A formal approach towards electrical verification of synchronous MOS circuits.

Ivo Bolsens, W. De Rammelaere, C. Van Overloop, L. Claesen, H. De Man [†]

IMEC Lab., Leuven, Belgium, Phone: +32-16-281203

Abstract

Careful observation of many circuit designs has shown that a rule base is too weak as a complete representation of the designer's knowledge. This paper describes the development of a formal theory that captures the designer's intuition of electrical correct digital circuits. To be able to produce software tools that assist the designer to ensure electrical performance and good digital operation, we developed a set of axioms and theorems that cover his expertise. The gap between theory and practice has been bridged by moulding the theory into software tools that allow for a flexible definition of good design practice. Rule based programming, symbolic analysis as well as guided simulation, are the key aspects of this expert analysis environment. Rules form the top level component of the verification environment. They are mainly used to speed up the whole system and to increase flexibility. The aim is to obtain a good compromise between execution efficiency and reliable results by a mixture of provable correct rules, guided simulation and fundamental algorithms, based on basic logical principles.

1 Introduction.

Due to the growing complexity of integrated circuits, a gap between high level design expertise and low level electrical

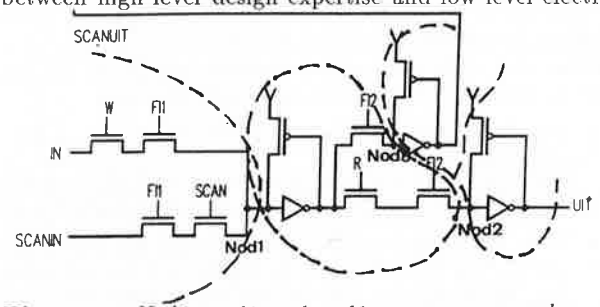

Figure 1. *Nod1, nod2 and nod3 are memory nodes. The CB boundaries are indicated.*

performance has been created. Therefore there is a need of software tools that provide the designer with expertise advice on circuit performance and electrical correctness. This requires a theory that links the intuitions of an expert circuit designer with the corresponding principles of a formal design theory. All concepts should be unambiguous when formulating such a theory. Therefore, in a first section, we define the basic terminology and definitions. Next we discuss the basic axioms and theorems that define good digital operation. This will be followed by the implications for implementing this theory into software tools. The last section contains some concluding remarks.

*This research is sponsored by the ESPRIT1058 project of the EC

[0]Professor at the K.U.Leuven

2 Terminology.

To provide the designer with an understanding into the overall circuit behaviour and to avoid detail, abstraction must be made of the device physics level. We operate on a network model as described in [1]. At any level of abstraction a circuit S(N,E) is described as a set of nodes N and a set of elements E. Each element is connected via its terminals to a set of nodes. Elements, nodes and terminals can have attributes such as capacitance, resistance and relations such as adjacency, output etc...

Because we are concentrating on digital, synchronous systems, the definition of clocks and derived clocks as well as memory nodes is of prime importance. The definitions below do not take into account the delay of signals. These effects are dealt with at the specification of correct transient behaviour. Figure 1 illustrates the main concepts.

Clock signal A clock signal C_i is a low impedant signal, characterized by a periodic waveform $V_{C_i}(t)$ with exactly two transitions during one clock period. A circuit node n_i that generates a clock signal is a clock i.e. $n_i \in Clocks(S(N,E))$.

Clock phase A clock phase ϕ is a set of closed time intervals T_i, characterized by a boolean function F_ϕ such that

$$(\forall t \in T_i : F_\phi(D(V_{C_j}(t)) \dots D(V_{C_k}(t))) = 1) \wedge$$
$$(\forall t \notin T_i : F_\phi(D(V_{C_j}(t)) \dots D(V_{C_k}(t))) = 0)$$

whereby

$V_{n_j}(t)$ represents the waveform of a signal n_j.
$D(v) : R \rightarrow \{0, 1, X\}$ maps voltage v onto one of three states.

Non-overlapping clock phases Two clock phases ϕ_i, ϕ_j do not overlap i.e.

$$\phi_i \cap \phi j = \emptyset \Leftrightarrow \forall t \in [0 \dots \infty] : F_{\phi_i}(t) \wedge F_{\phi_j}(t) = 0$$

Clock period,clocks, clock waveforms and clock phases have to be specified explicitly by the designer.
Clock phases should be specified in such a way that :

$$\forall \phi_i, \exists \phi_{k \neq i} : \phi_i \cap \phi_k = \emptyset$$

Ordering of clock phases We define an ordering of clock phases such that :

$$\forall t_1 \in \phi_{i+1} : \exists t_2 \in \phi_i \text{ such that } 0 \leq t_2 < t_1 .$$

Derived clock signal A node n_i generates a ϕ_{ij}-derived clock signal if

- n_i is an output node of a DC-connected network [2]

ISCAS'88

- and $\forall t$: IF $F_{\phi_i}(t) \wedge F_{\phi_j}(t) = 0 \Rightarrow D(V_{n_i}(t))) = 0$
  or
  $\forall t$: IF $F_{\phi_i}(t) \wedge F_{\phi_j}(t) = 0 \Rightarrow D(V_{n_i}(t))) = 1$.
- and $\forall t$: $Driven(n_i) = 1$. (The predicate Driven(n_i) is 1 if there exists a conducting transistor path to an external input.) [2]
- and $V_{n_i}(t)$ has zero or two transitions during each clock period.

In the context of this paper we concentrate on non-overlapping clock phases. However this does not affect the generality of the theory.

Logic node A logic node is a circuit node for which there exists at least one possible conducting path to VDD that is not passing via other logic nodes and there exists at least one possible conducting path to GND that is not passing via other logic nodes.

Memory node A node n_i is a ϕ_i memory node if

- n_i is the output of a DC-connected network.
- and $\forall t \in [0 \ldots \infty]$: IF $F_{\phi_i}(t) = 1 \Rightarrow$ There exists a *possible* conducting path to a logic node or an external input, that is no refreshing path..
- and there exists a clock phase ϕ_{i+1} that does not overlap with ϕ_i such that
  IF $F_{\phi_{i+1}} = 1 \Rightarrow$ There exists no possible conducting path to a logic node or an external input
  or
  There only exists a possible conducting path that restores the actual value $D(V_{n_i}(t)))$.

Intuitively one can state that a ϕ_i memory node *can* receive new information during phase ϕ_i and *has to* hold this information at least during ϕ_{i+1}.

Memorisation phase The memorisation phase ϕ of a ϕ_i memory node n_i is the maximum set of time intervals T_i characterized by the logic functions F_m such that

- $\forall t \in T_i$: There exists no possible conducting path that can alter the value of n_i
- and $\forall t \in T_i$: $F_m(D(V_j(t)) \ldots D(V_k(t))) = 1$
- and $\forall t \notin T_i$: $F_m(D(V_j(t)) \ldots D(V_k(t))) = 0$

whereby

{j ... k} are called control nodes (i.a. clocks).
F_m states the conditions that must hold for n_i to be in memorisation phase.

Combinational block. A combinational block (CB) is a subnetwork such that

- All terminals of the CB are external inputs or outputs of the circuit, memory nodes or derived clocks.
- and all external inputs, outputs, derived clocks and memory nodes of the CB are terminals of the CB.
- All DCN's of the CB form a connected graph.

Evaluation phase The evaluation phase ϕ of a ϕ_i memory node n_i is the maximum set of time intervals T_i, characterized by the boolean function F_e such that

- $\forall t \in T_i$: IF $n_i \in outputs(CB_i) \Rightarrow D(V(n_i)) = F(D(V_u) \ldots D(V_w$
- and $\forall t \in T_i$: $\exists$ a conducting path to at least one logic node or external input,
  that is no refreshing path.

- and $\forall t \in T_i$: $F_e(D(V_j(t)) \ldots D(V_k(t))) = 1$
- and $\forall t \notin T_i$: $F_e(D(V_j)) \ldots D(V_k(t))) = 0$

whereby

{j ... k} and {u ... w} are inputs of CB_i
{j ... k} are control nodes, {u ... w} are called data nodes.
$\{j \ldots k\} \bigcap \{u \ldots w\} = \emptyset$
F is a boolean function of inputs of CB_i.

3 Definition of correct digital behaviour

Given the above definitions, we are now able to formulate sufficient conditions for MOS circuits to behave correctly i.e. to have a correct *level sensitive* behaviour, independent of transition delays. The requirement of good digital behaviour of a network can be reduced to claiming correct behaviour of combinational blocks. Correct behaviour of a CB can further be divided into correct steady state behaviour, correct memorisation behaviour and correct transient behaviour. As a first axiom we state :

Axiom 1 : Steady state behaviour A combinational block shows a correct steady state behaviour, if every output node, for any logic combination of input signals, during the evaluation phase, can have one and only one steady state value that is recognized as a logic 1 or a logic 0.
This implies among others that during evaluation, a combinational block cannot have internal memory states that influence an output, because the output values may not depend on the sequence of input values.

The second basic axiom we postulate concerns the memorisation behaviour of memory nodes.

Axiom 2 : Memorisation behaviour A ϕ_i memory node n_i shows a correct memorisation behaviour :

- If F_e, characterizing the evaluation phase of n_i is periodic and
  if during all timepoints where $F_{\phi_1} \wedge \ldots \wedge F_{\phi_n} = 0$ and during $F_{\phi_{i+1}} = 1$, no new value can be presented to n_i.
- or if F_e is not periodic and if during all timepoints yields $F_e \oplus F_m = 1$, and during $F_m = 1$ no new value can be presented to n_i and during at least one clock phase which is a subset of the memorisation phase of n_i there is a conducting transistor path to refresh the stored value.

The third axiom concerning transient behaviour makes a distinction between derived clocks and CB memory nodes.

Axiom 3 : Transient behaviour

- A memory node n_i has a correct transient behaviour if, before the end of the evaluation phase, n_i reaches the steady state value depending on the input values, present at the beginning of the evaluation phase and n_i holds this steady state value until the beginning of the memorisation phase.

- A ϕ-derived clock n_i has a correct transient behaviour if as a consequence of a ϕ transition n_i evolves glitch-free to a steady-state value and keeps that value until the next ϕ transition. The transition time of n_i must be small enough to guarantee no races.

4 Software aspects

Above we stated three axioms that describe **sufficient** (not necessary) conditions for correct digital behaviour. Axiom three deals with the timing behaviour of a MOS network. We will limit ourselves to verification of correct steady state and memorisation behaviour. Checking the timing of a circuit is beyond the scope of this paper and is extensively handled in [5] [6].

To allow for an efficient and flexible verification tool, we opted for a mixture of **rule based programming and fundamental algorithms.** The basic flow of the verification process is depicted below :

```
Step 1 : Check user defined clocking information
         for its obedience to the restrictions of
         clock phases and waveforms.
Step 2 : Create a directed graph representation of
         the network by dividing it in DC-connected
         components.
Step 3 : Fire rules to detect logic nodes and create
         their pull-up and pull-down transmission
         functions.[1]
Step 4 : Detect derived clocks.
Step 5 : Detect asynchronous feedback loops.
Step 6 : Detect memory nodes :
         - Fire rules that find most commonly used
           register configurations.[1]
         - Execute algorithm  detect-memory-nodes
           on remaining part of the network
           (see algorithm 1)
Step 7 : Divide network into combinational blocks.
Step 8 : For every combinational block :
       - Fire rules to detect provable correct
         subcircuits.
       - If there still exist subnetworks that are
         not flagged correct : execute
         investigate-dcn-subnetwork
         (see algorithm 2)
```

Algorithms and rules are embedded in the expert system environment DIALOG. The rule mechanism and the nature of the rules are extensively documented in [1]. Algorithm 1 describes the general procedure to detect circuit nodes that obey the above definition of memory node.

Algorithm to detect ϕ_i memory nodes

Let the boundary conditions be :
$F_{\phi_i} = 0 \bigwedge F_{\phi_{i+1}} = 1 \bigwedge F_{\phi_k} = 0 \ (k \neq i+1)$
FOR all DCN_i **DO**
IF DCN_i not processed by a rule **DO**
FOR all $n_i \in outputs(DCN_i)$ **DO**
IF $(F_{pullup} = 0 \bigwedge F_{pulldown} = 0\ ) \bigvee$
$(F_{pullup} = n_i \bigwedge F_{pulldown} = \bar{n_i}) \bigvee$
$(F_{pullup} = n_i \bigwedge F_{pulldown} = 0) \bigvee$
$(F_{pullup} = 0 \bigwedge F_{pulldown} = \bar{n_i})$
is consistent with the boundary conditions **DO**
ASSIGN possible-memory-node to n_i

Let the boundary conditions be :
$F_{\phi_i} = 1 \bigwedge F_{\phi_k} = 0 \ (k \neq i)$
FOR all possible-memory-nodes n_i **DO**
IF $(F_{pullup} \neq 0 \bigwedge F_{pullup} \neq n_i) \bigvee$
$(F_{pulldown} \neq 0 \bigwedge F_{pulldown} \neq \bar{n_i})$
is consistent with the boundary conditions **DO**
ASSIGN memory-node to n_i

Algorithm 1

During a preprocessing step all register configurations which are present in the knowledge base of the given design style will be detected. Therefore algorithm 1 will only evaluate the remaining DCN-outputs. Given that $(IF\ x \Rightarrow y) \equiv (\overline{X}+Y \equiv 1)$, gives rise to a reformulation of algorithm 1 in terms of a tautology checking problem. Figure 2 depicts the general rule statement to recognize a *phase1* memory node in the case of a two phase non overlapping clocking strategy.

```
(Defrule  phase1-memory-nodes ()
 (*If
  ((property static-DCN-output node1)
  (attribute pull-up xf1 nod1)
  (attribute pull-down xf2 nod1)
  (true-tautology
   (or (not phase2) (and (not xf1) (not xf2))
       (and node1 (not xf2)) (and (not node1) (not xf1)))
  (*not (false-tautology
         (and phase1 (or xf1 xf2)))))
  *Then
  ((assign-attribute  memory-node phase1 node1))))
```

Figure 2.

An important part of this rule consists of verifying the consistency of boolean expressions. Hereby we are making use of a powerful tautology checking program that is successfully exploiting rule based techniques to cut down execution time. [3]

Once the memory nodes are detected, we can divide the network into combinational blocks, which will be investigated separately. For every CB , feedback loops are cut. Feedback loops are only allowed to restore a logic value, all others are reported as errors. The directed graph of DC-connected networks is leveled. For all input combinations of the CB that make the evaluation phase condition to hold, the CB is verified by recursively investigating all DCN's on possible charge-sharing or w/l errors that have a sensitive path to a CB output. The kernel of this procedure is depicted in algorithm 2.

The function Investigate(DCN_i) returns a list of output-results of the CB, generated for each possible combination of logic values of the nodes in the list short-circuit $\bigcup$ high-impedant.

The overall verification complexity equals $O(N^{CBinputs})$ whereby N can vary from 1 to 2.
In worst case, if no knowledge about the circuit nor the design style is present, N equals 2. However, experience has shown that in most cases large parts of the network will be processed by design style specific rules. We can reduce the execution time by leveling the DCN's of a CB in such a way that those being verified locally and found correct can be excluded from the list of subnetworks to be investigated. By this means we can diminish the number of input combinations to be examined. In the extreme case when a CB is completely approved by the rules of the knowledge base , N equals 1. This occurs for example when the CB is composed of static CMOS gates or is obeying the NORA-CMOS rules [4].

Algorithm to detect relevant problems in DC-connected networks

Investigate (DCN_i)
IF $i >$ total number of DC-connected networks in CB_i **DO**
 investigate $\Leftarrow$ return CB-output-values
 ELSE DO
  FOR all $k_o \in outputs(DCN_i)$ **DO**
   IF $F_{pullup} = 1 \bigwedge F_{pulldown} = 0$ **DO** $k_o \Leftarrow 1$
   IF $F_{pullup} = 0 \bigwedge F_{pulldown} = 1$ **DO** $k_o \Leftarrow 0$
   IF $F_{pullup} = 0 \bigwedge F_{pulldown} = 0$ **DO**
    push k_o on list high-impedant
   IF $F_{pullup} = 1 \bigwedge F_{pulldown} = 1$ **DO**
    push k_o on list short-circuit
  IF high-impedant $\bigcup$ short-circuit = () **DO**
   investigate $\Leftarrow$ investigate(DCN_{i+1})
  ELSE DO
   FOR all logic combination of nodes in short-circuit $\bigcup$ high-impedant **DO**
    push investigate(DCN_{i+1}) on investigate
    IF $\exists i, j \in investigate : i \neq j$ **DO**
     DCN_i has a sensitive path to CB-output
     FOR all i $\in$ short-circuit $\bigcup$ high-impedant **DO**
      Search subnetwork of DCN that forms a conducting path to i
      Investigate subnetwork for possible errors.
Return Investigate

Algorithm 2

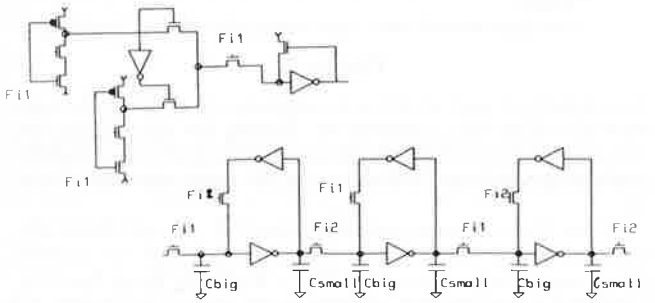

Figure 3. *Error situations, not detected by the program.*

Errors that occur due to transient effects, delays in signal transitions will not be detected by the above verification procedure. Examples of such errors are depicted in Figure 3.

Next we give an overview of the possible error configurations that are detected and reported during the execution of algorithm 2.

Violations against good design practice.

IF node $\in$ list high-impedant **DO**
 IF number of logic nodes in conducting part > 1 **DO**
  $ERROR - MESSAGE$: More than one logic node writes info to a high impedant output.
  ELSE IF number of logic nodes in conducting part $= 0$ **DO**
  $ERROR - MESSAGE$: No information is given to node, internal memory state detected.
  ELSE IF capacitance(logic-node) $>$ 3 x $\sum$ capacitance(conducting-part - logicnode) **DO**
  IF logic-node $\neq$ node **DO**
   $WARNING - MESSAGE$: info passed via charge-sharing from logic-node to output
  ELSE DO
   $ERROR - MESSAGE$: invalid value given to output via charge-sharing.

The nodes that are flagged because there exists a short circuit between VDD and GND are further analysed :

IF nr-logicnodes in conducting part >=1 **DO**
 Simulate conducting part with given input pattern
 IF D(V(node)) = X **DO**
  $ERROR - MESSAGE$: bad W/L ratios.
 IF D(V(node)) = 1 **DO**
  $WARNING - MESSAGE$: possible pseudo nmos or restoring logic.
 IF D(V(node)) = 0 **DO**
  $WARNING - MESSAGE$: possible restoring logic.
 ELSE DO
  $ERROR - MESSAGE$: probably pmos transistor in nmos path or vice versa.

5 Conclusion.

In this paper, we presented a formal view on the analysis of electrical behaviour of synchronous MOS circuits. Rule based techniques are used to derive the **intended behaviour** of the transistor schematics, by applying rules of common sense [7], to **increase efficiency** of the verification procedures by topological rules that recognize known circuit configurations and to **locally approve** specific subcircuits by applying provable correct rules. The combination of rule-based verification methods and fundamental algorithms, founded on a formal theory, allows to generate the relevant error messages. No restrictions are placed on the structure of the network. To our knowledge the mixture of expert system techniques and procedural programming is new in this area of research. The main effort is now concentrating on moulding the above in a usable software tool and extending the knowledge base to add more heuristics to the verification process to increase the execution efficiency.

Acknowledgment

The authors thank F. Van Aelten whose ideas contributed to the proposed verification strategy.

References

[1] H.J De Man, I. Bolsens, E. vanden Meersch and J. van Cleynebreugel : "Dialog : an Expert Debugging System for MOS VLSI Design". IEEE Transactions on CAD of Integrated Circuits and Systems, vol CAD-4, no 3, pp 303, July 1985.

[2] D. Dumlugol, H.J. De Man, P. Stevens, G. Schrooten : "Local Relaxation Algorithms for Event-Driven Simulation of MOS networks Including Assignable Delay Modelling", IEEE Transactions on CAD of Integrated Circuits and Systems, vol CAD-2, no 3, July 1983.

[3] P. Lammens : "The Function TC, a Tutorial", Esprit1058, Technical Report, Part 2, Leuven, December 1986.

[4] N.Goncalves, H. De Man :"NORA : A Racefree Dynamic CMOS Technique for Pipelined Logic Structures." IEEE Journal of Solid-State Circuits, Vol Sc-18, No 3, pp 261-266, June 1983.

[5] J.K. Oosterhout, "A Switch-level Timing Verifier for Digital MOSVLSI", IEEE Trans. on Computer-Aided Design, CAD-4, no 3,pp336-349, July 1985.

[6] E.Vanden Meersch, L. Claesen,H.De Man : "Slocop, A Timing Verification Tool for Synchronous CMOS Logic" Proc. ESSCIRC, pp. 205-207, Delft, The Netherlands, Sept 12-18, 1986.

[7] N.P. Jouppi : "Derivation of Signal Flow Direction in MOS VLSI" IEEE Transactions on CAD, Vol CAD-6, No 3, May 1987.

[8] J.A. Brzozowski, M.Yoeli : "Combinational static CMOS networks", Integration, the VLSI journal 5, pp 103-122, 1987.