A Hierarchical Timing Simulation Model

TZU-MU LIN AND CARVER A. MEAD

Abstract—**A hierarchical timing simulation model has been developed to deal with VLSI designs at any level of representation. A set of physically based parameters are used to characterize the behavior and timing of a semantic design object (cell) independent of its composition environment. As cells are composed, the parameters of the composite cell can be determined from those of the component cells either analytically or by simulation. Based on this model, a behavior-level simulator has been developed and combined with other tools to form an integrated design system that fully supports the structured design methodology.**

I. Introduction

THE EXPLOSIVE advances in VLSI technology have generated opportunities for revolutionary system designs. In order to successfully exploit these opportunities, there must be correspondingly aggressive advances in the supporting technologies and sciences. One question that must be addressed is how to manage the increasing complexity of VLSI systems. Under the well-known structured-design methodology [1], [2], a design is partitioned into several levels of hierarchy, typically from the architecture level, block level, logic level to circuit level. This partitioning helps designers focus on one particular level of design at any given time, and allows the complexity of a large system to be managed effectively.

A hierarchical design is best supported by a hierarchical simulator for determining its functionality and performance. The difficulty of a hierarchical simulator, however, is that consistency between different levels of representation cannot be easily ensured. As pointed out in [2], to ensure this consistency requires a good deal of discipline—in particular, a well-defined and consistent timing convention, and well-defined data types. If these disciplines are followed, then a system can be partitioned successively into hierarchical levels of semantic cells.[1]

The steady-state behavior of a semantic cell at any level of representation is the only information necessary to define its behavioral interface with other semantic cells. Furthermore, any legal interconnection of several semantic cells is itself a semantic cell, and the behavior of the composite cell can be derived from the behaviors of the component cells in a consistent manner. Based upon the fixed-point algorithm [5] to abstract the behavior of a semantic cell from its implementation, Chen developed a Universal Hierarchical Simulator (UHS) that can be applied to designs from transistor circuit-level to high-level communicating sequential processes [3]. The UHS differs in a fundamental way from "mixed mode" simulators, in that it does not constrain the design to certain levels determined *a priori*, but rather allows the user to define levels that correspond directly to the conceptual blocks of the design. The hierarchical nature of the UHS allows the implementation details to be hidden, and therefore yields a clear conceptualization of the design and a very efficient simulation.

The main concern of the UHS is the functional behavior of a design, not the delay in physical time units. Time delay information is very important to designers because a chip is not correct if it does not run at the desired speed. On the other hand, most simulators that offer accurate time delay information [6], [7] tend to carry too much analog detail; no simple abstraction or composition rules have been derived to allow hierarchical treatment of a complicated design. Traditional logic-delay simulators do not allow accurate enough modeling of the delays introduced by wiring in a VLSI chip. As the complexity of VLSI systems increases, the demand becomes more and more urgent for a UHS style simulation model with the capability of generating physical timing relations.

To establish such a hierarchical timing model, two pieces of work have been done: 1) an MOS transistor-level (bottom-level) delay model that serves as the basis, and 2) a general composition rule for deriving the behavior and timing of a high-level cell from those of the lower-level component cells. The transistor-level model was presented in [8], [9]; the general composition rule is discussed in this paper. For any timing discipline, there exists a way to partition a system such that every subsystem is a semantic cell. In this paper, two-phase synchronous sytems are used as an example to illustrate the principles used in our hierarchical timing model. These principles apply equally well to other timing disciplines [10], [11], and can be extended as necessary.

This paper uses the results of our transistor-level model which are summarized in Section II. In Section III, semantic cells of two-phase synchronous systems are characterized. Composition of semantic cells is discussed in Section IV. A set of parameters are used to characterize the behavior and timing of a semantic cell independent of its composition environment. As cells are composed, the resulting cell can be described in the same way by the same set of parameters. The number of such parameters

Manuscript received October 8, 1984. This work was supported by the System Development Foundation.

T-M. Lin is with Silicon Compilers Inc., 2045 Hamilton Avenue, San Jose, CA 95125.

C. A. Mead is with the Department of Computer Science, California Institute of Technology, Pasadena, CA 91125.

IEEE Log Number 8405891.

[1]As opposed to syntactic cells, which are only used for ease of specification, and do not provide any abstraction. The results are due to Chen [3], [4].

is linearly proportional to the number of connection ports of a cell. The parameters of the composite cell can be determined from those of the component cells either analytically or by simulation. In Section V, the timing of an nMOS static programmable logic array (PLA), as an example, is abstracted from its circuit structures into functional form. Data abstraction is discussed in Section 6. Embedded in the Smalltalk programming environment [12], [13], a hierarchical behavior-level timing simulator has been developed, and is discussed in Section VII. An integrated design system that fully supports the structured design methodology is presented in Section VIII.

II. MOS Delay Model

Every transistor group of an MOS circuit is modeled by an *RC* network for estimating delays [9], [14]. The definition of delay is based on that proposed by Elmore [15]. Three parameters *R*, *C*, and *D* are used to carry the delay characteristics of a two-port *RC* network, independent of its size and composition environment. *R* is the series resistance between the input and the output port, *C* is the total capacitance inside the network, *D* equals the internal delay, the delay of the output port when the input port is driven directly by the signal source, and the output port is open. These three parameters are well-defined in the simple *RC* case (*D* equals the *RC* time constant), and can be derived analytically as the networks are composed in various ways.

The above discussion applies to any general *RC* networks with parallel and bridge connections and initial charge distributions. For *RC* tree networks, an efficient algorithm (TREE) exists to calculate the delays of all the nodes in the following two steps.

1) The load capacitance C_i^L of every node i is accumulated and propagated from the loading ends towards the driving end of the tree. If node i is a leaf node, then $C_i^L = C_i$. Otherwise, $C_i^L = C_i + \Sigma_j C_j^L$, where index j ranges over all succeeding nodes of node i, and C_i is the node capacitance of node i.

2) The delay of every node is calculated incrementally from the driving end towards the loading ends: $T_i = T_{p(i)} + r_i C_i^L$, where $p(i)$ is the parent node of node i, $T_{p(i)}$ is the delay of node $p(i)$, and r_i is the resistance between node $p(i)$ and node i. (1)

This algorithm can be extended to deal with tree networks where every branch is a two-port *RC* network. The two steps of calculation are modified as follows (the differences are underlined).

1′) Load Capacitance C_i^L: if node i is a leaf node, then $C_i^L = C_i$. Otherwise $C_i^L = C_i + \Sigma_j(C_j^L + C_{i,j})$, where $C_{i,j}$ is the C parameter of the two-port *RC* network between node i and node j.

2′) Delay Value T_i: $T_i = T_{p(i)} + R_{p(i),i}\, C_i^L + D_{p(i),i}$, where $R_{p(i),i}$ and $D_{p(i),i}$ are the *R* and *D* parameters of the two-port *RC* network between node $p(i)$ and node i. (1′)

In this paper, the *R*, *C*, and *D* characterization of two-port *RC* networks is generalized to semantic cells at any level of representation. In this new context, *R* means the driving resistance of an output port, *C* means the loading capacitance of an input port, *D* carries the internal delay due to logic propagation, assuming the output port is open, and the input port is driven directly by the signal source. When cells are composed, the delay due to wires can be determined by the TREE algorithm based on the *R* parameter of the driving cell, the *C* parameters of the loading cells, and the structure and the *R*, *C* values of the interconnects (Section IV). Note that, as far as delay is concerned, a uniformly distributed *RC* line is equivalent to a three-element lumped *RC*-π network [9].

III. Semantic Cells in Two-Phase Synchronous Systems

In a two-phase synchronous system, all operations are initiated by global clocks. The period of a clock phase (the period from the rising edge of one clock to that of the other clock) is greater than the maximum amount of time necessary to complete any computations that occur during that phase. The results are then ready to be latched by the clock of the other phase. If the system is partitioned according to the phase relationships, then every partition of the network is a semantic cell. The reason is as follows: When clock ϕ_1 goes high, the inputs of all ϕ_1 cells switch to the results of the previous ϕ_2 computations, and remain stable during the rest of the clock phase. Although the outputs of these cells may switch several times during this period, the intermediate results are stopped by clock ϕ_2, and have no effect on the system behavior. Only the steady-state value of an output is of importance, and all internal nodes are stabilized at the end of this period. A ϕ_1 cell can be abstracted by its steady-state behavior to interface with the rest of the system, and thus is a semantic cell. Similarly for ϕ_2 cells.

A cell thus partitioned can be represented by the structure shown in Fig. 1. All inputs of the cell (except clocks) are controlled by pass transistors gated by a clock signal. All outputs are static with no pass transistor blocking the way. Such a cell, called a "clocked-cell" by Chen [3], is the primitive building block of any synchronous system.[2]

A semantic cell in a two-phase synchronous system is recursively defined: it is either a clocked-cell or a legal composition of semantic cells. A phase attribute is associated with each input to, and each output from a semantic cell indicating the active phase of the input or output port of the cell. A legal composition of semantic cells is such that the following two conditions are satisfied.

a) ϕ_1 inputs connect to ϕ_2 outputs, and vice versa.
b) The period for both phases is sufficient for all circuits active during that phase to reach their steady states. (2)

The checking of the first condition of (2) is purely syntactic. The second condition is a strong one, and can be checked for every cell without regard to how it will be

[2]Pass transistors are the most common clocking primitive in MOS designs. The same comments apply to any clocked signal gating discipline.

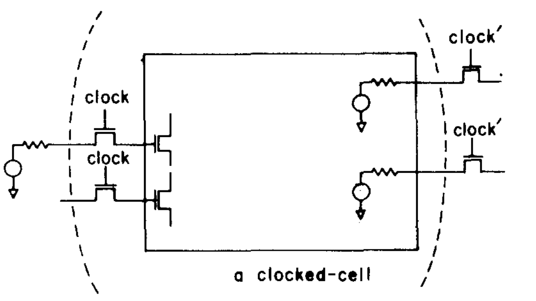


Fig. 1. Clocked-cell: building block for sychronous systems.

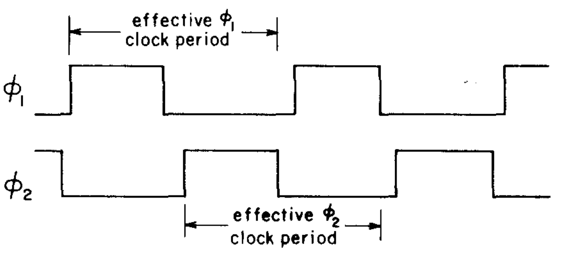

effective ϕ_1 clock period
ϕ_1
ϕ_2
effective ϕ_2 clock period

Fig. 2. Effective ϕ_1 and ϕ_2 clock periods.

interconnected. It is often desirable, however, to relax the second condition to allow the borrowing of time between ϕ_1 and ϕ_2, as often implemented in practical designs. A ϕ_1 cell is not required to reach its steady state before the rising edge of ϕ_2. As long as all inputs to ϕ_2 cells are stabilized by the falling edge of ϕ_2, a circuit can be made to function correctly. The "effective clock period" of a ϕ_1 cell starts from the rising edge of ϕ_1, through the rising edge of ϕ_2, until the falling edge of ϕ_2 (Fig. 2). Similarly for a ϕ_2 cell. To allow borrowing time between ϕ_1 and ϕ_2, condition (2.b) is replaced by the following three weaker conditions. Note that the term "period" refers to the effective clock period of a cell.

1) The network activities of two consecutive periods of the cell are loosely coupled, so that each period may be considered independently.
2) The response of the cell at any period can be described analytically with reasonable complexity.
3) The period of each phase is sufficient for all inputs of that phase to stabilize before the falling edge of that phase. (3)

One possible interpretation of conditions (3.1) and (3.2) is as follows.

- When the cell is excited at any period, all nodes in the cell stabilize to a fully charged or discharged state before the next period starts. (3.1′)

If (3.1′) is satisfied, then one needs not keep track of the stored charge of the internal nodes any more. Although it is necessary to record the logic states of latches of a sequential circuit, the number of latches is usually much smaller than the total number of internal nodes in the network.

Given one clock period, an input of a cell may switch once, more than once, or not switch at all. In general, there are infinite number of input patterns that need to be considered. To make the situation tractable, the following requirement is imposed.

- During any given period, the state of every input port switches at most once. (3.2′)

In summary, conditions (3.1′) and (3.2′) determine whether or not a clocked-cell is a semantic cell. Note that these two conditions refer to interactions among cells. Therefore, whether a cell is a semantic cell depends not only on its content, but also on how it is interconnected with other cells.

Depending on individual applications, conditions (3.1′) and (3.2′) can be further relaxed to assure that every clocked-cell is a semantic cell. However, more complicated expressions are required to describe the timing of such a cell. In this paper, conditions (3.1′) and (3.2′) are used as an example to illustrate the general idea of our timing model. These two conditions, or the stronger conditions (2.a) and (2.b) are believed to be satisfied by clocked-cells of most synchronous digital systems. The set of timing parameters presented at the end of Section IV are based on these two conditions.

Syntactically, a clocked-cell can be further decomposed into gate-level cells. According to the argument at the beginning of this section, clocked-cells are the smallest possible semantic cells in synchronous systems. With the relaxation of condition (2.b), however, it is possible to treat gate-level cells as semantic objects. In particular, if both conditions (3.1′) and (3.2′) are satisfied by the gate-level cells decomposed from a clocked-cell, then the timing of the clocked-cell can be derived from the timing values of these gate-level cells analytically. The PLA example of Section V is treated this way.

MOS transistors are, in general, bidirectional devices; the signal may flow in either direction. For a semantic cell, however, the direction of every connection port must be determined. Note that this restriction does not exclude the possibility of an *I/O* port. Although the direction of such a port changes dynamically, at any given clock period, it is either an input or an output. An illegal situation results when two input ports of a cell are shorted by a conducting path of pass transistors within the input network of the cell. We assume that some discipline has been applied in the input network to assure "no fighting" between driven signals [16].

IV. Composition of Semantic Cells

Consider a semantic cell with n input ports ($I_{1,\ldots,n}$) and m output ports ($O_{1,\ldots,m}$). From the previous discussions, every input or output state of the cell switches at most once during any given clock period. Suppose, during the current clock period, the input states of the cell switch to $VI_{1,\ldots,n}$ at time $TI_{1,\ldots,n}$, respectively. Note that all TI values of a semantic cell under condition (2) are equal to 0, because new input values enter the cell on the rising edge of the clock (the reference time). In general, the TI's may admit any nonnegative values. Suppose the output

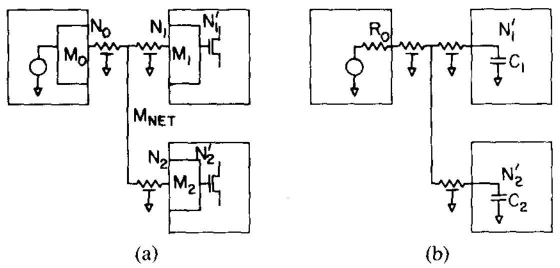


Fig. 3. Cells and interconnections.

states are updated to $VO_{1,\cdots,m}$, and stabilized at time $TO_{1,\cdots,m}$, respectively. Note that, in general, the values of the TI's depend on the driving resistances at the input ports, and the TO's depend on the loading capacitances at the output ports. Consider the general situation of interconnections among cells indicated in Fig. 3(a). An "interconnection" (or "net") is always of a tree structure, in which there are several loading nodes (referred to as nodes $N_{1,\cdots,s}$), and only one driving node (node N_o). Although there are more than one driving nodes in the case of a bus, we assume a discipline in which only one driving node is active at any given period. All nodes in the net are logically equivalent because there are no transistors separating them. However, due to the stray resistances and capacitances of the interconnection wires, these nodes are not electrically equivalent, and their delay values are different. According to our transistorlevel delay model, every driving or loading node of the net is contained in a transistor group that can be approximated by an RC network for estimating delays.

Note that the input port of the two-port RC network that contains the driving node N_o is connected directly to the signal source (this network is denoted by M_o). The output port of the two-port RC network that contains the loading node N_i of the net is open (this network is denoted by M_i, $i = 1, \cdots, s$). Let N_i' denote the node at the output port of M_i. Referring to Fig. 3(a), the net combines all these two-port RC networks into one RC network through which these cells interact (this resulting network is referred to as M_{NET}).

The delays of all nodes in network M_{NET} can be calculated using the extended TREE algorithm (1′).

Consider the RC network derived from M_{NET} by the following operations:

- Replace the two-port RC network M_o by R_o, where R_o is the R parameter of M_o.
- Replace the two-port RC network M_i by C_i, where C_i is the C parameter of M_i, $i = 1, \cdots, s$.

This derived RC network is indicated in Fig. 3(b). With the 3-element π-approximation, every branch of this network is a resistor so that the simple TREE algorithm (1) can be applied. Except for the R_o (driving resistance) and C_i's (loading capacitances), all capacitances and resistances in this derived network come from interconnection wires. The delay properties of a composition of cells are based on the following theorem:

Theorem 1. Let T_i^* and T_i denote the delays of node N_i' in the original and derived M_{NET}, respectively, $i = 1, \cdots, s$. Then $T_i^* = T_i + D_o + D_i$, where D_o and D_i are the D parameters of M_o and M_i, respectively.

Proof:

First note that the load capacitance of every node in the net is the same for both the original and the derived M_{NET}. Let t_o^*, t_o, t_i^*, and t_i be the delays of node N_o and N_i in the original and the derived M_{NET}, respectively. The proof of the theorem proceeds from the driving end towards the loading ends of M_{NET}.

- Node N_o: The signal source is the parent node of node N_o in both the original and the derived M_{NET}. In the original network, $t_o^* = R_oC_o^L + D_o$, where C_o^L is the load capacitance of node N_o. In the derived network $t_o = R_oC_o^L$. Therefore, $t_o^* = t_0 + D_o$.
- Node N_i: Note that all the branches in the net between node N_o and N_i are pure resistors. Therefore, the two algorithms add the same amount of delay to both t_o^* and t_o to obtain the values of t_i^* and t_i, respectively. Thus $t_i^* - t_i = t_o^* - t_o = D_o$.
- Node N_i': Node N_i is the parent node of N_i' in the original N_{NET}. Thus $T_i^* = t_i^* + D_i$. In the derived M_{NET}, T_i and t_i refer to the same node. Thus $T_i = t_i$.

Combining the results of the above three items, $T_i^* = T_i + D_o + D_i$. ∎

Note that D_o and D_i's in the above theorem are only functions of individual cells, and are independent of the interconnection. On the other hand, T_i is only a function of the interconnection, and is in dependent of the internal behavior of any of these cells. The ability to derive T_i^* from these three terms analytically makes if possible to abstract and compose timing of cells.

Timing Parameters

In summary, the behavior and timing of a cell with n input ports $(I_{1,\cdots,n})$, m output ports $(O_{1,\cdots,m})$, and t internal states $(S_{1,\cdots,t})$ can be characterized by the following set of parameters:

1) VO_i for $i = 1, \cdots, m$: the logic state of output O_i after the network has stabilized.

2) TO_i for $i = 1, \cdots, m$: the time when output O_i is stabilized (all the output ports are open and input ports directly driven by signal sources).

3) VS_i for $i = 1, \cdots, t$: the internal state S_i after the network has stabilized.

4) CI_i for $i = 1, \cdots, n$: the load capacitance of input I_i.

5) RO_i for $i = 1, \cdots, m$: the driving resistance of output O_i. (4)

These parameters are evaluated each time any input of the cell switches during a clock period. In general, these parameters are functions of

- VI_i for $i = 1, \cdots, n$: the state of input I_i that excites the cell.

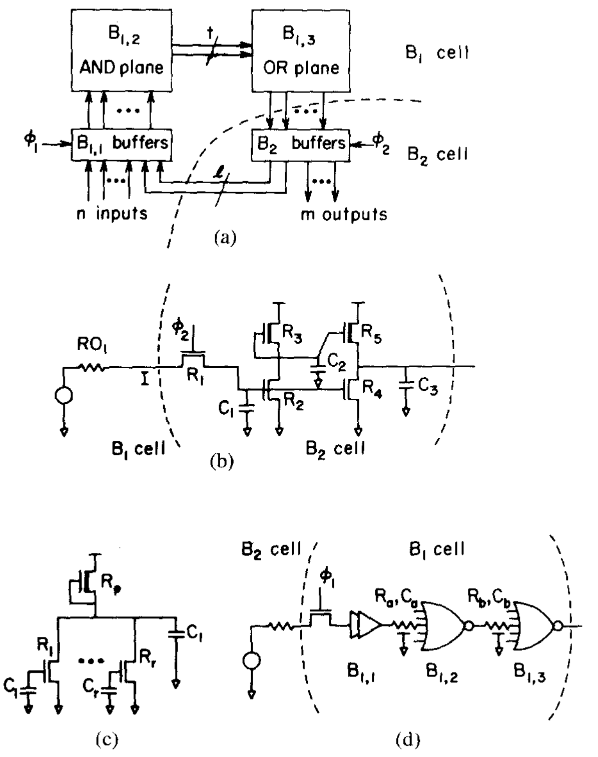


Fig. 4. Structure of a static nMOS PLA.

- TI_i for $i = 1, \cdots, n$: the time when the state of I_i switches to VI_i.
- $VS_i^{(0)}$ for $i = 1, \cdots, t$: the internal state S_i before the cell is excited.
- $VO_i^{(0)}$ for $i = 1, \cdots, m$: the state of output O_i before the cell is excited.

Among the five items of (4), RO's, CI's, and TO's are generalizations of the R, C, and D parameters of a two-port RC network. VO's and VS's describe the logical behavior of the cell. These two items are not necessary in a two-port RC network because the state of the output port simply follows that of the input port.

V. Abstraction of Circuit Behavior and Timing

Consider the structure of a static nMOS PLA shown in Fig. 4(a). According to the phase relationships, this circuit is partitioned into two clocked-cells: B_1 is active during ϕ_1, and B_2 is active during ϕ_2. The structure of B_2 is very simple: every feedback or output term corresponds to either an inverting or non-inverting buffer. Each buffer contains two inputs (I and ϕ_2), one output, and no internal states. The schematic diagram and associated circuit parameters for an inverting buffer are shown in Fig. 4(b). The set of parameters for describing the logic and timing of this buffer is indicated in Table I. VI_I is the state of input I, and TI_{ϕ_2} is the time when clock ϕ_2 rises. We assume that input I is stabilized before TI_{ϕ_2}; therefore TI_I is zero, and not shown in the table (the general case is discussed in [17]). A subscript $_2$ is associated with every parameter in the table indicating that they belong to clocked-cell B_2. The TO_2 value in the table is based on the assumption that the output state is switched. Otherwise, TO_2 is 0. Note that all the formulas in Table I are as accurate as if the circuit were simulated using the transistor-level delay model. The performance and accuracy of the transistor-level model are presented in [9]. Because of the simplicity of this model, the output timing can be explicitly expressed as a function of the input logic and timing. The advantages of functional evaluation over operational simulation are both faster execution speed and the possibility of abstraction for high-level representations.

TABLE I
Timing Parameters of an Inverting Buffer

	$VI_I = 1$	$VI_I = 0$
VO_2	0	1
TO_2	$TI_{\phi_2} + R_1C_1 + R_2C_2 + R_4C_3$	$TI_{\phi_2} + R_1C_1 + R_3C_2 + R_5C_3$
CI_2	C_1	C_1
RO_2	R_4	R_5

TABLE II
Timing Parameters of an r-Input nor Gate

case	1 $(0 \rightarrow 1)$	2 $(1 \rightarrow 0)$
TO	$TO = (\max_{i=1}^{r} TI_i) + R_pC_L$	$TO = TI_j + R_jC_L$
$CI_{1,\dots,r}$	$C_{1,\dots,r}$	$C_{1,\dots,r}$
RO	R_p	R_j

Clocked cell B_1 can be decomposed into three gate-level subcells: $B_{1,1}$ contains the input buffers. $B_{1,2}$ is the AND-plane. $B_{1,3}$ is the OR-plane. Both conditions (3.1′) and (3.2′) are satisfied by these three subcells. Therefore, the timing parameters of B_1 can be derived from those of the three subcells. $B_{1,1}$ can be treated in the same way as cell B_2. $B_{1,2}$ and $B_{1,3}$ are both (a collection of) multi-input NOR gates. Refer to Fig. 4(c) for the transistor circuit of an r-input NOR gate. The timing parameters of a single NOR gate are considered for the following two cases:

1) All the pull-down transistors are turned off.

2) Only one, say the jth, pull-down transistor is turned on.

The timing parameters in the above two cases are indicated in Table II.[3]

The values in case 1 are used for $0 \rightarrow 1$ transitions, and those in case 2 are used for $1 \rightarrow 0$ transitions. The delay values estimated for $1 \rightarrow 0$ transitions are always conservative since only one pull-down transistor is assumed. The general situation when more than one pulldown transistors (in series or in parallel) are turned on is discussed in [17]. For the PLA, the output delay of an AND plane under these conditions will be dominated by the pullup of the following OR plane, and hence the issue is largely academic.

Let $TO_{1,i}$, $RO_{1,i}$, and $CI_{1,i}$ be the timing parameters of cell $B_{1,i}$ for $i = 1, 2, 3$. These values can be determined from Table I and Table II, respectively. Cell B_1 is composed from these three subcells, and the stray capacitances and resistances of the interconnection wires are indicated in Fig. 4(d). The results are as follows:

[3]The composition of delays may be done either at the input side or at the output side. The TO values in the tables are composed at the input side.

- $TO_1 = TO_{1,1} + RO_{1,1}(C_a + CI_{1,2}) + \frac{1}{2} R_aC_a + TO_{1,2} + RO_{1,2}(C_b + CI_{1,3}) + \frac{1}{2} R_bC_b + TO_{1,3}$
- $RO_1 = RO_{1,3}$
- $CI_1 = CI_{1,1}$.

Up to this point, B_1 and B_2 are analyzed independently. Again, the TREE algorithm is used to combine these two cells together. During a ϕ_1 period, cell B_2 drives cell B_1, and the output timing of B_1 is equal to $TO_1 + RO_2 \cdot CI_1$. Likewise, the output timing of B_2, during a ϕ_2 period, is equal to $TO_2 + RO_1 \cdot CI_2$.

VI. Data Abstraction

The data used in previous sections are of type "bit." The timing discipline at this level of representation is the "non-overlaping tow-phase clock." This discipline assures that adjacent cells interact only at the bit level, not at the "analog-voltage" level. As a result, the behavior and timing of individual cells can be abstracted into functional form.

In addition to functional abstraction, our timing model also allows data abstraction. To illustrate the principle, consider an n-bit bit-serial multiplier such as the one proposed in [18]. In this implementation, every n consecutive bits into or out of the multiplier are interpreted as one unit (a serial word). The interaction between any two adjacent words are restricted to happen only at the word level, not at the bit level: a property that is true for any bit-serial data path. One common technique to assure this property is by using a "data-stationary" control synchronized with the least significant bit of data.

The timing of a word-level cell is also characterizable by the R, C, and D parameters. Taking the multiplier as an example, D is the minimum clock period required for any internal computation times the number of stages of the multiplier. The minimum clock period can be determined by simulating one stage of the multiplier using the bit level representation. The complexity of one such stage of circuit is manageable, and the critical path and the minimum clock period can be easily determined. The R and C parameters carry the driving and loading characteristics of the connection ports. These parameters are used to check if the clock period needs to be increased due to external connections. The consistency between the bit-level and the word-level representations can be established either by a comparison of simulation results, or by formal arguments such as the fixed-point approach described by Chen [3].

VII. Implementation of a Behavior-Level Simulator

Two kinds of cells are distinguished in our hierarchical timing model: leaf cells and composition cells. Leaf cells are the primitive components that have no subcomponents. A composition cell is a legal composition of leaf cells and other composition cells. With each leaf cell is associated a logic and timing description, which is valid for all possible input patterns and driving and loading conditions of the cell (as long as the composition preserves the semantics of individual component cells). To obtain such a description, any circuit or timing simulator can be used.

With each composition cell are associated a number of subcells which may be either leaf cells or composition cells, and a set of nets indicating how these subcells are connected. There is no explicit logic and timing description for a composition cell. However, it is possible to derive such a description from the descriptions of its subcells either analytically or by simulation. Once the logic and timing description of a composition cell is obtained, the composition cell is reduced to a leaf cell, and the details of its implementation can be eliminated.

Note that a leaf cell may be as simple as a single inverter, or as complicated as the entire data path of an ALU. Obviously, it requires the flexibility of a general purpose programming language to specify the behavior of such a cell. This flexibility is in contrast to most other simulators in which circuits are expressed in terms of only a few different types of primitives whose behaviors are very rigid, and predefined by the simulators.

Instead of designing yet another hardware description language, we embedded the simulator in an existing programming environment. Smalltalk [12], [13] was selected because its object-oriented programming model matches our semantic cell-oriented simulation model in a very natural way. The "messages," "methods," (functions) and "data" of an object correspond respectively to the interface parameters, internal behavior, and internal states of a cell in our simulation model. All debugging tools of the Smalltalk system can be directly applied to investigating and manipulating the design objects of HITSIM in a hierarchical manner.

7.1 Specification of a Cell

"Object," "Class," and "Instance" are the three major concepts in Smalltalk. All information in the Smalltalk system is represented as an object. Objects that respond to the same messages in the same way can share the same generic definition. The generic definition is called a class. Objects generated from this definition are called instances of the class.

In a structured VLSI design, a cell (or a family of cells) is often specified once, and may be instantiated in several different places. Using the Smalltalk terminology, specification of a cell corresponds to setting up a class, and the actual instantiations of the cell correspond to creating instances of the class.

Corresponding to the two kinds of cells in our hierarchical simulation model, there are two predefined classes in HITSIM: "LeafCell" and "CompositionCell." These two classes contain methods to transform cell specifications provided by the user into suitable classes and methods for performing timing simulation. All leaf cells specified by the user will become a subclass of the class "LeafCell"; similarly, for composition cells.

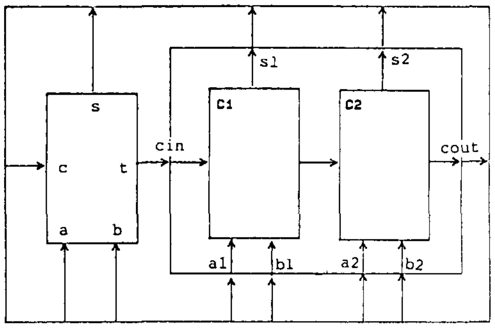


Fig. 5. A three-bit adder.

The three-bit adder shown in Fig. 5 is used as an example to illustrate the specification and simulation of cells in HITSIM. This adder is decomposed into two levels of hierarchy: the top-level composition cell consists of a one-bit adder (a leaf cell) and a two-bit adder (a composition cell). The two-bit adder, in turn, is composed of two one-bit adders.

A leaf cell can be specified by sending the following message to the class "LeafCell".

name: #⟨aString⟩
inputs: #(one or more ⟨inputSpec⟩'s)
outputs: #(one or more ⟨outputSpec⟩'s)
states: #(zero, one or more ⟨stateSpec⟩'s)
behavior: '⟨Smalltalk code⟩'. (5)

⟨aString⟩ of (5) specifies the name of the leaf cell. ⟨inputSpec⟩'s, ⟨outputSpec⟩'s, and ⟨stateSpec⟩'s specify the name and other attributes of the input ports, output ports, and internal states of the leaf cell:

1. One ⟨inputSpec⟩ corresponds to each input port, and consists of two items: the name and the loading capacitance of the input port.
2. One ⟨outputSpec⟩ corresponds to each output port, and consists of three items: the name of the output port, and two values of driving resistance: the first used for 1 → 0 transitions and the second used for 0 → 1 transitions.
3. One ⟨stateSpec⟩ corresponds to one state variable, and consists of only one item: the name of the variable.

⟨Smalltalk code⟩ in (5) is a text of Smalltalk source code for describing the logic and timing of the leaf cell: a mapping from the input states, current internal states and input timing to the output states, next internal states and output timing of the cell. Any construct of the Smalltalk language can be used in this text. If auxiliary variables are needed for the computation, they can be declared here.

The one-bit adder of Fig. 5 is specified as follows:

```
LeafCell name: #Add1
         inputs: #((a 1) (b 2) (c 1))
         outputs: #((s 42 12) (t 38 10))
         states: #()
         behavior: 's ← (a xor: b) xor:c
                    t ← a ifTrue:[b or:c] ifFalse:[b
                         and:c].
                    Ts ← PhyTime + 8
                    Tt ← PhyTime + 10'.          (6)
```

Upon receiving the above message, class "LeafCell" creates a subclass of its own, called "Add1" with the following six instance variables: *a*, *b*, *c*, *s*, and *t* in which the logic values of the corresponding input and output ports can be stored. *Ts* and *Tt* store the timing outputs *s* and *t*. The prefix *T* to the name of an output port is a convention adopted by HITSIM to associate timing with the output port. In general, if a cell contains *m* inputs, *n* outputs, and *l* internal states, then $m + 2n + l$ instance variables are created. These variables are referred to in the behavior section of message (6). Input timing is not always required to specify the logic and timing of a cell. For instance, the output timing of cell Add1 depends only on the time when the cell is excited ("PhyTime", a global variable used in HITSIM) so that no additional variables are created for individual input ports. In case the timing of a particular input port is important in the specification of a cell, it must be declared explicitly as a state variable.

After class "Add1" is created with its associated instance variables, the behavior section of the message is passed to the Smalltalk compiler, which returns a "compiledMethod" under message heading "cellExcited." Again, this heading is a convention used in HITSIM. This compiledMethod will be executed every time the message "cellExcited" is sent to an instance of the class. Note that users always interpret the data types of instance variables in their own way. For cell Add1, all input and output states are of type Boolean. For other cells, other data types may be used. During composition, proper coersion cells may be inserted between ports of different types. As to delay parameters, the unit for timing is nanoseconds; that for resistance is kilometers and that for capacitance is picofarads.

Remarks:

- Implied in message (6) is that, whenever an input of cell Add1 changes state, the same code (cellExcited) is executed. In many practical circuits, however, different actions may be required when different inputs change. The capability of defining input-specific actions is also included in HITSIM; examples can be found in [17].
- Clocks are not treated as special signals in HITSIM, although they can be made special by the user in the specification of the methods of the cell. For instance, a specification of the form "Phase_1 ifTrue:[· · ·]" is often used for a cell active during Phi1. When Phase_1 is low (false), effectively nothing happens when this specification is executed.

A composition cell is specified by sending the following message to the class CompositionCell:

name: #⟨aString⟩
inputs: #(one or more ⟨inputPortSpec⟩'s)
outputs: #(one or more ⟨outputPortSpec⟩'s)
subcells: #(one or more ⟨subcellSpec⟩'s)
connections: #(one or more ⟨connectionSpec⟩'s). (7)

Every ⟨inputPortSpec⟩ or ⟨outputPortSpec⟩ of (7) corresponds to an input or output port of the composition cell, and consists of only one item: the name of the port. The loading capacitances and driving resistances need not be specified because these values are all implicit in the connection list. Also, there are no explicit internal states of a composition cell.

Every ⟨subcellSpec⟩ of (7) corresponds to a subcell of the composition cell, and consists of two items: the instance name and the class name of the subcell. A subcell is either a leaf cell or a composition cell. Every ⟨connectionSpec⟩ of (7) corresponds to an interconnection net, and consists of two items: the name of the driving node, and a tree structure describing the topology and physical parameters of the net. This structure is typically generated from a router interface.

The two-bit adder of Fig. 5, for instance, is specified as follows. Similarly, for the three-bit adder.

```
CompositionCell name: #Add2
                inputs: #(a1 b1 a2 b2 cin)
                outputs: #(s1 s2 cout)
                subcells: #((C1 Add1) (C2 Add1))
                connections: #(((C1 t) 2 1 (C2 c)) · · ·
```

7.2 Simulation of a Cell

In HITSIM, simulation is always performed on a composition cell (referred to as the top-level composition cell). This composition cell may be as complicated as an entire system consisting of several levels of hierarchy, or as simple as a composition cell that contains only one leaf cell. Depending on the level of abstraction currently under investigation, a cell under the top-level composition cell may be represented either as a composition cell or as a leaf cell. Note that different data types may be used when the same design is represented at different levels of abstraction. Given a top-level composition cell and the level of abstraction of its component cells, the following actions are taken before the actual simulation starts.

1) An instance of the cell together with all the subcells under its hierarchy is created.

2) All the nets that span over more than one composition level are flattened, the delay values among the driving node and loading nodes of the net is calculated, and proper pointers among the nodes are established for preforming simulation. Note that, in addition to the input and output ports of the top-level composition cell, only leaf cells are involved in the simulation process. No overhead is spent on travelling through the intermediate-level composition cell. The nodes involved in the simulation process are classified into two groups: 1) driving end of a net which is either an input of the top-level composition cell or an output of a leaf cell; 2) loading end of a net which is either an output of the top-level composition cell or an input of a leaf cell.

HITSIM is an event-driven simulator. Associated with every event are the time to excite the event, the node to switch, and the target state of the node. When an event is scheduled, a pointer is established from the corresponding node to the event for possible cancellation or rescheduling of the event later. The following pseudo-code indicates the main loop of the simulation process.

```
while notEmpty(EventQueue) do
begin
  take the first event from the EventQueue;
  update the (global) physical time;
  case (the node corresponding to the event) of
    driving node of a net:
      schedule (cancel or reschedule) all the loading
      nodes of the net to switch
        at time determined by the delays calculated
        for the net;
    loading node of a net:
      for all the leaf cells that is affected by the node
      do
      begin
        execute the code that specify the behavior
        and timing of the cell;
        schedule (cancel or reschedule) the affected
        outputs of the cell to switch
        at time determined by the output timing;
      end
  end
end
```

7.3 Simulation Results

RC-based transistor-level simulators [9], [19] are capable of analyzing the timing of digital MOS circuits with resonable accuracy. For circuits containing less than one hundred transistors, this type of simulators run two to three orders of magnitudes faster than SPICE simulation [6]. The ratio grows drastically as the size of circuits increases. By imposing timing disciplines on the design and partitioning of circuits, HITSIM speeds up the simulation further by achieving the following:

1) functional abstraction: circuit timing is expressed in functional form and is directly executable.

2) data abstraction: different levels of representation can be used to express the behavior and timing of a design in a consistent manner.

The level of accuracy is the same as the *RC*-based transistor-level simulators, and the performance advantage is enormous. For a typical simulation run on a 32-bit bit-serial multiplier, the speedup is 10–30, if the circuit is simulated under the bit-level representation. If the word-level representation is used, the speedup is about 500–2000.

To compare HITSIM with other functional simulators, a PLA with 20 inputs and 60 min-terms is simulated using HITSIM with mixed integer and bit representations. Using the same representation, the behavior of the PLA is hard-coded in Pascal, and excuted on a HP-9836 workstation. This station is of comparable hardware capability as the Xerox Dolphin workstation in which the HITSIM is implemented. With the debugging flags on, the Pascal code runs about three times faster than the HITSIM, and with debugging flags off, thirty times faster. Note that the Pascal code, not a simulator, represents the fastest possible way to simulate the PLA. The HITSIM, on the other hand, is a simulator that can take any code that describes the behavior and timing of a design.

VIII. An Integrated Design System

The HITSIM simulator can be combined with other tools to form an integrated design system that fully supports the structured design methodology. The design flow of one such system currently under integration is indicated in Fig. 6.[4]

The blocks bounded by bold lines are programs, and those bounded by regular solid lines are data sets. The blocks bounded by dash lines are tasks that are currently performed by the user; automation of these tasks requires more disciplines on the design. The structure-level simulation indicated in Fig. 6 can be performed by any simulators that the user prefers. In addition to HITSIM, two other programs used are the Pooh leaf cell design and syntactic composition system[5] developed by Whitney [20] and the BBL general routing system developed by Chen, etc. [21]. These two systems are selected because they both fit in our general framework in a clean and natural way.

- The Pooh system manipulates and generates circuit schematics (listing of transistors and wires and their sizes) and design-rule-correct layouts based on a unified representation. This is quite a contrast to the traditional approach of extracting circuit schematics from physical layouts [22], a process that is not only timing consuming, but also incapable of determining the semantic boundaries within a circuit for performing hierarchical simulation.
- The BBL system handles arbitrarily shaped rectilinear blocks, minimizes layout area and assures 100 percent routing completion. This system also allows routing to be done in a hierarchical manner.

Given the specification of a target circuit, the user first determines the timing strategy, the set of leaf cells to be designed, and the composition hierarchy for building up the circuit. Every leaf cell can be designed using the Pooh system which, upon completion, generates the following four pieces of data: 1) the circuit schematics of the cell for performing timing simulation; 2) the driving resistances and loading capacitances of the ports required for HITSIM simulation; 3) the size of the cell and the coordinates of the ports for performing routing; and 4) the physical layout of the cell which will be combined with the router output to form the complete layout of the chip.

[4]The system can be used either in a top-down or in a bottom-up fashion. For ease of explanation, the bottom-up design flow is presented.

[5]The Pooh system consists of a physical leaf-cell design entry and a syntactic composition system. To run hierarchical simulation, the result of a syntactic composition must be a semantic cell. This resulting cell, called a functional block in many design systems, is used as a (logical or semantic) leaf cell in the HITSIM simulator.

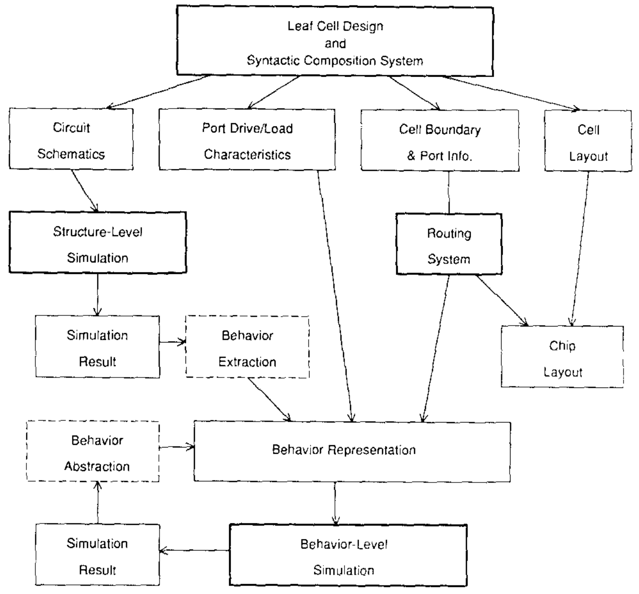

Leaf Cell Design and Syntactic Composition System
Circuit Schematics
Port Drive/Load Characteristics
Cell Boundary & Port Info.
Cell Layout
Structure-Level Simulation
Routing System
Simulation Result
Behavior Extraction
Chip Layout
Behavior Abstraction
Behavior Representation
Simulation Result
Behavior-Level Simulation

Fig. 6. An integrated design system.

Based on the simulation results, the user determines the behavior and timing of each leaf cell. The behavior and timing descriptions of a collection of leaf cells are used for performing HITSIM simulation, using the following information: 1) the driving resistances and loading capacitances of the ports generated by the Pooh system; and 2) the tree structure and physical parameters of the interconnects generated by the router. Note that both the behavior and timing specifications of the cells and the routing data are maintained in a hierarchical manner. With the proper functional and data abstraction of the composite cells, the user can flatten the design at any desired level for performing the HITSIM simulation.

Acknowledgment

This particular suitability of Smalltalk for an embedded behavior-level simulation environment was suggested by Prof. Marina Chen of Yale University. Some preliminary Smalltalk simulation experiments were performed in collaboration with Dr. Chen on "Dolphin" hardware at Xerox, Pasadena, CA, courtesy K. Laprade, R. Lansford, and D. Stewart. The HITSIM simulator presented in this paper was implemented on the experimental "Magnolia" workstation of Tektronix (Beaverton, Oregon) which was generously provided by W. Cunningham and K. Bradley.

References

[1] C. A. Mead and L. A. Conway, *Introduction to VLSI Systems*. Reading, MA: Addison Wesley, 1980.

[2] C. A. Mead, "Structural and behavioral composition of VLSI," in *Proc. IFIP Int. Conf. VLSI*, (Trondheim, Norway), Aug. 1983, paper TC 10/WG 10.5, pp. 3-8.

[3] M. C. Chen and C. A. Mead, "A hierarchical simulator based on formal semantics," in *Proc. 3rd Caltech Conf. VLSI*, Mar. 1983, pp. 207-223.

[4] M. C. Chen, "Space-time algorithms: Semantics and methodology," Ph.D. dissertation, Computer Science, Caltech, Pasadena, May 1983.

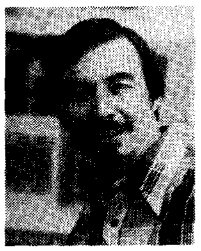
[5] D. Scott and C. Strachey, *Toward a Mathematical Semantics for Computer Languages*. New York: Polytechnic Inst. Brooklyn Press, 1971.

[6] L. W. Nagel, "SPICE2: A computer program to simulate semiconductor circuits," Electronics Research Laboratory, Univ. California, Berkeley, CA, ERL Memo ERL-M520, Dec. 1975, pp. 901-910.
[7] B. R. Chawla, H. K. Gummel, and P. Kozak, "MOTIS—An MOS timing simulator," *IEEE Trans. Circuits Syst.*, vol. CAS-22, no. 12, pp. 901-910, Dec. 1975.
[8] T-M. Lin and C. A. Mead, "Signal delay in general RC networks with application to timing simulation of digital integrated circuits," in *Proc. 3rd MIT Conf. Advanced Research in VLSI*, pp. 93-99, Jan. 1984.
[9] T-M. Lin and C. A. Mead, "Signal delay in general RC networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, Oct. 1984.
[10] C. Seitz, "System Timing," Introduction to VLSI Systems, Reading, MA: Addison-Wesley, 1980, chap. 7.
[11] M. C. Chen, "A methodology for hierarchical simulation of VLSI systems," Computer Science, Yale Univ., no. YALEU-DCS-RR-325, Aug. 1984.
[12] A. Goldberg and D. Roson, *Smalltalk-80, the Language and Its Implementation.* Reading, MA: Addison-Wesley, May 1983.
[13] A. Goldberg, *Smalltalk-80, the Interactive Programming Environment.* Reading, MA: Addison-Wesley, 1984.
[14] J. Rubinstein, P. Penfield, and M. Horowitz, "Signal delays in *RC* tree networks," *IEEE Trans. Computer-Aided Des.*, vol. CAD-2, no. 3, pp. 202-211, July 1983.
[15] W. C. Elmore, "The transient response of damped linear networks with particular regard to wideband amplifiers," *J. Appl. Phys.*, vol. 19, no. 1, pp. 55-63, Jan. 1948.
[16] M. Rem and C. A. Mead, "A notation for designing restoring logic circuitry in CMOS," in *Proc. 2nd Caltech Conf. VLSI*, pp. 399-411, Jan. 1981.
[17] T-M. Lin, "A hierarchical timing simulation model for digital integrated circuits and systems," Ph.D. dissertation, Computer Science, Caltech, Pasadena, July 1984.
[18] R. F. Lyon, "Two's complement pipeline multipliers," *IEEE Trans. Commun. Technol.*, vol. COM-24, no. 4, pp. 418-425, April 1976.
[19] C. J. Terman, "Simulation tools for digital LSI design," Ph.D. dissertation, M.I.T., Oct. 1983.
[20] T. Whitney and C. A. Mead, "Pooh: A uniform representation for circuit level designs," in *Proc. IFIP TC 10/WG 10.5 Int. Conf. VLSI*, (Trondheim, Norway), pp. 401-411, Aug. 1983.
[21] N-P. Chen, C-P. Hsu, and E. S. Kuh, "The Berkeley building-block layout system for VLSI design," in *Proc. Int. Conf. VLSI*, (Trondheim, Norway), pp. 37-44, Aug. 1983.
[22] C. Baker, "Artwork analysis tools for VLSI circuits," MIT/LCS/TR-239, May 1980.

Tzu-Mu Lin was born in Taipei, Taiwan, Republic of China, on September 15, 1956. He received the B.S. degree in electrical engineering from National Taiwan University, Taiwan, in 1978, and the Ph.D. degree in computer science from California Institute of Technology, in 1984.

From July 1980 to August 1984, he worked as a Graduate Research Assistant at California Institute of Technology. He was involved in many projects on integrated circuits and systems design and verification. He is currently with the Silicon Compilers Inc., where his main responsibility is to develop methodology and supporting software for assuring the timing integrity and analyzing the performance of compiled VLSI designs. His research interests also include silicon compilation, computer architecture, and concurrent processing.

*



Carver A. Mead, Gordon and Betty Moore Professor of Computer Science, has taught at the California Institute of Technology in Pasadena, CA, for over twenty years.

His current research focus and teachings are in the area of VLSI design, ultra-concurrent systems and physics of computation. He has worked in the fields of solid-state electronics and the management of complexity in the design of very large-scale integrated circuits. In addition to his wide range of interests in solid-state physics, microelectronics and biophysics, he has written, with Lynn Conway, the standard text for VLSI design, *Introduction to VLSI Systems.*

Among his many awards and honors, are the T. D. Callinan, the Electronics Achievement, the Harold Pender, the John Price Wetherhill Medal, and the Harry Goode Memorial Award. Dr. Mead is a fellow of the American Physical Society and a member of the National Academy of Engineering.