IUST
Iran University of Science and Technology

Iranian Journal of Electrical and Electronic Engineering

Journal Homepage: ijeee.iust.ac.ir

Research Paper

Static Timing Analysis for Critical Path Identification in Ternary Logic Circuits

S. Abolmaali*(C.A.)

Abstract: In this article, a critical path identification method is proposed for ternary logic circuits. The considered structure for the ternary circuits is based on 2:1 multiplexers. Sensitization conditions for the employed ternary multiplexers are introduced. Moreover, static timing analysis and dynamic programming are utilized in the identification of true and false paths of the circuit for obtaining more realistic results in a reasonable time. An event-driven simulation engine is also developed for confirming the sensitization state of the identified paths. Some ternary arithmetic logic circuits are designed to depict the effectiveness of the proposed identification method. Simulation results show the correctness and efficiency of the proposed method.

Keywords: Critical Path, False Path, Multiplexer, Static Timing Analysis, Ternary Logic.

1 Introduction

TERNARY logic is known as a promising method for reducing the power consumption and propagation delay of today's circuits. The three-valued logic provides the possibility of implementation of the logic circuits with less transistor count and interconnects. On the other hand, it is very important to correctly identify the longest path of the circuit to have an accurate quantity for the maximum delay of the circuit. Timing analysis methods equipped with proper path sensitization conditions are commonly used to find the longest sensitized (true) paths of the circuit.

The static timing analysis method equipped with the dynamic path sensitization obtain more realistic results since it considers the temporary sensitization of path components caused by the temporal signal transitions on the input lines of the components. Chen-Du criterion [1] and viability analysis [2] are two gate-level exact methods that use dynamic path sensitization in identifying the true critical paths of binary logic circuits.

Implementation of ternary circuits by CMOS devices has been presented in [3] and [4]. The work in [5] has introduced the ternary CMOS devices which are realizable and scalable. By using the proposed models, a synthesis method for circuits based on multi-valued logic, which is fully compatible with CMOS circuits, has been presented. In this work, the standard ternary inverter (STI) gate has been selected as the basic block of the circuits. Their proposed models have resulted in the enhancement of the static noise margin of the ternary logic gates.

Recently, several methods have been proposed to design ternary circuits using efficient transistor-level structures and emerging device technologies. Using CNTFET devices has attracted a lot of interest due to their low delay and low power characteristics [6]. Changing the threshold voltage of these devices by altering the diameters of their carbon nanotubes has made these devices suitable for implementing multi-valued logics.

Designing ternary circuits using ternary decoders and encoders has been presented in [7]. In this work, circuit delays are obtained by case, through Spice simulations. A CNTFET-based synthesis method has been proposed in [8] that uses unary operators of ternary logic to implement the circuit with 3:1 multiplexers. The proposed method leads to circuits with fewer transistors in comparison to the work of [7]. Worst-case delays of circuits are earned by Spice simulations of the synthesized circuits.

The authors of [9] have introduced a synthesis

Iranian Journal of Electrical and Electronic Engineering, 2022.
Paper first received 27 July 2021, revised 19 January 2022, and accepted 25 January 2022.
* 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.18.2.2245

technique, by utilizing CNTFET devices, which uses ternary-transformed binary decision diagrams to implement ternary circuits with 2:1 multiplexers. The transistor count of the obtained circuits has been reduced when compared to the results of [8]. Fan-Out of 4 (FO4) method is used to extract the worst-case delay of the critical path of the circuit. In this method, the delay is obtained when four STI inverters are bound to each circuit output.

The work in [10] has proposed a logic synthesis methodology that synthesizes ternary functions by utilizing ternary logic gates made by CNTFETs. The amount of power consumed by the third logic value is reduced by using the body effect. Their method reduces the average power consumption of the circuit in comparison to the power consumption of the previously mentioned methods. In this work, the propagation delay for each proposed ternary gate is acquired by Spice simulation. Then, the worst propagation delay of the circuit is calculated by adding gate delays of the critical path.

In [11], a structured methodology for designing ternary logic circuits using CNTFETs has been proposed. It produces sum-of-products expressions that are simplified by the Quine-McCluskey table or the Karnaugh map, for the circuit outputs. The power-delay-product and the transistor count are improved by this method. The worst-case delay of the circuits in this work is also obtained by adding delays of the circuit components that are obtained by Spice simulations.

Authors of [12] have presented the reduced-sized ternary logic gates by utilizing negative capacitance characteristics of the ferroelectric materials in the CNTFET devices. The basic ternary gates proposed by this work show significant improvement in the transistor count and the energy-delay product. The ternary memristor-CMOS logic family has been presented in [13] for building primitive ternary gates and ternary MIN and MAX gates. In comparison to the conventional CMOS logic, significant data density improvement has been achieved. In these two works, the delays are extracted by Spice only for the proposed ternary gates and components, and obtaining the circuit delay is not considered.

In work [14], a compact model of ternary CMOS has been used to design a ternary multiplier. Performance improvement and overall power reduction have been attained by the proposed multiplier circuit. To avoid simulating the multiplier circuits containing a large number of transistors with Spice, this work first has acquired the delays of the proposed gates for the transitions between different ternary values. Then, the maximum delay among these cases has been used for each gate in the critical path of the circuit to obtain the worst-case delay of the multipliers.

In the mentioned works, circuit delays are obtained either by Spice simulation of the whole of the circuit, which can be time-consuming especially when several critical paths are analyzed or by accumulating the delays of the gates located in the critical path. The focus of any of these works has not been static timing analysis. It should be mentioned that just adding the maximum gate delays of the circuit path which contains the most path gates is not considered as static timing analysis (although it provides the worst-case delay of the circuit). In static timing analysis, all possible paths of the circuit are analyzed for timing violation.

In addition, path sensitization has not been considered in any of the reviewed researches. The longest physical path of the circuit does not necessarily determine the worst-case delay of the circuit since it may not be sensitized. The longest circuit path is sensitized by a proper input vector which is achieved by sophisticated timing analysis and this issue has not been studied in these works. Furthermore, as stated before, since timing analysis is not the main issue of the previous works, they only find the delay of the longest circuit path and none of them identify several critical paths of the circuit.

To be useful in timing analysis, the basic components of the circuit should be easily included in the circuit synthesis procedure and should have straightforward sensitization conditions (like the ones in the binary logic based on the controlling and non-controlling values for primitive gates). Although the enumerated methods in [10] through [13] have adorable improvements in the characteristics of the ternary circuits, they are not general to be utilizable for the timing analysis of ternary circuits. In [10] and [11], the ternary functions are implemented by a pull-up pull-down structure similar to the CMOS logic circuits. Therefore, dedicated sensitization conditions should be considered for each ternary function, which complicates the timing analysis procedure. Also, no synthesis procedure has been introduced in the works [12] and [13] for implementing the ternary logic functions with the proposed ternary gates.

Methods presented in [7] through [9] are general enough that can produce logic circuits from ternary truth tables. As stated, the work in [9] has the best transistor count. It utilizes 2:1 ternary multiplexers and some ternary inverters to implement ternary functions. It is known that no sensitization condition is required for the inverters since they have only one input line. On the other hand, defining sensitization conditions for 2:1 multiplexers that have a low number of inputs is very acceptable. Thus in this work, a static timing analysis method equipped with dynamic path sensitization is proposed for identifying the longest true paths (LTPs) of the ternary circuits implemented by the method of [9].

To the best of our knowledge, this is the first static timing analysis method, equipped with dynamic path sensitization, for ternary circuits. It should be noted that, although the dynamic path sensitization is utilized in this proposed work, glitches are not considered in the timing analysis, like the existing sensitization criteria.

Considering glitches severely complicates the overall process and they are ignored here. Moreover, in this work, the identified critical paths and their worst-case delays are not utilized in the optimization of circuit parameters (like circuit delay and power consumption) and thus, the impact of one parameter optimization on the other parameters is not studied here.

The main contributions of this work are listed below:

1. Dynamic sensitization conditions for 2:1 multiplexers presented in [9] (that are used in constructing the ternary circuits) are introduced.
2. A dynamic programming procedure for identifying the true and false paths of the circuit by using static timing analysis is proposed.
3. An event-driven simulation engine is introduced that is utilized in confirming the sensitization state of the identified paths.
4. Carry skip adders, which are known to have many false paths, in addition to some other arithmetic circuits are designed by ternary logic as benchmark circuits to depict the effectiveness of the proposed identification algorithm in finding the true and false paths of the circuit.

The paper organization is as follows. Section 2 describes the proposed sensitization conditions for 2:1 multiplexers. The implementation considerations for the procedure of finding the longest true paths of the circuit are presented in Section 3. In Section 4, utilizing the dynamic programming in the path identification procedure is explained. The proposed algorithm for the mentioned path identification procedure is presented in Section 5. In Section 6 the experimental results are provided. Finally, the conclusions are made in Section 7 and the proposals for future works are also presented.

2 Proposed Sensitization Conditions

Fig. 1 (obtained from [9]) shows the implementation of PTI-Mux (a 2:1 ternary multiplexer with a PTI ternary inverter as its select input) and NTI-Mux (a 2:1 multiplexer with an NTI gate as its select input) by CNTFET devices that are used in the circuit synthesis procedure of [9]. The number beside each device shows the carbon nanotube diameter of the CNTFET device. Since PTI and NTI inverters only produce values 0 and 2 (and not 1), each multiplexer has only two data input lines. NTIB and PTIB components in Fig. 2(a) (obtained from [9]) are equivalent low-cost gates for NTI-Mux and PTI-Mux, respectively when the input lines of the multiplexers have special constant values. Fig. 2(b) (obtained from [9]) depicts the CNTFET-based implementation of the binary NOT required in the circuits of NTIB and PTIB components.

2.1 Circuit Paths

Fig. 3 (obtained from [9]) shows the circuit implementation for a sample ternary function with two

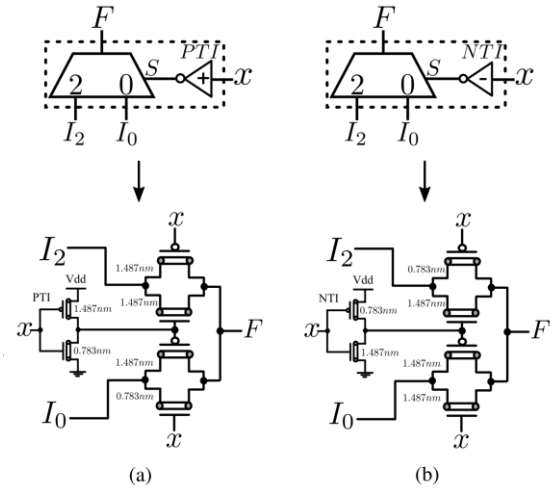

Fig. 1 Implementation of ternary 2:1 multiplexers of [9] by CNTFET devices; a) PTI-Mux and b) NTI-Mux.

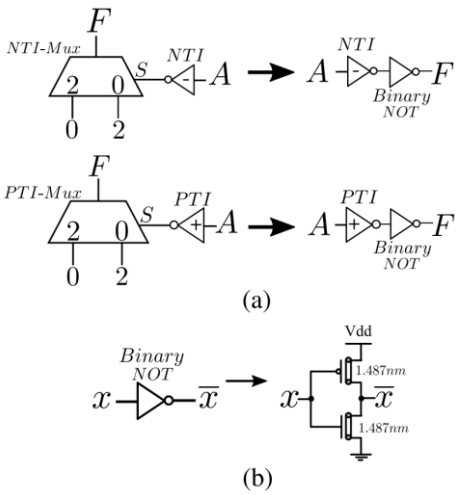

Fig. 2 a) NTIB and PTIB components and b) Implementation of binary NOT.

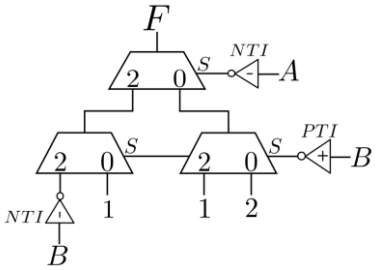

Fig. 3 Circuit implementation for a sample ternary function with two inputs.

inputs. In the circuits synthesized by the method of [9], primary inputs are used to feed the inverters connected to the select lines of the multiplexers. Only one primary input can be used in the data input lines of the multiplexers (input *B* in data line 2 of the left multiplexer). Although a circuit path in these circuits can be started from an input line of a multiplexer that has a constant value, in this work, the head of circuit paths can only be the circuit's primary inputs. This

approach is common in the existing timing analysis methods in which the pass of the transitions, created by the value changes in the circuit's primary inputs, is analyzed.

In this paper, a *physical path* denotes a circuit path constructed by circuit components and leads. A *stimulated path* is a physical path transmitting a transition through itself. For sensitizing a stimulated path, each path component should be sensitized.

2.2 Sensitization Conditions for 2:1 Multiplexers

In this work, like Chen-Du criterion and viability analysis, floating mode analysis [15] is considered in which, the initial value of each circuit node is considered as unknown value 'X' (it differs from the value used in the ternary literature instead of logic value '1'). All transitions on the circuit lines are from X to a known value 0, 1, or 2.

During static timing analysis of a binary circuit, sensitization of a circuit gate to pass a transition is determined by whether the gate inputs have controlling or non-controlling values. It is worth noting that the controlling and non-controlling values of a binary AND (OR) gate are 0 and 1 (1 and 0), respectively. However, here in which we use multiplexers as circuit components, the controlling and non-controlling values have no meaning and new sensitization conditions should be declared.

Consider a 2:1 multiplexer, named *M*, with input lines I_0, I_2, and *S* (direct select line of *M* after NTI or PTI) and output line *F*. For sensitizing *M* to pass transition *Tr* on one of its input lines toward *F*:

1) if *Tr* appears on I_0 at time *t*, *S* should be stable at value 0 at time $t_S \le t$;
2) if *Tr* appears on I_2 at time *t*, *S* should be stable at value 2 at time $t_S \le t$;
3) if *Tr* appears on *S* at time *t*, and *S* has value 0, I_0 should be stable at a known value 0, 1, or 2 (and not X) at time $t_{I0} \le t$;
4) if *Tr* appears on *S* at time *t*, and *S* has value 2, I_2 should be stable at a known value 0, 1, or 2 (and not X) at time $t_{I2} \le t$;

If the data line selected by *S* has an X value, no transition appears on *F* since its current value is X (avoided by conditions 3 and 4). If a partial stimulated path reaches I_0 or I_2 and the sensitization conditions for *M* are satisfied, the same transition on the data input line appears on *F*. However, other transition than the one that reaches *S* through a partial stimulated path may be created on *F*. For example, if a partial path having an X-to-0 transition terminates on *S*, and if I_0 has value 1 at the time the transition reaches to *S*, an X-to-1 transition is created on *F*.

3 Implementation Considerations for the Longest True Paths Identification Procedure

Instead of finding only the longest true path of the circuit, in this work, several longest true paths are identified since they are utilizable in the optimization algorithms like area reduction of the circuit by gate re-sizing. For the circuits implemented by Boolean logic, the sensitization criteria are formulated in the form of CNF expressions. As is known, the literals used in the CNF formulas for binary logic can get binary values 0 and 1. Boolean satisfiability solvers are used to check whether the CNF formula for sensitizing a circuit path is satisfiable or not.

As is apparent from the proposed sensitization conditions, the condition for sensitizing a ternary multiplexer is that one of its fan-ins should have a known value of 0 or 2, or should have each of known values 0, 1, or 2. For case 1 (2), it should be checked if *S* has a value 0 (2) or not. Thus, the decision here is binary and a CNF literal can be considered for *S* which shows if *S* has ternary value 0 or not. For *S* having ternary value 1 or not, and having ternary value 2 or not, two other literals can be defined. For case 3 (4) of the proposed sensitization conditions, it should be checked if I_0 (I_2) has value 0, or has value 1, or has value 2, or none of these values. Similar literals mentioned above for *S* can be considered for I_0 (I_2) and a CNF formula which expresses the conditions in case 3 (4) as a Boolean expression can be developed.

Therefore, the presented ternary sensitization conditions can also be formulated by CNF expressions. The satisfiability of expressions is investigated by SAT solver, MINISAT solver [16] in this work. Three binary variables v_0, v_1, and v_2 are considered for each circuit line. If v_i, for i = 0, 1 and 2, becomes 1 the line has value *i*. In this case, the two other binary variables have a value of 0.

3.1 CNF Clauses for Different Kinds of Circuit Lines

For a circuit line having constant value *C*, a CNF clause is considered indicating that v_C is 1. Two other clauses are included to state that the line does not have a value other than *C*. Since circuit primary input *PI* can have any of the values 0, 1, or 2, clauses related to the following Boolean expression should be considered:

$$PI_0.\overline{PI_1}.\overline{PI_2} + \overline{PI_0}.PI_1.\overline{PI_2} + \overline{PI_0}.\overline{PI_1}.PI_2 \quad (1)$$

For an NTI named *N* with input line *A*, clauses related to the following Boolean expressions are included:

$$N_0 = A_1 + A_2 \;;\; N_1 = 0 \;;\; N_2 = A_0 \quad (2)$$

Expressions for the other types of ternary inverters are similar.

A multiplexer named *M*, with input lines *A*, *B*, and *S*, requires clauses for the following Boolean expressions:

$$M_i = S_0 A_i + S_2 B_i \quad \text{for} \quad i = 1,\ 2,\ 3 \quad (3)$$

The multiplexers and inverters do not need the clauses

considered for primary inputs. The reason can be explained as follows. Mutual exclusiveness of variables v_i, for i = 0, 1, 2, is preserved for primary inputs and constant values. When an NTI named N is fed by the primary input or the constant value A, it is apparent from the equations of (2) that N_0 and N_2 cannot be 1 at the same time since A_1 + A_2 and A_0 are mutually exclusive.

Moreover, consider data input lines A and B of multiplexer M are driven by primary inputs or constant values. Since the select line S is driven by an NTI or PTI, S_1 is 0, and S_0 and S_2 are mutually exclusive (the reason was explained in the previous lines). If S has value 0, then M_i becomes equal to A_i, for i = 0, 1, 2. Since A_i's are mutually exclusive, M_i's will also be mutually exclusive. The same situation happens when S has value 2.

Therefore, variables v_i's for all inverters and multiplexers that are driven by primary inputs or constant values are mutually exclusive. Thus, the components at the next circuit level receive variables v_i's which are mutually exclusive, and by using similar analysis explained above, v_i's of their output lines become also mutually exclusive. A recursive procedure in this manner on the circuit components reveals that the variables v_i's for all remaining inverters and multiplexers are mutually exclusive as well.

4 Implementing the Path Identification Procedure Using Dynamic Programming

The *best-first* search method can be used to find the true critical paths one by one. In this method, recently-obtained sensitized partial paths are stored in a priority queue. Partial paths are ordered in the queue by using the path *esperance* (*ESP*), the delay of the longest path that can extend the partial path. If the esperance of a partial path P_1 is greater than the esperance of a partial path P_2, the sensitization analysis of P_1 is performed before the analysis of P_2.

During the path identification process, if a partial path is required to be extended by one of the fan-out lines of the last path component, the one that has the biggest Remaining Time (RT) is chosen. The remaining time of a line is the delay of the longest partial path (not necessarily sensitizable) from the line towards one of the circuit output lines.

4.1 Setting Value on a Circuit Line before a Specified Time

In the sensitization conditions presented in Subsection 2.2, it is necessary to set a value on a circuit line in a time no greater than a specified time. To achieve this, first, the CNF clauses of the line should be included based on the circuit component which drives the line and all the circuit components on the fan-in cone of the driving component. Then the clause for setting the required value is added. However, the satisfaction of these clauses does not guarantee that the value set is performed not after a specified time. In other words, the sensitization conditions expressed in this way are static, and not dynamic, since the occurrence time of the transitions is not considered at all.

To solve this problem, the set S_{TPP} of sensitized stimulated partial paths terminating on a circuit line are stored in that circuit line, for all lines of the circuit. Consider a multiplexer M with input lines l_i, for i = 0, 1, 2, and output line ol. Suppose that the current path which is under sensitization analysis is the path P_k, a sensitized stimulated partial path that reaches to l_k at time t_{spec}. Also consider that for sensitizing M to transmit the transition on P_k, it is required to set value v on l_j, for $j \neq k$, not after time t_{spec}.

Let the set $S_{TPP\text{-}j\text{-}v}$ be a subset of S_{TPP} of l_j containing those partial paths which deliver to l_j a transition to v. For a $P \in S_{TPP\text{-}j\text{-}v}$, we have ESP($P$) ≥ ESP($P_k$) since it has been analyzed before P_k in the best-first procedure. Suppose D_P is the delay of P and D_M is the delay of M. For a partial path Q terminating on one of the fan-ins of M with delay D_Q, ESP(Q) = D_Q + D_M + RT(ol). By knowing that the delay of P_k is equal to t_{spec}, we can write:

$$\begin{aligned} &ESP(P) \geq ESP(P_k) \rightarrow \\ &D_P + D_M + RT(ol) \geq t_{spec} + D_M + RT(ol) \rightarrow \\ &D_P \geq t_{spec} \end{aligned} \tag{4}$$

Thus, all paths in $S_{TPP\text{-}j\text{-}v}$ have delays greater than t_{spec}. On the other hand, there is no other partial path that can set value v on l_j after t_{spec}, since if so, it shall be included in $S_{TPP\text{-}j\text{-}v}$ during the best-first procedure. Therefore, it is required to add clauses that prevent all paths in $S_{TPP\text{-}j\text{-}v}$ from being sensitized to satisfy the timing requirement needed for setting value v on l_j. Let {l_1, l_2, …, l_n} be the circuit lines that construct path P. Suppose T_i is the transition on l_i. To sensitize path P, all conditions C_i's for transmitting all T_i's should be satisfiable. To prevent P from being sensitized, the contrary of the above condition should be provided, i.e., making at least one C_i not satisfiable. For $C_{!P}$, the condition of P not being sensitized, we can write:

$$C_{!P} = \overline{C_0} + \overline{C_1} + \cdots + \overline{C_n} \tag{5}$$

To prevent setting value v on l_j after time t_{spec}, all paths P_i's, i =1…m, in $S_{TPP\text{-}j\text{-}v}$ should not be sensitized. Thus, clauses related to the following Boolean expression should be included:

$$C_{!P_0}.C_{!P_1}.\ldots.C_{!P_m} \tag{6}$$

5 Overall Algorithm for Path Identification

The overall algorithm for identifying the longest true

paths of the circuit, using static timing analysis and dynamic programming method is presented in Fig. 4. In line 1, the graph of the synthesized circuit, G_C, is constructed. Then, by considering the subjects in Subsection 3.1, CNF formulas for each line of G_C are generated (according to the circuit component that drives the line). In line 3, the remaining times of all circuit lines are calculated by a simple block-based static timing analysis. These values are used in the computation of the esperances of partial paths. A priority queue, named *frontier*, is used to store the recently-obtained sensitized partial paths in order, for being used in the best-first search procedure. The ordering is descending based on the esperances of the partial paths.

Initially, each partial path contains only one primary input line. For each primary input line, three partial paths having to-0, to-1, or to-2 transitions are created and inserted into *frontier*. The main loop of the algorithm in line 5 continues to repetition when *frontier* is non-empty. In the mentioned loop, first, a partial path *cp* is brought out from *frontier*. Surely, it is the path with the greatest esperance. Then, the sensitization condition of *cp* is analyzed (line 5.2). The related procedure is explained later.

If the sensitization condition is not satisfied, the *cp* is a false path and it should be discarded (line 5.4.1). Otherwise, *cp* is added to the S_{TPP} set of the tail line of *cp*, tl_{cp}. If *cp* arrives in a primary output line, it is a complete true path and is reported (line 5.3.2.1). If the number of found complete paths reaches the demanded number of paths ($N_{demanded}$), the algorithm exits from the main loop. Otherwise, the next partial path in *frontier* should be analyzed. If *cp* is not a complete path yet, *cp* should be extended. Each fan-out line *fol* of tl_{cp} should be examined (line 5.3.3) since the extension through each *fol* may result in a false path.

During the extension of a partial path, it is possible that multiple transitions being considered for each *fol* of tl_{cp} to be examined. Thus, a set S_{TT} is considered for each circuit line that contains the candidate transitions. If the driving component of *fol*, C_{drive}, is a ternary inverter, only one transition, according to the type of inverter, is placed in S_{TT} of *fol* (line 5.3.3.1.1). If C_{drive} is a multiplexer and tl_{cp} is one of its data lines, S_{TT} of *fol* again has only one transition (same as the transition on tl_{cp}).

However, if tl_{cp} is the select line of a multiplexer, X-to-0, X-to-1, and X-to-2 transitions are considered for S_{TT} of *fol*. The reason can be explained in brief. Although the transitions of the lines of the partial path under analysis are known, the transitions on the other circuit lines are not designated yet. The SAT solver only reports that the constraints for sensitizing the lines of a partial path are satisfiable and proposes only one value assignment to the circuit lines which satisfies the conditions.

However, there may be several other satisfying value assignments. Therefore, the transition on the input data line of the multiplexer, selected by the value of the transition on tl_{cp}, is not known yet, and consequently, the selected data line may have any of the above-mentioned three transitions. Thus, any of these transitions should be considered for S_{TT} of the output line of the multiplexer (*fol*) and should be examined.

In the next step, *ext_path* is created by appending the *fol*, having a transition from S_{TT}, to *cp* (line 5.3.3.4.1). *ext_path* is inserted into *frontier* to be analyzed afterward.

In the *analyze_sensitization_of_path*() function, clauses required for sensitizing the last component of *cp* are added to the clauses for the other components of *cp*. If tl_{cp} is a primary input line, CNF formulas of expression (1) in addition to three more clauses are considered for the CNF formula of *cp* according to the transition considered for tl_{cp}. The additional three formulas are alike the ones for the lines with a constant value. If tl_{cp} is fed by an NTI (or another kind of ternary inverter), clauses related to the expressions in (2) (or similar ones for another kind of ternary inverter) are added to the CNF formula of *cp*.

For a tl_{cp} driven by a multiplexer *M* with select line *S*, clauses for formulas in expression (3) are added. Moreover, the clauses for all lines in the fan-in cone of *M* are also included. If the previous path line of tl_{cp} is a data line of *M*, three clauses are added to set the proper value on *S* for selecting the data line. Otherwise, according to the transition on *S*, one of the data lines, say L_D, is selected. Based on the considered transitions in S_{TT} of tl_{cp} (see lines 5.3.3.4 and 5.3.3.4.1), three clauses are added to make L_D have the proper transition.

Algorithm 1

1. construct the circuit graph, G_C
2. generate CNF formula for each line of G_C
3. find remaining times of all lines of G_C
4. fill *frontier* by partial paths having transitions on primary input lines
5. **while** *frontier* is not empty **do**
5.1. bring out *cp* from *frontier*
5.2. *sens_cond* = *analyze_sensitization_of_path*(*cp*)
5.3. **if** *sens_cond* is **true then**
5.3.1. add *cp* to S_{TPP} set of the tail line of *cp*, tl_{cp}
5.3.2. **if** tl_{cp} is a primary output line **then**
5.3.2.1. report *cp*
5.3.2.2. **if** $N_{found} == N_{demanded}$ **then**
5.3.2.2.1. **break**
5.3.2.3. **continue**
5.3.3. **for** each fan-out line *fol* of tl_{cp} **do**
5.3.3.1. **if** C_{drive} of *fol* is a ternary inverter
5.3.3.1.1. S_{TT} = *obtain_trans_type_of_ternary_inv*(*fol*, tl_{cp})
5.3.3.2. **else if** C_{drive} of *fol* is a Mux and tl_{cp} is a data input line of Mux
5.3.3.2.1. S_{TT} = *obtain_trans_type_of_ternary_mux*(*fol*, tl_{cp})
5.3.3.3. **else if** C_{drive} of *fol* is a Mux and tl_{cp} is the select line of Mux
5.3.3.3.1. S_{TT} = {$Tr_{to\text{-}0}$, $Tr_{to\text{-}1}$, $Tr_{to\text{-}2}$}
5.3.3.4. **for** each transition *Tr* in S_{TT} **do**
5.3.3.4.1. *ext_path* = {*cp*, *fol* with *Tr*}
5.3.3.4.2. insert *ext_path* into *frontier*
5.4. **else**
5.4.1. discard *cp* // this partial path is not useful

Fig. 4 Overall algorithm for identifying the longest true paths of the circuit.

As stated in Subsection 4.1, it is required the value setting of L_D happens not after the transition on S. To achieve this, CNF clauses of expression (6) for the related partial paths in S_{TPP} of L_D should be added to the CNF formulas of *cp*. After preparation of the formulas for *cp*, SAT solver is called to investigate the satisfiability of formulas.

6 Experimental Results

The unit delay model is used in this work to assign propagation delays to the circuit components. From the circuits in Figs. 1 and 2, it can be deduced that the output line value of a component is set by the transmission of a voltage level through a transistor or a transmission gate, and thus the considered assumption is not irrational. For the NTIs and PTIs driving the select lines of multiplexers, a separate unit delay is considered. Because of the appended Binary NOT, a 2-unit delay is assigned for NTIBs and PTIBs components.

Carry Skip Adders (CSKAs) are known to have many false paths. In this work, some ternary CSKAs are implemented by the method of [9] as benchmark circuits. Block diagram of a CSKA 2×2 is depicted in Fig. 5. For $i \in \{0, 1\}$, P_i denotes whether the i-th full-adder (FA_i) is in the propagation mode, based on the values of A_i and B_i. If both P_0 and P_1 have logical value 2, the multiplexer selects C_{in}. Otherwise, c_2 is placed on C_{out}.

The multiplexer-based implementation of the full-adder is obtained by using the synthesis method of [9], given the truth table of the full-adder. The truth table for the combination of the AND gate and the carry-select multiplexer is also provided for the synthesis tool. By connecting the obtained multiplexer-based circuits, the ternary circuit of CSKA is generated. In addition to the CSKA circuits, *sum_i*, *prod_i*, and *count_i* circuits are also included as benchmark circuits. *sum_i* circuit adds i ternary digits and *prod_i* circuit calculates the multiplication of i ternary digits. Also, the *count_i* circuit generates the number of 1's and 2's of i ternary digits. Note that these circuits are obtained totally by the synthesis method of [9], and not by the structural approach used for CSKA circuits.

6.1 Event-Driven Ternary Simulation Engine

An event-driven ternary simulation engine (TSE) is developed to verify the results of the identification algorithm. It performs some kind of dynamic timing analysis. All the possible combinations of values for primary inputs are applied to the engine and all the complete paths sensitized by each input vector are reported. To-X transition (transition to the unknown value) is also included in the simulation. They occur when a multiplexer selects a data line having an X value. If a to-X transition occurs on a circuit line at the same time as another transition to a non-X value, the to X transition is ignored.

The algorithm of the proposed simulation engine is depicted in Fig. 6. The simulation continues while there are no-handled events in the circuit lines. The transition events on all circuit lines are held in a sorted list named SL_{TE}. The ordering is ascending based on the event time. This list is initialized at the beginning of the simulation by the transition events on the primary input lines, according to the input vector *IV*. At each step, the earliest transition event (*ete*) is brought out from the sorted list, the simulation time is updated accordingly,

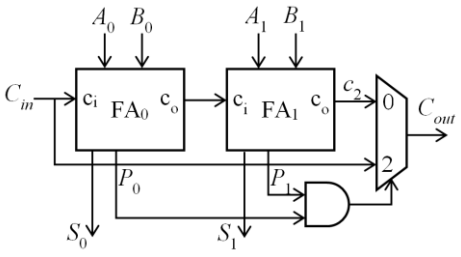

Fig. 5 Block diagram of a CSKA 2×2.

Algorithm 2

1. **for** each input vector *IV* **do**
1.1. insert transition events related to the primary input lines under *IV* to the sorted list SL_{TE}
1.2. **while** SL_{TE} is not empty **do**
1.2.1. bring out the earliest transition event *ete* from SL_{TE} and update the simulation time
1.2.2. l_c = the line that contains *ete*
1.2.3. **if** transition of *ete* is to the current value of l_c **then**
1.2.3.1. **continue**
1.2.4. **for** each fan-out line *fol* of l_c **do**
1.2.4.1. **if** C_{drive} of *fol* is a ternary inverter **then**
1.2.4.1.1. **if** *ete* == $Tr_{to\text{-}X}$ **then**
1.2.4.1.1.1. create new transition event *new_te* with $Tr_{to\text{-}X}$ for *fol*
1.2.4.1.2. **else**
1.2.4.1.2.1. create *new_te* for *fol* according to inverter type of C_{drive}
1.2.4.2. **else** // it is a ternary 2:1 multiplexer
1.2.4.2.1. **if** l_c is connected to the first data input line of C_{drive} **then**
1.2.4.2.1.1. **if** the select input of C_{drive} has value 0 **then**
1.2.4.2.1.1.1. create *new_te* same as *ete* for *fol*
1.2.4.2.2. **else if** l_c is connected to the second data input line of C_{drive} **then**
1.2.4.2.2.1. **if** the select input of C_{drive} has value 2 **then**
1.2.4.2.2.1.1. create *new_te* same as *ete* for *fol*
1.2.4.2.3. **else if** l_c is connected to the select input of C_{drive} **then**
1.2.4.2.3.1. **if** *ete* == $Tr_{to\text{-}X}$ **then**
1.2.4.2.3.1.1. create *new_te* with $Tr_{to\text{-}X}$ for *fol*
1.2.4.2.3.2. **else if** *ete* == $Tr_{to\text{-}0}$ and first data input line has value *v* **then**
1.2.4.2.3.2.1. create *new_te* with $Tr_{to\text{-}v}$ for *fol*
1.2.4.2.3.3. **else if** *ete*==$Tr_{to\text{-}2}$ and second data input line has value *v* **then**
1.2.4.2.3.3.1. create *new_te* with $Tr_{to\text{-}v}$ for *fol*
1.2.4.2.3.4. **else if** *ete* == $Tr_{to\text{-}1}$ **then**
1.2.4.2.3.4.1. report an error!
1.2.4.3. **if** *new_te* == $Tr_{to\text{-}X}$ and *fol* has another transition at current simulation time to a non-X value **then**
1.2.4.3.1. remove *new_te*
1.2.4.4. **else**
1.2.4.4.1. insert *new_te* in SL_{TE}
1.3. store complete circuit paths sensitized by *IV*

Fig. 6 Algorithm of the proposed event-driven ternary simulation engine.

and the line that contains the event (l_c) is determined. If the transition is to the value that l_c currently has, the event is ignored. Otherwise, new transition events (*new_te*s) are created for the circuit lines (*fol*s) driven by the fan-out components of l_c.

If the fan-out component C_{drive} is a ternary inverter, a to-X transition event is created for *fol* if *ete* is the to-X transition. If *ete* is not the to-X transition, a proper transition event is created for *fol* according to the type of ternary inverter. For a ternary 2:1 multiplexer, the output transition depends on the place of the current transition. If l_c is connected to the first data input of the multiplexer, the value of the select input of the multiplexer is examined. If it is value 0, the same transition as *ete* (albeit for a time after the current time, according to the delay of C_{drive}) is created for *fol*. Otherwise, the transition is blocked and no event is created for *fol*. A similar situation holds when l_c is connected to the second data input of the multiplexer.

When l_c is connected to the select input of the multiplexer, if *ete* is a to-X transition, a to-X transition is created for *fol*. If *ete* is a to-0 transition and the current value of the first data input of the multiplexer is value *v*, a to-*v* transition event is created for *fol*. The same process is performed when l_c has a to-2 transition. A to-1 transition on l_c is treated as an exception since the select line of the 2:1 multiplexer can only have values 0 and 2. After the creation of *new_te*, this event is also inserted in SL_{TE} to be processed in the future, as the simulation time advances. After handling all the transition events, those complete circuit paths that are sensitized by the input vector are stored. The above-mentioned process is repeated for the other input vectors.

6.2 Results

The proposed path identification algorithm, in addition to the developed simulation engine, is implemented in C++. A machine, with an Intel Core i5 CPU (7500 at 3.40GHz) and 8GB of RAM, running with the Linux operating system, is employed to obtain the results.

Table 1 shows information about the number of components and physical paths of the benchmark circuits. The number of PTI- and NTI-Muxs, in addition to the number of ternary inverters used in the synthesized circuit, are presented under columns N_{MUX} and N_{INV}, respectively. The number of complete physical paths (N_{PP}) and the number of false complete physical paths (N_{FPP}) of the circuits are reported in the two next columns. The last column depicts the percentage of N_{FPP} relative to N_{PP}. Note that the considered paths are complete. False physical paths are those not sensitized by any input vector.

Since the proposed algorithm uses many calls to the SAT solver for identifying the sensitization state of circuit paths, obtaining all false paths of the circuit by the proposed algorithm is slower than using the simulation engine. Thus, the reported values for N_{FPP} are provided by the simulation engine. Since identifying all false paths of CSKA 8×2 and 8×4 circuits takes a very long time (because of the large number of circuit paths, and the excessive number of input vectors, i.e., $3^{(2\times8+1)}$ = 129,140,163 vectors), they are not included.

Table 1 Number of components and physical paths of benchmark circuits

Circuit	N_{MUX}	N_{INV}	N_{PP}	N_{FPP}	[%]
FA	17	0	28	11	39.3
CSKA 2×2	42	4	178	87	48.9
CSKA 4×2	84	9	2740	1633	59.6
CSKA 4×4	90	4	2330	950	40.7
CSKA 8×2	168	19	275777	—	—
CSKA 8×4	180	9	286133	—	—
sum_4	38	2	115	43	37.4
sum_6	112	2	1190	417	35.0
sum_8	194	2	11597	3999	34.5
prod_4	27	1	50	13	26.0
prod_6	89	2	295	81	27.4
prod_8	210	2	1511	414	27.4
count_4	43	2	131	47	36.2
count_6	88	2	1280	439	34.3
count_8	142	2	13925	4553	32.7
Average					36.8

It is obvious from the outcomes that a significant percentage of the physical paths in ternary circuits are false under the unit delay model so that on average, about 36.8% of the physical paths of the circuits are false. Even for a simple FA with only 17 multiplexers, about 40% of the physical paths are indeed false.

Execution times of the proposed path identification algorithm and the event-driven ternary simulation engine are shown in Table 2. Column labeled D_{LTP} presents the delay of the longest true path of the circuit in unit time. D_{LTP} is greater for CSKA 4×2 than CSKA 4×4, and for CSKA 8×2 than CSKA 8×4. The reason is that along the path from the primary inputs lines to the primary output lines, for CSKA N×2, the number of multiplexers used for the carry selection is more than that for CSKA N×4.

The execution time in milliseconds for identifying the first longest true path is presented in the next column (t_{1st}). The execution times in milliseconds for identifying the 10th, 20th, 30th, 40th, and 50th longest true paths are reported in the next five columns. For FA, t_{50th} is empty since it has 37 longest true paths. Note that in this table, the results for the *stimulated* paths are reported. For each physical path, three stimulated paths are activated by placing transitions to ternary values on the head line of the physical path. This is the reason that the number of true paths of FA (= 37) is greater than the number of physical paths reported in Table 1 (= 28). As expected, the longest run times are for CSKA 8×2. Obtaining the first longest true path not later than 4 seconds and the 50th longest true path during 94

Table 2 Execution times of the proposed identification algorithm and the event-driven simulation engine.

Circuit	D_{LTP}	t_{1st} [ms]	t_{10th} [ms]	t_{20th} [ms]	t_{30th} [ms]	t_{40th} [ms]	t_{50th} [ms]	t_{TSE}
FA	7	216	1767	3293	4596	5698	—	138 ms
CSKA 2×2	12	448	2938	5413	8674	10156	11211	5.4 s
CSKA 4×2	21	1156	8174	12616	22254	27566	33497	1695.7 s
CSKA 4×4	20	987	5113	9577	14135	18600	22727	1825.1 s
CSKA 8×2	39	3923	21137	34975	58726	74371	93837	214 days*
CSKA 8×4	37	3297	13859	26390	38937	51531	64703	249 days*
sum_4	9	329	2016	4360	6375	7860	9513	2.4 s
sum_6	13	641	2984	5641	8969	12219	14734	139 s
sum_8	17	998	4125	7891	11938	15516	18984	3675.1 s
prod_4	10	297	1797	3985	5766	9000	9391	1.4 s
prod_6	14	594	2594	5484	7734	10255	12107	97.5 s
prod_8	18	2453	4937	7828	10718	13640	16703	4316.7 s
count_4	9	363	2362	4605	6781	8253	9477	2.8 s
count_6	12	604	2341	4413	6996	9587	11592	102.7 s
count_8	15	710	2935	5629	8513	11028	14237	2615.5 s

* Approximate run-times

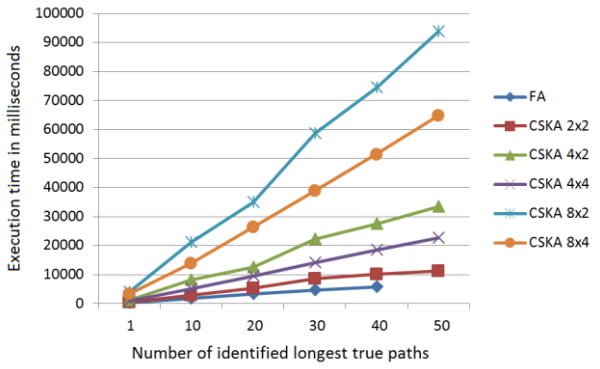

Fig. 7 Execution times of the proposed path identification algorithm for FA and CSKA circuits.

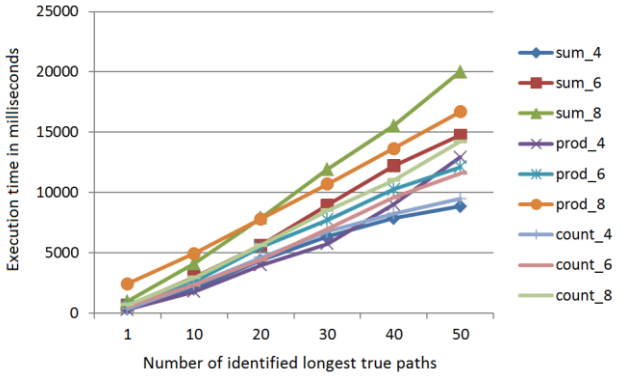

Fig. 8 Execution times of the proposed path identification algorithm for *sum_i*, *prod_i*, and *count_i* circuits.

seconds, for the considered benchmarks, is acceptable.

Moreover, by looking at the charts in Fig. 7, it can be observed that the run time increases almost linearly with the number of identified longest true paths, for the FA and CSKA benchmark circuits. The charts in Fig. 8 which are for the similar execution times for *sum_i*, *prod_i*, and *count_i* circuits confirm the linear increase of rum times with the number of identified longest true paths.

The last column depicts the execution times for the simulation of the benchmark circuits by the simulation engine under all possible input vectors. For 8-bit CSKAs the presented run times, which are quite long, are approximate. As stated in the previous lines, analyzing all the circuit paths under all input vectors is very time-consuming for these benchmarks and therefore it is ignored. Instead, the reported run times are obtained by measuring the simulation times of these circuits for 1000 input vectors and then scaling them for all 129,140,163 possible input vectors.

It should be mentioned that for all the benchmark circuits, except for the 8-bit CSKAs, all the true and false circuit paths are obtained by the simulation engine. Thus, by searching in the set of the true paths found by the simulation engine, examining the correctness of the proposed algorithm in the identification of the longest true paths of the circuit is possible.

On the other hand, it is reminded that if the SAT solver reports the satisfaction of the sensitization conditions of a circuit path, an input vector that provides the required values for the circuit lines can be extracted from the SAT solver report. Thus, for each 8-bit CSKA, the correctness of the algorithm can be inspected by performing simulation on each identified true path of the adder circuit under the input vector proposed by the SAT solver report. It should be noted that for both mentioned investigation methods, the results of the simulation runs confirm the correctness of the proposed algorithm in identifying the true critical paths.

Moreover, it should be mentioned that for the small circuits (like full-adder, CSKA 2×2, *sum_4*, and *prod_4*), the simulation engine can simulate all input vectors and evaluate the sensitization of all paths in less time than the proposed algorithm. However, for the larger circuits, if the number of demanded critical paths is restricted, the proposed algorithm identifies them very faster since the simulation engine must examine all the input vectors.

6.3 Comparison with Static Timing Analysis Equipped by Static Sensitization

It is noteworthy that to the best of our knowledge, there is no static timing analysis method for identifying the longest true paths of ternary circuits. Therefore, there is no other related work for comparing the quality of the presented method, and the correctness and the efficiency of the proposed path identification algorithm are verified by employing the developed TSE. Comparison of the obtained results with the outcomes of the Spice simulations is also not included. The reason is that Spice considers glitches and complex waveform shapes in its analysis [17]. Comprising such details in the proposed method strongly complicates the analysis and thus they are not included in the introduced algorithm.

However, to provide a comparison with another timing analysis method (in addition to the comparisons with TSE), the proposed method is compared with a static timing analysis which utilizes static path sensitization. As mentioned in the Introduction section, this work employs dynamic path sensitization. On the other hand, in static sensitization, delays are not investigated at all and only *stable* values on the circuit lines are considered. Of course, dynamic sensitization provides more accurate results.

Comparison between these two sensitization methods is reported in Table 3. N_{LTP} denotes the number of longest true paths considered for the comparison. In general, value 500 is devoted to this parameter. In cases where the number of all true paths identified by the dynamic sensitization is fewer, the number of paths is denoted in this column. $\overline{D_P^D}$ means average of the delays of true paths identified by the dynamic sensitization. $\overline{D_P^S}$ has the same meaning for the static sensitization. It should be mentioned that the delay of the longest true path of the circuit is the same for both sensitization methods, for all the considered circuits (reported under column D_{LTP} in Table 2).

By comparing $\overline{D_P^D}$ and $\overline{D_P^S}$ values in the table, it can be observed that in most cases $\overline{D_P^S}$ is greater than $\overline{D_P^D}$. This means that more critical paths are sensitized by static sensitization. However, for the FA circuit which is a small circuit, this relation is inverse. For CSKA 8×4, sum_6, and sum_8 circuits, these average delays are the same.

Since in static sensitization, fulfilling the timing relations between signal transitions is not included, overall sensitization conditions for sensitizing a circuit path are reduced in comparison to the dynamic sensitization conditions. Therefore, the number of CNF clauses decreases in the case of static sensitization and as a result, the satisfaction of these clauses can be easier. Thus, the number of critical paths being sensitized by static sensitization becomes greater. This can explain why in most cases $\overline{D_P^S}$ is greater than $\overline{D_P^D}$.

Table 3 Comparison between STAs with dynamic and static path sensitizations.

Circuit	N_{LTP}	$\overline{D_P^D}$	$\overline{D_P^S}$	t_{50th}^D [ms]	t_{50th}^S [ms]
FA	36	5.53	5.48	5698	4828
CSKA 2×2	171	9.07	10.16	11211	13141
CSKA 4×2	500	18.68	19.09	33497	34125
CSKA 4×4	500	18.31	18.63	22727	22315
CSKA 8×2	500	38.49	38.61	93837	94651
CSKA 8×4	500	36.86	36.86	64703	64875
sum_4	180	7.83	8.06	9513	9359
sum_6	500	12.39	12.39	14734	14282
sum_8	500	17	17	18984	19002
prod_4	69	7.53	8.54	9391	8954
prod_6	290	11.03	12.44	12107	12094
prod_8	500	17.15	17.25	16703	16313
count_4	195	8.13	8.32	9477	9284
count_6	500	10.90	11.27	11592	11814
count_8	500	14.22	14.26	14237	14195

On the other hand, dynamic sensitization can sensitize a path component that is not sensitized by static sensitization. This can be done by providing *temporal* sensitization conditions that the static approach does not profit from it. This can explain why in the case of FA, $\overline{D_P^D}$ is greater than $\overline{D_P^S}$.

The execution time in milliseconds for identifying the first 50 longest true paths by dynamic and static sensitization methods are denoted by t_{50th}^D and t_{50th}^S, respectively. Since the set of conditions for sensitizing a path by static approach is smaller than that for dynamic approach, it is expected the execution time of SAT solver to be shorter for the case of static sensitization and it is expected t_{50th}^S to be smaller than t_{50th}^D. Although this is true in some cases (for just a few tens of milliseconds), in other cases t_{50th}^S is greater. It should be noted that the sensitized paths by the two approaches are not necessarily identical. Thus, it seems that utilizing dynamic sensitization results in finding critical paths by examining fewer paths in comparison to the static approach. In summary, using static sensitization for speeding up the path identification process does not always work.

7 Conclusion

In this paper, path sensitization conditions, for ternary circuits based on 2:1 multiplexers, were introduced. An algorithm, implemented by dynamic programming, for identifying the longest true paths of ternary circuits based on the introduced sensitization conditions was presented. The correctness of the proposed algorithm in finding the longest true paths of the circuit was confirmed by an event-driven ternary simulation engine. The efficiency of the proposed algorithm was quite satisfactory for the considered benchmark circuits.

In the future, considering the realistic delay models (instead of the unit delay model) will be studied. Moreover, obtaining the longest true paths considering process variation will be investigated. In real-time systems, satisfying the timing constraints of the circuit is very important to avoid system failure. Critical paths obtained by the proposed work and their worst-case delays can be utilized in examining the satisfaction of the circuit timing constraints. Assessment of the impact of the identified critical paths of a ternary circuit which is utilized in a real-time application in meeting the system constraints and in system optimization is also a good subject to be studied in the future.

Intellectual Property

The authors confirm that they have given due consideration to the protection of intellectual property associated with this work and that there are no impediments to publication, including the timing of publication, with respect to intellectual property.

Funding

No funding was received for this work.

CRediT Authorship Contribution Statement

S. Abolmaali: Idea & conceptualization, Research & investigation, Analysis, Software and simulation, Verification, Original draft preparation, Revise & editing.

Declaration of Competing Interest

The authors hereby confirm that the submitted manuscript is an original work and has not been published so far, is not under consideration for publication by any other journal and will not be submitted to any other journal until the decision will be made by this journal. All authors have approved the manuscript and agree with its submission to "Iranian Journal of Electrical and Electronic Engineering".

References

[1] H. C. Chen and D. H. 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, Feb. 1993.

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

[3] P. C. Balla and A. Antoniou, "Low power dissipation MOS ternary logic family," *IEEE Journal of Solid-State Circuits*, Vol. JSSC-19, No. 5, pp. 739–749, Oct. 1984.

[4] A. Heung and H. T. Mouftah, "Depletion/enhancement CMOS for a lower power family of three-valued logic circuits," *IEEE Journal of Solid-State Circuits*, Vol. 20, No. 2, pp. 609–616, Apr. 1985.

[5] S. Shin, E. Jang, J. W. Jeong, and K. R. Kim, "CMOS-compatible ternary device platform for physical synthesis of multi-valued logic circuits," in *IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, Novi Sad, Serbia, pp. 284–289, 2017.

[6] J. Appenzeller, "Carbon nanotubes for high-performance electronics—Progress and prospect," in *Proceedings of the IEEE*, Vol. 96, No. 2, pp. 201–211, Feb. 2008.

[7] S. Lin, Y. Kim, and F. Lombardi, "CNTFET-based design of ternary logic gates and arithmetic circuits," in *IEEE Transactions on Nanotechnology*, Vol. 10, No. 2, pp. 217–225, Mar. 2011.

[8] B. Srinivasu and K. Sridharan, "A synthesis methodology for ternary logic circuits in emerging device technologies," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 64, No. 8, pp. 2146–2159, Aug. 2017.

[9] C. Vudadha, A. Surya, S. Agrawal, and M. B. Srinivas, "Synthesis of ternary logic circuits using 2:1 multiplexers," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 65, No. 12, pp. 4313–4325, Dec. 2018.

[10] S. Kim, S. Y. Lim, S. Park, K. R. Kim, and S. Kang, "A logic synthesis methodology for low-power ternary logic circuits," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 67, No. 9, pp. 3138–3151, Sep. 2020.

[11] A. D. Zarandi, M. R. Reshadinezhad, and A. Rubio, "A systematic method to design efficient ternary high-performance CNTFET-based logic cells," *IEEE Access*, Vol. 8, pp. 58585–58593, 2020.

[12] M. K. Q. Jooq, M. H. Moaiyeri, and K. Tamersit, "Ultra-compact ternary logic gates based on negative capacitance carbon nanotube FETs," *IEEE Transactions on Circuits and Systems II: Express Briefs*, Vol. 68, No. 6, pp. 2162–2166, 2020.

[13] X. Y. Wang, P. F. Zhou, J. K. Eshraghian, C. Y. Lin, H. H. C. Iu, T. C. Chang, and S. M. Kang, "High-density memristor-CMOS ternary logic family," *IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 68, No. 1, pp. 264–274, Jan. 2021.

[14] Y. Kang, J. Kim, S. Kim, S. Shin, E. S. Jang, J. W. Jeong, K. R. Kim, and S. Kang, "A novel ternary multiplier based on ternary CMOS compact model," in *IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL)*, pp. 25–30, May. 2017.

[15] S. Abolmaali, "Area reduction of combinational circuits considering path sensitization." *Iranian Journal of Electrical and Electronic Engineering*," Vol. 17, No. 3, pp. 1730–1730, Sep. 2021.

[16] N. Eén and N. Sörensson, "An extensible SAT-solver," in *International Conference on Theory and Applications of Satisfiability Testing*, Springer, Berlin, Heidelberg, pp. 502–518, May 2003.

[17] S. Abolmaali, "Efficient delay characterization method to obtain the output waveform of logic gates considering glitches," *Iranian Journal of Electrical and Electronic Engineering*, Vol. 15, No. 4, pp. 485–501, Dec. 2019.

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 timing analysis, approximate computing, ternary logic circuits, and low-power design.