Verifying Real-Time Properties of MOS-Transistor Circuits

J. Frößl, Th. Kropf

Institut für Rechnerentwurf und Fehlertoleranz (Prof. D. Schmid)
Universität Karlsruhe, Kaiserstraße 12, 76128 Karlsruhe, Germany
{froessl,kropf}@informatik.uni-karlsruhe.de
http://geothe.ira.uka.de/hvg/

Abstract: *A verification approach which allows the verification of functional and timing behavior of circuits at transistor level is presented. It is aimed at the verification of asynchronous interfaces and standard-cell library modules. In contrast to other approaches, timing is explicitly considered, allowing to verify timing-dependent effects with a high degree of accuracy. To conveniently specify desired properties, a specification language based on Linear Quantized Temporal Logic (QLTL) is provided. For an efficient verification, input constraints, necessary for a proper circuit functioning, are converted into input constraining automata, reducing the reachable state space and providing a model linearisation, necessary to prove linear QLTL formulas by branching CTL model checking.*

1 Introduction

VLSI-technology steps forward to smaller layout structures and higher clock-frequencies resulting in more compact and faster chips. As an undesirable side effect the timing of VLSI circuits gets more and more critical. Parasitic layout elements e.g. capacities and resistances of signal lines gain strong influence on the correct functioning of the circuits. Thus, incorporating real time properties during the validation or verification phase is a must to further guarantee the correctness of critical high speed VLSI implementations.

This paper introduces a methodology to verify the function and the timing of MOS-transistor circuits. We use the transistor level as the basis for our verification approach since this is the highest abstraction level still related to the layout topology and able to uniformly represent layout parasitcs jointly with functional elements. Formal verification avoids expensive analog simulations with huge amounts of stimuli and complicated analysis of their results. Furthermore verification is able to prove the absence of design errors with regard to a given specification and thus is superior to comparable simulation techniques.

Our approach is well suited to verify small and medium sized transistor circuits e.g. standard cells or asynchronous interface modules mostly hand crafted and often tricky in their circuit structure. Nevertheless their correctness directly determines the quality of standard cell libraries and thus the correctness of whole designs.

Considering the trade-off between circuit complexity and model accuracy, our emphasis lies on a fine granularity of the underlying model, simultaneously representing behavior and time. The verification approach is based on a behavioral circuit model, automatically extractable from SPICE transistor netlists, which accurately represents function and timing [1]. To verify a given circuit, first the formal behavioral model is extracted from the transistor-level circuit. Then environmental assumptions, describing the legal input sequences and the desired properties to be verified are stated. To ease this task, we implemented a specification language based on quantized linear time temporal logic [2]. To allow an efficient verification, the input constraints are transformed into *input constraining automata*, restricting the legal input sequences. These automata restrict the reachable state space and linearise the underlying model. Thus, although not possible in general, in our restricted context the linear time specification may be translated into equivalent branching time CTL formulas, allowing efficient CTL model checking [3].

Other approaches verify the functional and the timing behavior separately. Functional verification is performed by extracting a gate level description, which is analysed on a clock-cycle by clock-cycle basis, leading to classical FSM based verification [4]. Timing is considered by verifying that the combinational parts of the circuit operate within the time-bounds given by the clock period. This leads to algorithms, based e.g. on the detection of critical and false paths. Although this separated treatment is sufficient for synchronous gate-level designs, it fails for asynchronous circuits, where the overall operation depends on both, function and timing.

There are only few approaches which have considered the combined verification of function and timing. Leeser [5] has performed a deductive style reasoning in interval temporal logic. However, the approach allows only the semi-automated proof of small example circuits. A slightly higher degree of automation has been achieved using a BDD-based extension of Leeser's formalization [6]. However, they explicitly model the bidirectional signal flow of MOS-transistors, what leads to a complex formal model. In our approach, only a simpler unidirectional signal flow has to be considered due to a circuit analysis preceding the actual verification. Approaches like [7] perform a verification of asynchronous circuits on the gate-level, i.e. whole gates are modelled using an upper and lower bound of the delay, neglecting the dependency of the delay on actual input values. Moreover only certain kinds of input constraints are admissible. Other approaches cope with delay-insensitive circuits and hence do not consider quantitative timing at all.

2 Modelling MOS-Transistor Circuits

This section gives a brief summary of the used circuit model, presented previously in more detail [1]. Circuits at the analog transistor level are usually modelled using real numbers $\mathbb{R}$ for time and signal values. In order to enable discrete domain verification methods we perform *data* and *timing abstraction* before the actual verification. *Structural abstraction* is furthermore used to translate a transistor circuit into a network of *timed transition systems* (TTS), which are structure-free functional units. Using this model we are able to represent the functional and timing behavior of MOS-transistor circuits with a discrete model on a high level of accuracy.

Data abstraction maps the node voltages of the transistor circuit given as real numbers $\mathbb{R}$ onto a three valued logic $\mathbb{T} = \{0, 1, X\}$. Value 0 represents an uncharged node, value 1 represents a circuit node charged to supply voltage level, and X represents a circuit node carrying an undefined (intermediate) or unknown (initial) voltage level [8]. Although we are able to cope with $\mathbb{T}$, we will not use the X value in the following to simplify the presentation. This restricted logic is however sufficient for all examples presented in this paper.

Timing abstraction maps the time scale based on real numbers $\mathbb{R}$ to a discrete time scale using natural numbers $\mathbb{N}$. Natural numbers represent multiples of a minimum time step of length T_{min} that may be set to arbitrary values depending on the desired accuracy of the abstracted model.

Structural abstraction is based on partitioning the transistor circuits into *channel connected subnetworks* (CCS) [9]. A transistor network is partitioned into a network of CCSs by grouping together all transistors, connected via their drain and source connections. As an example figure 2-1. shows an RS-flipflop,

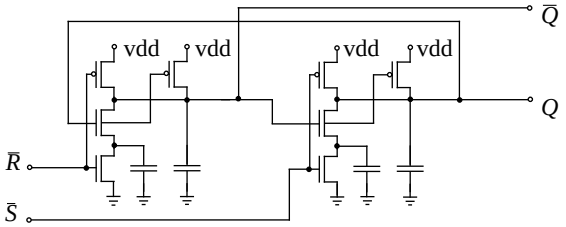

Figure 2-1: CMOS RS-flipflop

built up with two CMOS nand-gates. The CCS-partitioning of the RS-flipflop results in two CCSs representing the two nand-gates, the RS-flipflop consists of (figure 2-2) The signal flow

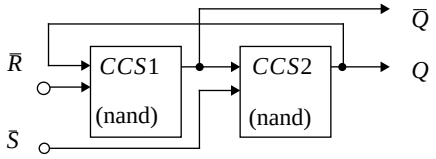

Figure 2-2: CCS-partitioning of the RS-flipflop

between different CCSs is *unidirectional*, inside a CCS *bidirectional*. Additionally to the functional elements, the CCSs include parasitic layout elements, as resistors and capacities of the internal lines. The node capacities of the CCS output nodes are increased by the load of the connected transistor gate capacities.

For performing a structural abstraction, each CCS is treated as an independent unit and mapped onto a behaviorally equivalent TTS. Figure 2-3 shows the transistor circuit of one nand-

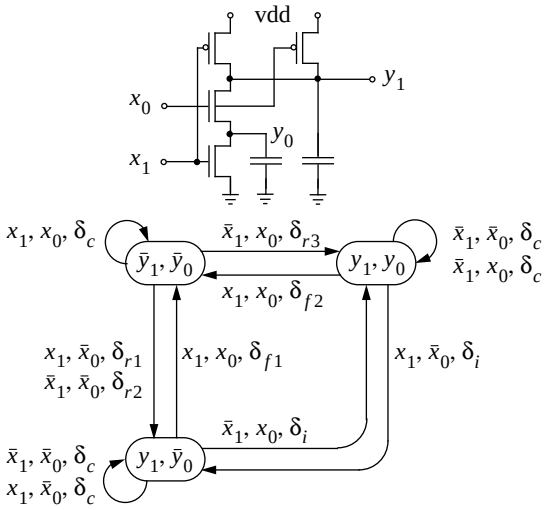

Figure 2-3: Timed transition system of a nand-gate

CCS and its equivalent TTS state-graph. Each internal node of a CCS is interpreted as a TTS state variable, able to take one of the symbolic values 1 and 0. The subset of internal nodes connected to inputs of other CCSs are defined as the output variables of the TTS, all transistor gate inputs of a CCS are the inputs of the TTS. Each transition of a TTS is labelled with its triggering input $\hat{x}_n, \ldots, \hat{x}_0$ with $\hat{x}_i \in \{x_i, \bar{x}_i\}$ and a delay value δ. This delay is equivalent to the maximum time, the internal node-capacities need to change from the charge value corresponding to the actual state, to the charge value corresponding to the next state of the transition. The function, i.e. the different state transitions are extracted using ANAMOS [10], the transition delays are extracted by a detailed RC-analysis.

In the example the two internal nodes (state variables) of the nand-gate allow the encoding of four different symbolic states. One of them ($\bar{y}_1, y_0$) is physically not realizable (figure 2-3). Such *unreachable states* are automatically eliminated during the analysis process and considerably reduce the complexity of the TTSs.

The semantics of the timed transition system is given by the exact conditions to perform a state transition. For each state of the timed transition system there exists at least one (*stable*) input for which the state passes to itself. All other (*unstable*) inputs cause *timed transitions* to next states. The change from a stable to an unstable input valuation first initializes a transition to a next state; the transition itself is performed only when the delay time has entirely passed. During the initialisation phase the actual state remains active. The destination state is stable under the valuation which caused the transition. In *fundamental mode* the environment of the circuit ensures, that input changes can only take place if the timed transition module is in a stable state and no timed transition is actually initialized. *Non fundamental mode* also allows input changes during the initialization phase. Each input change occurring in the initialization phase causes the immediate abort of the initialized transition, i.e. the TTS remains in its actual state. Modelling non funda-

mental mode this way suppresses all inputs which are shorter than the delay of the transition they want to trigger. Although this particular *inertial delay* mode cannot cover all different delay effects of analog MOS-transistor circuits it has proven as a good approximation for a wide range of design styles, validated in many detailed simulations of extracted timed transition models.

3 Circuit specification

In contrast to gate level implementations each transistor level circuit is inherently asynchronous. Such circuits only work properly, if the sequences of values applied to the inputs x_i conform to certain *input constraints* $\varphi(x_0, \ldots, x_n)$. The actual *circuit specification* ψ relates the values at the outputs y_k to the values applied to the inputs x_i, leading to a relation $\psi(x_0, \ldots, x_n, y_0, \ldots, y_m)$. As stated before, ψ may become valid only under the assumption φ, leading to the overall specification of formula 3-1.

$$\varphi(x_0, \ldots, x_n) \rightarrow \psi(x_0, \ldots, x_n, y_0, \ldots, y_m) \tag{3-1}$$

To formulate such specifications we developed a language based on *Linear Temporal Logic* (LTL) introduced subsequently. We choose linear instead of *branching time logic*, which is of course better suited for verification purposes, because the semantics of linear time logics is much easier to understand. Moreover, in branching time temporal logics the temporal operators are not commutative and it is therefore not possible to write down specifications composed of arbitrarily nested predicates as defined in the following [11].

Due to the lack of space we are not able to introduce temporal logics in detail (see e.g. [12]). In the sequel we use the linear temporal operators $\bigcirc$ „next“, $\Box$ „always“ and $\Diamond$ „eventually“ in their commonly used meaning. The semantics of LTL is defined with respect to linear temporal structures $\mathcal{TS}_l$ which are infinite chains of succeeding states s_i (figure 3-1). Each state is labelled with the atomic propositions true in

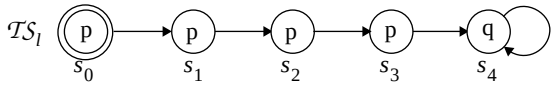

Figure 3-1: Linear temporal structure

that state. A finite part $\sigma = s_i, s_{i+1}, \ldots s_n$ of the infinite chain is called a *trace* of $\mathcal{TS}_l$.

Additionally we will use bounded or quantized temporal operators $\bigcirc^n$, $\Box^n$ and $\Diamond^n$ which facilitate the specification of linear traces with a fixed length n [2]. The quantized operators are only abbreviations of LTL-formulas what is shown by their fixed point equations

$$\bigcirc^n p = p \text{ for } n=0,\ \bigcirc^n p = \bigcirc(\bigcirc^{n-1} p) \text{ for } n \in \mathbb{N} \tag{3-2}$$

$$\Box^n p = p \text{ for } n=0,\ \Box^n p = p \wedge \bigcirc(\Box^{n-1} p) \text{ for } n \in \mathbb{N} \tag{3-3}$$

$$\Diamond^n p = p \text{ for } n=0,\ \Diamond^n p = p \vee \bigcirc(\Diamond^{n-1} p) \text{ for } n \in \mathbb{N} \tag{3-4}$$

We will use the abbreviation QLTL for the linear temporal logic only containing quantized temporal operators. Each QLTL-formula can be reduced to a LTL-formula only containing next-operators.

In our context, the input sequence assumptions mainly define minimum intervals between possible value changes. As shown already in section 2, parasitic elements of transistor level implementations cause inertial delays, and thus a transistor circuit absorbs all input signals shorter than a minimum delay. Behavioral properties to be verified require e.g. that after a specified delay time (or delay interval) the correct output value is available at the outputs after an input change has occurred. Hence many interesting properties may be verified using statements containing *change* and *stable* propositions.

The QLTL *temporal predicates* "stable" $st(x, n, m)$ and "change" $ch(x, n, m)$ are defined as:

$$st(x, n, m) = \bigcirc^m(\Box^n x \vee \Box^n \neg x) \tag{3-5}$$

$$ch(x, n, m) = \bigcirc^m(\Diamond^n(x \wedge \bigcirc\neg x) \vee \Diamond^n(\neg x \wedge \bigcirc x)) \tag{3-6}$$

with $m, n \in \{\mathbb{N} \cup 0\}$. The *stable predicate* is true for the signal[1] x at the time point t_i iff the valuations of signal x are identical at the time points $t_{i+m} \le t_j \le t_{i+m+n}$. The *change predicate* is true for the signal x at time point t_i iff the valuations of signal x at the time points $t_{i+1+m} \le t_j \le t_{i+1+m+n}$ are at least in one time point different from the valuation of x at time point t_{i+m}. The parameter m of the predicates shifts the scope of the formula from the actual time t_i to $t_i + m$. The parameter n of the stable predicate indicates the interval length, i.e. the number of time points the signal has to be stable. For the change predicate the parameter n gives the interval length in which a signal change must occur.

Based on the introduced temporal predicates we may specify the behavior of the nand-gate example as follows[2]:

$$\begin{aligned} &(ch(x_0) \vee ch(x_1) \rightarrow \\ &st(x_0, \Delta_{min}, 1) \wedge st(x_1, \Delta_{min}, 1)) \rightarrow \\ &(ch(x_0) \vee ch(x_1) \rightarrow \bigcirc(\neg(x_0 \wedge x_1) \leftrightarrow \bigcirc^{\delta_{max}} \Box^{\delta_{stab}} y_1)) \end{aligned} \tag{3-7}$$

The specification can be separated into two parts: the *input constraints* φ (first two lines of formula 3-7) and the actual *behavioral specification* ψ of the nand-gate (third line of formula 3-7). The input constraints φ claim that changes of the input signals x_0 and x_1 only occur with a minimum distance of Δ_{min} time points. Thus, an input change must always be followed by a Δ_{min} units lasting phase of stability. The behavioral specification ψ of the nand-gate states that if an input change occurred, at most δ_{max} time points later the nand-relation holds and also holds for $\delta_{stab} = \Delta_{min} - \delta_{max} + \delta_{min}$ further time points. From a designers point of view δ_{max} is the maximum and δ_{min} is the minimum tolerable delay of a desirable implementation. If we make sure that the change rate of the inputs is no greater than the maximum delay of the circuit ($\delta_{max} \le \Delta_{min}$), the given specification holds for the time point right after an input change.

4 Verification

In the preceding sections we showed how transistor circuits can be modelled as networks of TTSs and their behavior can be specified using LTL, extended by quantized operators and tem-

1. In our context, signals are discrete sequences of state variable valuations.
2. Parameters m, n of the predicates equal to 0 are omitted.

poral predicates. The resulting verification task, given by formula 4-1, is to check whether the abstracted MOS-transistor circuit model $\mathcal{M}$ implies the given specification or not.

$$\mathcal{M} \models \quad \Box [\varphi(x_0, \ldots, x_n) \rightarrow \psi(x_0, \ldots, x_n, y_0, \ldots, y_m)] \qquad (4\text{-}1)$$

Since transistor-level circuits do not have a reset state, formula 4-1 must initially hold for all those states, which are physically possible. Having used QLTL as the underlying formal language, this verification task can be interpreted as a *linear real time model checking problem*, for which no efficient proof algorithms exist. However, efficient model checking methods exist for *branching time computational tree logic* (CTL, [13]). Thus we will show in the following how in our context linear real time model checking can be transformed into a model checking problem for CTL, by:

- *transforming the TTSs of section 2 into a unit delay temporal structure,*
- *representing the input constraints* φ *more efficiently as so-called input constraining automata, and*
- *converting the QLTL specification into CTL formulas.*

In contrast to linear time logics, the semantics of branching time logics [12] are declared with respect to branching temporal structures $\mathcal{TS}_b$ (figure 4-1).

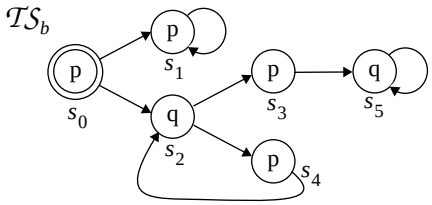

Figure 4-1: Branching temporal structure

Such logics use *path quantifiers* to determine whether a temporal operator should be valid for all (A) or only for some (E) paths, starting from a given state. CTL, as a specific branching time logic, demands a path quantifier preceding each single temporal operator. In CTL the „next", „always" and „eventually" operators are expressed by X, G, and F respectively. The combination of path quantifiers and temporal operators thus allows six basic CTL-operators: EX, AX, EG, AG, EF and AF. Similar to QLTL we can introduce quantized variants of the CTL operators. Formulas 4-2 and 4-3 give the definition by the fixed point equations for the next-operator.

$$\begin{aligned} \mathsf{EX}^n p &= p && \text{for } n = 0 \text{ and} \\ \mathsf{EX}^n p &= \mathsf{EX}(\mathsf{EX}^{n-1} p) && \text{for } n \in \mathbb{N} \end{aligned} \qquad (4\text{-}2)$$

$$\begin{aligned} \mathsf{AX}^n p &= p && \text{for } n = 0 \text{ and} \\ \mathsf{AX}^n p &= \mathsf{AX}(\mathsf{AX}^{n-1} p) && \text{for } n \in \mathbb{N} \end{aligned} \qquad (4\text{-}3)$$

CTL model checking requires the model being described as a (branching) temporal structure with unlabelled edges although each TTS transition is labelled with input variables and a transition delay.

Input variables may be modelled in a temporal structure by adding them as additional state variables to the structure, which would directly result from the states and transitions of a TTS. The unbounded input variables result in an indeterminism of the equivalent temporal structure (figure 4-2).

For modelling the transition delay time, each single timed transition is split up into a chain of succeeding states (figure 4-2). The number of necessary interim states depends on the given minimum delay T_{min}. Each transition of the resulting state chain represents one minimum delay time step. Obviously this transformation may strongly increase the number of necessary states and thus the accuracy of the model, increasing with smaller minimum delays, trades of with its complexity and requires a careful choice of this parameter.

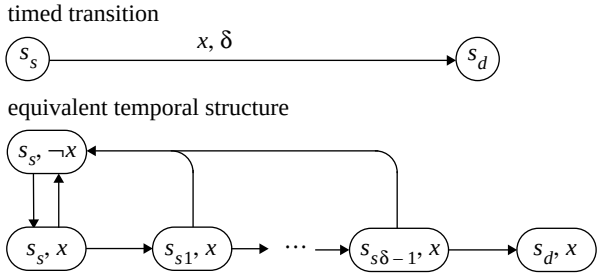

Figure 4-2: Transformation of a timed transition into an equivalent temporal structure

Each model checking problem for LTL can be translated to an equivalent CTL model checking problem with additional fairness constraints [14]. LTL model checking is linear in terms of the model but exponential in terms of the specification [15] whereas CTL model checking is only linear with regard to the model and the specification [16]. Since we may not expect any decrease of the problem complexity by moving from LTL to CTL model checking, this transformation needs to require exponential effort in the worst case and thus may result in very large CTL formulas. A simple example for this is demonstrated with the following QLTL formula:

$$st(x_0, n) \wedge st(x_1, n) \wedge \ldots \wedge st(x_k, n) \qquad (4\text{-}4)$$

It specifies a linear sequence of states, where all of the variables x_i have stable values. Expressing the same using CTL leads to formula 4-5, where each possible stable valuation must be treated separately.

$$\begin{aligned} &((x_0 \wedge \ldots \wedge x_k) \wedge \mathsf{EX}(\ldots \wedge \mathsf{EX}(x_0 \wedge \ldots \wedge x_k)\ldots)) \vee \\ &((\neg x_0 \wedge \ldots \wedge x_k) \wedge \mathsf{EX}(\ldots \wedge \mathsf{EX}(\neg x_0 \wedge \ldots \wedge x_k)\ldots)) \vee \\ &\qquad \ldots \\ &((\neg x_0 \wedge \ldots \wedge \neg x_k) \wedge \mathsf{EX}(\ldots \wedge \mathsf{EX}(\neg x_0 \wedge \ldots \wedge \neg x_k)\ldots)) \end{aligned} \qquad (4\text{-}5)$$

The path binding of the formula, implicitly given for LTL, must be artificially build up for the CTL formula using this recursive formula structure. The resulting CTL formula is however exponential larger than the semantically equivalent LTL formula. Conversely, checking a LTL formula for a branching temporal structure requires the examination of the specification with regard to each single path of the structure, where the number of possible paths may be exponential in the structure size [12].

Another problem for the translation of a LTL specification into an equivalent CTL one is the choice of the path quantifiers. The decision whether to use universal or existential quantified temporal operators in the CTL formula highly depends on the

graph structure of the branching temporal structure we want to verify. We would obtain an easy solution of this problem, if we are able to eliminate the indeterminism in the temporal structure, i.e. if we are able to transform the branching temporal structure into an equivalent linear temporal structure. In this case each LTL operator could be replaced by its universal quantified CTL correspondent and the LTL and CTL model checking problems are equivalent [12].

However, our abstracted circuit model $\mathcal{M}$ is a *branching* temporal structure and thus contains indeterminism, but only as a means to model the input variables. Combining our circuit model with an additional *deterministic input constraining automaton* $\mathcal{M}_{ic}$ as shown in figure 4-3, an overall deterministic

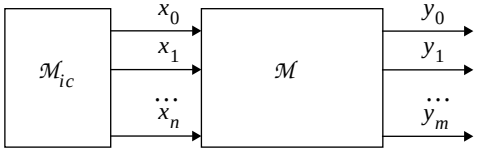

Figure 4-3: Eliminating indeterminism by using an input constraining automaton

model would result. Such an "ideal" input constraining automaton would degenerate the overall temporal structure to a single linear chain of succeeding states.

Unfortunately, using this approach to transform our specifications into equivalent CTL formulas would results in huge input automata, since we would have to encode all possible input combinations as a linear state sequence in $\mathcal{M}_{ic}$. Nevertheless using the input constraints φ to restrict the input sequences is possible. The idea is to cover the input constraints φ by a input constraining automaton $\mathcal{M}_{\varphi}$ according to figure 4-3 and to set up an alternative but equivalent proof goal (formula 4-6).

$$\mathcal{M} \times \mathcal{M}_{\varphi} \models \Box\psi(x_0, \ldots, x_n, y_0, \ldots, y_m) \tag{4-6}$$

The construction of such an input constraining automaton starts with rewriting the QLTL formula that states the input constraints into its *timed disjunctive form* (formula 4-7) with $\hat{x}_i \in \{x_i, \bar{x}_i\}$ and $\kappa_{ij} \in \{true, false\}$.

$$\varphi = \bigvee_{tp_{ij} \in TP} \bigwedge_{j=0}^{depth} \bigcirc^j \underbrace{\bigwedge_{i=0}^{n-1} \kappa_{ij} \hat{x}_i}_{p} \tag{4-7}$$

The constant κ_{ij} determines whether a specific variable is part of the product term p form or not. The formula is a disjunction of *timed product terms* tp_{ij} which are „normal" product terms p in conjunction with quantized next operators $\bigcirc^j$ where j ranges from 0 to the greatest sequential *depth* of φ.

Using a semantic tableau each tp_{ij} is separately translated into a finite linear sequence of states, where the timed product term holds for the first state of the sequence. In order to generate the input constraining automaton the single sequences must be concatenated to a branching temporal structure. The concatenation algorithm must ensure that φ holds for each state of the generated input constraining automata i.e. each state of the automaton must be the start state of at least one of the generated state sequences, representing a single timed product term. This rule ensures, that the given input constraints hold for each state of the input automaton. Additionally, to ensure that the input automaton is able to generate all input sequences allowed by the input constraints, each generated state sequence must be found in the automaton. Figure 4-4 shows the branching tem-

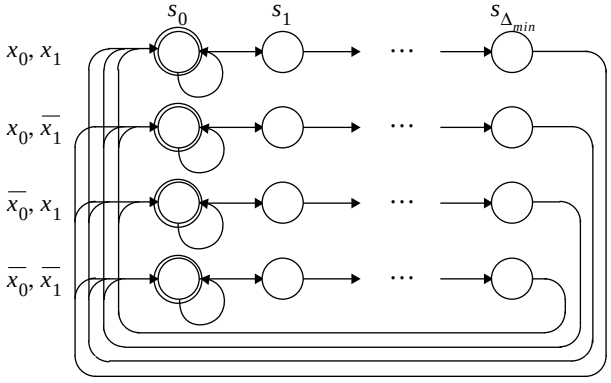

Figure 4-4: Input automaton representing the nand-gate input constraints

poral structure $\mathcal{M}_{\varphi}$ of the resulting input constraining automaton representing the input constraints of the nand gate example given with formula 4-8.

$$ch(x_0) \vee ch(x_1) \rightarrow st(x_0, \Delta_{min}, 1) \wedge st(x_1, \Delta_{min}, 1) \tag{4-8}$$

Due to the restricted set of stability requiring input constraints the temporal structure $\mathcal{M}_{\varphi}$ is piecewise linear, i.e. linear chains of states are interrupted only by few branching segments.

As mentioned above, the translation of the *behavioral specification* part of the overall LTL specification highly depends on the structure of the generated input constraining automaton. In the following we assume that each part of the behavioral specification ψ can always be represented as a QLTL formula with the form given with formula 4-9, where we call α the *anchor* and β *behavior* of the behavioral specification.

$$\alpha(x_0, \ldots, x_n) \rightarrow \beta(x_0, \ldots, x_n, y_0, \ldots, y_m) \tag{4-9}$$

Such a formula can be translated into CTL by replacing each LTL operator of α by its existential quantified CTL counterpart and each LTL operator of the behavior part β by its universal quantified CTL counterpart if we can ensure that the input constraining automaton has the following property:

- *At each state in the input constraining automaton which is the beginning of a path for which α holds, α must also hold for all other paths starting from this state.*

This property makes sure that at each state α holds β might also hold for all paths starting from this state, and thus allows the universal quantification of the behavior part, leading to formula 4-10.

$$\begin{aligned}&(x_0 \wedge \mathsf{EX}\neg x_0) \vee (\neg x_0 \wedge \mathsf{EX} x_0) \vee \\ &(x_1 \wedge \mathsf{EX}\neg x_1) \vee (\neg x_1 \wedge \mathsf{EX} x_1) \rightarrow \\ &\mathsf{AX}\left[\neg(x_0 \wedge x_1) \leftrightarrow \mathsf{AX}^{\delta_{max}}(\mathsf{AG}^{\delta_{stab}} y_0)\right]\end{aligned} \tag{4-10}$$

This formula can be expanded to a simple CTL formula using the fixpoint equations for the quantized CTL operators given above. Now the verification of the generated circuit model with respect to the CTL specification can be performed using a standard CTL model checker.

5 Results

The resulting verification tasks were performed with the CTL model checker SMV, developed by Clarke and McMillan [13]. Table 5-1 gives the results for MOS-transistor standard-cell elements, such as gates and flipflops and oscillators consisting of inverter loops with various numbers of inverters, with size up to 66 transistors. The verification goals comprised a functional and timing characterization in case of the standard-cells and a verification of the oscillator frequency. The circuit models were abstracted from their layout descriptions and are based on a standard cell library. The results table gives the number of transistors, the number of state variables necessary to represent the circuit model as temporal structure, the number of state variables of the input automaton and the number of actually reachable states of the system representing the combination of circuit model and input automaton. This reachable state space is given as an absolute value and in terms of virtual state variables. Furthermore we related the reachable state space to the original state space of the circuit.

The runtimes were measured on a SUN SPARC10 workstation. An interesting result was that the reachable state space representing the combination of input constraining automaton and the actual circuit is often much smaller than the state space of the circuit alone, since the input constraints restrict the number of allowed input sequences and thus restrict the set of reachable states.

1	2	3	4	5	6	7	8
circuit name	#transistors	#state vars $\mathcal{M}$	#state vars $\mathcal{M}_\varphi$	reach. states $\mathcal{M}\times\mathcal{M}_\varphi$	col. 5 relative to $2^{col.3}$	Bdd-size	SMV runtime
inverter	2	5	4	32	100%	133	0.2 s
nand2	4	9	4	423	82%	327	0.5 s
ab∨¬c	8	14	7	6543	39.9%	909	1.6 s
ab∨cd	12	21	9	101717	4.8%	2313	34.3 s
RS-FF	8	14	7	2817	17.2%	1186	7.4 s
D-FF	28	49	7	2^28.2	5.5e-5%	11028	11525 s
oscil.3	6	12	-	48	1.2%	185	0.2 s
oscil.17	34	56	-	2^20.5	2.1e-9%	1864	42.9 s
oscil.33	66	132	-	2^36.5	1.9e-27%	3785	1211 s

Table 5-1: Results for various example circuits

6 Conclusions

In this paper an approach has been presented, which allows the verification of function and real time properties of MOS-transistor circuits. In contrast to most other approaches, verification is performed with a high degree of accuracy on transistor level, such that our approach may be used e.g. for standard-cell library cell characterization. Interesting properties and input restrictions may be conveniently described by a specification language based on QLTL. The circuit behavior and timing is uniformly represented by timed transitions systems. Although for our verification tasks different proof algorithms are possible in general, we showed in this paper, how in our restricted context the original linear real time model checking problem may be translated in a CTL model checking problem. The high model accuracy and the need to represent explicit timing information obviously results in large models to be verified. However, the elimination of unreachable states and a high degree of structural similarities, leading to acceptable BDD sizes in the underlying prover, allowed the verification of circuits sizes, which are sufficient for the intended application like the verification of standard-cells or asynchronous interface modules.

7 References

[1] J. Frößl and Th. Kropf. A New Model to Uniformly Represent the Function and Timing of MOS-Transistor Circuits and its Application to VHDL-Simulation. In *European Design Automation Conference*, pages 343-348, 1994.

[2] A. Pnueli and E. Harel. Applications of Temporal Logic to the Specification of Real Time Systems. In M. Joseph, editor, *Formal Techniques in Real Time and Fault Tolerant Systems*, pages 84–98. Springer, 1988.

[3] J.R. Burch, E.M. Clarke, K.L. McMillan, D.L. Dill, and L.J. Hwang. Symbolic Model Checking: 10^{20} States and Beyond. In *Symposium on Logic in Computer Science*, 1990.

[4] T. Kam and P. A. Subrahmanyam. Comparing Layouts with HDL Models: A formal Verification Technique. In *Intl. Conference on Computer Aided Design*, pages 588–591, 1992.

[5] M.E. Leeser. Reasoning about the Function and Timing of Integrated Circuits with Interval Temporal Logic. *IEEE Transactions on CAD*, 8(12):1233–1246, December 1989.

[6] M. Fujita, Y. Matsunaga, and T. Kakuda. Automatic and Semi-Automatic Verification of Switch-Level Circuits with Temporal Logic and Binary Decision Diagrams. In *Intl. Conference on Computer Aided Design*, pages 38–41, 1990.

[7] S. Devadas, K. Keutzer, S. Malik, and A. Wang. Verification of Asynchronous Interface Circuits with Bounded Wire Delays. In *IEEE International Conference on Computer-Aided Design*, pages 188–195, 1992.

[8] R. E. Bryant. Boolean Analysis of MOS-Circuits. *IEEE Transactions on Computer Aided Design*, CAD-6(4):634–649, July 1987.

[9] R.E. Bryant. A Switch Level Model Simulator for MOS Digital Systems. *IEEE C-33*, pages 160–177, 1984.

[10] Randal E. Bryant, Derek Beatty, Karl Brace, Kyeongsoon Cho, and Thomas Sheffler. Cosmos: A Compiled Simulator for MOS-Circuits. In *Design Automation Conference*, volume 27, pages 9–16, 1987.

[11] L. Lamport. Sometimes is Sometimes "not never"-on the Temporal Logic of Programs. In *Proceedings of the ACM Symposium on Programming Languages*, pages 174–185, 1980.

[12] E. Allan Emerson. Temporal an Modal Logic. In *Handbook of Theoretical Computer Science*, chapter 16, pages 996–1072. Elsevier Science Publishers (Editor: J. van Leeuwen), 1990.

[13] K. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem.* PhD thesis, Carnegie Mellon University, 1992.

[14] E. Clarke, O. Grumberg, and K. Hamaguchi. Another Look at LTL Model Checking. In D. M. Gabbay and H. J. Ohlbach, editors, *Temporal Logic*, number 827 in LNAI. Springer, July 1994.

[15] O. Lichtenstein and A. Pnueli. Checking that Finite State Concurrent Programs Satisfy their Linear Specification. In *Proceedings of the 12th. ACM Symposium on Principles on Programming Languages*, pages 97–107, 1985.

[16] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic Verification of Finite State Concurrent Systems Using Temporal Logic. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.