* is a true path. However, if the delay of *g2* increases to 25 ps, when the rising transition appears on the online input of *g3* at time 25 ps, the side input has had value 0 for the last 5 ps and *g3* cannot be sensitized. In this case, where *PI2* has a falling transition, *P2* cannot be sensitized in another way and therefore it becomes a false path. Thus, down-sizing a gate may cause a true path to becoming false.

From the above examples, it may be concluded that increasing the delay of a circuit gate may not change the viability of a path, and a true path may remain true. These paths may provide new late side input for circuit gates on false paths not containing the resized gate, and cause these false paths to becoming true (first example). On the other hand, when a circuit gate is down-sized, true paths that pass through the gate may become false since the delays of these true paths are increased and these true paths may become longer than the viable paths terminating at the late side inputs (second example).

4 Proposed Algorithm of Gate Resizing Equipped with Viability

In this section, the general algorithm for gate resizing is presented which uses viability in timing analysis required for the investigation of sensitization state of circuit paths after resizing of circuit gates.

The feasibility of this algorithm for large circuits is also discussed. Related pseudo-code is depicted in Fig. 4, Algorithm 1. It should be explained here that as stated in subsection 2.3, since the viability of a path is dependent on the viability of other circuit paths, the proposed algorithm is path-based in which many circuit paths are traversed.

In this algorithm first, the longest true path of the circuit is attained by considering viability analysis to obtain the timing constraint of the circuit. Then, the size of all circuit gates is down-sized. After that, the proposed metric of this work (introduced in subsection 4.2.1) is calculated for each circuit node. In the next step, a set of starting partial paths is prepared to be extended and analyzed. After that, the main loop of the algorithm is started. In this loop, a partial path is analyzed under viability conditions and if the evaluation is satisfiable, the path is extended. If a path under analysis reaches a primary output, some candidate nodes of it are up-sized to fulfill the circuit timing constraint. Then the circuit is updated according to the changes made in the gate delays. If the delay of an obtained complete true path is not greater than the circuit timing constraint, the loop is terminated. The longest true path of the circuit after gate resizing is attained again using viability analysis to be compared with the circuit timing constraint.

4.1 Obtaining the Timing Constraint of the Circuit

The algorithm is started by extracting the graph of circuit structure, which results in graph *C*. Circuit gates and the connections between them are modeled by graph nodes and edges, respectively. Since viability conditions are stated by CNF formulas, at the next step the CNF formulas for each node of *C* are created according to their related gate types. These formulas are used during the execution of the algorithm repeatedly.

As stated in subsection 2.3, paths that are analyzed under viability conditions are ordered by their esperances (esperance is the potential full length of path). Using the Maximum Remaining Time (MRT) is one reasonable approach to obtain the esperance. An MRT value is assigned for each circuit node. It is specified as the delay of the longest extension path from the circuit node to a primary output node. Step 3 of the algorithm calculates the MRT value for each circuit node. This process is performed by a backward block-based timing analysis from the primary output nodes of the circuit. Path sensitization is not considered here to accelerate the algorithm execution.

In step 4, delay of the longest true path of the circuit, D_{LTP}, is obtained by implementing the viability function using the dynamic programming approach explained in subsection 2.3. This value is assumed as the timing constraint of the circuit.

4.2 Required Pre-Processing

To perform gate resizing, one approach is the utilization of gate slacks. Positive slack means the delay of the gate can be increased (the size of the gate can be decreased) while the timing constraints are not violated. However, the approach used in this work for gate

Algorithm 1

```
1.  make the graph of circuit, $C$
2.  create CNF formula for each node of $C$
3.  call calculate_remaining_times_of_all_nodes($C$)
4.  $D_{LTP}$ = find_longest_true_path_of_circuit($C$)
5.  double the delay of all nodes of $C$
6.  call calculate_remaining_times_of_all_nodes($C$)
7.  calculate the proposed metric for all nodes of $C$
8.  call fill_frontier_by_paths_contains_transitions_on_primary_input_nodes($C$, frontier)
9.  while frontier is not empty do
9.1.    pick up cur_path from frontier
9.2.    psi = evaluate_viability_of_path_if_needed(cur_path, should_be_added)
9.3.    if psi is true then
9.3.1.      if should_be_added is true then
9.3.1.1.        add cur_path to tvp_vector of last node of cur_path
9.3.2.      if cur_path terminates at a primary output then
9.3.2.1.        if $D_{cur\_path} \leq D_{LTP}$ then
9.3.2.1.1.          break
9.3.2.2.        up-size sufficient available nodes in cur_path to satisfy timing constraint $D_{LTP}$
9.3.2.3.        update tvp_vector of each node of $C$
9.3.2.4.        update remaining times of nodes of $C$ if required
9.3.2.5.        update esperances of paths in tvp_vector of nodes of $C$
9.3.2.6.        call fill_frontier_by_paths_contains_transitions_on_primary_input_nodes($C$, frontier)
9.3.2.7.        continue
9.3.3.      for each fon from nodes on fan-out of last node of cur_path do
9.3.3.1.        extension_path = {cur_path, fon}
9.3.3.2.        add extension_path to frontier
9.4.    else
9.4.1.      remove cur_path  // this path is useless
10. $D'_{LTP}$ = find_longest_true_path_of_circuit($C$)
11. Compare $D_{LTP}$ and $D'_{LTP}$
```

Fig. 4 General algorithm for gate resizing benefits from viability in the required timing analysis.

resizing is not based on the slack values. The reason is explained briefly. When a true path shorter than the timing constraint of the circuit is found, some path gates, from several candidate path gates, can be down-sized according to their slacks. Note that several selection combinations from candidate gates might exist.

After the down-sizing of the gates, as explained in Section 3, new paths may become true which were false before the down-sizing of the gates. The length of one or more of these paths might be longer than the timing constraint of the circuit. Therefore, the gate resizing algorithm should cancel the performed resizing, which can introduce heavy overhead since the information related to the gate delays and the viability of analyzed paths should be modified. In addition, the algorithm should find another gate resizing solution, either by modifying the resizing performed in the previous step, or by selecting another combination of path gates from available candidate gates, which satisfies the timing constraint. This process is very complex and if the newly made decision leads to timing constraint violation again, the process may need to be repeated several times which has a large execution overhead.

Instead, in this work, the size of each node of *C* is halved at the beginning, and then the sizes of some nodes are doubled every time a timing constraint violation occurs during the execution of the algorithm. Doubling the size of nodes is terminated when a viable path with a delay lower than or equal to D_{LTP} is found. Although there is no guarantee that the performed resizing is the optimum one, every time a true path is encountered, examining several solutions to fulfill the circuit timing constraint is omitted. Note that in the gate resizing procedure, the amount of decrease in the size of gates is not unlimited. In many previous works, doubling the gate delays is selected as the constraint, which is also used in this work.

In step 5 of the proposed algorithm, the delays of all nodes of *C* are doubled due to the above explanation. Since the gate delays are changed, it is required to re-calculate the remaining times of gates that are used in the viability analysis. This is performed in step 6 of the algorithm.

When a complete viable path is obtained by the algorithm which has a delay greater than D_{LTP}, it provides some candidate gates for resizing which their sizes are not fixed yet. These nodes are named *available* nodes in this article. Among them, some nodes may be selected that after being resized they can fulfill the timing constraint. In this article, two approaches are utilized for node selection. The first one, M_{IN_ORDER}, selects available gates in their order of appearance in the path. This is the preliminary method to be used for comparison to the proposed method of this article.

4.2.1 Proposed Metric

The introduced metric of this work is based on the viability conditions and the number of paths traversing through the node, which is an important factor in the efficiency of the proposed metric. Changing the delays of some available nodes alters the viability of paths traversing through these nodes. Therefore, selecting some available nodes which changing their delays leads to maximum or minimum changes in the viabilities of the traversing paths can have influences on the performance or quality of the algorithm, and thus can be used as a selection metric. Keeping this in mind, from the available nodes, a subset that fulfills circuit timing constraint D_{LTP} should be selected and all of its nodes should be resized. Indeed, the simultaneous selection of multiple nodes is considered. After changing the node delays, the viability value of some circuit nodes is changed. Maximizing or minimizing the aggregate of these viability value changes can be used as a heuristic for the node selection process.

However, the above-mentioned approach imposes a heavy overhead. First, a viability analysis should be performed on a part of circuit paths to reflect the applied delay changes. This process is time-consuming by itself since it is path-based and needs several calls to the SAT solver. In addition, this process should be repeated for all subsets of available nodes that satisfy D_{LTP}. For example, to select 5 nodes from 10 available nodes, the number of total subsets becomes 252 which requires a long processing time. Besides, after examining a subset, the viability information of the processed paths that are stored in circuit nodes should be removed before processing another subset, which is time-consuming.

Therefore, a greedy approach is employed in this work for node selection. Among the available nodes, the node having the maximum value of the proposed metric ($M_{PROPOSED\_MAX}$) or the minimum value ($M_{PROPOSED\_MIN}$) is selected first. The proposed metric is a combination of viability conditions and the number of partial paths ending to and originating from the circuit nodes. Modifying the delay of such gate may cause considerable changes in the sensitization of circuit paths and thus selecting such an important gate is reasonable.

One approach, to include the viability conditions in the selection of the best available node, can be considering the viabilities of all paths traversing through each available node. In this approach, the viabilities of all circuit paths should be attained. Then, the sum of viabilities of paths traversing through each circuit node should be obtained. As is obvious, this is a very time-consuming procedure and it should be noted that this procedure is performed during the execution of the general algorithm. Therefore, a heuristic method is required for estimating the viabilities of all paths passing through each circuit node.

The heuristic is defined based on the viability conditions introduced in subsection 2.1. Each side input *si* of a circuit gate *g* on the path *P* should have either non-controlling value (condition 1), or should be at the tail of one or more viable partial path which has delay not shorter than the partial path of *P* that terminates in online input *oi* of *g* (condition 2).

Signal probability is used in this work for evaluating condition 1 for *g*. The signal probability is the probability of a circuit signal to have logical value 1 under all input vectors. Since examining the circuit under all input vectors is impossible for large circuits, a probability-based method is used. In this method, a probability relation is introduced for the signal on the output pin of each gate type. Consider $\mathcal{P}_1$ and $\mathcal{P}_2$ are the signal probabilities of input pins of a two-input gate. The signal probabilities $\mathcal{P}_S$ for different gate types are depicted in Table 1.

To obtain the signal probability for each gate *g*, a breadth-first traverse of the circuit graph *C* is performed. For primary input nodes, $\mathcal{P}_S$ is set to 0.5 which means that the probability of a primary input to have a value 0 or 1 is equal. For the other nodes of *C*, formulas in Table 1 are used during the traverse of *C*. Note that in a circuit structure, $\mathcal{P}_S$'s of the gates are dependent to each other. However, for the sake of simplicity, considering node dependencies and conditional probabilities is ignored in this work. It should be noted that setting only stable value 1 is analyzed in this work and the transient signal values are not considered.

The obtained $\mathcal{P}_S$'s are used to determine the probability of a circuit line being 0 or 1. In other words, they are used to attain the probability of a circuit line, which feeds a circuit gate, having a non-controlling value that is utilizable in condition 1 of the viability.

Several approaches can be employed in evaluating condition 2 of viability, which are listed below:

Using a simplified version of viability analysis: in this approach, still, the dynamic programming implementation of viability analysis, presented in subsection 2.3, is utilized. However, instead of checking the satisfiability of complex CNF files, signal probabilities are used for checking the possibility of

Table 1 Signal probability $\mathcal{P}_S$ for different gate types.

Gate Type	$\mathcal{P}_S$	Gate Type	$\mathcal{P}_S$
Buffer	$\mathcal{P}_1$	Inverter	$1-\mathcal{P}_1$
AND	$\mathcal{P}_1\times\mathcal{P}_2$	NAND	$1-\mathcal{P}_1\times\mathcal{P}_2$
OR	$1-(1-\mathcal{P}_1)\times(1-\mathcal{P}_2)$	NOR	$(1-\mathcal{P}_1)\times(1-\mathcal{P}_2)$

setting a side input to the non-controlling value. Since all partial paths of the circuit should be analyzed in this approach, this method is very time-consuming.

Using MATs: instead of traversing all partial paths of the circuit, one heuristic is employing MATs of nodes. MATs of online and side input nodes of a circuit node are used instead of delays of the online and side partial path in condition 2. The reason is that if the MAT of a side input is greater than the MAT of the online input, then with a high probability the side input is visited earlier in viability analysis through a partial path longer than the partial path terminating at the online input.

Using the average value of delays of partial paths terminating at circuit node: This approach is similar to the above one, except that the average value of delays of partial paths terminating at the side and online inputs of a circuit node are used instead of MAT values.

Using aggregate of the lengths of paths traversing through circuit node: In this approach, for each circuit node, the aggregate value of the lengths of paths (instead of the number of paths mentioned previously) traversing through the node is used.

Using aggregate sensitization probabilities of paths traversing through circuit node: This approach is similar to the above one, except that the aggregate sensitization probabilities of paths are used instead of the aggregate lengths. Sensitization probabilities of paths are obtained by considering only static sensitization. The probabilities of setting the non-controlling value on side inputs are attained by signal probabilities.

Other similar approaches may be proposed. A comprehensive investigation of using the above approaches as heuristic methods for the algorithm was performed. Results showed that maximizing or minimizing of the metrics proposed in the above approaches does not outperform the efficiency of the approach presented in this work.

The proposed metric of this work for circuit node *g* is formulated as:

$$m_g = \left(\sum_{oi} \sum_{si \neq oi} \left(\mathcal{P}N_1 + \mathcal{P}N_2 \right) \right) \div K \times N_{PPo} \tag{4}$$

$$\mathcal{P}N_1 = \mathcal{P}_S\left(si\right) \times N_{PPi}\left(oi\right) \tag{5}$$

$$\mathcal{P}N_2 = \left(1 - \mathcal{P}_S\left(si\right)\right) \times \left(MAT\left(si\right) > AD_{PPi}\left(oi\right)\right) \times N_{PPi}\left(sd\right) \tag{6}$$

As explained in the previous paragraph, considering metrics only based on viability conditions is not beneficial. Thus, a combination of viability conditions and the number of paths traversing through circuit nodes is employed.

In the above formula, $\mathcal{P}N_1 + \mathcal{P}N_2$ is calculated for each online input *oi* of *g* and each side input *si* of the considered *oi*. $\mathcal{P}_S\left(si\right)$ is the probability of *si* having value 1. The above formula is written for gate types with a non-controlling value 1. For the other gate types, $1 - \mathcal{P}_S\left(si\right)$ is used in calculating $\mathcal{P}N_1$. $N_{PPi}\left(oi\right)$ and $N_{PPi}\left(si\right)$ denote the number of partial paths terminate at *oi* and *si*, respectively. In calculating $\mathcal{P}N_2$, the term $1 - \mathcal{P}_S\left(si\right)$ is the probability of *si* having value 0. For the gate types with non-controlling value 0, the term $\mathcal{P}_S\left(si\right)$ is used instead.

$AD_{PPi}\left(oi\right)$ represents the average value of delays of partial paths terminating at *oi*. The term N_{PPo} is the number of partial paths originate from *g*. Parameter *K* is the number of times parameter N_{PPi} is summed in calculating m_g for each fan-in node of *g*. For example, for the first fan-in fi_1 of a gate with three fan-in nodes, N_{PPi} is summed two times in $\mathcal{P}N_1$ where fi_1 is as *oi*, and two times in $\mathcal{P}N_2$ where fi_1 acts as *si*, one time for fi_2 and one time for fi_3. Therefore, *K* is 4 for 3-inputs gate types. For 2- and 4-inputs gate types, *K* is 2 and 6, respectively. The reason for using parameter *K* is to count N_{PPi} only one time in the formula of m_g for each fan-in node of *g*.

The total number of complete paths traversing through *g*, and through all fan-in nodes *fi*, is obtained by:

$$N_{CP}\left(g\right) = \left(\sum_{fi} N_{PPi}\left(fi\right) \right) \times N_{PPo} \tag{7}$$

Considering viability conditions in this formula results in the formula for m_g. Formula (5) is related to condition 1 of viability, i.e. setting non-controlling value on *si*. Formula (6) incorporates condition 2 of viability in the metric. It relates to the case that *si* has a controlling value (term $1 - \mathcal{P}_S(si)$), and there exists a side path longer than the online partial path ($MAT(si) > AD_{PPi}(oi)$).

For checking the existence of a longer side path, *MAT* is used for *si* since the longest partial path terminates at *si* is the most useful one. In addition, $AD_{PPi}(oi)$ is used for *oi* here because the metric m_g is unique for all paths traversing through *g*. Moreover, N_{PPi} for *oi* is included in (5) since when the non-controlling value places on *si*, the number of paths traversing through *si* is not important. However, when the length of side paths and online paths is compared in (6), $N_{PPi}(si)$ becomes more important. Using $N_{PPi}(si)$ in (5) or $N_{PPi}(oi)$ in (6) degrades the efficiency of m_g, according to the simulations performed on the considered benchmarks. For gate types with only one fan-in node, formula (7) is used directly for obtaining m_g instead of (4) since the viability conditions are not applicable in this case.

4.2.2 Frontier Queue

After performing the required pre-processing computation, the main path-based procedure of Algorithm 1 is started (line 8 in Fig. 4). This procedure is based on the dynamic programming method presented in subsection 2.3. The object named *frontier* is a priority queue containing partial paths and is used in the best-first procedure utilized in the dynamic programming method. Partial paths are ordered in the frontier based on their esperances. First, for each primary input node a partial path, containing only that node, is created and inserted in the frontier (line 8). The number of these partial paths is double the number of primary input nodes. One path for rising and one path for falling transition on each primary input node are considered.

4.3 Main Loop of the Algorithm

Then, the main loop of the algorithm starts. The procedure is repeated while the frontier is not empty. First, a partial path *cur_path* is picked up from the frontier. Note that this partial path has the greatest esperance among the partial paths in the frontier. Then, the viability of *cur_path* is analyzed, if it is required, and the result is stored in variable *psi*. As stated previously, after changing the delays of some circuit nodes, it is required to analyze the viability of previously analyzed paths again. If the delay changes do not affect the viability of a path, the previously obtained viability information is preserved.

For each circuit node, a vector named *tvp_vector* is considered which contains the viable paths terminating at the node. Also, when delay changes are performed in line 9.3.2.2 of Algorithm 1, *tvp_vector*'s of all affected nodes are updated in line 9.3.2.3 to reflect the delay changes. This concept is explained later in this section.

In function *evaluate_viability_of_path_if_needed*, *tvp_vector* of the last node of *cur_path* is searched. If *cur_path* exists in the vector, the *should_be_added* variable is set to 'false' and the function returns the value 'true'. Otherwise, the value of *psi* is obtained by performing viability analysis on *cur_path*, and the value of *should_be_added* is set to 'true'. If the *psi* value is 'true', the process of *cur_path* should be continued. Otherwise, the partial path *cur_path* is a false path and should be discarded (line 9.4.1).

For *psi* having value 'true', if *should_be_added* is 'true', *cur_path* should be inserted in *tvp_vector* of the last node of *cur_path*. After that, it is checked if *cur_path* terminates at a primary output node (*cur_path* is a complete path). If so, a sequence of activities is performed. The immediate condition which is checked is whether the delay of *cur_path*, D_{cur_path}, is less than or equal to the timing constraint D_{LTP}. If it is, no further resizing process is required and the main 'while' loop of the algorithm is terminated. Otherwise, the nodes of *cur_path* are analyzed to be resized.

A sufficient number of available nodes is up-sized in this step under the constraint of D_{LTP}. One of the node selection approaches (subsection 4.2) is used here to select enough number of nodes from the available nodes. The pseudo-code in Fig. 5 is a general code that chooses a sufficient number of path nodes based on the maximum metric value of a selected approach. The code for approaches that use the minimum values of the related metrics is very similar.

The function in this figure takes path *p* and chooses the available nodes based on metric *param*. *path_delay* is the delay of *p*. Initially, the function places all available nodes of *p* in *available_nodes_list*. If the list is not empty a loop for selecting the best nodes is started in line 4. Node *max_cn*, obtained by those lines of the algorithm labeled by prefix 4.2, has the maximum parameter value *max_param* among the available nodes. After it, the delay of *max_cn* is decreased and thus *path_delay* is also decreased. *max_cn* is removed from *available_nodes_list* then by setting one of its attributes to 'true'. If the current value of *path_delay* is not greater than D_{LTP}, or no available node remains, the loop is terminated.

After performing delay changes, the *tvp_vector* of all circuit nodes are processed. If a path in *tvp_vector* of a circuit node has one or more nodes that their sizes are fixed in the previous step, it is removed from the vector. The reason is that by altering the delay of a path, a true path may become false, and thus, it has no place in *tvp_vector*.

Since delays of some circuit nodes are changed and consequently delays of several paths may be altered, the viability analysis of the circuit paths should be performed again from the beginning. It is required that the remaining times of the circuit nodes to be updated (line 9.3.2.4). In addition, the esperances of paths in *tvp_vector* of circuit nodes should be updated in the next step. The reason is that if *cur_path* is one of the paths in this vector and the object of this path is used instead of *cur_path*, its esperance should be updated by new changes in the node delays to allow correct extension of the path.

Because the delays of some nodes are changed, it seems that before beginning the next iteration of the

Function *general_function_for_upsizing_available_nodes*(*p*)

```
1.        path_delay = p.delay
2.        add all available nodes of path p to available_nodes_list
3.        if available_nodes_list is empty then
3.1.          return
4.        while (true)
4.1.          max_param = 0.0
4.2.          for each node cn in available_nodes_list do
4.2.1.            if cn.param > max_param then
4.2.1.1.              max_param = cn.param
4.2.1.2.              max_cn = cn
4.3.          path_delay = path_delay – max_cn.delay_change
4.4.          max_cn.delay_change = 0.0
4.5.          max_cn.node_delay_fixed = true
4.6.          remove max_cn from available_nodes_list
4.7.          if path_delay ≤ D_LTP or available_nodes_list is empty then
4.7.1.            break
```

Fig. 5 General code which up-sizes a sufficient number of available nodes.

main algorithm, it is required to update the MAT and AD_{PPi} of circuit nodes since they are used in calculating the metric m_g (Formulas (4) and (6)). However, after running several simulations on the considered benchmarks, it concluded that updating the mentioned parameters degrades the efficiency of the proposed method. Therefore, it is sufficient to calculate the MAT and AD_{PPi} parameters for circuit nodes only once and at the beginning of the algorithm.

Now that updating the circuit elements is completed after delay changes, the exhaustive viability analysis should be performed from the beginning again. First, the function in line 8 of the algorithm is re-called to fill the frontier by proper preliminary paths. All partial paths that are remained in the frontier from previous steps of the algorithm (before delay changes) are removed at the beginning of this function. Then, the 'continue' instruction repeats the while loop in line 9 from the beginning, allowing the algorithm to be executed for the new version of the circuit.

If *cur_path*, which is true, is not terminated at a primary output node (line 9.3.3), it should be extended through the fan-out nodes of its last node. Each fan-out node *fon* of the last node of *cur_path* is added to the last node of *cur_path* to create a new *extension_path*. Each created path is inserted into the frontier to be processed in the next iterations of the algorithm.

After processing several partial and complete paths in the main loop of the algorithm, delays of complete paths become less than or equal to D_{LTP}. In this situation, the loop is terminated which means that the length of all remaining paths is not greater than the timing constraint. After it, in line 10, the longest true path of the circuit and its delay, D'_{LTP}, are obtained again. D'_{LTP} is compared to D_{LTP} to ensure fulfilling the circuit timing constraint.

5 Implementation Considerations

The mentioned algorithm is implemented in C++. Both rise and fall transitions on input and output lines of circuit gates are considered. For gate delay, arrival time, remaining time, and esperance of each circuit node, separate values for rising and falling transitions are utilized. Besides, for each circuit node, two different *tvp_vector*s for the mentioned transition types are considered, which is not stated in the algorithm for abbreviation.

A lookup-based method is utilized to obtain gate delays by the Spice simulations with the aid of linear regression. NanGate45 Open Cell library [27] is employed in the simulations. Gate type, the capacitive load on the gate output pin, the input pin which a transition places on it, slope delay of transition on the input pin, and the transition kind are the parameters that index the lookup tables. Considering linear regression and not considering the other circuital parameters for obtaining the gate delays lead to the introduction of errors in the delay values. Anyhow, the introduced errors are negligible.

From the available gate types in the above-mentioned library, gate types inverter and buffer, in addition to AND, OR, NAND, and NOR types with 2, 3, and 4 input pins are considered. The test circuits are first synthesized by considering these gate types before being used in the proposed algorithm.

It should be mentioned that the load capacitances of circuit gates are changed by gate resizing which itself alters the gate delays. However, to prevent the algorithm to become more complicated, this change is ignored. In addition, using signal probabilities, as stated in subsection 4.2.1, does not mean that the statistical method, like the researches in [28, 29], is utilized here. All delay values in this work are deterministic and process variation is not considered in this article. Moreover, the impact of the proposed approaches and methods on the circuit power consumption is not studied in this manuscript.

6 Experimental Results

A machine containing an Intel Xeon CPU (5680 at 3.33GHz) and 8GB of RAM, with the Linux operating system, is utilized to obtain the outcomes. ISCAS'85 benchmark circuits are used to compare the results of running the proposed algorithm when different metrics M_{IN_ORDER}, $M_{PROPOSED\_MAX}$, and $M_{PROPOSED\_MIN}$ are employed. To have a comparison with another method, the results obtained by using the Brand-Iyengar sensitization method are also included. It should be mentioned that finding several sensitizable paths by the Brand-Iyengar method is also path-based.

For $M_{PROPOSED\_MAX}$ and $M_{PROPOSED\_MIN}$ methods, lines 1 through 8 of the algorithm in Fig. 4, except the lines 3 and 4, are related to the required pre-processing before the beginning of the main loop of the algorithm in line 9. This pre-processing takes less than 1 second for all benchmark circuits and thus is not reported in the results.

Table 2 shows the area improvements of different methods in percent for the considered circuits. Columns 3 through 6 are results for Brand-Iyengar (identified by ***Br_Iy*** in the table), M_{IN_ORDER}, $M_{PROPOSED\_MIN}$, and

Table 2 Area improvements of different methods.

Circuit	# of Gates	*Br_Iy* [%]	M_{IN_ORD} [%]	M_{PR_MIN} [%]	M_{PR_MAX} [%]
c499	426	17.6	25.5	24.0	31.8
c880	294	68.2	77.9	79.5	80.1
c1355	436	22.9	25.9	27.1	28.0
c1908	320	46.1	67.9	67.3	68.8
c2670	518	65.3	74.4	74.9	82.8
c3540	635	52.3	57.6	58.1	67.3
c5315	1324	67.1	73.9	71.4	82.0
c6288	1472	43.7	52.5	49.2	58.7
c7552	1639	76.5	81.9	78.5	88.0
Average		51.1	59.7	58.9	65.3

$M_{PROPOSED\_MAX}$ methods, respectively. It is obvious from the table that the results of any of the proposed methods outperform the Brand-Iyengar results. For the c1908 circuit, $M_{PROPOSED\_MAX}$ even obtains 22.6% more area improvement in comparison to the Brand-Iyengar. The reason is that the Brand-Iyengar sensitization criterion is not exact and results in pessimistic outcomes. Indeed, it finds more sensitizable paths which leads to more candidate nodes for up-sizing.

Also, it is apparent from the table that the $M_{PROPOSED\_MAX}$ method attains the best results for area improvement for all benchmark circuits. Therefore, maximizing the proposed metric m_g (subsection 4.2.1) leads to better area improvement in comparison to minimizing the mentioned metric. $M_{PROPOSED\_MIN}$ results are worse than M_{IN_ORDER} results in more than half of the cases. For the c7552 circuit, the percentage reaches 88% for $M_{PROPOSED\_MAX}$ which is considerable. On average, 8.6%, 7.8%, and 14.2% growth in area improvement is achieved relative to the Brand-Iyengar method, for M_{IN_ORDER}, $M_{PROPOSED\_MIN}$, and $M_{PROPOSED\_MAX}$ methods, respectively. In addition, $M_{PROPOSED\_MAX}$ has 5.6% more growth in area improvement in comparison to M_{IN_ORDER}.

Fig. 6 presents the number of true paths found by each method. It can be deduced that the Brand-Iyengar finds almost always the most number of true paths, while $M_{PROPOSED\_MAX}$ finds the least number of true paths in most cases.

The number of nodes that are down-sized by each method is depicted in Fig. 7. It is apparent that $M_{PROPOSED\_MAX}$ always obtains the most number of down-sized nodes, while Brand-Iyengar down-sizes the least number of nodes in most cases. The number of found true paths and the number of down-sized nodes cannot lonely determine the total area improvement. The number of down-sized nodes in each path and the amount of change in the size of each down-sized node are also important. However, from the last two figures, it is deduced that Brand-Iyengar which finds the most number of true paths, leaves the least number of nodes for down-sizing. For $M_{PROPOSED\_MAX}$, it is completely different. These results agree with the results of Table 2.

Fig. 8 shows the execution time of each method in seconds for all benchmark circuits. The values are tractable and are less than 80 minutes. The figure states that for the first six circuits, Brand-Iyengar has lesser execution time, in comparison to M_{IN_ORDER} and $M_{PROPOSED\_MAX}$. The reason is that Brand-Iyengar sensitization conditions are simpler than the viability conditions, and consequently, SAT solving of the Brand-Iyengar conditions is performed with more speed.

However, for the last three circuits, which have more circuit nodes and therefore more circuit paths, Brand-Iyengar has greater execution time. The reason can be explained by the results of Fig. 6. The figure points out that the Brand-Iyengar method finds more true paths. To find more true paths, it is required to analyze more false and true partial paths. Also, when a true path is found in the proposed algorithm, a sequence of time-consuming processing is performed for up-sizing some candidate nodes and for updating the circuit for the next algorithm iteration.

Therefore, although the SAT solving for Brand-Iyengar is faster, finding more true paths by this criterion leads to analyzing more partial paths and to more processing for updating the circuit elements, which results in longer execution time. Fig. 8 also states that $M_{PROPOSED\_MAX}$ has more execution time in the first six circuits, in comparison to Brand-Iyengar and M_{IN_ORDER}. However, for the last three circuits, which have more circuit paths, $M_{PROPOSED\_MAX}$ has better performance. The reason, according to Fig. 6, is that the number of analyzed true paths in this method is considerably smaller than the other methods.

For example, for c5315, the execution time of

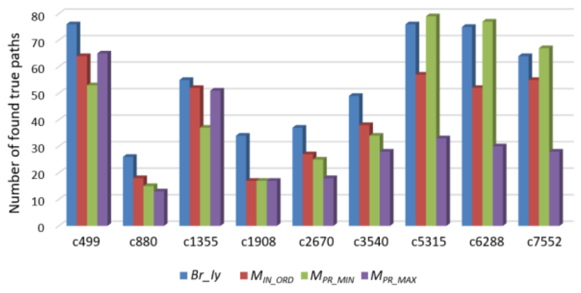


Fig. 6 Number of true paths found by each method.

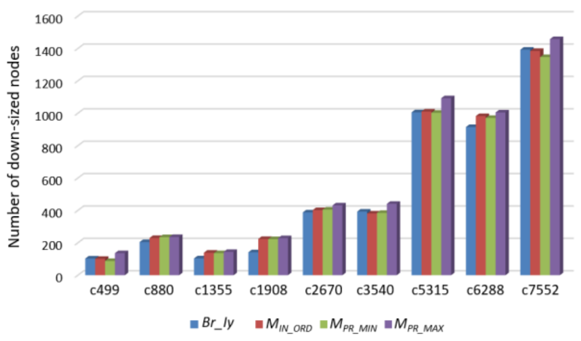


Fig. 7 Number of nodes down-sized by each method.

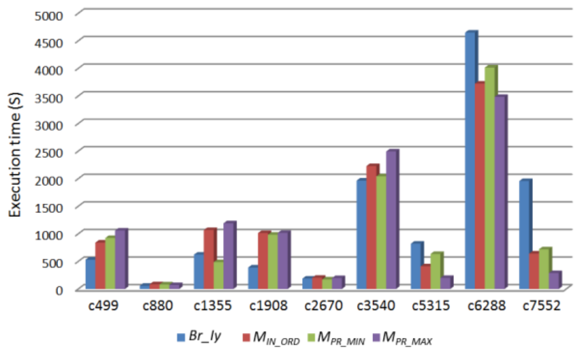


Fig. 8 Execution time of each method.

$M_{PROPOSED\_MAX}$ is 202 seconds which is less than one half of the execution time for M_{IN_ORDER} (412 seconds), and less than one-fourth of the execution time for Brand-Iyengar (822 seconds). For c7552, the execution times are 290, 643 (less than one half), and 1959 (less than one-sixth) seconds, for $M_{PROPOSED\_MAX}$, M_{IN_ORDER}, and Brand-Iyengar, respectively. The average execution times of benchmark circuits for the considered methods are 1243.6, 1136.5, 1119.1, and 1112.5 seconds for Brand-Iyengar, M_{IN_ORDER}, $M_{PROPOSED\_MIN}$, and $M_{PROPOSED\_MAX}$, respectively.

Since the proposed method uses a heuristic approach by utilizing a metric to select more useful nodes for resizing among the candidate nodes, there is no guaranty that the proposed method always selects the best nodes for resizing. However, considering viability analysis in timing analysis, along with the $M_{PROPOSED\_MIN}$ metric, leads to better overall area reduction.

7 Conclusions and Future Works

This article has proposed an incremental path-based algorithm for area reduction of digital circuits. The article has shown that considering exact path sensitization by using viability analysis has led to better area reduction, in comparison to using a non-exact criterion. Decreasing the size of all circuit gates at the beginning of processing, instead of using gate slacks, has resulted in tractable execution time for the presented path-based method. Utilizing the proposed metric in the presented heuristic approach has led to better average performance and area improvement in comparison to a non-exact sensitization criterion. In the future, area reduction by the proposed method in the presence of process variation can be investigated.

References

[1] C. Visweswariah, K. Ravindran, K. Kalafala, S. G. Walker, S. Narayan, D. K. Beece, J. Piaget, N. Venkateswaran, and J. G. Hemmett, "First-order incremental block-based statistical timing analysis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 10, pp. 2170–2180, 2006.

[2] P. C. McGeer and R. K. Brayton, "Efficient algorithms for computing the longest viable path in a combinational network," in *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 561–567, 1989.

[3] H. C. Chen and D. C. Du, "Path sensitization in critical path problem (logic circuit design)," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 2, pp. 196–207, 1993.

[4] H. R. Lin and T. T. Hwang, "On determining sensitization criterion in an iterative gate sizing process," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 18, No. 2, pp. 231–238, 1999.

[5] P. Girard, C. Landrault, S. Pravossoudovitch, and D. Severac, "A gate resizing technique for high reduction in power consumption," in *Proceedings of 1997 International Symposium on Low Power Electronics and Design*, pp. 281–286, 1997.

[6] C. Chen, X. Yang, and M. Sarrafzadeh, "Potential slack: an effective metric of combinational circuit performance," in *IEEE/ACM International Conference on Computer Aided Design*, pp. 198–201, 2000.

[7] C. Chen, A. Srivastava, and M. Sarrafzadeh, "On gate level power optimization using dual-supply voltages," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 9, No. 5, pp. 616–629, 2001.

[8] D. Nguyen, A. Davare, M. Orshansky, D. Chinnery, B. Thompson, and K. Keutzer, "Minimization of dynamic and static power through joint assignment of threshold voltages and sizing optimization," in *Proceedings of the 2003 International Symposium on Low Power Electronics and Design*, pp. 158–163, 2003.

[9] K. Banovic and H. Abdulhamid, "Algorithms for budget management with gate-sizing and other low-power applications," in *2006 IEEE International Conference on Electro/Information Technology*, pp. 290–294, 2006.

[10] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G. J. Nam, C. J. Alpert, and I. L. Markov, "RUMBLE: An incremental timing-driven physical-synthesis optimization algorithm," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 27, No. 12, pp. 2156–2168, 2008.

[11] T. Luo, D. Newmark, and D. Z. Pan, "Total power optimization combining placement, sizing and multi-Vt through slack distribution management," in *Asia and South Pacific Design Automation Conference*, pp. 352–357, 2008.

[12] G. Flach, T. Reimann, G. Posser, M. Johann, and R. Reis, "Effective method for simultaneous gate sizing and Vth assignment using Lagrangian relaxation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 33, No. 4, pp. 546–557, 2014.

[13] J. Jung, G. J. Nam, L. N. Reddy, I. H. R. Jiang, and Y. Shin, "OWARU: Free space-aware timing-driven incremental placement with critical path smoothing," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 37, No. 9, pp. 1825–1838, 2017.

[14] F. Peng, C. Yan, C. Feng, J. Zheng, S. G. Wang, D. Zhou, and X. Zeng, "A general graph based pessimism reduction framework for design optimization of timing closure," in *2018 55th ACM/ESDA/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2018.

[15] H. C. Chen, S. Cheng, Y. C. Hsu, and D. H. C. Du, "A path sensitization approach to area reduction," in *Proceedings of 1993 IEEE International Conference on Computer Design ICCD'93*, pp. 73–76, 1993.

[16] D. Brand and V. S. Iyengar, "Timing analysis using functional analysis," *IEEE Transactions on Computers*, Vol. 37, No. 10, pp. 1309–1314, 1988.

[17] H. R. Lin and T. T. Hwang, "Power reduction by gate sizing with path-oriented slack calculation," in *Proceedings of ASP-DAC'95*, pp. 7–12, 1995.

[18] J. Y. Jou and D. S. Chou, "Sensitisable-path-oriented clustered voltage scaling technique for low power," *IEE Proceedings-Computers and Digital Techniques*, Vol. 145, No. 4, pp. 301–307, 1998.

[19] A. B. Kahng, S. Kang, R. Kumar, and J. Sartori, "Slack redistribution for graceful degradation under voltage overscaling," in *15th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 825–831, 2010.

[20] Y. T. Chung and J. H. R. Jiang, "Functional timing analysis made fast and general," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 32, No. 9, pp. 1421–1434, 2013.

[21] C. N. Lai and J. H. R. Jiang, "Path-specific functional timing verification under floating and transition modes of operation," in *Proceedings of the 54th Annual Design Automation Conference*, p. 38, 2017.

[22] S. M. Ebrahimipour, B. Ghavami, and M. Raji, "Adjacency criticality: a simple yet effective metric for statistical timing yield optimisation of digital integrated circuits," *IET Circuits, Devices & Systems*, Vol. 13, No. 7, pp. 979–987, 2019.

[23] A. Sharma, D. Chinnery, T. Reimann, S. Bhardwaj, and C. Chu, "Fast Lagrangian relaxation based multi-threaded gate sizing using simple timing calibrations," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 39, No. 7, pp. 1456–1469, 2019.

[24] J. Jung and T. Kim, "Statistical viability analysis for detecting false paths under delay variation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 32, No. 1, pp. 111–123, 2012.

[25] J. Benkoski, E. Vanden Meersch, L. Claesen, and H. De Man, "Efficient algorithms for solving the false path problem in timing verification," in *IEEE International Conference on Computer-Aided Design*, Washington, DC, 1987.

[26] L. Guerra e Silva, J. Marques-Silva, L. M. Silveira, and K. Sakallah, "Satisfiability models and algorithms for circuit delay computation," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 7, No. 1, pp. 137–158, 2002.

[27] Nangate—The standard cell library optimization company, 2016. [Online]. Available: http://www.nangate.com.

[28] S. Abolmaali, N. Mansouri-Ghiasi, M. Kamal, A. Afzali-Kusha, and M. Pedram, "Efficient critical path identification based on viability analysis method considering process variations," *IEEE Transactions on Very Large Scale Integration Systems*, Vol. 25, No. 9, pp. 2668–2672, 2017.

[29] S. Abolmaali, M. Kamal, A. Afzali-Kusha, and M. Pedram, "An efficient false path-aware heuristic critical path selection method with high coverage of the process variation space," *ACM Transactions on Design Automation of Electronic Systems*, Vol. 23, No. 3, p. 32, 2018.

S. Abolmaali was born in Semnan, Iran, on August 21st 1980. He received the B.Sc. in 2004, the M.Sc. degree in 2007, and the Ph.D. degree in 2018 all in Computer Engineering from the University of Tehran, Tehran. He is currently the Assistant Professor with the Electrical and Computer Engineering Department of the Semnan University, Iran. His current research interests include statistical static timing analysis, approximate computing, digital circuit testing, hardware/software co-design, and low-power design.