Formal Verification of C-element Circuits

Chao Yan* Florent Ouchet+ Laurent Fesquet+ Katell Morin-Allory+
*Department of Computer Science +TIMA Laboratory
University of British Columbia, Canada Grenoble INP, UJF, CNRS, France
chaoyan@cs.ubc.ca firstname.lastname@imag.fr

It is well known that the correct behavior of asynchronous circuits is not guaranteed when the inputs switch too slowly. The erroneous behavior is generally difficult to be spotted by simulation based methods. We applied formal methods to study the analog switching behavior of a full-buffer circuit composed of C-elements. We used our reachability analysis tool COHO to compute all reachable states of two C-element designs and verified several analog properties. Based on these properties, we identified a sufficient condition under which the full-buffer circuit always supports the designed handshaking protocol. We also improved the COHO tool to automate the verification process, reduce error and improve performance.

I. INTRODUCTION

Formal verification of asynchronous or analog circuits is a promising research area. As manufacturing technologies progress to ever smaller feature sizes, transistors become less-and-less ideal, and phenomena that were insignificant in an older technology may cause design failures in a more advanced process. As a consequence, analog design errors account for a growing percentage of design re-spins. Furthermore, the widely used simulation based methods have the weakness that it is difficult to simulate all corner cases and find potential bugs. Formal verification is attractive for circuit design as it formally models a circuit and considers all possible behaviors of the model. On the other hand, there are many properties of circuit-level design that make it attractive for formal approaches. Key circuits tend to be small, thus, the problems are much smaller than those for system-level verification. Circuit-level design is the domain of design experts who expect to spend a substantial amount of time on each cell designed. Thus, they can consider working with a verification expert if that interaction leads to a reduction in design time or risk.

It is well studied that delay insensitive (DI) circuits may not work correctly when the inputs switch too slowly [1], [2]. In [3], the authors reported some cases where slow transition time might affect correct behavior of C-elements. Under these circumstances, the full-buffer circuit as shown in Figure 1 does not satisfy the designed handshake protocol. This motivated our work of applying formal methods to verify C-elements and the full-buffer circuit.

This paper presents a reachability analysis based verification method and describes the verification of the full-buffer circuit as shown in Figure 1. In particular:

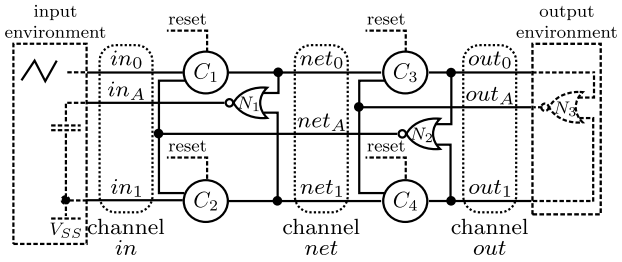


Fig. 1. The full-buffer circuit

- We formally verified two C-element designs: a *weak-feedback C-element* and a *conventional C-element*. We specified circuit properties to be checked based on linear time logic (LTL) and Brockett annuli, and computed all possible circuit states with our reachability analysis tool COHO. By establishing an invariant set of reachable regions, we verified that the analog behaviors of these two C-element designs satisfy their designed specifications.
- We verified the full-buffer circuit by decomposing it into C-elements, inverters and NOR gates. Firstly, we showed that the output signals of these circuit elements (*e.g.,* C-elements) satisfy the same Brockett annulus with the one used to specify their input signals. Based on this property, we found a sufficient condition under which the full-buffer circuit always supports its designed handshaking protocol. This method can be applied to other complex asynchronous circuits.
- We improved COHO by providing a hybrid automata based interface which automates reachability computation for circuit verification and supporting more features to reduce error and improve performance, such as *slicing*, *assume-guarantee strategy*, *etc.* These improvements facilitate the verification.

After surveying related work in Section II, Section III introduces the C-element circuits to be verified and provides specifications of these circuits. This section also presents a potential bug of the full-buffer circuit which was not spotted by simulations in [3]. Section IV describes our reachability analysis algorithms and new features of COHO. Section V presents details of our approach to compute reachable regions of C-elements. In Section VI, we present the properties that have been verified based on reachable regions. We also analyze the performance of COHO in this section. Section VII concludes the paper and presents some future work.

II. Related Work

The possibility of erroneous switching behavior in asynchronous modules as a result of slow input transition was pointed out and analyzed in [1], [2]. C-elements are widely used in asynchronous circuits to hold control states, especially in micro-pipeline [4] based circuits, such as the full-buffer circuit. In [3], the authors showed by simulations that the correctness of the full-buffer circuit depends on the input slopes of C-elements and the handshake protocol might be violated when the slopes are very small. The authors also provided an explanation for the unexpected phenomenon. They defined two threshold voltages of C-elements: the low to high threshold V_{TH} is the minimum voltage of inputs which makes the output switch from low to high, and the high to low threshold V_{TL} is the maximum voltage of inputs which makes the output become low. When V_{TL} is greater than V_{TH} in a C-element implementation, the output of the circuit may oscillate and produce extra data to the full-buffer circuit.

Besides simulations, there are several other methods to analyze asynchronous or analog circuits. In [5], an efficient method was presented to find all equilibrium points of a ring oscillator, and the stability of equilibria was analyzed based on eigenvalues of the Jacobian matrix. Recently, formal methods for verifying asynchronous designs have received intense attention, as these methods apply formal models and consider all possible circuit behaviors automatically. We summarize some of this work here noting that some comprehensive surveys are presented in [6]–[10].

The first circuit-level verification work is by Kurshan and McMillan [11]. They modeled a digital arbiter circuit by differential equations and employed COSPAN to construct the reachable space. Several reachability analysis tools and techniques have been developed in the past ten years. Frehse *et al.* implemented PHAVer [12] which provides more efficient and robust implementations of the HyTech algorithms [13], [14]. The LEMA [15], [16] tool applies *labeled hybrid Petri net (LPTN)* to model circuits and uses a *simulation aided verification* method to generate LPTN models automatically. CheckMate [17], [18] computes convex polyhedral approximations of the reachable regions for systems with non-linear dynamics by using numerical optimization methods to find extremal trajectories. The d/dt tool [19], [20] performs reachability analysis of continuous or hybrid systems modeled by linear differential inclusions of the form of $dx/dt = Ax + Bu$, where u is an external input taking values in a bounded convex polyhedron. d/dt represents the reachable sets as non-convex orthogonal polyhedra [21], i.e. finite unions of full-dimensional hyper-rectangles, and approximates the reachable state using numerical integration and polyhedral approximation. These tools have been used to verify several analog circuits with low-dimensional state spaces including tunnel diode oscillators, voltage controlled oscillators and Sigma-Delta modulators [22]–[25]. However, these reachability based methods tend to suffer from state-space explosion problems. Therefore, these existing examples are restricted to simple circuits and simple properties.

III. C-elements

A. Circuits

Figure 1 shows the asynchronous circuit to be verified. It depicts a full-buffer composed of two cascaded half-buffers. The communication channels, *i.e.*, *in*,*net* and *out* are dual-rail encoded. The communications of data inside the channel are implemented using state-based communication protocols. The data senders and receivers are synchronized using handshake phases: the sender's transition from one to another phase requires receiver's acknowledgment. Figure 2 highlights the sequential steps found in a four-phase handshake protocol. The data is issued by the sender block and the receiver drives the acknowledgment signal as follows:

- during phase 1, the receiver is ready to accept data;
- in phase 2, the sender emits data through the communication channel;
- the receiver acknowledges this data in phase 3;
- the sender removes the data in phase 4.

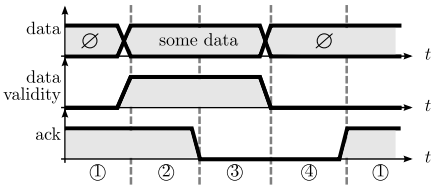
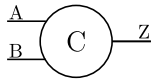


Fig. 2. 4-phase handshake protocol timeline



A	B	Z
0	0	0
0	1	memory
1	0	memory
1	1	1

Fig. 3. C-element

Components $C_1, \cdots, C_4$ are symmetrical 2-inputs C-elements, also called Muller gates. The C-elements are used as state-based memory for synchronization protocol since there is no global clock. Figure 3 denotes the usual gate symbol and its truth table: its output Z switches when two inputs A and B have the same digital value. This gate implements a rendez-vous between sender and receiver wires. Sender and receiver ends are wired to C-element's inputs. When both ends are ready, the C-element output Z switches: the handshake protocol moves to the next phase.

The C-elements are usually designed as library standard cells. We focused our studies on the weak-feedback C-element structure [4] and the conventional C-element structure [26]. Figure 4 and Figure 5 denote their transistor networks. For both circuits, the input stage is built by stacking the transistors T_{P1}, T_{P2}, T_{N2} and T_{N1}. The transistors T_{N3} and T_{P3} make the memory point output stage. Finally, the memory point feedback stage is made of the remaining transistors. The weak-feedback C-element has six nodes: two inputs A and B, one output Z and two internal nodes P and N. The conventional C-element has two more internal nodes P_2 and N_2.

The transistors are sized according to input-to-output delay, output transition time and consumption constraints in order to match manufacturer rated values. This constraint based design may not give a correct C-element behavior under some

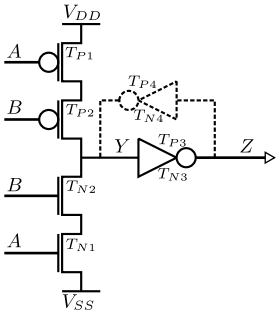


Fig. 4. Weak-feedback C-element

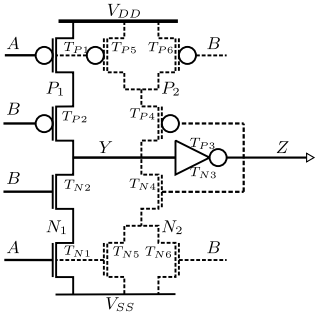


Fig. 5. Conventional C-element

circumstances [3]. Therefore, it is necessary to develop novel techinques to ensure correctness of the C-element and full-buffer circuits.

B. Small Signal Analysis

In [3], simulations show that the communication protocol of the full-buffer circuit loses its synchronization when thresholds of the C-elements are disordered. However, C-elements with correct ordered thresholds cannot guarantee correct behavior of the full-buffer circuit. If one internal node of the full-buffer circuit has a much larger capacitance than others, it not only increases the propagation delay but also stalls the handshaking process. Such large capacitances and delays may happen together when a wire is physically longer than others. Let us simplify the full-buffer circuit by considering only one half-buffer, connecting the channel acknowledge signal to its input and connecting its output to the output environment acknowledge. Figure 6 shows a 3-stage ring oscillator composed of weak-feedback C-elements with thresholds $V_{TL} = 0.69$ and $V_{TH} = 0.94$. When all nodes x_1, x_2, x_3, x_4 have the same capacitance, the circuit oscillates as designed. However, the circuit does not oscillate any more when we add a large capacitance to node x_2. The phenomenon is illustrated in Figure 7 where the x axis is time and the y axis represents voltages of x_1, x_2, x_3, x_4. Similarly, when the capacitance of x_2 is much smaller than other nodes, the circuit fails to oscillate either. The circuit oscillates with a very low frequency if all nodes have large but comparable capacitances.

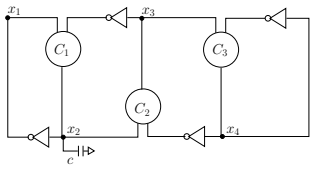


Fig. 6. A 3-state ring oscillator

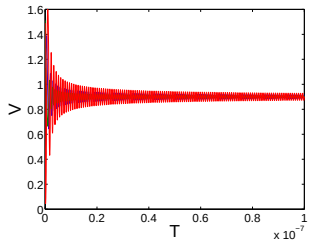


Fig. 7. The circuit does not oscillate

We analyzed the unexpected behavior using a small signal analysis method as presented in [5]. First, We identified all DC equilibrium points by searching the whole state space with TSMC 180nm process models. To determine whether or not a DC equilibrium is stable, we constructed a linear model for circuit behavior in a small neighborhood of the equilibrium:

$$\dot{V} \approx J_{eq}(V - Veq) \tag{1}$$

where $\dot{V}$ is the time derivative of V, V_{eq} is the voltage vector for the equilibrium point, and J_{eq} is the Jacobian matrix.The solution to Equation 1 is

$$V(t_0 + t) = Veq + e^{tJ_{eq}}(V - Veq) \tag{2}$$

If J_{eq} has at least one eigenvalue with positive real parts, then the DC equilibrium is unstable because $e^{tJ_{eq}}$ goes to infinity as t increases. Otherwise, we say the equilibrium is stable. The result shows that real parts of eigenvalues become negative when the capacitance of x_2 reaches an upper or lower bounds, which explains the simulation results.

C. Specifications

As described above, simulation based methods cannot always find all bugs of circuit designs. Therefore, it is necessary to apply formal methods to analyze the C-element and full-buffer circuits. One challenge of formal methods is to formally specify circuit properties. We applied linear time logic (LTL) [27] to specify properties of C-elements. The specification of output Z is shown in Figure 8.B. In English, the "guarantee" clause says that if inputs A and B are both high (low), then the output Z will eventually be high (low). On the other hand, the inputs A and B cannot vary arbitrary. In the full-buffer circuit or other micro-pipeline [4] based circuits, A is the output of previous C-element, which cannot switch unless its inputs switches to the same level in advance. However, input B of previous C-element is the inverted version Z of the current C-element. Therefore, when A is high (low), it cannot switch to low (high) before Z becomes high (low). This property is specified by $\Box A \overset{\mathbf{U}}{\Rightarrow} Z$ which is short for $A \Rightarrow (A \,\mathbf{U}\, Z)$, *i.e.*, if A holds in the current state, A will continue to hold until a state in which Z holds. It is similar for input B. The specification of inputs is shown as the "assume" clause in Figure 8.A.

A. Assume (environment controls inputs A, B)

$$(\Box A \overset{\mathbf{U}}{\Rightarrow} Z) \wedge (\Box \neg A \overset{\mathbf{U}}{\Rightarrow} \neg Z) \wedge (\Box B \overset{\mathbf{U}}{\Rightarrow} Z) \wedge (\Box \neg B \overset{\mathbf{U}}{\Rightarrow} \neg Z)$$

B. Guarantee (C-element controls output Z)

$$\Box(A \wedge B \Rightarrow \Diamond Z) \wedge \Box(\neg A \wedge \neg B \Rightarrow \Diamond \neg Z)$$

Fig. 8. Discrete Specifications of C-elements

Circuit-level verification requires continuous specifications. To support continuous variables and specify analog signals, we introduce Brockett annuli into LTL. Figure 9 shows a Brockett annulus [28]. When a variable is in region 1, its

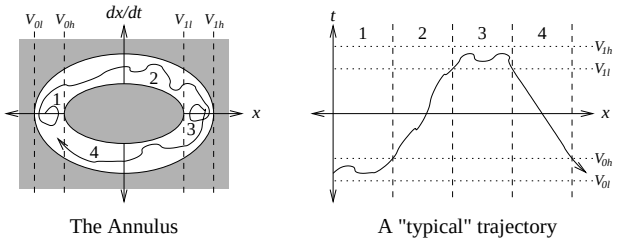


Fig. 9. Brockett Annulus

value is constrained but its derivative may be either positive or negative. Thus, region 1 of the annulus specifies a logically low signal: it may vary in a specified interval around the nominal value for low signals. When the variable leaves region 1, it must be increasing; therefore, it enters region 2. Because the derivative of the variable is positive in region 2, it makes a monotonic transition leading to region 3. Regions 3 and 4 are analogous to regions 1 and 2 corresponding to logically high and monotonically falling signals respectively. Furthermore, the horizontal radii of the annulus define the maximum and minimum high and low levels of the signal (i.e. V_{0l}, V_{0h}, V_{1l}, and V_{1h} in Figure 9). We also add constraints specifying the minimum dwell time t_l, t_h in regions 1 and 3, *i.e.*, the minimum duration of sojourns in region 1 and 3. We specify a Brockett annulus by $B([V_{0l},V_{0h},V_{1l},V_{lh}],[b_i,b_o],[t_l,t_h])$, where b_i is the vertical radius of the inner ellipse and b_o is the vertical radius of the outer ellipse.

A Brockett annulus provides the mapping from continuous trajectories to discrete traces. We write $B_i(x)$ to indicate variable x is in region i of the annulus. If a trajectory is in region B_1 for variable x, then its discrete abstraction is unambiguously low (i.e. false); likewise if it is in region B_3, then its clearly high. If the trajectory is in region B_2 (resp. B_4), then it *may* be treated as high (resp. low), but it is not required to do so until the signal enters region B_3 (resp. B_1). We say that a signal makes a rising transition when it enters region 2 of its Brockett annulus and a falling transition when it enters region 4. Using the mapping described above, we translate the discrete specification as shown in Figure 8 into the continuous specification as shown in Figure 10.

A. Assume (environment controls inputs A, B)

$$(\Box B_3(A) \overset{\mathbf{U}}{\Rightarrow} B_3(Z)) \wedge (\Box B_1(A) \overset{\mathbf{U}}{\Rightarrow} B_1(Z)) \quad \wedge$$
$$(\Box B_3(B) \overset{\mathbf{U}}{\Rightarrow} B_3(Z)) \wedge (\Box B_1(B) \overset{\mathbf{U}}{\Rightarrow} B_1(Z))$$

B. Guarantee (C-element controls output Z)

$$\Box(B_3(A) \wedge B_3(B) \Rightarrow \Diamond B_3(Z)) \quad \wedge$$
$$\Box(B_1(A) \wedge B_1(B) \Rightarrow \Diamond B_1(Z))$$

Fig. 10. Continuous Specifications of C-elements

The Brockett annulus construction is also used to specify input signals as it allows a large class of signals to be described in a simple and natural manner. Given any trajectory, $x(t)$ that is contained in the interior of the annulus, any trajectory $x'(t)$ that is obtained from a small, differentiable perturbation of $x(t)$ is also in the annulus. This is in contrast with traditional circuit simulators that test a circuit for specific stimuli such as piecewise linear or sinusoidal waveforms. Thus, a Brockett annulus can be given that contains all trajectories that will occur during actual operation, something that traditional simulation cannot achieve.

IV. Verification Platform

Our approach for circuit verification is based on reachability analysis, *i.e.*, computing all possible circuit states and then checking properties on these states. We developed a reachability analysis tool COHO for asynchronous or analog circuits and hybrid systems. In this section, we first present the basic reachability algorithm of COHO, noting more details and verification examples were presented in [29]–[34]. However, computing circuit states with the old version COHO requires lots of manual work which is tedious and error-prone. To automate reachability computation, we improved COHO by developing an easy-to-use interface based on hybrid automata and providing more useful features.

A. Reachability Algorithms

The reachability problem, *i.e.*, determining if a target state is reachable from an initial state, is usually undecidable for hybrid systems with non-trivial continuous dynamics [35]–[37]. COHO developed techniques and algorithms to approximate reachable regions and bound solutions of non-linear ODEs. All of these techniques over-approximate the forward reachable region and ensure soundness.

COHO uses *projectagons* to represent reachable regions in continuous spaces. A projectagon represents an object by its projection onto two-dimensional subspaces. Conversely, given a set of projection polygons, the projectagon is the object obtained by intersecting the prisms obtained by inverse-projecting each projection polygon back into the full-dimensional space. Most operations on projectagons can be performed efficiently by manipulating the individual polygons, avoiding explicit operations on high-dimensional objects. This representation corresponds to circuit designers' intuitive notion of how a circuit works. Typically, each signal is controlled by a small number of signals. Pairing a node with each of its controlling nodes naturally captures the causal behavior of the circuit. Projectagons can represents non-convex polyhedron accurately with efficient operations. This property is essential for circuit verification because the reachable region of complex circuits are usually non-convex. A non-convex region can be represented by a union of convex objects, such as the orthogonal polyhedra [21] representation used in D/DT. The method can be applied to low-dimensional (*e.g.*, 2-3) systems but is inefficient for high-dimensional systems as the number of convex objects generally increases exponentially.

COHO's reachability algorithm [32], [38] computes reachability through a sequence of time steps. In each step, COHO

computes a projectagon that contains all reachable points at the end of the step. The key to COHO's approach is that extremal trajectories originate from the boundary of the faces of the projectagon, and these faces correspond to edges of the projection polygons. This allows COHO to compute reachability by working on one edge at a time: COHO moves each edge forward, projects it back down to its polygon's subspace, and uses these projections to compute a bounding polygon for each projection polygon at the end of each time-step.

Given the netlist of a circuit, COHO automatically constructs an ODE model using modified nodal analysis. By treating all capacitances as having constant values to ground, a simple ODE is generated as

$$\dot{V} \quad = \quad C^{-1}MI_{ds}(V), \tag{3}$$

where I_{ds} is a function of transistor currents, C is a diagonal matrix of nodes capacitances and M is a connectivity matrix for transistors to nodes. More details of the modeling method were represented in [34]. However, nonlinear ODEs presented above cannot be solved analytically; thus, reachability computation must be based on approximate, numerical methods. COHO computes bounds on the points reachable from a projectagon face during a time step by approximating ODEs by *linear differential inclusions:*

$$AV+b-u \quad \leq \quad \dot{V} \quad \leq \quad AV+b+u \tag{4}$$

where $AV+b$ is a linear approximation of the circuit model; and u is a vector that upper bounds the approximation error.

We use HSPICE to obtain tables of I_{ds} data for a particular process on relatively fine grid and use this data in our reachability calculations. Because I_{ds} is monotonic in transistor node voltages, it is straightforward to obtain linear inclusions that contain the HSPICE model based on the tabulated data. Inclusions for dynamics of inputs A and B can be constructed from the Brockett annulus construction described in Section III.C. Substituting these linear inclusions into Equation 3 provides the linear differential inclusion required for COHO. This approach is simple and general: we can generate accurate models for any process with vendor provided SPICE models. Furthermore, our table is interval value based, *i.e.,* both lower and upper bounds of the current are provided for each grid point. Therefore, the conservative table allows non-deterministic which can be used to model PVT variations or noise. Compared with other methods such as the simulation aided verification method used in LEMA [16], our approach has the benefit that we only simulate a small number of basic devices, *e.g.,* transistors, instead of the whole circuit or macro-models.

B. Hybrid Automata Based Interface

Given the mathematical model as shown in Equation 3, COHO can compute forward reachable regions from an given initial region. However, it is non-trivial to compute all circuit states using the fundamental reachability analysis algorithms directly. To automate reachability computation, we implemented a general interface for reachability analysis in COHO. The interface employs a standard model for hybrid systems: *hybrid automata* [39], [40]. The hybrid automaton used in COHO is a tuple $\mathbf{M} = (\mathbf{Q}, \mathbf{X}, \mathbf{F}, \mathbf{T}, \mathbf{I}, \mathbf{G}, \mathbf{R}, \mathbf{S}_0)$ where

- $\mathbf{Q}$ is a finite set of *discrete states*.
- $\mathbf{X} \in \mathbb{R}^n$ is the *continuous state space*, where n is the number of continuous variables. $\mathbf{S} = \mathbf{Q} \times \mathbf{X}$ is the *state space* of the system.
- $\mathbf{I} : \mathbf{Q} \rightarrow 2^{\mathbf{X}}$ is a collection of *invariants*. $\mathbf{I}(q)$ is the condition of continuous variables which must be satisfied in the state q. The condition is described by a system of inequalities, *e.g.,* $(x_1 \leq 1) \vee (x_1 + 2x_2 \geq 10)$.
- $\mathbf{F} : \mathbf{Q} \rightarrow (\mathbf{X} \rightarrow R^n)$ is a set of *continuous dynamics*. For each state q, the evolution of continuous variables is governed by the deterministic or non-deterministic dynamics $\mathbf{F}(q)$.
- $\mathbf{T} \subseteq \mathbf{Q} \times \mathbf{Q}$ is a set of *discrete transitions*. Each transition $t = (q, q')$ identifies a *source state* q and a *target state* q'.
- $\mathbf{G} : \mathbf{Q} \rightarrow (\mathbf{X} \rightarrow 2^{\mathbf{X}})$ assigns each state a *guard condition* which determines the pre-condition of discrete transitions.
- $\mathbf{R} : \mathbf{T} \rightarrow (2^{\mathbf{X}} \rightarrow 2^{\mathbf{X}})$ is a collection of *reset maps*. For each transition t, $\mathbf{R}(t)$ alters the continuous variables in the source state q, which will be used in the target state q'.
- $\mathbf{S}_0 = \mathbf{Q}_0 \times \mathbf{X}_0 \subseteq \mathbf{Q} \times \mathbf{X}$ is the *initial region* of the automaton.

With the hybrid automata based interface, it is much easier for users to describe all reachability computations required to verify a property of a circuit. Given a hybrid automaton, many functions and features can be implemented easily. First, stop-resume computation is supported automatically by saving a check point after completing the reachability computation in an automaton state. Second, COHO can adjust function parameters to optimize performance or accuracy in each state independently. For example, computations in a state where the continuous dynamic converges rapidly can be optimized for performance; and can be optimized for accuracy otherwise to avoid false negative results. This is important for successfully verifying a complex circuit, since reachability computation is time-consuming and reducing approximation error is crucial for a successive verification. Third, COHO can partition a state of a hybrid automaton into several discrete states by *slicing* a monotonically changing variable. For example, the output Z of C-elements increases monotonically when inputs A and B are both clearly high. The corresponding state can be partitioned into several sub-states by slicing the value of Z. The reachability computation is performed in the first sub-state and then in the second sub-state using the result from the first one as the initial region, and so on. The slicing strategy avoids computing differential inclusion models as shown in Equation 4 for large regions. This feature is essential for our verification of C-element circuits as described in the next section.

V. Reachability Computations

We applied COHO to compute reachable regions of two C-elements as shown in Figure 4 and Figure 5. We first modeled all input transitions in one hybrid automaton and then applied an assume-guarantee strategy to partition the hybrid automaton into two independent automata. Several techniques, including the slicing method, are used to speed up computations and reduce approximation error. We now describe the reachability computation in greater details.

A. Constructing Hybrid Automata

In order to obtain conservative results, it is necessary to compute circuit states under all possible input transitions. As shown in Section III.C, input signals are specified by Brockett annuli. However, reachability computations of regions 2 and 4 must be performed separately, *i.e.*, in different states of a hybrid automaton. This is because regions 2 and 4 of an annulus admit signals with the same range, thus, are indistinguishable without extra information. Therefore, transitions of one input signal can be modeled by four *states* denoted as S_1, S_2, S_3, S_4, corresponding to four regions of the Brockett annulus construction. Reachable states of a circuit with only one input signal can be obtained by performing reachability computations in these four states. Noted that there are minimum dwell time requirements in low and high regions, trajectories cannot leave $S_1(S_3)$ before staying in the state for the minimum time t_l (t_h).

Transitions of two input signals A, B are more complicated. Assuming two signals are independent, there are $4^2 = 16$ possible concurrent transitions, which are denoted as $S_{<i,j>}, i, j \in \{1,2,3,4\}$, where $i(j)$ is the region of signal $A(B)$ (*i.e.*, regions 1-4 correspond to low/rising/high/falling signals respectively). We assume that the minimum duration time is T for regions 1 and 3 of two annuli, otherwise, we can set T as the minimum value of them. Due to the minimum dwell time requirement, trajectories in $S_{<1,1>}$ cannot enter $S_{<2,1>}$ unless A has stayed in region 1 for at least T time. Similarly, trajectories cannot enter $S_{<1,2>}$ unless B has stayed in region 1 for T time. Therefore, there are four kinds of trajectories in this state. The first case is when both inputs satisfy the dwell time requirement thus trajectories can leave the current state at any time and enter $S_{<2,1>}, S_{<1,2>}$ or $S_{<2,2>}$. The second (third) case is when only $A(B)$ satisfies the dwell time requirement and trajectories can only enter $S_{<2,1>}$ ($S_{<1,2>}$). The final case is when none of inputs satisfy the dwell time requirement and trajectories cannot leave the current state. Furthermore, it is possible that B does not satisfy the dwell time requirement when A leaves region 1 whereas B starts to satisfy the requirement when A is in region 2. This generates infinite number of possible transitions, *e.g.*, B has stayed in region 1 for a time t when A leaves region 1. It is similar for trajectories in states $S_{<1,3>}, S_{<3,1>}$ and $S_{<3,3>}$.

However, it is generally impossible to record the exact time of each trajectory during reachability computation. We employ a conservative approximation technique to solve the problem. The method first measures the maximum rising time t_{max} from the inner bound of the Brockett annulus and then uses $T' = T - t_{max}$ as the minimum dwell time of region 1. This method allows trajectories to leave region 1 before time T, whereas ensures that B cannot leave region 1 when A is in region 2. Using this over-approximated technique, all concurrent input transitions are enclosed by a finite number of states.

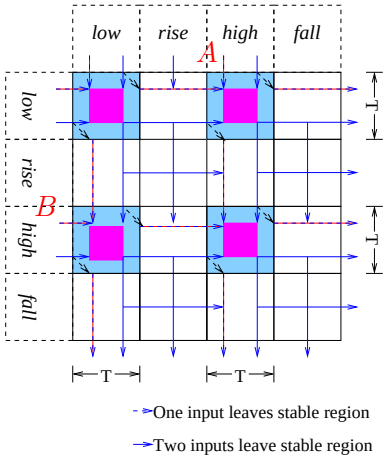


Fig. 11. Modeling input transitions

Figure 11 illustrates the concurrent input transitions for circuits with two independent inputs. Each input signal has four discrete states, thus, there are 16 combinations and four of them have dwell time requirements for both inputs. The solid arrows denote transitions of trajectories where both inputs satisfy their dwell time requirements and are allowed to make the transition. The dashed arrows denote transitions of trajectories where only one input satisfies its dwell time requirement. For examples, trajectories can go to $S_{<1,2>}, S_{<2,1>}$ or $S_{<2,2>}$ if both inputs satisfy the dwell time requirement when leaving $S_{<1,1>}$. If these trajectories enters $S_{<1,2>}$ and then $S_{<1,3>}$, they can either stay in $S_{<1,3>}$ for T' time or go to $S_{<2,3>}$ immediately as A satisfies the dwell time requirement of region 1. However, trajectories of the latter case cannot go to $S_{<2,4>}$ further because B has not satisfied its dwell time requirement of region 3 yet. It is similar for trajectories originated from states $S_{<1,3>}, S_{<3,1>}$ or $S_{<3,3>}$. With this analysis, we can see that there are totally 32 states. This approach can be extended to higher dimensions for circuits with more than two inputs.

Not all of these states are reachable for C-elements. As shown in Figure 8.A, input A can not switch to low when input B is switching from low to high, and vice versa. Therefore, trajectories from state $S_{<1,1>}$ must go to state $S_{<1,3>}$ or $S_{<3,1>}$ first and then stay in state $S_{<3,3>}$ for T time. It is similar for trajectories from state $S_{<3,3>}$. This reduces the number of reachable states from 32 to 16. A hybrid automaton with these states can be constructed to compute reachable regions of C-elements under all allowed input transitions.

B. The Assume-Guarantee Strategy

Reachability analysis is usually quite expensive, *e.g.,* it may take several days to complete the computation for a circuit with more than six nodes. Therefore, it is demanding to partition the reachability computation into several phases and perform computations in parallel. Therefore, we split the hybrid automaton constructed above into several smaller ones. To make reachability computation in each phase independent of each others, we applied an *assume-guarantee* strategy. For each phase, we *assume* that the initial circuit states are bounded by a region and compute forward reachable regions from the initial region. At the end, we *show* that all assumptions are correct based on reachable regions computed in all phases in order to establish an invariant set. Generally, we partition reachability computation based on input transitions as described above. For example, for a circuit with one input, its reachability computation can be partitioned into two phases which correspond to the rising and falling stages of the input signal respectively. For a circuit with two inputs, the reachability computation as shown in Figure 11 is usually partitioned into four phases: the first phase starts from state $S_{<1,1>}$ and ends in states $S_{<3,3>}$; the second phase is from state $S_{<1,3>}$ to state $S_{<3,1>}$; the third phase is from state $S_{<3,1>}$ to state $S_{<1,3>}$; and the last phase is from state $S_{<3,3>}$ to state $S_{<1,1>}$. This strategy enables us to complete reachability computation, debug the code, and address issues of over approximation in a reasonably timely manner.

For the C-element verification, we partitioned the reachability computation into two phases and constructed two hybrid automata correspondingly. The first one corresponds to the low to high transition phase where output Z switches from low to high. The initial region is estimated based on trajectories leaving state $S_{<1,1>}$. Then reachability computations are performed in states $S_{<1,2>}, S_{<2,1>}, S_{<2,2>}, S_{<2,3>}, S_{<3,2>}$ and ended in state $S_{<3,3>}$. The second automaton corresponds to the high to low transition phase similarly. The initial regions for these two hybrid automata are listed in Table I.D.

C. Verifying the Weak-feedback C-element

We first computed reachable regions of the weak-feedback C-element as shown in Figure 4. The circuit has six nodes, thus reachable regions are six-dimensional. Table I.C lists the projection polygons that we used in each phase. These were chosen with several considerations. First, we included at least one projection polygon for each variable to bound the resulting projectagon in all dimensions. Therefore, plane (Y,Z) is selected to bound the value of Z. Second, we chose projections that corresponded to logical dependencies between changing signals. For example, the value of signal Y is determined by signals B, P, N and output Z. Therefore, if Y is in a large interval, it is better to include planes $(Y,Z),(P,Y),(N,Y)$ and (B,Y). However, large number of projectagon polygons reduces the performance significantly. We can drop a plane if all projection polygons on it are close to rectangles, which means that these two variables are roughly independent. When inputs A and B are both high, we know the value of P is quite stable and slightly depends on the value of A. Thus, we dropped plane (A,P) in state $S_{<3,3>}$. However, we included the plane in state $S_{<2,2>}$ because signal P is falling when inputs A and B switch from low to high. Noting that we used different projection polygons in different states of a hybrid automaton. This improves performance without introducing large approximation error.

To reduce approximation error and improve performance, we sliced monotonically switching variables. We first sliced input signals A and B by 0.2 volts in regions 2 and 4. This resulted in 49 sub-states for state $S_{<2,2>}$ or state $S_{<4,4>}$. We also sliced Y into four equally spaced intervals as shown in Table I.C in phase one when Y starts to fall. In state $S_{<3,3>}$, output Z starts to rise when Y is low enough. Therefore, we sliced both Y and Z in order to reduce approximation error. With these slicing variables, the first hybrid automaton has totally 193 states. It is similar for the second hybrid automaton.

We also added several pre-proved invariants to COHO, as shown in Table I.B. We established these conditions by examining the time derivatives of signal voltages at the entire state space. They can also be checked by other tools such as HYSAT [41]. Generally, COHO produces more accurate reachable regions than interval propagation based tools, such as HYSAT and HSOLVER [42]. However, COHO is more expensive than these tools and reachable regions computed by COHO might violate these invariants because of over-approximation errors. The combination of reachability analysis and static analysis is necessary for many verification problems.

D. Verifying the Conventional C-element

The conventional C-element has two more nodes (P_2,N_2) than the weak-feedback C-element, thus, the corresponding ODE model is eight-dimensional. However, the reachability computation on the accurate model is more expensive than that for the weak-feedback C-element. Therefore, we simplified the circuit as a four-dimensional system by ignoring internal nodes P_1,P_2,N_1,N_2. Our method assumes that these internal nodes have no capacitance and voltages on these nodes are equilibria that balance the currents flowing through the upper and lower N/P-channel transistors. With this assumption, we created a three-dimensional model for the nMOS tetrode composed of T_{N1} and T_{N2} as shown in Figure 5 and another such model for the pMOS tetrode composed of T_{P1} and T_{P2}. This assumption is reasonable as the internal nodes have much smaller capacitances than other nodes of the circuit. In fact, many designers would instinctively ignore the contributions of these tiny capacitors. Similarly, we created simplified models for macro-devices composed of T_{N4},T_{N5} and T_{N6} (or T_{P4},T_{P5} and T_{P6}) as shown in Figure 5. These macro-models have four terminals A,B,Y and Z, thus the tabulated data generated by HSPICE simulations consumes large amount of memory. To solve the problem, we proposed two solutions. The first method uses coarser grids and reduces

A. Transistor sizes	
Weak-feedback	$T_{P1,P2} = 8.8u$, $T_{N1,N2} = 4u$, $T_{P3} = 4.4u$, $T_{N3} = 2u$, $T_{P4} = 2.2u$, $T_{N4} = 1u$
Conventional	$T_{P1,P2} = 8.8u$, $T_{N1,N2} = 4u$, $T_{P3} = 4.4u$, $T_{N3} = 2u$, $T_{P4,P5,P6} = 2.2u$, $T_{N4,N5,N6} = 1u$

B. Specifications and additional invariants		
Circuits	Input specifications	Invariants
Weak-feedback	$B([0,0.15,1.65,1.8],[0.75e10,1.75e10],[1e-9,1e-9])$	$0 \le [A,B,Y,Z] \le 1.8$, $0 \le N \le 1.45$, $0.45 \le Y \le 1.8$, $N \le Y \le P$
Conventional	$B([0,0.15,1.65,1.8],[0.75e10,1.75e10],[1e-9,1e-9])$	$0 \le [A,B,Y,Z] \le 1.8$

C. Projectagon polygons and slicing variables			
States	Projections (weak-feedback)	Projections (conventional)	Slicing Variables
$1.S_{<2,2>}$	(Y,Z);(N,Y);(*B*,Y);(*A*,P)	(*A*,Y); (*B*,Y); (Y,Z)	slice *A*,*B* by 0.2 volts, split *Y* into $[1.8,1.35,0.9,0.45,0]$ when $min(A,B) > 0.9$
$1.S_{<3,3>}$	(Y,Z);(N,Y);(P,Y);(*A*,*B*)	(*A*,Y); (*B*,Y); (Y,Z)	slice *Y* by 0.2 volts, split *Z* into $[0,0.2,0.5,1.0,1.8]$
$2.S_{<4,4>}$	(Y,Z);(P,Y);(*B*,Y);(*A*,N)	(*A*,Y); (*B*,Y); (Y,Z)	slice *A*,*B* by 0.2 volts, split *Y* into $[0,0.45,0.9,1.35,1.8]$ when $max(A,B) < 0.9$
$2.S_{<1,1>}$	(Y,Z);(P,Y);(N,Y);(*A*,*B*)	(*A*,Y); (*B*,Y); (Y,Z)	slice *Y* by 0.2 volts, split *Z* into $[1.8,1.6,1.3,0.8,0]$

D. Initial and ending hyper-rectangles										
Phases	Weak-feedback C-element						Conventional C-element			
	A	*B*	*P*	*N*	*Y*	*Z*	*A*	*B*	*Y*	*Z*
1, init	$[0.00,0.20]$	$[0.00,0.20]$	$[1.60,1.80]$	$[0.00,1.80]$	$[1.60,1.80]$	$[0.00,0.20]$	$[0.00,0.20]$	$[0.00,0.20]$	$[1.60,1.80]$	$[0.00,0.20]$
1, end	$[1.60,1.80]$	$[1.60,1.80]$	$[1.44,1.80]$	$[0.00,0.07]$	$[0.00,0.12]$	$[1.61,1.80]$	$[1.60,1.80]$	$[1.60,1.80]$	$[0.00,0.01]$	$[1.75,1.80]$
2, init	$[1.60,1.80]$	$[1.60,1.80]$	$[0.00,1.80]$	$[0.00,0.20]$	$[0.00,0.20]$	$[1.60,1.80]$	$[1.60,1.80]$	$[1.60,1.80]$	$[0.00,0.20]$	$[1.60,1.80]$
2, end	$[0.00,0.20]$	$[0.00,0.20]$	$[1.72,1.80]$	$[0.00,0.38]$	$[1.68,1.80]$	$[0.00,0.16]$	$[0.00,0.20]$	$[0.00,0.20]$	$[1.79,1.80]$	$[0.00,0.05]$

TABLE I
REACHABILITY SUMMARY OF C-ELEMENT VERIFICATION

the number of grids of the table to a reasonable value. The second method combines these two nMOS (pMOS) at the bottom and apply the maximum (minimum) value of *A* and *B* as the gate value. With these simplifications, we completed reachability computation within an acceptable time.

VI. RESULTS

We implemented the methods described above and applied them to the verification of the weak-feedback C-element shown in Figure 4 and the conventional C-element shown in Figure 5. Transistor models used during reachability computation are generated by HSPICE simulations for the TSMC 180nm, 1.8V, bulk CMOS process. Transistor sizes of these circuits are listed in Table I.A. Table I.B also lists Brockett annuli for input specifications. In the implementation, we increased the maximum low value from 0.15 to 0.2 and reduced the minimum high value from 1.65 to 1.6 to ensure the signal changes monotonically in regions 2 and 4 using the differential inclusion model as shown in Equation 4.

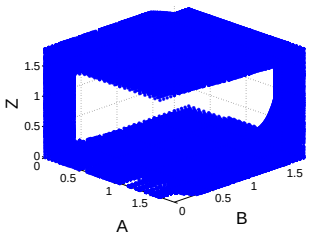


Fig. 12. Reachable regions of the weak-feedback C-element

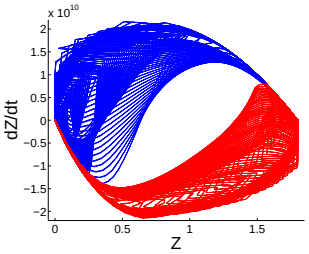


Fig. 13. Brockett annulus of the weak-feedback C-element

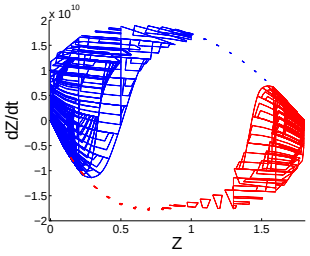


Fig. 14. Brockett annulus of the conventional C-element

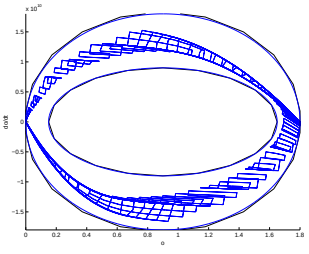


Fig. 15. Brockett annulus of the "inverted" weak-feedback C-element

A. Verified Properties

With the methods and parameters described above, we have computed reachable regions for both weak-feedback and conventional C-elements. From Table I.D, we can see that the reachable region at the end of each phase is contained by the initial hyper-rectangle of the other phase. Therefore, we have established an invariant set for the C-element. Figure 12 illustrates the reachable regions of the weak-feedback C-element by projecting them onto the (A,B,Z) subspace. The figure shows that output *Z* stays as low until both inputs are logically high and *Z* does not fall until inputs are clearly low. The analog behavior of C-elements supports specifications as shown in Figure 8.B and Figure 10.B.

For each projectagon computed by COHO, we computed the time derivative of signal *Z*. We note that $\dot{Z}$ is negative monotonic in *Z* and *Y*. Thus, the extremal values of *Z* vs. $\dot{Z}$ occur on the boundary of the (*Y*, *Z*) projection. For each edge of the (*Y*, *Z*) projection, COHO computes a linear differential inclusion model as shown in Equation 4 and uses this model to find the reachable combination of *Z* and $\dot{Z}$. From these, we constructed a Brockett annulus that is satisfied by *Z* and $\dot{Z}$. Figure 13 shows the result for the weak-feedback C-element and Figure 14 shows the result for the conventional C-element. From these plots, we can see that the Brockett annulus of the conventional C-element is tighter than the one

of the weak-feedback C-element. This is because there is no fight between the feedback stage and the output stage in the conventional C-element which creates a tighter annulus; and we applied simplified models as described in Section V.D which reduces approximation errors. The simplified models are suitable to find obvious errors more efficiently. Although the computed result may not be an over-approximation of real circuit states, it estimates the shape of reachable regions. Therefore, we can perform two reachability computations: one with simplified models and the other with accurate models. The result of the first computation can be used to optimize COHO for performance or accuracy in the later computation, *e.g.*, choosing projection polygons, estimating COHO parameters, slicing variables, etc.

Figure 13 shows that output Z does not satisfy the Brockett annulus in Table I.B. To solve the problem, we appended an inverter to output of the weak-feedback circuit and performed a separate reachability analysis for the inverter. We specified the input signal of the inverter by the Brockett annulus in Figure 13 and computed the Brockett annulus of the output. The result is shown in Figure 15 in which blue polygons illustrate the Brockett annulus of the output signal, and black ellipsoids denote the Brockett annulus used in the specification (Table I.B). From the plot, we can see that the output of the "inverted" C-element satisfies the same specification with input signals A and B. Similar result was obtained for the conventional C-element.

Obviously, we can apply COHO to compute reachable regions of NOR gates used in the full-buffer circuit as shown in Figure 1. The NOR gate has four nodes, thus, the reachability computation required to verify the circuit is more efficient. It is easier to show that the input and output of a proper sized NOR gate satisfy the same Brockett annulus. Therefore, all internal signals of the full-buffer circuit satisfy the same Brockett annulus, if signals from input and output environments also satisfy this Brockett annulus. Under this condition, the full-buffer always supports the designed handshake protocol, thus, the correctness of the circuit is guaranteed. The method by showing all subcomponents satisfy the same specification can be applied to verify the correctness of other complex asynchronous circuits that employ the micro-pipeline [4] structure.

During the verification, we failed to show that Z satisfies a Brockett annulus if the keeper inverter is too large or small. When the keeper is too big, input stage transistors do not have enough power to switch the state of memory point. On the other extreme, the voltage of Z may reach high (low) before inputs switch to logic high (low), which indicates the circuit is not robust to noise. The failures of verification can help designers to find potential bugs in circuit designs.

B. Performance

We also evaluated the performance of COHO. We ran all reachability computations on a 64-bit Linux server with sixteen 2.27GHz Xeon processors and 32G bytes of memory. Although COHO is partitioned into separate MATLAB and JAVA processes, one COHO process only uses one CPU at a time and consumes about 1-2G bytes memory. We used the *time* function of the *bash* shell to measure the run-time of the JAVA process and used the *cputime* function to measure the computation time of each state in a hybrid automaton. Generally, the JAVA process consumes 40%-50% of total run-time and linear program solver consumes approximately 30% of total run-time.

We computed reachable regions of the weak-feedback C-elements with different sets of parameters. We first performed a reachability computation with eight projection polygons ((Y,Z), (N,Y), (P,Y), (B,Y), (B,P), (B,N), (A,P), (A,N)) to obtain accurate results. But we did not slice variables Y and Z. The total run-time of this reachability computation is about 144 hours for phase one and 81 hours for phase two. We then enabled the slicing strategy in state $S_{<3,3>}$ and reduced the run-time of phase one to 90 hours. Finally, we used the set of parameters listed in Table I. This reduces the run-time to 32 hours and 29 hours for phase one and phase two separately. From these data, we can see that the performance of COHO highly depends on the number of projection polygons. The slicing technique also improves performance by about 50% as reachability computation in a smaller region is faster. On the other hand, the slicing method does not improve accuracy significantly. That is because we used a large set of projection polygons which ensures accurate results. However, we cannot reduce the number of projection polygons to the minimum value in order to obtain the best performance. For example, we failed to verify the weak-feedback C-element when we dropped the (Y,N) plane in state $S_{<2,2>}$ of phase one. The run-time for the conventional C-element verification is a little more than 1 hour for each phase. This demonstrates that reducing the number of system dimensions can improve performance significantly.

VII. CONCLUSIONS

We applied formal methods to compute reachable regions of C-elements and found a sufficient condition which guarantees the correct behavior of an asynchronous full-buffer circuit. The success of these verifications demonstrates that reachability analysis can be applied to verify asynchronous circuits and find potential bugs which are difficult to be spotted by simulation based methods. We also provided a standard, hybrid automaton based interface for COHO and implemented more features such as slicing, assume-guarantee strategy, *etc.* These improvements make COHO an easy-to-use platform and also speed up the reachability analysis.

There are many directions for future work. First, our verification method and tool can be applied to verify more asynchronous or analog circuits. Second, we would like to compare COHO in run-time and accuracy to other tools, such as D/DT, PHAVER, LEMA, HYSAT, *etc.* Unfortunately, the other tools that we are aware of have not formally verified practical circuits. We plan to use commonly used hybrid system examples as the testbench in the first step.

COHO is still slow for practical use. A promising research direction is to improve the performance of COHO by parallel computing. For example, independent reachability computation on projectagon faces can be easily parallelized. Noting reachability computations in many hybrid automaton states can be computed at the same time, the hybrid automata based interface provides more parallelisms.

ACKNOWLEDGMENTS

We thank Mark Greenstreet and Ian Mitchell for helpful discussions and the anonymous reviewers for their valuable comments. This work was funded in part by an NSERC Discovery Grant, an NSERC CRD Grant, and research funding from Intel, all of which we gratefully acknowledge.

REFERENCES

[1] J. E. Robertson, "Problems in the physical realization of speed independent circuits," in *Switching Circuit Theory and Logical Design, 1961. SWCT 1961. Proceedings of the Second Annual Symposium on*, Oct. 1961, pp. 106 –108.

[2] K. van Berkel, "Beware the isochronic fork," *Integration, the VLSI Journal*, vol. 13, no. 2, pp. 103 – 128, 1992.

[3] F. Ouchet, K. Morin-Allory, and L. Fesquet, "Delay insensitivity does not mean slope insensitivity!" in *Asynchronous Circuits and Systems (ASYNC), 2010 IEEE Symposium on*, May 2010, pp. 176 –184.

[4] I. E. Sutherland, "Micropipelines," *Communications of the ACM*, vol. 32, no. 6, pp. 720–738, 1989.

[5] M. R. Greenstreet and S. Yang, "Verifying start-up conditions for a ring oscillator," in *GLSVLSI '08: Proceedings of the 18th ACM Great Lakes symposium on VLSI*. 2008, pp. 201–206.

[6] M. H. Zaki, S. Tahar, and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, vol. 39, no. 12, pp. 1395 – 1404, 2008.

[7] E. Asarin, T. Dang, G. Frehse, A. Girard, C. L. Guernic, and O. Maler, "Recent progress in continuous and hybrid reachability analysis," in *In Proc. IEEE International Symposium on Computer-Aided Control Systems Design. IEEE Computer*. 2006.

[8] E. Barke, D. Grabowski, H. Graeb, L. Hedrich, S. Heinen, R. Popp, S. Steinhorst, and Y. Wang, "Formal approaches to analog circuit verification," in *DATE*. 2009, pp. 724–729.

[9] F. Wang, "Formal verification of timed systems: A survey and perspective," in *Proceedings of the IEEE*, 2004, p. 2004.

[10] B. I. Silva, O. Stursberg, B. H. Krogh, and S. Engell, "An assessment of the current status of algorithmic approaches to the verification of hybrid systems," *40th Conference on Decision and Control*, Dec. 2001.

[11] R. P. Kurshan and K. L. McMillan, "Analysis of digital circuits through symbolic reduction," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 10, no. 11, pp. 1356–1371, Nov. 1991.

[12] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," *Int. J. Softw. Tools Technol. Transf.*, vol. 10, no. 3, pp. 263–279, 2008.

[13] T. A. Henzinger, P.-H. Ho, and H. Wong-toi, "HyTech: The next generation," in *In Proceedings of the 16th IEEE Real-Time Systems Symposium*. 1995, pp. 56–65.

[14] T. A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-toi, "Beyond HyTech: Hybrid systems analysis using interval numerical methods," in *in HSCC*. 2000, pp. 130–144.

[15] S. Little, N. Seegmiller, D. Walter, C. Myers, and T. Yoneda, "Verification of analog/mixed-signal circuits using labeled hybrid Petri nets," in *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*. 2006, pp. 275–282.

[16] S. Little, D. Walter, K. Jones, and C. Myers, "Analog/mixed-signal circuit verification using models generated from simulation traces," *Automated Technology for Verification and Analysis, Lecture Notes in Computer Science*, vol. 4762, pp. 114–128, 2007.

[17] A. Chutinan, "Hybrid system verification using discrete model approximations," Ph.D. dissertation, Carnegie Mellon University, 1999.

[18] B. I. Silva, K. Richeson, B. Krogh, and A. Chutinan, "Modeling and verifying hybrid dynamical systems using CheckMate," in *Proceedings of the 4th International Conference on Automation of Mixed Processes (ADPM 2000)*, Sept. 2000, pp. 323–328.

[19] T. Dang, "Verification and synthesis of hybrid systems," Ph.D. dissertation, Institut National Polytechnique de Grenoble, 2000.

[20] E. Asarin, T. Dang, and A. Girard, "Reachability analysis of nonlinear systems using conservative approximation," in *HSCC*. 2003, pp. 20–35.

[21] O. Bournez, O. Maler, and A. Pnueli, "Orthogonal polyhedra: Representation and computation," in *Hybrid Systems: Computation and Control*. 1999, pp. 46–60.

[22] T. Dang, A. Donzé, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *FMCAD*. 2004, pp. 21–36.

[23] S. Gupta, B. H. Krogh, and R. A. Rutenbar, "Towards formal verification of analog designs," in *Proceedings of 2004 IEEE/ACM International Conference on Computer Aided Design*, Nov. 2004, pp. 210–217.

[24] G. Frehse, B. H. Krogh, and R. A. Rutenbar, "Verifying analog oscillator circuits using forward/backward abstraction refinement," in *DATE '06: Proceedings of the conference on Design, automation and test in Europe*. 2006, pp. 257–262.

[25] S. Little, "Efficient modeling and verification of analog/mixed-signal circuits using labeled hybrid petri nets," Ph.D. dissertation, University of Utah, 2008.

[26] A. J. Martin, *Formal Program Transformations for VLSI Circuit Synthesis*. 1990, pp. 59–80.

[27] A. Pnueli, "The temporal semantics of concurrent programs," *Theoretical Computer Science*, vol. 13, pp. 45–60, 1981.

[28] R. W. Brockett, "Smooth dynamical systems which realize arithmetical and logical operations," in *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*. 1989, vol. 135, pp. 19–30.

[29] C. Yan, M. R. Greenstreet, and M. Laza, "A robust linear program solver for reachability analysis," in *Proceedings of the First International Conference on Mathematical Aspects of Computer and Information Sciences* , July 2006, pp231–242.

[30] C. Yan, "Coho: A verification tool for circuit verification by reachability analysis," Master's thesis, The University of British Columbia, Aug. 2006.

[31] C. Yan and M. R. Greenstreet, "Circuit level verification of a high-speed toggle," in *FMCAD*. Nov. 2007, pp. 199–206.

[32] C. Yan and M. R. Greenstreet, "Faster projection based methods for circuit level verification," in *ASP-DAC*. Jan. 2008, pp. 410–415.

[33] C. Yan and M. R. Greenstreet, "Verifying an arbiter circuit," in *FMCAD*. Nov. 2008, pp. 1–9.

[34] C. Yan, M. R. Greenstreet, and J. Eisinger, "Formal verification of arbiters," *The 16th IEEE International Symposium on Asynchronous Circuits and Systems*, May 2010.

[35] R. Alur and D. L. Dill, "A theory of timed automata," *Theoretical Computer Science*, vol. 126, no. 2, pp. 183 – 235, 1994.

[36] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" in *Journal of Computer and System Sciences*. 1995, pp. 373–382.

[37] G. Lafferriere, G. J. Pappas, and S. Yovine, "Decidable hybrid systems," Department of Mathematical Science, Portland State University, Portland, OR, Tech. Rep., 1998.

[38] M. R. Greenstreet and I. Mitchell, "Reachability analysis using polygonal projections," in *HSCC '99: Proceedings of the Second International Workshop on Hybrid Systems*. 1999, pp. 103–116.

[39] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P. H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine, "The algorithmic analysis of hybrid systems," *Theoretical Computer Science*, vol. 138, no. 1, pp. 3 – 34, 1995.

[40] T. A. Henzinger, P.-H. Ho, and H. Wong-toi, "Algorithmic analysis of nonlinear hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, pp. 225–238, 1996.

[41] C. Herde, A. Eggers, M. Fränzle, and T. Teige, "Analysis of hybrid systems using HySAT," in *ICONS '08: Proceedings of the Third International Conference on Systems*. 2008, pp. 196–201.

[42] S. Ratschan and Z. She, "Constraints for continuous reachability in the verification of hybrid systems," in *AISC*. 2006, pp. 196–210.