Timing Analysis of Combinational Circuits using ADD's *

R. Iris Bahar Hyunwoo Cho † Gary D. Hachtel Enrico Macii ‡ Fabio Somenzi

University of Colorado
Dept. of Electrical and Computer Engineering
Boulder, CO 80309

Abstract

This paper presents a symbolic algorithm to perform timing analysis of combinational circuits which takes advantage of the high compactness of representation of the Algebraic Decision Diagrams (ADD's). The procedure we propose, implemented as an extension of the SIS synthesis system, is able to provide more accurate timing information than any other method presented so far; in particular, it is able to compute and store the true delay of the gate-level representation of the circuit for all possible input vectors, as opposed to the traditional methods which consider only the worst-case primary inputs combination. Furthermore, the approach does not require any explicit false path elimination. The information calculated by the timing analyzer has several practical applications such as determining the sets of critical input vectors, critical gates, and critical paths of the circuit, which may be efficiently used in the process of re-synthesizing the network for low-power consumption.

1 Introduction

In the design of modern IC's, ensuring that timing constraints are met is a fundamental issue. Circuit simulators such as SPICE are occasionally used for this purpose, but simulation is typically too time consuming to be applied to an entire VLSI circuit. An alternative way to estimate the delay of a digital network applies *timing analysis* methods; the advantage of these methods over the traditional simulation-based approaches is that they do not need test vectors to be supplied by the user; the timing behavior of the circuit is determined either by implicit input vector enumeration, or by implicit exploration of the paths of the network. Even if symbolic techniques are now available, input enumeration-based techniques may still be too expensive to be successfully used; therefore, in the last few years, researchers have spent considerable efforts in the development of path-oriented timing analysis techniques. These methods may provide only a rough estimate of the delay of the circuit, due to the possible presence in the network of *false paths*, that is, of paths that can not be sensitized by any input vector. The problem to be solved is then the calculation of the *true delay* of the circuit, that is, the delay of the circuit calculated along the longest sensitizable path, also called the *critical path* of the circuit. Since the delay of a circuit influences the choice of the clock cycle, a large difference between the longest false path and the critical path will result in posing unnecessarily conservative timing constraints on the design.

Traditional path-oriented techniques for timing analysis and verification consist of two processes: Path generation and path sensitization. In path generation, a list of topologically longest paths is generated and sorted in non-increasing order. In path sensitization, paths in the list are checked for being sensitizable, and false paths are eliminated from the list until the critical path is found. The choice of the sensitization conditions to be used to identify the critical path influences the correctness of the analysis [1]. If *static* sensitization conditions are used [2], the delay estimation is optimistic; the logical state of the gates along the path is not always properly taken into account, and this may lead to an underestimation of the critical path. In the case of *dynamic* sensitization [3], the problem which arises is not related to the correctness of the delay computation, but rather to its robustness. The circuit that the timing verifier manipulates is an idealized circuit, and the delays are given as a range of values from which the timing analyzer chooses the maximum value. The exact time at which a node settles to its final value is unknown, implying problems with maintaining *monotone speed-up* [1]. Identifying the critical path by means of the *viability* [4] criterion, on the other hand, results in a pessimistic timing analysis; the selected critical path might actually be a false path. The same thing can happen if the true path of the network is detected using the conditions proposed in [5], or the equivalent and independently derived *co-sensitization* conditions published in [6]. Beside sensitization-based methods, other timing analysis strategies have been proposed in the last few years; some of them are based on explicit critical path extraction [7], others on the computation of the true delay using timed test generation-like [8] or probabilistic [9] techniques.

In this paper we present a symbolic algorithm to perform timing analysis of combinational circuits which takes advantage of the high compactness and ease of manipulation of the ADD data structure [10]. The analysis procedure we propose, implemented as an extension of the SIS synthesis system [11], is able to provide more accurate timing information than any other method presented so far. In particular, it is able to compute and store the true delay (i.e., the length of the critical path) of the gate-level representation of the network for all possible combinations of the primary inputs, as opposed to the traditional methods which consider only a single worst-case input vector. Furthermore, the approach does not require any explicit false path elimination. The information calculated by the timing analyzer has several practical applications such as determining the set of critical input vectors (i.e., input vectors which sensitize a critical path), the set of critical gates (i.e., the set of gates which belong to a critical path), and the set of critical paths of the circuit, which may be efficiently used in the process of re-synthesizing the network for low-power consumption.

The rest of this paper is organized as follows. Section 2 gives definitions and notation for subsequent usage. Section 3 describes our ADD-based algorithm for true delay calculation. Section 4 shows preliminary timing analysis results obtained on selected benchmark circuits, some of which are from [12]. Finally, Section 5 is devoted to conclusions and future work.

*This work was supported in part by NSF/DARPA grant MIP-9115432 and SRC contract 92-DJ-206.

†Hyunwoo Cho is with Motorola Inc., Austin, TX 78735.

‡Enrico Macii is also with Politecnico di Torino, Dipartimento di Automatica e Informatica, Torino, ITALY 10129.

2 Background

2.1 Definitions and Notation

In the following section we give the most fundamental definitions concerning combinational circuits, and paths in combinational circuits. The notation we use throughout this paper is partially taken from [13].

A *combinational circuit* (or *network*) is a directed acyclic graph composed of gates (or nodes) and connections (or edges) between gates. Given some input vector x, a k-input gate g_i has k delays, $d(g_{ij}, x)$, one for each input. A *path* in a combinational circuit is a sequence of gates, $(g_0, \ldots, g_n)$, where the output of gate g_i, $0 \leq i < n$, connects to an input pin of gate g_{i+1}.

If the output of a gate, g_i, is connected to an input of a gate, g_j, then g_i is a *fanin* of g_j. Gate g_j is a *fanout* of gate g_i. The *length* of a path, $P = (g_0, \ldots, g_n)$ is defined as $d(P, x) = \sum_{i=0}^{n-1} d(g_{ij_i}, x)$, where j_i is the pin connected to the previous gate g_i in the path. If no delay model is specified, the *delay* (or *topological delay*) of a combinational circuit is the maximum over all x of the length of its longest path. An *event* is a transition $0 \rightarrow 1$ or $1 \rightarrow 0$ at a gate. Given a sequence of events, $\{e_0, e_1, \ldots, e_n\}$, occurring at gates $\{g_0, g_1, \ldots, g_n\}$ along a path, such that e_i occurs as a result of event e_{i-1}, the event e_0 is said to *propagate* along the path. A *controlling value* at a gate input is the value that determines the value at the output of the gate independent of the other inputs. For example, 0 is a controlling value for AND/NAND gates, while 1 is a controlling value for OR/NOR gates. A *non-controlling value* at a gate input is the value which is not a controlling value for the gate. For example, 1 is a non-controlling value for AND/NAND gates, while 0 is a non-controlling value for OR/NOR gates.

The *arrival time*, $AT(g_i, x)$, is the time at which the output of gate g_i settles to its final value if input vector x is applied at time 0. Under a specified delay model, a path $P = (g_0, \ldots, g_n)$ is said to be *sensitizable* if an event e_0 occurring at gate g_0 can propagate along P. The *critical path* of a circuit is the longest sensitizable path under a specified delay model; if a path is not sensitizable, then it is a *false path*.

2.2 Operating Modes and Delay Models

Let us consider the operation of a circuit over the period of application of a sequence of input vectors. Let v_i be the vector applied at time t_i. In the *floating mode* of operation, the nodes of the circuit are not assumed to be ideal capacitors, and therefore their state is unknown until it is set by the current input vector. In the *transition mode* of operation, on the other hand, the nodes of the circuit are assumed to be ideal capacitors, and hence they retain their value set by the previous input vectors until the current vector forces the voltage to change. Thus, in the transition mode of operation, the timing response of each gate of the circuit to v_i is also a function of v_{i-1}, while in the floating mode of operation the response to v_i depends only on v_i itself.

The overall timing response of a circuit is computed by first estimating the delay of each gate in the circuit. The simplest way to model the delay of a gate is to assign to each pin j of gate g a fixed number, $d(g_j, x)$, which represents the time required by the output to assume the correct value in response to a voltage change on input j under the input vector x. This model, called the *fixed gate delay model*, indicates that if an event occurs at pin j of gate g at time t_0, the output of gate g will settle to its final value at time $t_0 + d(g_j, x)$. In reality, $d(g_j, x)$ is typically an upper bound on the expected delay, so in fact the actual delay may be some value in the range $[t_0, t_0 + d(g_j, x)]$. This is called the *upper-bounded* gate delay model. The upper-bounded model

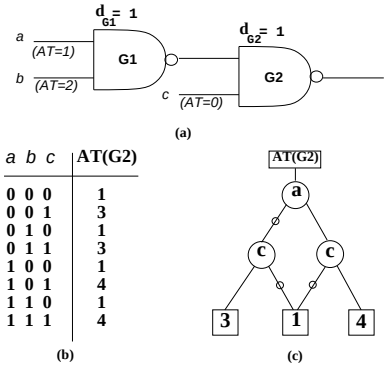

Figure 1: (a) A Combinational Circuit. (b) Its Output Arrival Times. (c) The Corresponding ADD.

in floating mode guarantees the *monotone speed-up* property: If any gate along the path to some gate g is made faster, then the arrival time at the output of g will not increase. Another model, the *bounded gate delay model*, assumes that the delay for pin j of gate g is specified as a range, $[d^l(g_j, x), d^u(g_j, x)]$, given by the lower and upper bounds on the actual delays. However, when the objective is true delay calculation, such a model does not give any advantage over the upper-bounded gate delay model.

2.3 Algebraic Decision Diagrams

In this section we briefly summarize the main characteristics of Algebraic Decision Diagrams (ADD's) [10]. For a more extensive treatment of this subject the reader can refer to [14].

An ADD is a directed acyclic graph $(V \cup \Phi \cup T, E)$, representing a set of functions $f_i : \{0,1\}^n \rightarrow S$, where S is the finite carrier of the algebraic structure over which the ADD is defined. V is the set of the internal nodes. The out-degree of $v \in V$ is 2. The two outgoing arcs for a node $v \in V$ are labeled *then* and *else*, respectively. Every node $v \in V$ has a label $l(v) \in \{0, \ldots, n-1\}$. The label identifies a variable on which the f_i's depend. Φ is the set of the function nodes: The out-degree of $\phi \in \Phi$ is 1 and its in-degree is 0. The function nodes are in one-to-one correspondence with the f_i's. T is the set of terminal nodes. The value at the terminal node is stored as a floating point number rather than encoding time as a binary representation for efficiency of manipulation. Each terminal node t is labeled with an element of S, $s(t)$. The out-degree of a terminal node is 0. E is the set of edges connecting the nodes of the graph; (v_i, v_j) is the edge connecting node v_i to v_j. The variables of the ADD are ordered; if v_j is a descendant of v_i (i.e., $(v_i, v_j) \in E$), then $l(v_i) < l(v_j)$.

An ADD represents a set of boolean functions, one for each function node, defined as follows:

1. The function of a terminal node, t, is the constant function $s(t)$. The constant $s(t)$ is interpreted as an element of a boolean algebra larger than or equal in size to S.
2. The function of a node $v \in V$ is given by $l(v) \cdot f_{\text{then}} + l(v)' \cdot f_{\text{else}}$, where '$\cdot$' and '$+$' denote boolean conjunction and disjunction, and f_{then} and f_{else} are the functions of the *then* and *else* children.
3. The function of $\phi \in \Phi$ is the function of its only child.

For example, given the circuit of Figure 1(a), the table of Figure 1(b), which provides the arrival time of the output node of gate $G2$ for each input vector, can be represented with the ADD of Figure 1(c).

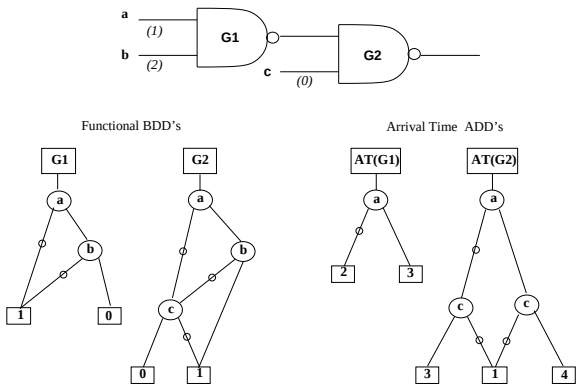

Figure 2: A Combinational Circuit, and its Associated Functional BDD's and Arrival Time ADD's.

```
ComputeAT(BG,G,AT,DELAY) {
  if (non_contr_value(BG) && all_const(AT)) {
    forall (fanin_i)
      time = Max(AT_i, non_contr_delay(DELAY_i))
    return(time);
  }
  if (contr_value(BG) && all_const(AT) && all_const(G)) {
    forall (fanin_i such that contr_input(G_i))
      time = Min(AT_i,contr_delay(DELAY_i))
    return(time);
  }
  if (table_lookup(cache,(BG,G,AT,DELAY),R))
    return(R);
  v = top_var(BG,G,AT);
  R_t = ComputeAT(BG_v,G_v,AT_v,DELAY);
  R_e = ComputeAT(BG_v',G_v',AT_v',DELAY);
  R = v · R_t + v' · R_e;
  table_insert(cache,(BG,G,AT,DELAY),R);
  return(R);
}
```

Figure 3: The Arrival Times Computation Algorithm.

3 ADD-Based Timing Analysis

The problem of calculating the timing response of a combinational circuit can be formulated as follows: Given a combinational circuit, find the set of input vectors for which the length of the critical path, under a specified mode of operation and a gate delay model, is maximum; the length of the critical path gives the overall circuit delay.

Using the ADD data structure, we are able to find more than just the overall circuit delay. In fact, the algorithm described in the next section allows us to compute and store the length of the critical path *for each* input vector. The amount of timing information we are able to compute is then much greater than what is usually computed by traditional delay analyzers. This additional information is useful in applications like re-synthesis for low-power.

In the realization of our ADD-based timing analysis tool, we have made the following assumptions:

1. The circuit is assumed to operate in *floating mode*.
2. The *monotone speed-up gate delay model* is used.
3. Gate delays are modeled as pin-to-pin delays
4. For each input pin j of gate g, $d(g_j, x)$ is composed of the intrinsic delay of the gate plus the delay due to the driving and load factors of the gate when input vector x is applied to the gate.
5. Gate delays may be different for rising and falling input transitions depending on the library used to map the circuit. Also, different pins may be responsible for different gate delays.

3.1 The Algorithm

Given a gate g of the network and its input vector x, the arrival time at its output, $AT(g,x)$, is evaluated in terms of its controlling values, and its fanin gate delays, $d(g_j,x)$. Let g^j be the gate connected to g via pin j.

If at least one fanin g^j of g has a controlling value,

$$AT(g,x) = \min_j\{(AT(g^j,x) \mid f_{g^j} = \mathit{controlling}) + d(g_j,x)\}. \quad (1)$$

If all fanins of g have non-controlling values,

$$AT(g,x) = \max_j\{AT(g^j,x) + d(g_j,x)\}. \quad (2)$$

Using the circuit represented in Figure 2, we can now go through an example of how the arrival times are calculated. For the sake of simplicity, assume unit time gate delay for all gates and all its pins, i.e., $d(g_j,x) = 1$. The arrival time for inputs a, b, and c of the circuit is shown in parenthesis. Assume we want to calculate $AT(G2,011)$, the arrival time at the output of gate $G2$, given the input vector $\{a,b,c\} = (011)$. We start with the evaluation of gate $G1$. The arrival time at the output of gate $G1$ is controlled by the input a. Therefore, the arrival time at its output is $AT(a,011) + d(G1_a,011) = 1 + 1 = 2$, and its functional value is set to 1. All fanins to $G2$ have non-controlling values, therefore, the arrival time of $G2$ is determined by the latest arriving input. That is, $AT(G2,011) = \max\{(AT(G1,011)+d(G2_{G1},011)),(AT(c,011)+d(G2_c,011))\} = AT(G1,011) + d(G2_{G1},011) = 2 + 1 = 3$.

We are now ready to see how ADD's are used to compute the arrival times of a circuit for all combinations of input values. In Figure 3 we show the pseudo-code of the procedure which computes the arrival time ADD for a simple gate in a circuit. The algorithm requires four parameters, the functional BDD of the gate, BG, and three arrays, G, AT, and $DELAY$, containing the functional BDD's, arrival time ADD's, and rise and fall gate delay information respectively, for each fanin to the gate. The arrival times are computed for all combinations of input values. This is accomplished by splitting on the top variable among all the arguments, BG, G, and AT, until a constant value of 1 or 0 has been reached for BG, the functional BDD of the gate. This constant, 1 or 0, determines whether the gate has any controlling values at its fanins. For example, if the gate is a NAND, then a constant 1 at its output implies at least one fanin is at a controlling value of 0. Likewise, if the gate is a NOR, then a constant 1 implies all fanins are non-controlling, or 0. If all fanins of a gate are non-controlling (i.e., $\mathit{non\_controlling}(BG)$ is true) then we already know that the BDD's of all the fanins are also constant, and we only need to check if the arrival times of these fanins are constant as well. If so, then we have reached a terminal case, and we can compute the arrival time for the gate given the input values associated with the splitting order. This computation is done in line 4 of the pseudo-code using Equation 2.

If on the other hand, it has been determined that at least one fanin of a gate is controlling (i.e., $\mathit{controlling}(BG)$ is true), then it is not necessarily the case that all fanins are constant; only one controlling fanin needs to be constant. Since the arrival time at the output of the gate is determined by the earliest arriving controlling input, we need to know exactly which inputs are controlling. Therefore, it may be necessary to further split on variables until both arrival time ADD's and functional BDD's of all fanins are constant. The arrival time computation is done in line 9 of the code using Equation 1.

Going back to the example in Figure 2, we can now see how the arrival time ADD is constructed using again the input vector $\{a,b,c\} = (011)$. We first split the BDD of $G2$, along with the BDD's and ADD's of $G1$ and c with the variable $a = 0$, i.e., $\{a,b,c\} = (0xx)$. Although the BDD and ADD of $G1$ are now constant ($\mathrm{BDD}(G1,0xx) = 1$, $\mathrm{AT}(G1,0xx) = 2$), the BDD's for $G2$ and c are not constant, so we split again with $c = 1$. Notice that although $l(b) < l(c)$, b is no longer contained in the BDD's or ADD's, so we split on c instead. Now, we have reached constant values for the BDD's ($\mathrm{BDD}(G2,0x1) = 0, c = 1$), and the arrival time can be computed as $AT(G1,0x1)+d(G2_{G1},0x1) = 2+1 = 3$. Going now to the arrival time ADD of $G2$ we see that the arrival time pointed to by the vector $\{a,b,c\} = (011)$ is 3. Other arrival times are computed in a similar fashion, by splitting on the top variable v and recursively calling the procedure on the *then* and *else* branches until a terminal condition is reached. The ADD is constructed by combining the ADD's of these branches, R_t and R_e, and forming the result, $R = v \cdot R_t + v' \cdot R_e$. Notice by the example shown in Figure 2, the arrival time ADD of a gate may contain more or fewer nodes than its functional BDD. Also, an arrival time ADD of a gate may not be a function of the same variables as its BDD.

Once the arrival time computation is complete, we have available not only the critical path delay at the output of the gate, but also the exact set of input values that produce that delay. Tracing the arrival time ADD for $G2$, we see that the critical path of 4 is produced by the input vectors $\{a,b,c\} = (101),(111)$. In fact, we can extract the exact set of input values for *any* possible arrival time seen at the output of the gate. The ADD structure also lends itself well to the handling of don't care combinations at the inputs. For instance, if certain primary input combinations will never happen in a circuit, then the arrival times for these input values need not be computed, thereby simplifying the overall timing analysis.

To speed up the computation time, a local hash table is used for each gate to store intermediate results as its arrival time is computed. BG, the functional BDD of the gate, is not strictly required in the recursion or hashing function because it is logically implied by the BDD's for the inputs. However, it is faster to check for the terminal cases and to detect that two subproblems are not identical if BG is available.

3.2 Input Variable Ordering

Although in some instances the arrival time ADD of a gate may be smaller in size than its functional BDD, in practice the ADD usually runs about 5 to 10 times larger than its BDD in terms of number of nodes. The size of the ADD's of the gates depends partially on the structure of the circuit and to a large extent on the input variable ordering. Finding a good input variable ordering for the ADD's is key to calculating the arrival times with efficient time and memory usage.

Much recent work has been done in the area of reducing the size of the functional BDD representation of a circuit using smart variable ordering methods [16, 17, 18, 19, 20]. A good variable order for the functional BDD's of a circuit often correlates with a good variable ordering for the arrival time ADD's. However, an ordering that is "good enough" to efficiently store the functional BDD may not be good enough for the arrival time ADD. Also, as with BDD's, a good ordering method for one particular circuit's ADD is not necessarily a good ordering method for a different circuit. Using a simple depth-first search ordering algorithm may work well for a fanout-free circuit, for example, but may cause an ADD size explosion for a different circuit. The dependency of the size of the ADD to the structure of the circuit suggests using an ordering that is more sensitive to the fanout of the circuit. A different method may provide a better ordering than depth-first search, but this still may not be good enough given that on average ADD's grow at a faster rate than BDD's. Initial experiments suggest using a dynamic variable ordering, such as those presented in [17, 18, 20]. Since the time to run a dynamic variable ordering procedure depends on the initial ordering of the variables, a good approach would be to start with an ordering sensitive to fanout, and then proceed to dynamically reorder the variables as the ADD's grow in size above a certain threshold.

Circuit	*PI*	*PO*	*Gates*	*Top. Delay*	*True Delay*	*Time (sec)*
C432	36	7	197	24	23	1954.9
C499	41	32	530	24	mem-out	
C880	60	26	357	15	15	435.0
C2670	233	140	810	26	time-out	
C5315	178	123	1710	34	34	11800.3
C7552	207	108	2776	39	38	86.7
CBP16	33	17	278	51	25	4.8
CBP32	65	33	496	99	41	17.6
CBP64	129	65	992	195	73	74.9
CBP128	257	129	1984	387	137	431.9
Ablock	52	16	609	54	30	30.1

Table 1: Results for Arrival Time Calculation.

3.3 Reducing Memory Requirements

Generating the ADD's for the arrival time of all the gates of a circuit may require a large amount of memory. If one is interested only in detailed information on the outputs of a circuit, two approaches can be used to significantly reduce this memory requirement: deleting unused ADD's and BDD's, and sorting the primary outputs. As the arrival time of a circuit is being calculated, we move from the inputs forward toward the outputs visiting all the gates in a depth-first search manner. Using this method, we guarantee that all fanins of a gate have been visited and their arrival times computed before the gate itself is visited. Furthermore, once all the fanouts of a gate have been visited, then the ADD and BDD of that gate will no longer be needed to compute the arrival time ADD of any other gate. ADD's and BDD's of these gates may be freed, effectively reducing memory usage. The other effective approach involves pre-sorting the primary outputs according to non-increasing static delay. The output with the longest static delay is processed first. If the true delay of this output is greater than or equal to the longest static delay of the next output, then the critical path has been found, and we do not need to proceed any further.

4 Experimental Results

In Table 1 we report the results for computing the true delay for some combinational benchmark circuits using ADD's we obtained on a DEC-Station 5000 with 80 Megabytes of RAM. Most of the examples are from the ISCAS'85 set [12]. Also presented are data for 16, 32, 64, and 128-bit Carry-Bypass Adders, and for a component module, called `Ablock`, of a greatest-common-divisor circuit, which contains two 16-bit Carry-Bypass Adders and some multiplexing logic. The circuits were mapped onto a NAND/inverter library using the SIS [11] technology mapper. This limited library was used only to simplify the initial coding of the algorithm; the code can easily be expanded to handle other gates. Arrival times were computed using unit gate delays. In particular, column *Circuit* indicates the circuit name, and columns *PI, PO*, and *Gates* indicate the number of primary inputs, primary outputs, and number of gates respectively after technology mapping of each circuit. Column *Top. Delay* gives the static delay computed by SIS, and column *True Delay* gives

the true delay calculated by our ADD-based timing analyzer. Finally, column *Time* reports the CPU time in seconds needed to calculate the arrival time of the critical path.

Of course, no comparison is possible with results obtained using existing timing analysis tools because, as mentioned earlier in the paper, the amount of information calculated by our procedure is much more extensive than the simple worst-case, true delay of the circuit; this leads, obviously, to run times of the ADD-based method that are sensibly higher than what has been published, for example, in [8]. However, there is a class of circuits, the Carry-Bypass Adders, for which our procedure is very competitive also in terms of run time. Carry Bypass Adders are known to be particularly difficult to handle by timing analysis algorithms based on false paths elimination. Consequently, true delay results on these circuits are not available in the literature, but for devices of a limited size, and the CPU time required to complete the true delay computation is usually much higher than the time taken by our ADD-based method.

The hash table proved to be quite efficient in our computations; hit rates averaged about 30%. Although these examples were run using unit gate delay, the tool supports non-unit delays on gates, as well as skewed rise/fall delays for each fanin pin and delays due to fanout load. In general it takes longer to compute arrival times when the circuit does not have unit gate delay. In the case of the Carry-Bypass Adders and the `Ablock`, this extra time is nominal (about 10% longer). For other circuits, computation times may be several times longer. These time should go down as improvements to the algorithm are made, since the size of arrival time ADD's (25-50% larger) does not indicate major problems in terms of memory requirements.

5 Conclusions and Future Work

Designing digital circuits that satisfy given timing constraints is one of the main requirements in the process of fabricating modern VLSI devices. Various timing analysis techniques have been presented over the years; most of them were based on explicit false paths detection and elimination, others used probabilistic or test generation-like approaches.

In this paper we have presented an ADD-based method to perform true delay computation in combinational circuits. The algorithm is able to efficiently compute and store the length of the critical path for all possible input vectors. Experimental results, though preliminary, are very encouraging, especially on some circuits, like the Carry-Bypass Adders.

The amount of timing information provided by our tool allows us to approach the problem of re-synthesizing combinational circuits for low power; this is the direction we are currently moving towards. The exact knowledge of the arrival times at each gate of the network can be used to identify critical input vectors, critical gates, and critical paths. Combinational circuits can be re-synthesized for low-power using two approaches: First, reducing glitches caused by transition delays when primary inputs switches values, and second by relaxing arrival times on gates that are not on the critical path. We want to choose a delay that has the effect of both reducing glitches due to input transitions, and reducing capacitive load at the output of the gate.

Variable ordering plays a fundamental role in all the BDD/ADD based symbolic algorithms. We are currently investigating the relationship between functional BDD's and arrival time ADD's with respect to their variable orderings. Many algorithms and heuristics are available to find good orderings for the functional BDD's of a circuit, but there is no guarantee that the same ordering criteria work when applied to arrival time ADD's. Furthermore, we are trying to apply to ADD's dynamic reordering techniques, as the ones suggested by Bernard Plessier [17], Eric Felt and co-workers [18], and Rick Rudell [20].

Acknowledgments

We wish to thank Seh-Woong Jeong, Rick Rudell, and Jerry Yang for providing us with good BDD input variable orderings for the ISCAS'85 benchmark circuits.

References

[1] P. C. McGeer, R. K. Brayton, Integrating Functional and Temporal Domains in Logic Design, Kluwer Academic Publishers, 1991.

[2] J. Benkoski, E. V. Meersch, L. Claesen, H. De Man, "Efficient Algorithms for Solving the False Path Problem in Timing Verification", ICCAD-87: IEEE International Conference on Computer Aided Design, pp. 44-47, Santa Clara, CA, November 1987.

[3] D. H. C. Du, S. H. C. Yen, S. Ghanta, "On the General False Path Problem in Timing Analysis", DAC-26: ACM/IEEE Design Automation Conference, pp. 555-560, Las Vegas, NV, June 1989.

[4] P. C. McGeer, R. K. Brayton, "Efficient Algorithms for Computing the Longest Viable Path in a Combinational Network", DAC-26: ACM/IEEE Design Automation Conference, pp. 561-567, Las Vegas, NV, June 1989.

[5] H. C. Chen, D. H. C. Du, "Path Sensitization in Critical Paths", IEEE Transactions on Computer Aided Design, Vol. CAD-12, No. 2, pp. 196-207, February 1993.

[6] S. Devadas, K. Keutzer, S. Malik, "Delay Computation in Combinational Logic Circuits: Theory and Algorithms", ICCAD-91: IEEE International Conference on Computer Aided Design, pp. 176-179, Santa Clara, CA, November 1991.

[7] H. Chang, J. A. Abraham, "VIPER: An Efficient Vigorously Sensitizable Path Extractor", DAC-30: ACM/IEEE Design Automation Conference, pp. 112-117, Dallas, TX, June 1993.

[8] S. Devadas, K. Keutzer, S. Malik, A. Wang, "Computation of Floating Mode Delay in Combinational Circuits: Practice and Implementation", International Symposia on Information Sciences, pp. 88-75, Fukuoka, Japan, July 1992.

[9] S. Devadas, H. F. Jyu, K. Keutzer, S. Malik, "Statistical Timing Analysis of Combinational Circuits", ICCD-92: IEEE International Conference on Computer Design, pp. 38-43, Cambridge, MA, October 1992.

[10] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, "Algebraic Decision Diagrams and their Applications", ICCAD-93: ACM/IEEE International Conference on Computer Aided Design, pp. 188-191, Santa Clara, California, November 1993.

[11] E. M. Sentovich, et al., SIS: A System for Sequential Circuits Synthesis, Technical Report Memorandum No. UCB/ERL M92/41, University of California at Berkeley, May 1992.

[12] F. Brglez, H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran", ISCAS-85: IEEE International Symposium on Circuits and Systems, Kyoto, Japan, June 1985.

[13] K. Keutzer, S. Malik, A. Saldanha, "Is Redundancy Necessary to Reduce Delay?", IEEE Transactions on Computer Aided Design, Vol. CAD-10, No. 4, pp. 427-435, April 1991.

[14] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic Decision Diagrams and their Applications, Internal Report, VLSI/CAD Research Group, Dept. of Electrical and Computer Engineering, University of Colorado at Boulder, April 1993.

[15] R. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation", IEEE Transactions on Computers, Vol. C-35, No. 8, pp. 79-85, August 1986.

[16] S.W. Jeong, B. Plessier, G. D. Hachtel, F. Somenzi, "Variable Ordering for Binary Decision Diagrams", EDAC-92: IEEE European Conference on Design Automation, pp. 447-451, Brussels, Belgium, March 1992.

[17] B. Plessier, A General Framework for Verification of Sequential Circuits, Ph. D. Thesis, Dept. of Electrical and Computer Engineering, University of Colorado at Boulder, May 1993.

[18] E. Felt, G. York, R. K. Brayton, A. Sangiovanni-Vincentelli, "Dynamic Variable Reordering for BDD Minimization", EuroDAC-93: IEEE European Design Automation Conference, pp. 130-135, Hamburg, Germany, September 1993.

[19] H. Fujii, G. Ootomo, C. Hori, "Interleaving Based Variable Ordering Methods for Ordered Binary Decision Diagrams", ICCAD-93: ACM/IEEE International Conference on Computer Aided Design, pp. 38-41, Santa Clara, CA, November 1993.

[20] R. Rudell, "Dynamic Variable Ordering for Ordered Binary Decision Diagrams", ICCAD-93: ACM/IEEE International Conference on Computer Aided Design, pp. 42-47, Santa Clara, California, November 1993.