Project No. 1058

Accurate Timing Verification Algorithms for Synchronous MOS Circuits. *

E. Vanden Meersch, L. Claesen, H. De Man †

IMEC vzw, Kapeldreef 75, 3030 Leuven, Belgium, Tel +32-16-281203

Abstract

In this paper, new algorithms for timing verification and analysis of synchronous MOS circuits are presented. It is shown that a very wide class of timing faults, including faults related to the use of level sensitive latches, conditional or derived clocks and precharging techniques, although apparently each of a different nature, can be dealt with in a unified way. Constraints are formulated which guarantee the absence of such faults in the circuit for specified clock waveforms. A rule-based method is used to transform the network of MOS transistors, obtained from the physical layout by extraction, into a network of unidirectional subcircuits. Each subcircuit is characterised by a logic model, which is used to accurately derive the timing constraints. To verify these constraints, an algorithm is adopted which has been presented in [10]. The timing model of the circuit, required by this algorithm, is derived from the created subcircuit structure. Because of the rule-based implementation, the resulting timing verifier can be used for a wide range of static and dynamic nMOS and CMOS circuit designs and requires no user input besides the circuit and the clock waveforms, which greatly enhances the efficiency of its use.

1 Introduction

Timing verifiers have become recognized tools for the verification of the performance of electronic designs at transistor level [4,5,16]. Commonly, they calculate for each node in a given circuit the settling time, or the latest point in time at which a signal transition can take place. The required user input consists of the settling times at the primary circuit inputs, which is a distinct advantage over simulation. The user of the tool is not required to specify detailed input excitations: the calculated settling times are the worst case for all possible excitations.

In the discussion of the functionality and implementation of a timing verifier at transistor level (i.e. which handles netlists extracted from mask layout), three main issues arise:

1.1 The signal flow modeling problem

The following problem is crucial in timing verification at transistor level: From the connectivity of the MOS devices, which are essentially bidirectional in nature between the source and drain terminals, a unidirectional model for the propagation of signal transitions is to be constructed.

*Work sponsored by the EC under the ESPRIT-1058 project

†Professor at Kath. Univ. Leuven

In order to compute the latest arrival time at a node, all the possible ways in which a signal transition can propagate up to this node must be known. In all known timing verifiers, implicit use is made of knowledge about circuit design techniques in order to derive this unidirectional model. This means however that deviations of the target design style require modifications of the tool, specific preprocessing of the circuit, or additional user information about the purpose of the circuitry [7], [14], [10].

For this reason, a timing verifier inevitably has a specific class of circuits as its field of application. The introduction by designers of new types of latches or gates, bootstrapping techniques, bleeder transistors... is always liable to impair the modeling algorithm for the signal flow in such devices, on which the timing analysis is based. This property is in contrast to analog simulators, which can be used for any nMOS or CMOS circuit, irrespective of its purpose.

In this paper we propose a method which makes the knowledge used in the signal flow modeling algorithm explicit, by performing a rule-based transformation of the transistor circuit to a network of unidirectional subcircuits. From this subcircuit structure, the signal flow model can be formally derived, as well as the timing constraints to be verified. The rule base contains a description of the possible subcircuits that can occur in the designs that are to be handled by the timing verifier, together with a model for the logic behaviour of the subcircuits. Modifying or adding the description of a subcircuit requires no low-level programming and is therefore relatively easy.

1.2 The delay modeling problem

In this paper, we will refer to the model for the signal flow in the circuit as the 'timing model'. It is usually represented as a directed graph [5], in which the edges represent signal propagation through a small part of the circuit. The edges are weighted with the associated propagation delay. Several methods have been published to calculate this delay [6], [11], [13], [12], [15]. Each method offers a different trade-off between accuracy, computational effort and generality. In [15], the approximation is made that a signal transition at a node takes place whenever a path of transistors starts conducting. This single path can be approximated as an RC-chain which (dis)charges the load capacitor at the end of the chain. This method offers very high efficiency but suffers from lack of generality. Furthermore, the delay of a gate is in many cases determined by the complex interaction of several transistor paths that simultaneously switch on and/or off (this is already the case in a basic CMOS invertor if the input transition is slow). The simple model may therefore lead to considerable inaccuracy, if not tuned to the design style.

We decided to use local analog simulation to extract delay values. This method offers very high accuracy and generality, but is computationally expensive.

1.3 Timing faults in synchronous circuits

As mentioned before, a timing verifier for combinational circuits calculates settling times at each node. It is up to the designer to draw conclusions about whether or not the circuit can be inserted in a synchronous system without causing timing faults. More advanced timing verification tools allow to directly analyse synchronous circuits. The important problem that arises here is to determine when the calculated settling times with respect to the different transitions of the clock actually imply a timing fault in the circuit.

Important work in the direction of automating the timing verification for synchronous circuits has been done in [10]. This author describes how the timing analysis of synchronous circuits, irrespective of the number of clocks, can be performed by applying a PERT critical path evaluation of the timing model during each phase of the clock cycle. This is repeated over a number of clock cycles until the settling times converge.

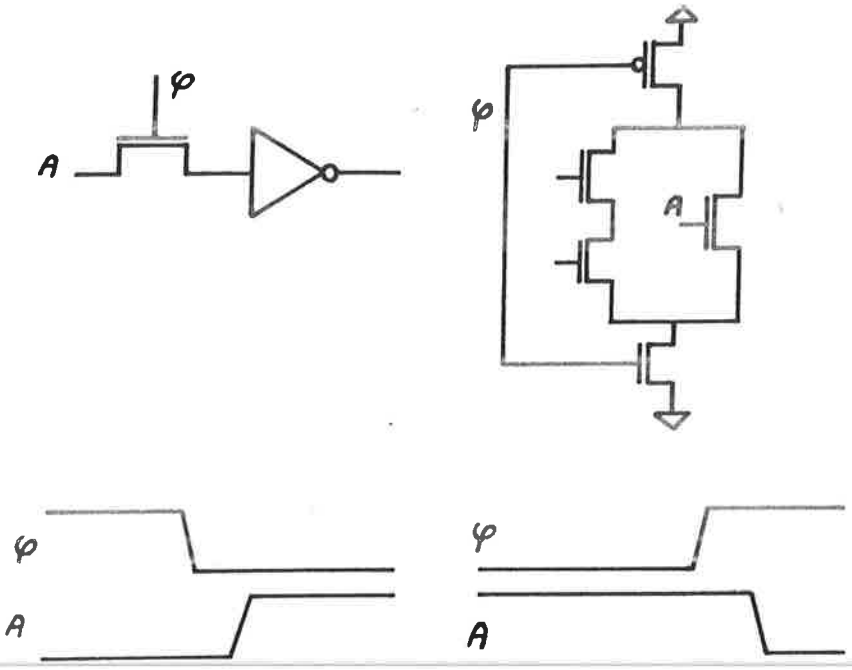


Figure 1: Examples of timing faults in clocked circuits.

In combination with this analysis algorithm, a rigourous method is needed to derive the constraints on the settling times which are to be verified. In this paper, it will be shown how a set of constraints can be derived which cover timing faults due to the improper timing of signals at inputs of level sensitive latches and precharged circuits. Also the specific timing constraints related to the use of conditional clocks are dealt with in a unified way. In figure 1, two situations are shown which lead to a timing fault in the circuit. In the first example, the latch output has not assumed the correct value before the start of the 'hold phase'. The signal A arrives too late to be gated in the latch. In the second example, the precharged output of the gate unintentionally discharges. Both timing faults are seemingly unrelated. It can however be seen that in both examples the final output state is a storage (or memory) state which stores the wrong logic value.

The basic observation that, when a memory state (M) can exist during a certain phase of the clock period, it must store the correct logic value, will be generalised and put in a form that is suitable for computer verification.

In Section 2, we will discuss the rule-based method to partition an MOS circuit into a network of unidirectional subcircuits. In Section 3 the logic model of a subcircuit will be defined as an expression of four operators. The possible output states that are considered are High, Low, Error and Memory (1,0,E,M). A proof will be given that the operators are sufficient to model arbitrary subcircuits. In Section 4, the timing constraints will be formulated. Section 5, 6 and 7 deal with the timing model and the timing analysis algorithm, which is needed to verify the timing constraints. Section 8 discusses the problem of compatibility of logic states at different nodes in the circuit. It will be shown that a detection of statically incompatible logic states is needed to avoid false timing errors and to solve the false path problem in the timing analysis.

2 The unidirectional model

In this section we discuss the transformation of an MOS transistor network to a network of subcircuits. As pointed out above, the purpose of this tranformation is to obtain a unidirectional model of the circuit: as opposed to transistors, in which the direction of signal flow between source and drain terminal depends on its connections with the environment, a subcircuit is required to have a number of inputs and one output. Each node in the network of subcircuits is connected to at most one subcircuit output. This implies that the direction of signal flow in the transformed network is unambiguous.

First, a notation for networks is given which will be used throughout this paper. Next, the formal restrictions of the network of subcircuits will be given. Finally, the rule-based implementation of the actual transformation algorithm is discussed.

Both the initial transistor circuit and the result of the transformation are networks, which can be described as a set of nodes N and a set of elements E. Each net $n \in N$ is associated to a set of elements $\alpha(n)$. Each element $e \in E$ is associated to a set of nets $\beta(e)$. In our case, different types of relations exist between the elements and the nodes, and the lists $\alpha(n)$ and $\beta(e)$ are differentiated accordingly. Using this notation, the initial network of MOS transistors is described as $E1 = \{m1, m2, \ldots\}$, $N1 = \{n1, n2, \ldots\}$. If net $\mathbf{n}$ is connected to the source or drain of transistor $\mathbf{m}$, this is denoted as

$$\begin{aligned} & m \in \alpha_{io}(n) \\ \text{or}: \quad & n \in \beta_{io}(m) \end{aligned}$$

If net $\mathbf{n}$ is connected to the gate of transistor $\mathbf{m}$, this is denoted as

$$\begin{aligned} & m \in \alpha_{g}(n) \\ \text{or}: \quad & n \in \beta_{g}(m) \end{aligned}$$

The network of subcircuits after transformation is described as $E2 = \{s_1, s_2, \ldots\}$ and N2, which is a subset of N1. E2 is a set of sets s_i of elements from E. If net $\mathbf{n}$ is connected to the input of subcircuit $\mathbf{s}$, this is denoted as

$$\begin{aligned} & s \in \alpha_{in}(n) \\ \text{or}: \quad & n \in \beta_{in}(s) \end{aligned}$$

If net $\mathbf{n}$ is connected to the output of subcircuit $\mathbf{s}$, this is denoted as

$$\begin{aligned} & s \in \alpha_{out}(n) \\ \text{or}: \quad & n \in \beta_{out}(s) \end{aligned}$$

Each subcircuit represents a set of transistors. Irrespective of the actual partitioning of the transistor circuit into subcircuits, the following formal restrictions must be satisfied: Each transistor must belong to (at least) one subcircuit, and the inputs and output of each subcircuit must be connected to an internal transistor of the subcircuit.

$$\begin{aligned} & m \in E1 \;\Rightarrow\; \exists\, s \in E2;\; m \in s \\ & n \in \beta_{in}(s) \cup \beta_{out}(s) \;\Rightarrow\; \exists\, m \in s;\; n \in \beta_{io}(m) \cup \beta_{g}(m) \end{aligned}$$

In addition, in order to insure that the direction of signal flow in the network is unambiguous, for every node and subcircuit must hold:

$$\begin{aligned} & |\beta_{in}| \geq 1 \\ & |\beta_{out}| = 1 \\ & |\alpha_{out}| \leq 1 \end{aligned}$$

```
(RULE exor)
(IF (
      (nmos e6 n4 n3 n6)
      (nmos e4 n1 n4 n2)
      (nmos e2 n1 n2 n4)
      (pmos e1 n3 n2 n1)
      (pmos e3 n1 n3 n2)
      (pmos e5 n5 n3 n4)
      (vdd n5)
      (gnd n6))
THEN (
      (subcircuit s1)
      (name exor)
      (output n1)
      (input n2 n3)
      (element e1 e2 e3 e4 e5 e6)
      (function (or (and (not n3) n2) (and (not n2) n3))))))
```

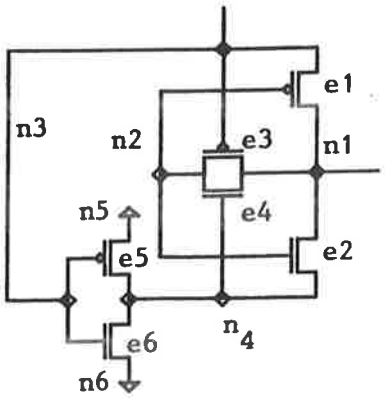


Figure 2: Example of a subcircuit partitioning rule.

In Section 5, additional criteria will be formulated for a transistor network to form a valid subcircuit. Within all of these formal restrictions, a partitioning of transistors must be chosen which yields subcircuits with minimal size and optimal electrical properties. We preferred not to choose a hard-coded solution to the partitioning problem, as this would inevitably limit the use of the timing verifier to a certain class of circuits. In order to avoid this bottleneck to the applicability of the program, a rule-based implementation has been developed.

Basically, the rule base contains a list of connectivity patterns of transistors which correspond to the possible types or classes of subcircuits, together with the behavioural model of the subcircuits. All occurences of these types in a given circuit are identified, using pattern matching techniques [2]. Figure 2 gives an example of a rule which describes a pass-gate implementation of the EXOR operation. Adding or modifying such a rule is easy and can, if necessary, be done by a designer.

It will be described below how the subcircuit partitioning is done by pattern matching on the basis of these rules. In the first two phases of this process, the rules are compiled. This is only necessary after a change of the rule base.

1. **Expansion of the pattern.** The specification of the transistors is expanded into more basic connectivity conditions (denoted by Ci):

```
(nmos e1 n1 n2 n3)
      ⇓
(ntype e1)          C1
(gate e1 n1)        C2
(io e1 n2)          C3
(io e1 n3)          C4
```

2. **Translation of the pattern.** The arguments in the connectivity conditions are treated as variables to which a transistor (the arguments with name e1, e2,..) or a node (the arguments with name n1, n2,..) can be assigned. The principle of the translation is the following: each connectivity condition Ci corresponds to a statement Si with a block that will be executed repeatedly (if the statement is a loop) or conditionally (if the statement is a test). The connectivity conditions are translated sequentially, and the corresponding statements are nested:

   C1 C2 C3 ... → S1{S2{S3{ ... }}}

   The statements are chosen in such a way that the inner block of each statement is executed with the already encountered variables assigned to all possible sets of nodes and transistors that satisfy the preceding connectivity conditions.

   At the moment a certain condition is to be translated, 0,1 or 2 of its variables are assigned a node or a transistor. Suppose e1 is known at the moment the following conditions are to be translated:

   ... preceding conditions
   (io e1 n3)
   (ntype e1)
   (io e2 n3)
   (gate n3 e1)
   ... remaining conditions

   It can be seen that each of the following statements creates an inner block which is traversed for all instantiations of the translated part of the pattern.:

   ... translation of the preceding conditions
   FOR n3 := all nodes $\in \beta_{io}$(e1) {
    IF e1 is an ntype transistor {
     FOR e2 := all transistors $\in \alpha_{io}$(n3) {
      IF n3 $\in \beta_{g}$(e1) {
       ... translation of the remaining conditions
   ...}}}}...

   As a consequence of the translation principle, the innermost block of the generated code is executed for each instance of the pattern in the circuit. In this position, the THEN-part of the rule is inserted.

3. **Execution of the code.** The compiled code implements the transformation from transistor network to network of subcircuits, and is linked to the SLOCOP timing verifier [16]. This way, the tool is 'customized' for a particular design style. Because subcircuits are usually small, the pattern matching is quite fast: a circuit can be partitioned at a rate of 300 transistors/CPU second on a VAX 8600 computer. The efficiency of the generated code strongly depends on the order in which the connectivity conditions are translated; the rule is preprocessed in order to determine an optimal order before translation.

The currently available rule base in SLOCOP [16] contains descriptions which cover complete classes of subcircuits, such as complementary CMOS gates. This involves more sophisticated connectivity conditions, like for instance the condition that a transistor path must or must not

exist between specified nodes. A class of subcircuits is usually described by means of several rules. The basic translation process for each rule is however the same.

3 The logic model of the subcircuits

In the rule base, the possible generic subcircuits are described manually in the LEXTOC language [2] for a given design style. We will now discuss how the logic model for a subcircuit can be specified in the LEXTOC language for use in the SLOCOP [16] timing verifier. This model is the basis for the derivation of the timing constraints in the circuit.

From the logic point of view, the exact voltage at the input or output of a subcircuit is not relevant. For each technology, there exists a function $D(v) : \quad R \rightarrow \{0,1,E\}$, which maps a voltage v onto one of three states:

0 : logic low
1 : logic high
E : error

The **E** indicates an indetermined signal state that is not accepted by the logic model that is build of the subcircuit.

For a given subcircuit s, the following notation is introduced to denote inputs and output of a subcircuit s and the logic states at these nodes:

$$\beta_{out}(s) = \{n_0\}$$
$$\beta_{in}(s) = \{n_1, n_2, \ldots n_N\}$$

$$D(v(n_0)) = y$$
$$D(v(n_i)) = x_i$$

$$X = [x_1 \ldots x_N]^T$$

The output of an arbitrary subcircuit of MOS transistors can, in accordance with previous work [8] be characterised by four states, which lead to a logic model as a function:

$$F : \{0,1\}^N \rightarrow \{0,1,E,M\}$$

with:

$$F(X) \in \{0,1,E\} \Rightarrow F(X) = y$$
$$F(X) = M \Rightarrow y \text{ is not determined by } X$$

. Physically, the M state indicates that the output of the subcircuit is in a bistable state. This means that both an electrically stable 0 or 1 state can exist at the output, together with these input states. Common ways to implement a bistable state is by electrical isolation of a node or by a loop with an even number of inversions. Figure 3 shows examples of subcircuits whose output state is M.

Because of the introduction of the M and E state, the basic operators of boolean algebra are not sufficient to describe the function F(X) and must be extended. We will use the following set of operators in order to obtain a concise expression for F(X):

conjunction (AND)

IF $x_i = E \ (1 \leq i \leq N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N = E$

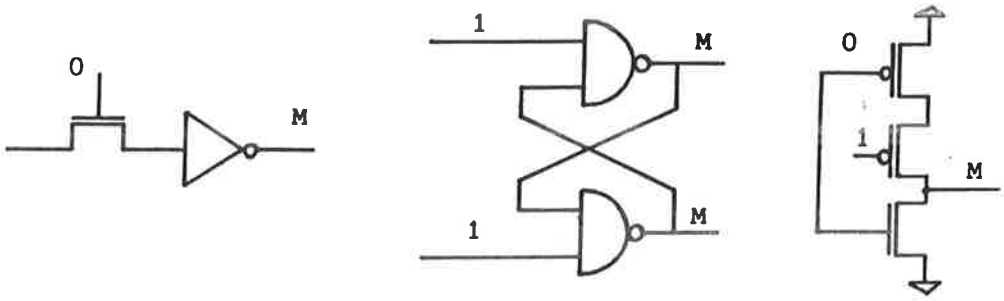


Figure 3: Examples of the bistable (M) state.

ELSE IF $x_i = 0 \quad (1 \leq i \leq N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N = 0$
ELSE IF $x_i = 1$ $(i = 1 \ldots N)$ THEN $x_1 \wedge x_2 \ldots \wedge x_N = 1$
OTHERWISE $x_1 \wedge x_2 \ldots \wedge x_N = M$

disjunction (OR)

IF $x_i = E \quad (1 \leq i \leq N)$ THEN $x_1 \vee x_2 \ldots \vee x_N = E$
ELSE IF $x_i = 1 \quad (1 \leq i \leq N)$ THEN $x_1 \vee x_2 \ldots \vee x_N = 1$
ELSE IF $x_i = 0$ $(i = 1 \ldots N)$ THEN $x_1 \vee x_2 \ldots \vee x_N = 0$
OTHERWISE $x_1 \vee x_2 \ldots \vee x_N = M$

negation (NOT)

IF $x = E$ THEN $-x = E$
IF $x = 1$ THEN $-x = 0$
IF $x = 0$ THEN $-x = 1$
IF $x = M$ THEN $-x = M$

selection

IF $x_i = 0$ $(i = 1 \ldots N)$ THEN $(x_1, x_2 \ldots, x_N)|(y_1, y_2 \ldots, y_N) = M$
ELSE IF $x_i = 1$ $(1 \leq i \leq N)$ AND $x_j = 0$ $(j = 1 \ldots N, i \neq j)$ THEN
$(x_1, x_2 \ldots, x_N)|(y_1, y_2 \ldots, y_N) = y_i$
ELSE IF $x_i = M$ $(1 \leq i \leq N)$ AND $x_j = 0$ $(j = 1 \ldots N, i \neq j)$ THEN
$(x_1, x_2 \ldots, x_N)|(y_1, y_2 \ldots, y_N) = M$
OTHERWISE
$(x_1, x_2 \ldots, x_N)|(y_1, y_2 \ldots, y_N) = E$

The select operator is necessary as it is the only operation which can evaluate to E or M when all its operands are 0 or 1.

In order to be able to specify the function F(X) of classes of subcircuits, an additional construct in the rule base language is necessary: the **transmission function** TF between two nodes in a subcircuit is a boolean function of the subcircuit inputs which evaluates to 1 if a conducting transistor path between the two nodes exists. The transmission function avoids that logic equations have to be specified in the rule-base for each instance of a specific type of gate (see for example the pull up and pull down configurations in a static gate in figure 4).

In the methods as implemented in the SLOCOP timing verifier [16], a LEXTOC rule base,

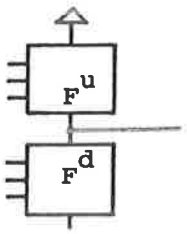


Figure 4: CMOS pullup-pulldown transistor configuration.

together with the logic functions for the subcircuits has to be defined for a specific design style.

Recently however, useful methods have been published for the automatic derivation of algebraic equations for transistor circuits [17,18]. Future research should try to build on such a method, in order to automate the generation of the logic function of the subnetworks. This will avoid the current manual specification of the logic function in the rule base.

In the rule base, prefix notation is used for the subcircuit expression. Use is made of the five constructs below:

(not x1 **)**	negation operator
(and x1 x2 ...**)**	conjunction operator
(or x1 x2 ...**)**	disjunction operator
(select (x1 x2 ...)(y1 y2 ...)**)**	selection operator
(tf n1 n2**)**	transmission function

In figure 5, a number of examples of subcircuits are shown together with the specification of the logic model F(X), as specified in the LEXTOC rule base for CMOS design.

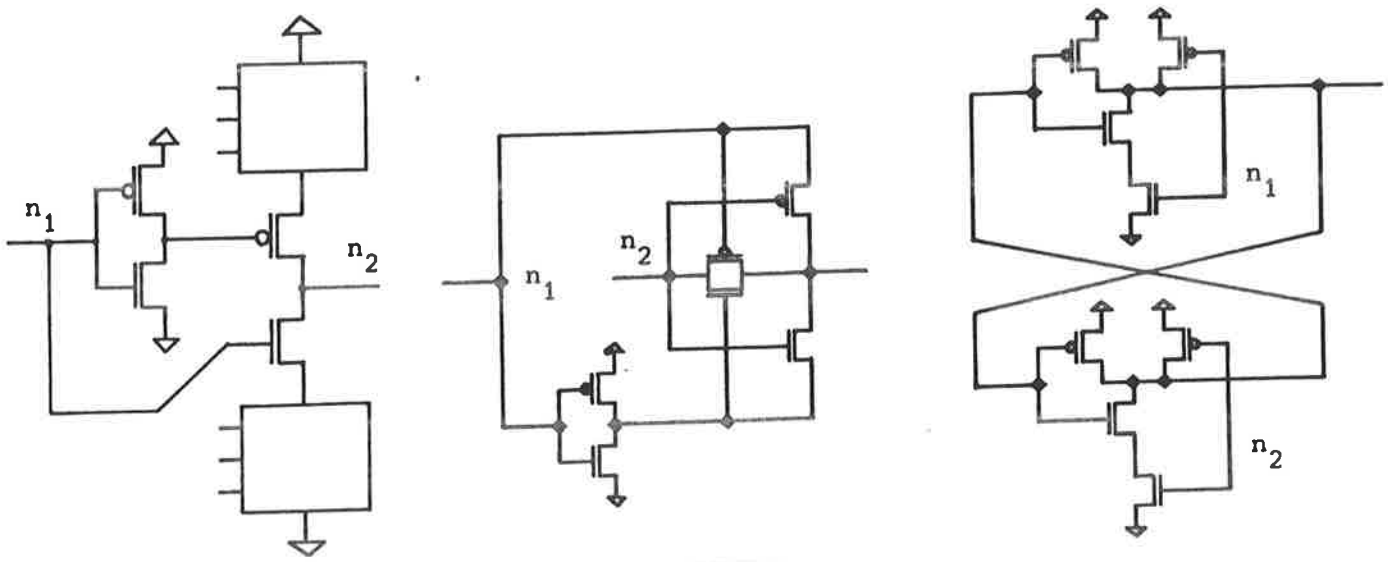
Rule (i) illustrates the use of the transfer function **tf** for generic building blocks. Rule (ii) illustrates how an exor build up of pass transistors is modelled logically. Rule (iii) describes the function of a RS type flip flop build out of two CMOS nand gates.

4 Timing constraints in synchronous circuits

Timing constraints in synchronous circuits follow from the presence of subcircuits which can have an M state at the output during certain phases of the global clock period. By definition of the M state, the actual logic value at the subcircuit output is determined by the preceding *sequence* of events at the subcircuit inputs. The purpose of the timing constraints is to formulate conditions under which this sequence of events can or will lead to an incorrect logic value for the M state at the subcircuit output. First, notation will be presented to denote clocks and clock waveforms. Next, we introduce the concept of 'settling time' of a node. Finally, the timing constraints are discussed and formulated in terms of the settling times of the nodes in the circuit.

To the clock inputs of a synchronous circuit, periodic waveforms are applied. The following notation will be used:

Clocks: $nc_i \in N2, \quad i = 1 \ldots q$
Logic state of the clocks: $D(v(nc_i)) = xc_i \in \{0, 1\}$
$XC = [xc_1 \quad \ldots \quad xc_q]^T$
Clock waveform: $XC_1 \quad \ldots \quad XC_p$
The time point at which $XC_{i-1} \rightarrow XC_i$ is called $tc_i \quad (tc_1 = 0)$



(i) (select ((and n1 (tf n2 vdd)) (and n1 (tf n2 gnd))) (n1 (not n1)))
(ii) (or (and (not n1) n2) (and (not n2) n1))
(iii) (select ((not n1) (and n1 (not n2))) ((not n1) n2))

Figure 5: Subcircuits with a corresponding logic model specification.

Each time interval (tc_i, tc_{i+1}) during which the clocks are stable at the value XC_i is called a clock phase.

For an arbitrary node **n** in the circuit, the settling times during a certain clock phase are defined as the latest possible time points at which a signal transition can take place after the beginning of the clock phase, under the assumption that the states of the clock would be kept at the value of this phase. The following notation is used:

U(n): latest possible time point at which node n changes from 0 to 1
D(n): latest possible time point at which node n changes from 1 to 0

The settling times of internal nodes in the circuit are determined by the settling times of the primary inputs. The timing constraints will be expressed in terms of these settling times. In the following section, the algorithm to calculate U(n) and D(n) for each node in the circuit and for each clock phase will be discussed.

A timing fault in a circuit occurs when the M state at a certain node and during a certain clock phase does not store the intended logic value. This condition imposes constraints on the settling times at certain nodes in the circuit.

In order to formulate the constraints, it must be observed that for each subcircuit, during a certain clock phase XC_i, a number of inputs $\hat{n}$ have no determined logic state, while the others are determined by the states of the clocks. If all possible combination of states $\hat{X}$ are applied at the inputs $\hat{n}$, the set Θ_i of potential states (0,1,E or M) at the output of the subcircuit during the i-th clock phase is obtained:

$$\exists\ \hat{X}: \quad F(X) = 0 \Leftrightarrow 0 \in \Theta_i$$
$$\exists\ \hat{X}: \quad F(X) = 1 \Leftrightarrow 1 \in \Theta_i$$
$$\exists\ \hat{X}: \quad F(X) = M \Leftrightarrow M \in \Theta_i$$

The following timing constraints can then be formulated, which must hold for each subcircuit and each clock phase:

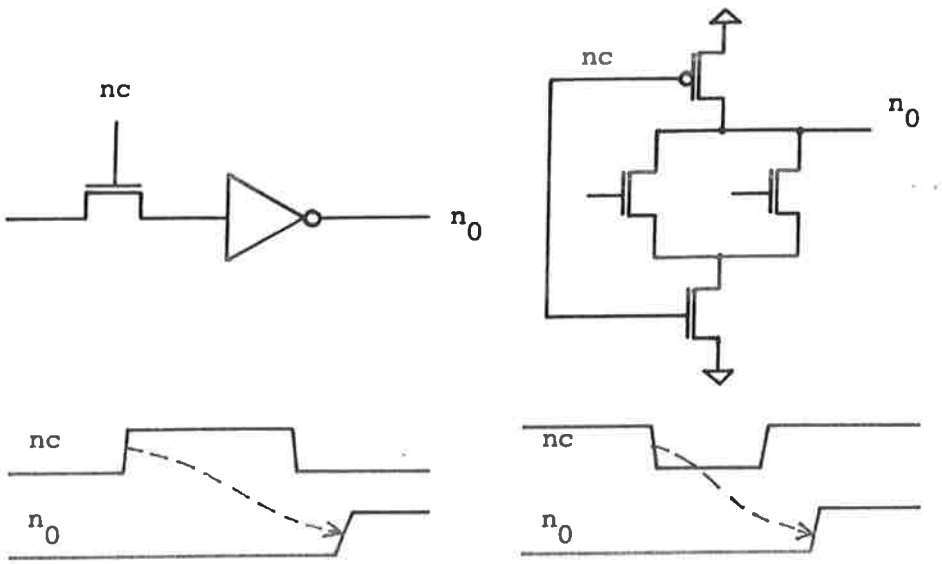


Figure 6: Timing faults covered by the first timing constraint.

Timing constraint 1.

If

$$M \in \Theta_i \quad \wedge \quad \Theta_{i-1} \setminus \Theta_i \neq \{\}$$

Then

$$1 \in \Theta_{i-1} \setminus \Theta_i \quad \Rightarrow \quad U(n_0) < tc_i$$
$$0 \in \Theta_{i-1} \setminus \Theta_i \quad \Rightarrow \quad D(n_0) < tc_i$$

If a determined boolean state (1 or 0) can exist during a certain clock phase but not during the next clock phase, it must have settled before the start of the second phase. This is based on the consideration that this state may have to be be stored by the M state during the second phase. Figure 6 shows two examples of timing faults which can occur when the constraints are not satisfied. In the first case (6a), the output of the latch has not reached the intended state before the latch 'closes'. In the second case (6b), the output of the dynamic gate is not yet precharged before the end of the precharging phase of the clock period.

Timing constraint 2.

If

$$M \in \Theta_i \quad \wedge \quad \Theta_i \setminus \{M\} \neq \{\}$$

Then

$$\forall\ \hat{X}_1, \hat{X}_2 \text{ during } XC_i$$

$$\begin{cases} F(X_1) \neq M \\ F(X_2) = M \\ X_2 = X_1^{\bar{t}_1} \\ x_{1,t} = 0 \end{cases} \Rightarrow \quad U(n_t) < tc_i$$

$$\begin{cases} F(X_1) \neq M \\ F(X_2) = M \\ X_2 = X_1^{\bar{t}} \\ x_{1,t} = 1 \end{cases} \Rightarrow \quad D(n_t) < tc_i$$

If during a certain clock phase both an M state and a determined boolean state are possible at the output, no input transition must take place at the subcircuit during this clock cycle that changes the output from the determined boolean state to the M state. This is based on the consideration that the purpose of the M state can be to store the value of the previous clock cycle. Figure 7 shows timing faults which are excluded by the second timing constraint. Signal nc is directly influenced by the primary clocks. n_1 is an input signal to the subnetwork. In 7a, there is a time interval during which an unintended state can 'slip' through the latch, because the condition signal A is not stable in time. In the case of 7b, the output of the precharged gate may discharge because the inputs have not all stabilized at the end of the precharging phase of the clock period.

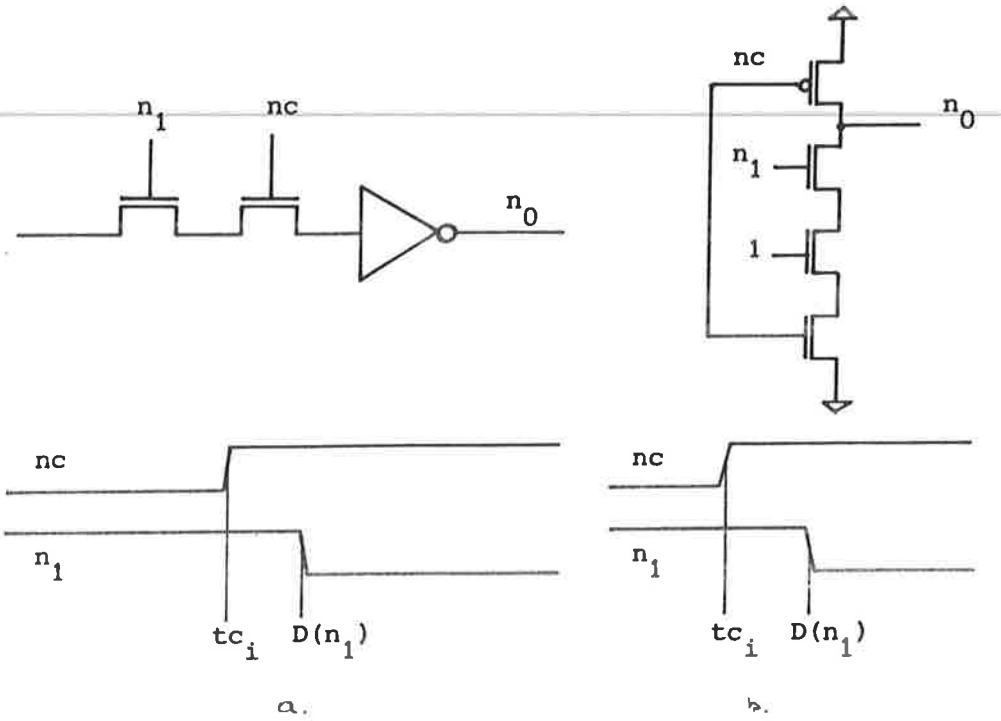


Figure 7: Timing faults covered by the second timing constraint.

5 The timing model

In this and the following two sections, we will discuss the timing analysis algorithm. The purpose of this algorithm is to calculate the settling times U(n) and D(n) for every node n in the circuit during each clock phase, which enables to verify the timing constraints. For the purpose of timing analysis, all known timing verifiers implicitly[2] or explicitly [3] use a signal flow model that takes the form of a directed graph, referred to as 'dependency graph' or 'causality graph'. An edge in this graph signifies that a signal transition at its tail is followed by a signal transition at its

[2]E.g. in [14] where the model is dynamically extracted from the circuit during the critical path analysis
[3]E.g. in [10]

head after the time delay associated with this edge. The timing model we present here is more detailed in the sense that the signal propagation is not considered to be *unconditional*, but the logic conditions for each edge are known and stored with the edge. These conditions can be used to eliminate false paths from the timing analysis. This is the topic of the last section. The actual algorithm that we use for timing analysis has first been presented in [10]. It will be discussed in more detail in the next sections.

To derive the timing model for a circuit, consider for each subcircuit the set of sequences $X_1 \ldots X_L$ of input vectors $X_i \in \{0,1\}^N$ which cause the state of the output y to change:

$$y_{L-1} \neq y_L \tag{1}$$

Each vector must be different from its predecessor in exactly one entry: $X_i = {X_{i-1}}^{\bar{t}}$, where the notation $X^{\bar{t}}$ is used to denote a vector of boolean values which is different in one entry from the vector X :

$$X = [x_1 \ldots x_N]^T \text{ with } x_i \in \{0,1\}$$
$$X^{\bar{t}} = [x_1 \ldots x_{t-1} \quad -x_t \quad x_{t+1} \ldots x_N]^T$$

It must be noted that the expression F(X) for the subcircuit does not contain sufficient information to automatically determine all sequences that satisfy (1). The reason is that when F(X)=M, the logic value y of the output is undetermined, by definition of the M state. Apparently, the model we use is inadequate and the problem can only be solved by using a considerably more complex finite state model for a subcircuit, instead of a simple equation. However, we observe that it is usually possible to choose a subcircuit partitioning in such a way that the following holds:

Subcircuit property

For every sequence $X_1 \ldots X_L$

with $X_i \in \{0,1\}^N$ and $\forall i \ \exists t : X_{i+1} = {X_i}^{\bar{t}}$

holds: $F(X_i) = M \Rightarrow y_i = y_{i-1}$

This property of the sequential behaviour of the subcircuit states that when a single transition at one input causes the output to become bistable, the logic value at the output remains unchanged. As a corrollary of this property, every sequence $X_1 \ldots X_L$ with

$$\begin{aligned} &F(X_1), F(X_L) \in \{0,1\} \\ &F(X_1) \neq F(X_L) \\ &F(X_2) \ldots F(X_{L-1}) = M \\ &\forall i \ \exists t : X_{i+1} = {X_i}^{\bar{t}} \end{aligned} \tag{2}$$

propagates a signal transition. The input n_t makes a transition from $x_{t,L-1} \rightarrow x_{t,L}$. The output makes a transition $y_{L-1} \rightarrow y_L \quad = \quad F(X_1) \rightarrow F(X_L)$. To build the timing model for the subcircuit, consider all possible sequences $X_1 \ldots X_L$ that satisfy (2). Each of those sequences is represented as an edge in a directed graph[4] $\mathcal{G}(\mathcal{V}, \mathcal{E})$, where the vertices are the inputs and outputs of subcircuits and every pair X_{L-1}, X_L (with $X_L = {X_{L-1}}^{\bar{t}}$) corresponds to an edge E in $\mathcal{E}$ from n_t (the subcircuit input that makes a transition) to n_0 (the subcircuit output). The other entries in X_L form the set of propagation conditions $(n_i, x_{i,L}) \quad i = 1 \ldots N, \ i \neq t$ associated to this edge. This set is denoted by $\Lambda(E)$.

The graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a model for the signal flow in the circuit. In order to obtain the timing model, a delay value is to be associated to each edge. In the approach as used in the SLOCOP program, the propagation delay of the input transition to the output transition is calculated by

[4]This graph is in fact another form of the 'causality graph' of [10]

x_1	1	0	0	1
x_2	1	1	1	1
x_3	1	1	0	0
x_4	0	0	0	0
x_5	don't care			
x_6	don't care			
F(X)	0	M	M	1
y	0	0	0	1

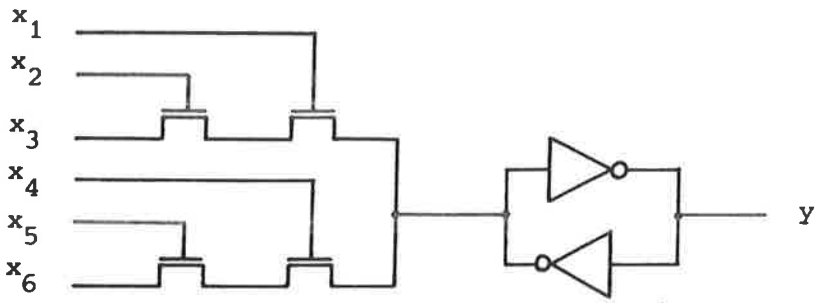


Figure 8: Don't care inputs for signal propagation.

x_1	1	0	0	1
x_2	0	0	1	1
x_3	0	0	0	0
x_4	0	0	0	0
F(X)	1	M	M	0
y	1	1	1	0

x_1	0	0	1
x_2	1	1	1
x_3	1	0	0
x_4	0	0	0
F(X)	1	M	0
y	1	1	0

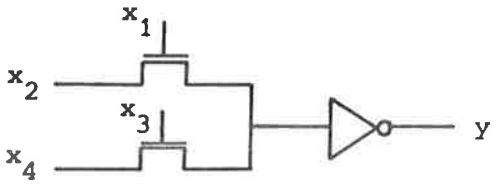


Figure 9: Two equivalent initialisation sequences.

an analog simulation of the subcircuit to which the input excitation $X_1 \ldots X_L$ is applied. In order to reduce the number of simulations to be done and the number of edges in the graph, three observations are made:

1. Don't cares. If the logic value x_k at a particular subcircuit input n_k is not changed throughout the sequence $X_1 \ldots X_L$, and the sequence still satisfies (2) with n_k set to $-x_k$, only one edge E is added to $\mathcal{G}(\mathcal{V}, \mathcal{E})$, without a state for n_k in $\Lambda(E)$. It is assumed that the state of n_k has no influence on the delay of the edge, so only one simulation is to be done. Examples of don't care inputs are shown in figure 8. The notion of don't care input (which can assume both 0 and 1 in all vectors of $X_1 \ldots X_L$ without changing the response of the output) is generalised to sets of inputs that can assume any combination of states. For some subcircuits this can lead to collapsing large number of edges.

2. Equivalent initialisation. The first L-1 vectors in the sequence initialise the output. It is assumed that every initialisation sequence which leads to the same vector X_{L-1} is equivalent and will give rise to the same edge in the timing model (i.e. the propagation delay is not influenced by the history preceding the first state y_{L-1}). An example of equivalent initialisation is shown in figure 9.

3. Length of the initialisation sequence. It is assumed that the output of each subcircuit can be initialised with a sequence of at most three vectors. This means that only sequences of length $L \leq 4$ are to be considered, which makes is possible to exhaustively evaluate all possible sequences $F(X_1) \ldots F(X_L)$ to verify whether they satisfy (2) or not. As shown in figure 10, for a subcircuit with a basic latch function, a sequence of length 3 is needed to initialise the output: no shorter sequence is possible. This imposes a minimum length of L=4. Informally, this assumption can be refrased as the requirement that the subcircuit has no internal states; a subcircuit with internal states (e.g. a shift register) could require much longer initialisation sequences. If the only memory function of the subcircuit is at its output, sequences of limited length are sufficient.

This assumption has consequences on the granularity that is chosen for the sequential behavior of the subnetworks in the rule base definition.

x_1	1	0	0	1
x_2	0	0	1	1
F(X)	1	M	M	0
y	1	1	1	0

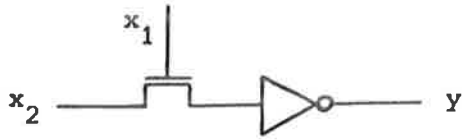


Figure 10: Initialisation sequence to get a 1→0 transition at the output of a latch.

6 Timing analysis of combinational circuits

The goal of timing analysis is to calculate for each node n in the timing model the settling times for a logic high and low signal, respectively denoted with

U(n): latest timepoint at which node n changes 0→1
D(n): latest timepoint at which node n changes 1→0

In a combinational circuit, we assume that for each node n with $|\alpha_{out}(n)| = 0$, the values U(n) and D(n) are given. These values represent the settling times of the inputs of the circuit.

The values of U(n) and D(n) for all other nodes can then be obtained by evaluating all subcircuits in topological order (i.e. a subcircuit is not handled before all the subcircuits that drive it) as follows:

$$U(n_0) = \text{MAX} \begin{pmatrix} U(n_i) + \text{ weight of edge } n_i \rightarrow n_0 & \text{with } T_i = 0{\rightarrow}1 \text{ and } T_o = 0{\rightarrow}1 \\ D(n_i) + \text{ weight of edge } n_i \rightarrow n_0 & \text{with } T_i = 1{\rightarrow}0 \text{ and } T_o = 0{\rightarrow}1 \end{pmatrix}$$

$$D(n_0) = \text{MAX} \begin{pmatrix} U(n_i) + \text{ weight of edge } n_i \rightarrow n_0 & \text{with } T_i = 0{\rightarrow}1 \text{ and } T_o = 1{\rightarrow}0 \\ D(n_i) + \text{ weight of edge } n_i \rightarrow n_0 & \text{with } T_i = 1{\rightarrow}0 \text{ and } T_o = 1{\rightarrow}0 \end{pmatrix}$$

where the maximum is taken over all edges $n_i \rightarrow n_0$ and T_i, T_o denote the input and output transition of the edge. This algorithm is known as the PERT algorithm [3] and used in several existing timing verifiers.

7 Timing analysis of synchronous circuits

The analysis of synchronous circuits is different from the analysis of combinational circuits because the timing model is no longer an acyclic graph. Therefore, the PERT algorithm can not be used. However, during each clock phase the propagation conditions of a number of edges in the graph are in logic contradiction (see Section 8) with the states of the clocks during this phase. For a synchronous circuit, it can safely be assumed that elimination of these edges from the graph yields an acyclic graph during each clock phase. This is in the assumption that small cycles can be taken up in subnetworks that are described in the rule base. Under this condition, it is possible to repeat the PERT evaluation described in the previous section for each clock phase. Each evaluation gives the worst case settling times U(n) and D(n) for every node n in the circuit, in response to a rising or falling edge of the clock waveforms. The algorithm is summarized below:

REPEAT
  FOR every phase in the clock period **DO**
    Eliminate edges from the SPG with incompatible conditions;
    Determine U(n) and D(n) by the sensitizable critical path
    algorithm on the remaining graph [1];
    Verify timing constraints

ENDFOR;
UNTIL convergence: U and D are incremented with exactly one clock period for all nodes.

The advantage of this algorithm lies in the absence of any assumptions about the synchronising actions of the clocks. Signals are not assumed to be stable at latch inputs or outputs at certain time points of the clock period [10]. The algorithm correctly handles signals that propagate through latches during the time interval that these are in 'sample' mode. As an illustration, figure 11 shows a 2-phase clocked circuit and a simplified timing model. The table below gives the settling times as computed by the algorithm.

Clock transition		A	B	C	D
0	nc_1 $0 \rightarrow 1$	12	-	18	-
20	nc_2 $0 \rightarrow 1$	12	38	18	44
45	nc_1 $0 \rightarrow 1$	60	38	63	44
65	nc_2 $0 \rightarrow 1$	60	83	63	89
90	nc_1 $0 \rightarrow 1$	105	83	108	89

Table 1. Execution trace of the timing analysis algorithm for the model of figure 11.

The falling clock transitions do not appear in the table as they cause no signal propagation and therefore do not change the settling times (they could however have an influence if precharged gates were used). It can be seen that after a number of iterations, the settling times are merely incremented by one clock cycle and no more iterations need be done.

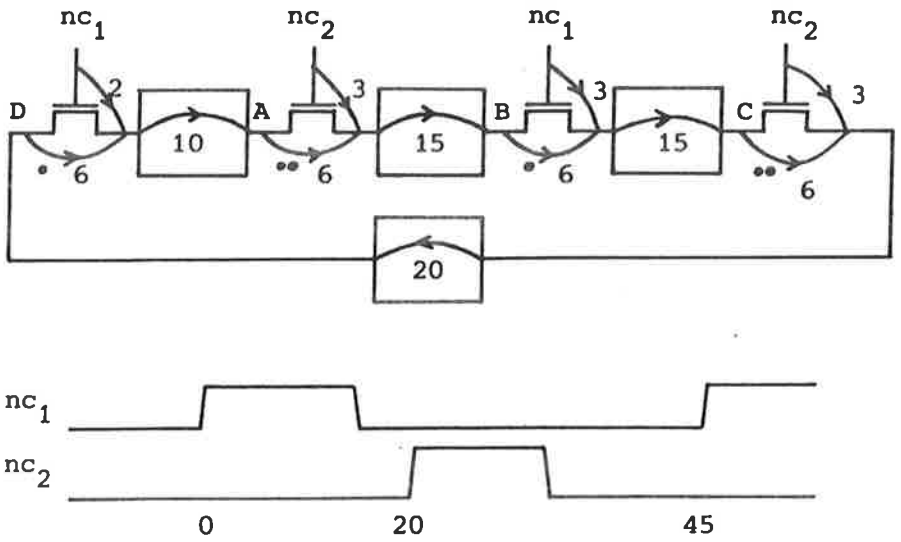


Figure 11: Simplified timing model and clock waveforms. The edges marked with • are eliminated during $nc_1 = 0$; those with •• during $nc_2 = 0$.

8 Compatibility of logic states

Both the timing model and the timing constraints are derived from the logic model of the subcircuits. These subcircuits are defined for a specific design style in the LEXTOC rule base description. It will be shown in this section that another benefit follows from the availability of

this logic model in the timing verifier. Consider the following situation: A logic state (0 or 1) can be imposed on an arbitrary number of nodes in the network. If the states are arbitrarily chosen, it is possible that they can never occur simultaneously in the circuit, because of logic incompatibility (e.g. a 0 state may be assigned to both the input and output of an invertor). We will denote a set of (node, logic state) pairs with the symbol Λ. It is then possible to define a boolean function $\varepsilon(\Lambda)$, which determines the compatibility of the set Λ:

$\varepsilon(\Lambda) = 1 \Leftrightarrow \Lambda$ is not contradictory
$\varepsilon(\Lambda) = 0 \Leftrightarrow \Lambda$ is contradictory

An implementation of the function ε is given below, based on the D-algorithm which has been adapted to the set of operators used in the logic model of the subcircuits. Essentially, this algorithm for propagation of logic states in a network of logic operators is similar to event-driven logic simulation at gate level, except for the fact that signals can also propagate from the output of an operator to the input. Below, the evaluation procedure of the operators is given.

- $y = -x$
  1. IF x is known, set y to $\bar{x}$
  2. IF y is known, set x to $\bar{y}$
- $y = x_1 \wedge x_2 \ldots \wedge x_N$
  1. IF $x_i = 1$ for all i, set y to 1
  2. IF y=1 , set x_i to 1 for all i
  3. IF $x_i = 0$ for any i, set y to 0
  4. IF x_j is unknown and,
     $x_i = 1$ for all i$\neq$j and,
     $y = 0$, set x_j to 0
- $y = x_1 \vee x_2 \ldots \vee x_N$
  1. IF $x_i = 0$ for all i, set y to 0
  2. IF y=0 , set x_i to 0 for all i
  3. IF $x_i = 1$ for any i, set y to 1
  4. IF x_j is unknown and,
     $x_i = 0$ for all i$\neq$j and,
     $y = 1$, set x_j to 1
- $y = (s_1, s_2 \ldots, s_N)|(d_1, d_2 \ldots, y_N)$
  IF $s_i = 1$ for any i,
  1. set s_j to 0 for all $j \neq i$
  2. IF d_i is known, set y to d_i
  3. IF y is known, set d_i to y

The procedure determines all necessary implications of logic states at inputs and output of each operator. Logic contradiction is detected when both a 1 and a 0 state are implied at the same node. This causes the function ε to return 0. Note that only the 0 and 1 state are to be considered, as the M state does never cause any implications and an implication of an E state is taken to be equivalent to contradiction.

We will demonstrate the use of the function $\varepsilon(\Lambda)$ with three applications in the timing verifier:

Elimination of edges during each clock phase In the timing analysis algorithm, a number of edges can be eliminated from the timing model during each phase of the clock period. Each edge E in the graph has a number of propagation conditions associated to it, denoted with Λ(E). Whether an edge is to be eliminated or not can be established by evaluation of the function ε for the propagation conditions of the edge, together with the

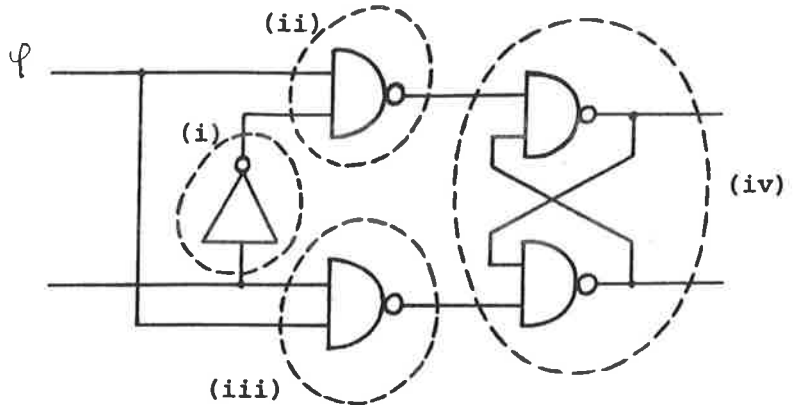


Figure 12: Partitioning of a D-flipflop, giving rise to false timing constraints.

states of the clocks during that particular clock phase:

$$\begin{cases} \Lambda(E) \\ (nc_i, xc_i) \quad i = 1..q \end{cases}$$

This is in contrast to the method used in LEADOUT [10] where simulation is used to determine the relevant transitions under each clock phase.

Elimination of false timing constraints In Section 4, it is explained how timing constraints are derived from the logic model of the subcircuits. In practice this requires the evaluation of the subcircuit expression for all possible input vectors X of each subcircuit during each phase of the clock period. If applied literally, the two rules give rise to many false timing constraints. These false constraints can be eliminated by verifying the logic consistency of each tested input vector by evaluation of the function ε.

Figure 12 shows a first example. It is possible that, according to the rule base for subcircuit partitioning, the D-flipflop is split into four subcircuits as indicated with dashed lines (it is not possible to further partition the cross-coupled NAND gates as this would give rise to unwanted cycles in the timing model). When the clock φ is 0, both inputs of the cross-coupled nands are 0 and the set Θ of possible output states is {M}. When φ is 1, and with the subcircuit (iv) considered separately, the set Θ that will be obtained is {0,1,M}. As a consequence, the second timing constraint applies. However, if the input vectors to the subcircuit (iv) which are not logically consistent with the environment are rejected, the possibility of the M state is ruled out because the inputs to subcircuit (iv) are always complementary. Therefore, the timing constraint is detected to be false.

A second example is given in figure 13. During each phase of the clock period, M is a potential output state of the multiplexor, which gives rise to a number of timing constraints. However, the invertor excludes the M state and hence the timing constraints. Again this can be detected by application of the function ε to all tested input vectors of subcircuit (ii), to verify whether they are indeed possible.

Elimination of false paths The remaining subgraphs of the causality graph in each clock phase are assumed to be acyclic, as all cycles are assumed to be detected by the rule base. In each such subgraph, the largest sensitizable paths have to be found. However due to the

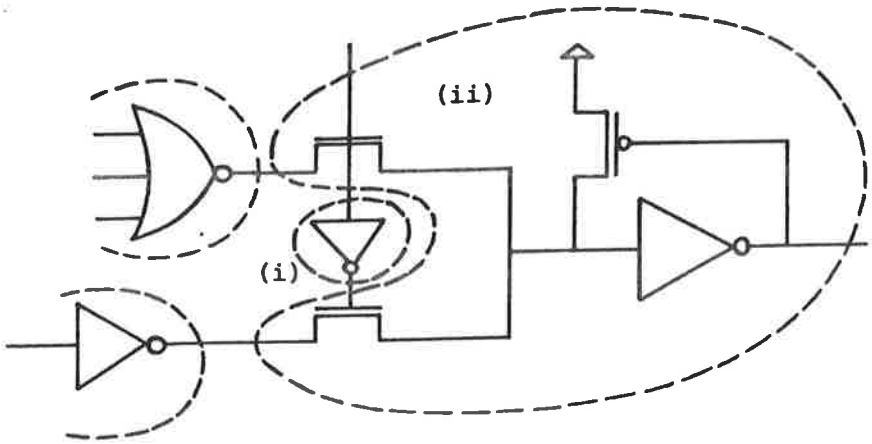


Figure 13: Partitioning of a multiplexor circuit.

fact that timing verification is signal value independent it can occur that paths found by a PERT analysis can not logically be excited and that the path found is false. In order to avoid this, the logical compatibility for the edges in a path have to be taken into account. In [1] efficient algorithms which effectively solve these problems are described.

9 Conclusions

An alternative view on timing faults in synchronous MOS circuits, and the way these faults can be verified has been presented. A rule-based method is presented which solves the signal flow modeling problem in timing verifiers in a flexible way. The rule base also contains information to build a logic model for the circuit. To our knowledge, all existing timing verifiers ignore the logic behaviour of the circuitry to be analysed. We showed that the availability of a logic model allows to automatically derive a detailed timing model and the timing constraints which are to be verified. Additional advantages of the logic modeling lie in the possibility to rule out false timing errors and exclude false paths from the analysis. The rule based method for timing verification as used in SLOCOP [16] has been combined with the algorithms for timing verification of multi-phase clocked systems where signal delays can propagate over level sensitive latches as published by T.Szymanski [10]. Because of the rule-based subcircuit partitioning, the SLOCOP timing verifier has been used for a wide range of static and dynamic nMOS and CMOS circuit designs. The rule based approach for timing verification is operational in the SLOCOP [16] timing verifier. Current work concentrates on implementing the concepts presented in this paper.

References

[1] J. Benkoski, E.Vanden Meersch, L.Claesen, H.De Man, *Efficient Algorithms for Solving the False Path Problem in Timing Verification*, Proceedings IEEE ICCAD conference, Santa Clara, November 1987.

[2] H. De Man, I. Bolsens, E. Vanden Meersch, J. Van Cleynenbreughel, *DIALOG, An Expert Debugging System for MOSVLSI Design*, IEEE Transactions on Computer-Aided Design, CAD-4, No.3, June 1985, pp. 303-311.

[3] Shimon Even, *Graph Algorithms*, Computer Science Press, 1979.

[4] Robert B. Hitchcock, *Timing Verification and the Timing Analysis Program* Proceeding of the 19th Design Automation Conference, 1982, pp.594-604.

[5] David E. Wallace, Carlo H. Sequin, *Plug-in Timing Models for an Abstract Timing Verifier*, Proceedings of the 23rd Design Automation Conference, 1986, pp. 683-689.

[6] Seung H. Hwang, Young H. Kim, A.R. Newton, *An accurate delay modeling technique for switch-level timing verification*, 23rd Design Automation Conference, June 29- July 2, 1986, pp. 227-233.

[7] Norman P. Jouppi, *TV: an nMOS Timing Analyzer*, Proceedings of the Third Caltech VLSI Conference, 1983, pp. 72-85.

[8] G. Ditlow, W. Donath, A. Ruehli, *Logic Equations for MOSFET Circuits*, International Symposium on Circuits and Systems, IEEE, 1983, pp. 752-755.

[9] *Diagnosis of automata failures: A calculus and a new method*, IBM J. Res. Develop., Oct. 1966, pp. 278-281.

[10] Thomas G. Szymanski, *LEADOUT: A Static Timing Analyzer of MOS Circuits*, IEEE ICCAD '86, Santa Clara, CA, Nov. 1986, Digest of Technical papers, pp. 130-133.

[11] Mark D. Matson, Lance A. Glasser, *Macromodeling and optimisation of digital MOS VLSI circuits*, IEEE Transactions on Computer-Aided Design, Vol. CAD-5, No. 4, October 1986, pp. 659-678.

[12] D. Etiemble, V. Adeline, Nguyen H. Duyet, J.C. Ballegeer, *Micro-computer Oriented Algorithms for Delay Evaluation of MOS Gates*, Proceedings of the 21st Design Automation Conf. pp. 358-364.

[13] Zhong L. Mo, Michael R. Lightner, *A Two Parameter Delay Model for Switch Level Simulation*, 1984, ?

[14] John K. Ousterhout, *Crystal: a Timing Analyzer for nMOS VLSI Circuits*, Proceedings of the Third Caltech VLSI Conference, 1983, pp. 58-69.

[15] John K. Ousterhout, *Switch-Level Delay Models for Digital MOS VLSI*, Proceedings of the 21st Design Automation Conference, 1984, pp. 542-548.

[16] E.Vanden Meersch, L.Claesen, H.De Man, *SLOCOP: a Timing Verification Tool for Synchronous CMOS Logic*, ESSCIRC '86 Delft, pp.205-207.

[17] R.E.Bryant, *Algorithmic Aspects of Symbolic Switch Network Analysis*, IEEE Trans. Computer-Aided Design of Integrated Circuits, July 1987.

[18] R.E.Bryant, *Boolean Analysis of MOS Circuits*, IEEE Trans. Computer-Aided Design of Integrated Circuits, July 1987.