Circuit Level Verification of a High-Speed Toggle

Chao Yan and Mark R. Greenstreet
Department of Computer Science
University of British Columbia

***Abstract*—As VLSI fabrication technology progresses to 65nm feature sizes and smaller, transistors no longer operate as ideal switches. This motivates verifying digital circuits using continuous models. This paper presents the verification of the high-speed, toggle flip-flop proposed by Yuan and Svensson [1]. Our approach builds on the projection based methods originally proposed by Greenstreet and Mitchell [2], [3]. While they were only able to demonstrate their approach with two- and three-dimensional systems, we apply projection based analysis to a seven-dimensional model for the flip-flop. We believe that this is the largest verification to date of a digital circuit using non-linear circuit-level models.**

In this paper, we describe how we overcame problems of numerical errors and instability associated with the original projection based methods. In particular, we present a novel linear-program solver and new methods for constructing accurate linear approximations of non-linear dynamics. We use the toggle flip-flop as an example and consider how these methods could be extended to verify a standard cell library for digital design.

I. Introduction

Deep-submicron technologies simultaneously confront designers with transistor behaviors that require circuit-level models to produce working designs and with integration densities that require working at high-levels of abstraction. Due to leakage currents, small transistors do not operate as the ideal switches that have been the foundation of synthesis and switch-level simulation tools for the past twenty years. Other deep-submicron challenges include crosstalk, power-supply noise, and random parameter variations. Current design tools do not provide a satisfactory way of dealing with these issues. Typically, coarse approximations or heuristics are used to validate designs, an approach that simultaneously sacrifices performance and fails to guarantee working silicon. Formal methods can help address these challenges by verifying proper behaviors at low-levels of abstraction and ensuring that the abstractions of these low-level behaviors to higher levels of abstraction are sound.

A circuit with a continuous model has an uncountably infinite state space, typically $\mathbb{R}^d$ where d is the number of nodes in the circuit. Furthermore, the non-linear differential equations that describe circuit behavior do not have closed form solutions; thus, purely symbolic methods are unlikely to succeed for circuit-level verification. To address these problems, tractable approximation methods are needed. To ensure soundness, these approximations must over approximate the reachable space. While such approximation-based techniques cannot be complete, we want the approximations to be accurate enough that the verification method can be applied to real designs. Finally, the reachability computations must be sufficiently efficient as to enable the verification of real designs in a tolerable amount of time.

The current work builds upon the projection based methods for a tool called "Coho" originally proposed in [2], [3]. Coho represents a region in $\mathbb{R}^d$ by its projection onto two-dimensional subspaces. The intuition behind this approach is that the input/output behaviors of digital circuits can be captured by these projections that correspond to pairs of causally related variables.

The framework presented in [2], [3] was only applied to simple two- and three-dimensional models due to numerical conditioning problems that often occurred in solving the linear-programs that were used when calculating the reachable space. We present our solution to these numerical problems. We also describe places where the original formulation for Coho resulted in large over approximations and present refinements that enable much more accurate calculations.

We apply our modified version of Coho to the verification of a toggle flip-flop that was originally presented by Yuan and Svensson [1]. This toggle is a relatively simple circuit consisting of nine transistors and seven nodes. This makes it similar in size to many cells in a typical library for standard cell design. For example, if a design has a critical timing path that cannot be satisfied by gate sizing or local logical changes, a designer might identify a custom logic function that could be implemented to achieve an acceptable delay. In current design flows, adding a cell to the library is an arduous task, in large part due to the large number of circuit simulations that must be manually run and checked; the time required to perform such simulations is often unacceptable for the project schedule. Thus, a design team in this situation can be forced to either make large architectural changes or to lower performance targets. Having verified the toggle, we can realistically hope to verify cells that a designer wants to add to a cell library. Similar opportunities to improve the design flow arise when trying to satisfy cross-talk requirements, when using dynamic or highly-skewed logic gates, when integrating self-timed circuits in a synchronous design, etc.

The toggle is an attractive first example for circuit-level verification because it has an interesting sequential behavior that we can specify very precisely in both discrete and continuous formulations. It exhibits a critical race which places demands on the accuracy of the approximations made by any reachability tool. Finally, with seven nodes, it presents an opportunity to verify discrete properties of a seven-dimensional, non-linear system. We are not aware of any previous examples of verifying digital circuit behaviour from non-linear circuit models for circuits of more than four nodes.

II. Related Work

The problem of verifying systems with a combination of discrete and continuous models has been a focus of the hybrid systems research in the past ten years. This section first examines prior work in verifying circuits and hybrid systems and then examines the prior work on Coho upon which our current research is built.

A. Verification of Hybrid Systems

One of the earliest tools for verifying hybrid systems was *HyTech* [4], [5]. Based on *linear hybrid automata* [6], HyTech models continuous variables as piecewise linear functions of time. While non-linear systems can be approximated by piecewise linear ones [7], the number of pieces required to obtain a given degree of accuracy grows exponentially with the dimensionality of the system, exacerbating the high complexity of the model-checking algorithm. Even with the simplistic assumptions that each variable evolves as a piecewise linear function of time, nearly all properties of linear hybrid automata are undecidable [8], [9]. Thus, heuristics and approximation methods are required for verifying real circuits.

The ideas in HyTech have been extended recently by Frehse in the implementation PHAVer [10]. PHAVer uses arbitrary precision integer and rational arithmetic to avoid problems of numerical overflow that limited HyTech. Furthermore, PHAVer uses the Parma Polyhedra Library [11]. PHAVer has verified larger problems than HyTech could including a non-linear, tunnel-diode oscillator circuit and a VCO [**?**]. We are not aware of any applications of PHAVer to digital circuit designs.

The *d/dt* tool [12] 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 [13], i.e. finite unions of full-dimensional, fixed size hyper-rectangles, and approximates the reachable state using numerical integration and polyhedral approximation. To the best of our knowledge, all examples computed with d/dt have been low, dimensional, with two- or three-dimensions, which we believe reflects the high-complexity of representing the reachable space as an explicit set of hypercubes. Earlier, Kurshan and McMillan [14] used a similar approach, for the verification of a simple arbiter, a four-dimensional, non-linear problem.

CheckMate [15] is a Matlab based tool for modeling, simulating and verifying properties of a class of hybrid systems: *threshold-event-driven hybrid systems*. Checkmate can model systems with non-linear dynamics by computing a convex polyhedral approximation of the reachable region. Checkmate has recently been used to verify simple circuits including a tunnel-diode oscillator and a Sigma-Delta modulator [16], both modeled with three-dimensional state spaces. As with PHAVer, we are not aware of any applications of Checkmate to digital circuit designs or to circuits with higher dimensional models.

B. Coho

Coho represents reachable sets with *projectagons*. A projectagon is the high dimensional (and potentially nonconvex) bounded polytope formed by the intersection of a collection of prisms. Each prism is unbounded in all but two dimensions, and in those two dimensions the cross-section of the prism is a bounded polygon. The projection polygons are not required to be convex; thus, non-convex, high-dimensional objects can be represented by projectagons. The high-dimensional object represented by a projectagon is the set of all points that satisfy the constraints of each projection. As an example, Figure 1 shows how a three-dimensional object (the "anvil") can be represented by its projection onto the xy, yz, and xz planes.

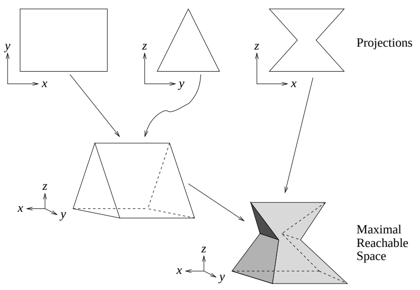


Fig. 1. A Three-Dimensional "Projectagon"

The basic ideas behind Coho's reachability analysis are straightforward. We require that the time derivatives for the state variables are finite and therefore that trajectories are continuous. This, the extremal trajectories emanate from the faces of the projectagon. At each time step in its reachability analysis, Coho derives a convex polytope that contains all points within a preset distance of the current projectagon. Coho then derives bounds on the time derivatives of the state variables within this polytope, and uses these to determine a time step size for which all trajectories are guaranteed to remain in the polytope. Coho then moves each face according to its maximum outward derivative to obtain a projectagon that contains the reachable space at the end of the time step. We describe these steps in greater detail below.

Let P_0 denote the projectagon for the reachable space at the beginning of a time step, and let λ be a positive real.‘ To derive a convex polytope that contains all points within distance λ of P_0, we observe that the convex-hull of a projectagon is contained in the projectagon obtained from the convex hulls of each of the projection polygons. Thus, Coho computes the hulls of these two dimensional polygons. Coho then moves all edges of these hulls outward by λ. These bloated hulls are the projection polygons of a projectagon that contains all points that are within distance λ of the original projectagon. We call this bloated version of the original projectagon P_{bloat}. Coho represents P_{bloat} as a linear program.

Faces of the projectagon correspond to edges of the projection polygons. Thus, the version of Coho described in [3] operated on one projection polygon edge at a time. We describe this approach here and describe our refinement of this method in Section IV-C. For each polygon edge, Coho constructs an oriented rectangle that contains all points within distance λ of the edge. The intersection of the prism for this rectangle with P_{bloat} contains all points within distance λ of the current face. We call this region P_{bloat}, and Coho represents it with a linear program.

Coho then calls the user supplied circuit model with P_{bloat}. The model returns a matrix A, vectors b, and positive vector u such that

$$Av + b - u \quad \leq \quad \dot{v} \quad \leq \quad Av + b + u \tag{1}$$

for any point $v \in P_{bloat}$. Based on this linearized model and the linear program P_{bloat}, Coho constructs a linear program for all points reachable from the face at the end of the time step. Coho then projects the feasible region of this linear program back onto the basis variables for the polygon edge associated with the face to obtain the time advanced edge. Details are given in [3].

We now focus on those aspects of the Coho algorithm that we had to modify in order to successfully verify the toggle. The modifications that we made should be applicable to other circuit and hybrid systems verification problems as well. First, Coho makes extensive use of linear programs (LPs). Earlier experience with Coho showed that these LPs were often highly ill-conditioned, causing off-the-shelf linear program solvers such as Matlab's linprog and a direct implementation of Simplex to fail. In Section IV-A, we describe our robust linear program solver for the LPs that arise in Coho. Second, Coho was originally implemented with uniform bloats for all variables. We found that this led to both large error bounds when linearizing the circuit model and to Coho taking very small time steps. Section IV-B describes how we modified Coho to use different bloat amounts for different variables.

Each time a projection polygon edge is advanced, a new polygon is produced. The union of these "edge polygons" forms the boundary of the projection polygon at the end of the time step. This process can lead to a rapid increase in the number of projection polygon edges. Section IV-C describes how the original Coho simplified the projection polygons at each step to control this blow-up. In the same section, we present our modifications to the original algorithm to reduce the amount of over approximation at critical phases of the verification.

III. The Yuan-Svensson Toggle

Figure 2 shows the toggle circuit from [1]. Transistors are labeled with their shape factors and the capacitor on the q output represents a load equivalent to the gate capacitance of transistors with the a total shape factor of 36; this is the load that the toggle places on its clock input. We use this load to verify that the output of one toggle can drive the clock input of another to implement a ripple counter.

The operation of this circuit can be understood by using a simple switch model starting from a state where the ϕ input is low. In this case, y is driven high, z is floating, and x is the logical negation of z. Figure 3 shows the state transition

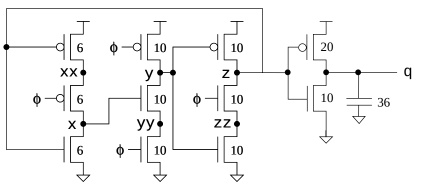
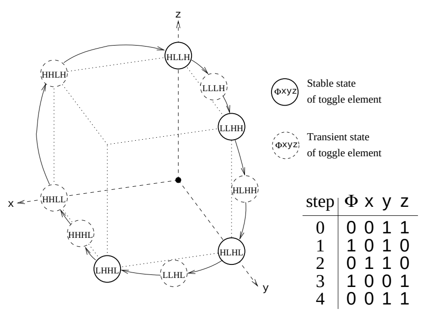


Fig. 2. Yuan and Svensson's Toggle Circuit



step	Φ	x	y	z
0	0	0	1	1
1	1	0	1	0
2	0	1	1	0
3	1	0	0	1
4	0	0	1	1

Fig. 3. State Transition Diagram for the Toggle

diagram for the toggle starting from the state where z is high when ϕ is low – the other case, with z high, is reached on step 2 of the figure. Note that from step 2 to 3 in the figure, all three of x, y and z change values. This is a critical race for the toggle. We further note that if the rise or fall times for ϕ are too large, the toggle will fail.

To specify the desired continuous behavior of the toggle circuit, we use the Brockett annulus construction [17] shown in Figure 4. When a variable is in region 1, its 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 small 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. Because transitions through regions 2 and 4 are monotonic, traversals of these regions are distinct events. This provides a topological basis for discrete behaviors. 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

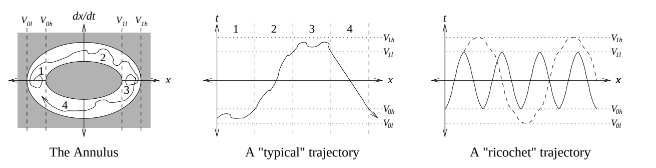


Fig. 4. Brockett's Annulus

figure 4). The maximum and minimum rise time for the signal correspond to trajectories along the upper-inner and upper-outer boundaries of the annulus respectively. Likewise, the lower-inner and lower-outer boundaries of the annulus specify the maximum and minimum fall times.

Note that a signal may remain in regions 1 or 3 arbitrarily long. This is essential when verifying the toggle where we must show that the output satisfies the constraints assumed of the input, even though the period of the output is twice that of the input. In addition to the constraints captured by the geometry of the annulus, we add constraints as in [20] for the minimum time that ϕ must remain in region 1 before entering region 2, and likewise for region 3. This construction allows a large class of input signals to be described in a simple and natural manner.

We specify the behavior of the toggle as a safety property. In particular, we use Coho to find an invariant subset of $\mathbb{R}^d$ such that all trajectories in this set have a period twice that of the clock signal. This notion can be formalized using a Poincaré section [18]. Let ϕ be the continuous signal corresponding to Φ, and let c be some constant with $V_{0h} < c < V_{1l}$. Consider the intersection of the invariant set with the $\phi = c$ hyperplane. These intersections form a Poincaré map [18]. We verify that these intersections form four disjoint regions (two for rising ϕ crossing c, and two falling crossings), All trajectories must visit these four regions in the same order. Thus, the $\phi = c$ plane partitions the invariant set of the continuous model into four, disjoint regions that map to the four discrete states of the discrete model.

Toggle elements can be composed to form a ripple-counter if there is an output variable such that for all trajectories in the manifold, the value and derivative of this variable satisfy the constraints of the input ring. It must also be shown that this output satisfies the minimum high and low time constraints. Section V shows that the toggle circuit satisfies all of these properties.

IV. Making Coho Work

A. Solving LPs in Coho

Coho makes extensive use of linear programs (LPs), and the original version of Coho was hindered because these LPs are occasionally extremely ill-conditioned. The LPs in Coho have constraints that correspond to the convex hulls of projection polygons. These LPs are naturally written with the form

$$\min_x c^T x \text{ s.t. } Ax \ \leq \ b \tag{2}$$

where each row of A has at most two non-zero elements. We call such an LP a *Coho LP*. The dual of a Coho LP is a standard form LP:

$$\min_\pi -b^T \pi \text{ s.t. } A^T \pi \ = \ c \tag{3}$$

Note that A^T has at most two non-zero elements in any column. In [19], Laza presented a linear-time algorithm for solving the linear systems that arise when applying the Simplex algorithm to the dual of a Coho LP. With Laza's algorithm, the tableau entries are computed from the original data at each pivot step; the only data carried forward from one pivot to the next is the original LP and the set of columns in the basis. By avoiding the rank-one updates of the tableau from the traditional formulation of Simplex, Laza's approach avoids error propagation from one pivot step to the next. His linear-time linear system solver makes this approach as efficient as traditional Simplex.

We implemented Laza's approach and included an arbitrary precision rational arithmetic package. Most computations are performed using interval arithmetic, and the arbitrary precision package is used for highly ill-conditioned bases and to verify the final solution. Our implementation has eliminated numerical stability problems from the LP solver. As an added benefit, our LP solver is guaranteed to find the exact optimum for the linear program in all cases. This guarantee allowed us to simplify many other parts of the Coho code.

B. Better Bloating

In the original formulation of Coho, the projection polygons were bloated equally in all directions to form the bloated convex hull that contained the trajectories for a time step. When verifying the toggle, we found that this resulted in large over approximations. Depending on the circuit state, some variables will have much larger time derivatives than others. If the state is bloated by the same amount for all variables, then the model will be evaluated with a much larger region than needed for the slow moving variables, and this causes the error term in the linear approximation of the model to be large.

We solved this problem by linearizing the models twice at each time step. The first phase is performed as in the original version of Coho, but we also keep track of the maximum magnitude of the derivative for each variable. At the end of the first phase, our new version computes the size of the allowed time-step. It also computes the bloat amount needed for each variable. In the second phase, faces are moved forward in time. In our new version, we linearize the model again for each face based on the per-variable bloat amounts computed from the first phase. This change enabled a dramatic reduction in the magnitude of the error terms in the linearizations of the model.

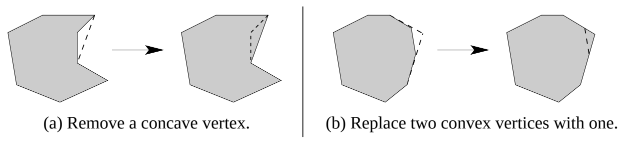

(a) Remove a concave vertex.
(b) Replace two convex vertices with one.

Fig. 5. Simplifying Projection Polygons

C. Simplifying Projection Polygons

At the end of each time step, Coho reduces the number of projection polygon vertices to keep their sizes tractable. As shown in Figure 5(a), Coho can simply delete convex vertices to produce an approximation of the original polygon.

Figure 5(b) shows the replacement of two consecutive, convex vertices with a single vertex, again producing an over approximation. Simplification serves two purposes. First it keeps the number of constraints in the LP for the bloated convex hull of the projectagon small. Second, it reduces the number of projectagon faces (i.e. projection polygon edges) that need to be advanced at each time step. These are separable concerns.

Our new version of Coho computes the convex hull of each polygon and simplifies it using the simplification shown in Figure 5(b) to control the number of constraints in the LP for the bloated convex hull, P_{bloat}. Then, we modified Coho to advance convex sequences of edges. Coho constructs LPs for these sequences and moves them forward in a single step. Of course, we still need to limit the growth of the number of polygon edges. The new Coho eliminates concave vertices as shown in Figure 5(a). Furthermore, once Coho identifies a convex sequence of vertices to advance, it can eliminate vertices as shown in Figure 5(b). Because we only have the extra edges for a single projection polygon at a time, we can maintain a much more detailed representation of the polygon without greatly increasing the time for the computation.

V. Verifying the Toggle

The verification that we present of the toggle resembles an earlier verification result presented in [20]. There are two significant differences in our approach and the earlier work. Most significantly, we include nodes xx, yy and zz in our model. This results in a seven-dimensional state space rather than the four-dimensional model used in [20]. Second, we model the drain-source currents of the transistors based on tabulated data obtained from HSPICE thus our results are based on the BSIM-3 models for the TSMC 180nm process. In [20], a simple, first-order, long-channel, MOSFET model was used for transistor current. Using a realistic model forced us to address other real-world issues, most notable of which was the leakage currents of the transistors. Thus, we added "keepers" to nodes x, y and z. While such practicalities can seem like a nuisance from a formal verification perspective, they show that our approach is solidly connected to the issues that challenge circuit designers for deep submicron processes.

We started our verification by simulating the toggle using a circuit simulation package that we have developed for use in Matlab. This approach had two advantages. First, we were able to informally validate our design before embarking on the more time-intensive effort of formal verification. Second, we modified the simulator to generate the linearized models with error bounds as required by Coho. Thus, we used the same circuit description for simulation and verification.

Our specification for the toggle requires it to have an invariant set that has twice the period of the input clock, ϕ. Accordingly, our reachability calculation is carried out for two periods of ϕ. We break each of these periods into the two phases: one for the rising transition of ϕ and the time that ϕ is high; and the other for the falling transition and low time. This partitioning had two advantages. First, we estimate a bounding hyper-rectangle for the end of each phase based on the simulation results. With these estimates, we divide the task of verifying the toggle into four separate proof obligations where each obligation is of the form:

> **Assume** that the circuit state is in hyper-rectangle Y_i at the end of phase i.
> **Show** that the circuit state will be in hyper-rectangle Y_{i+1} at the end of phase $i+1$.

By showing that the last phase leads to a hyper-rectangle that is contained in the initial hyper-rectangle of the initial phase, we establish that the reachable set that we have computed is invariant. Second, by separating the proof obligations, we can work on them in parallel. This was critical. Our current version of Coho is a prototype, and it is quite slow. It takes about a day to complete the reachability analysis for a single phase. Parallel computation allowed us to complete the verification, debug the model, and address issues of over approximation in a reasonably timely manner.

A. Modeling the Circuit

We model transistors as voltage controlled current sources, and all capacitors as having fixed values with one terminal connected to ground. For transistor j, let $ids_j(v_s, v_g, v_d)$ be the current that flows from the drain to the source when the voltages on the source, gate and drain are v_s, v_g and v_d respectively. We obtained our ids functions by tabulating data on a 0.01 volt grid for $(v_s, v_g, v_d) \in [-0.3, 2.25]^3$ for transistors in the TSMC 180nm, 1.8 volt, bulk CMOS process. We ignore gate leakage which is negligible in the 180nm process.

By Kirchoff's current law, the total current flowing out of each node through the capacitors connected to the node must equal the total current flowing into the node through the transistors. The current through a capacitor is $c\dot{v}$ where c is the capacitance of the capacitor and $\dot{v}$ is the time derivative of the voltage across the capacitor. This yields:

$$\dot{V} \quad = \quad C^{-1}\,M\,ids(V) \qquad (4)$$

where V is the vector of node voltages (one element for each node of the circuit); $ids(V)$ is the vector of drain-to-source currents (one element for each transistor); M maps transistors to nodes with $M(i,j) = 1$ if the source of transistor j is connected to node i, $M(i,j) = -1$ if the drain of transistor j is connected to node i, and $M(i,j) = 0$ otherwise. C is the matrix of inter-node capacitances. Because we model all capacitances as being fixed and to ground, C is fixed and diagonal.

Equation 4 is the basis for our circuit modeling. To create a linear model as in Equation 1, it suffices to linearize ids. Here, we'll present our initial approach. Given a linear program that contains the bloated face for which Coho needs a model, we obtain upper and lower bounds for v_s, v_g and v_d. We then compute a linear regression on the points in this bounding box using tables of precomputed sums. Noting that the ids function has exactly one inflection, (along the $v_s = v_d$ plane) enables efficient computation of the worst-case errors of the least-squares model. We then adjust the constant term from the linear-regression to balance the positive and negative worst-case errors. We now have

$$\begin{array}{ll} & (C^{-1}\,M\,A_{ids})V + (C^{-1}\,M\,A_{ids})b_{ids} - (C^{-1}\,M\,A_{ids})u_{ids} \\ \leq & \dot{V} \\ \leq & (C^{-1}\,M\,A_{ids})V + (C^{-1}\,M\,A_{ids})b_{ids} + (C^{-1}\,M\,A_{ids})u_{ids} \end{array} \quad (5)$$

where A_{ids} is computed from the linear regression and b_{ids} and u_{ids} are determined by the error analysis.

This model is simple, efficient to compute and worked fairly well for the toggle example. Section V-B describes some specific situations where the error term from this model was too large to complete the verification of the toggle and how we refined the model to achieve a successful verification.

Finally, we need a model for the input signal, ϕ, which is specified as satisfying a Brockett annulus. We obtained a candidate annulus by simulating the toggle with inputs of varying amplitude and frequency and observing the Brockett annulus satisfied by the toggle's output. We described this annulus by giving polygons for the inner and outer rings. Figure 6 shows the annulus that we used. We set the minimum low and high times for ϕ to 0.5ns. The circuit model uses the LP for the current bloated face to obtain bounds on ϕ. The model computes the center path for ϕ in this interval, computes the minimum, least-squares approximation of this path, and finds the worst-case errors for this approximation.

B. The Reachability Computation

As described above, we divided the reachability computation into four phases according to the four state transitions shown in Figure 3. Table I summarizes this analysis. We start each phase with the projectagon for the starting hyper-rectangle and note the bounding hyper-rectangle of the projectagon for the reachable region at the end of each phase. Note that the starting hyper-rectangle for each phase contains the ending hyper-rectangle of the previous phase, and the starting hyper-rectangle for the phase 1 contains the ending hyper-rectangle for phase 4. Thus, we've established an invariant set. Furthermore, the hyperrectangles for the four phases are pairwise disjoint. Thus, this invariant set has a period of two with respect to the clock input, ϕ.

Table I also lists the projection polygons that we used for each phase. These were chosen with two considerations. First, we chose projections that corresponded to logical dependencies between changing signals. Thus, in the first phase when z changes, we include z vs. zz and z vs. x (because the falling edge of z enables a rising edge of x). Second, we included at least one polygon for each variable to bound the resulting projectagon in all dimensions.

We now describe the verification of each of these phases in greater detail.

Phase 1: $(\phi = 0, x = 0, y = 1, z = 1) \rightarrow (\phi = 1, x = 0, y = 1, z = 0)$*:* In this phase, ϕ makes a low-to-high transition, and z goes from high-to-low. This phase starts with $\phi = 0.2$ volts at which point ϕ is already in region 2 of the Brockett annulus. This ensures that $\dot{\phi}$ is strictly positive. This phase includes the rising transition of ϕ (region 2 of the annulus) and the time that ϕ is high (region 3). We compute the union of the reachable regions for all times starting once ϕ has been high for the minimum required time. When this union computation reaches a fixpoint, we have the reachable set for the end of this phase. In the next phase, ϕ will enter region 4 of the Brockett annulus. We over approximate the annulus by lowering the minimum high value for ϕ to 1.6 volts (from 1.65) during this phase of the verification. This allows us to use a starting value of 1.6 volts for ϕ in the next phase which will ensure that $\dot{\phi}$ is strictly negative.

The linear model for $\dot{\phi}$ has large errors if the interval for ϕ is too large. Thus, we "sliced" the space into regions corresponding to 0.1 volt wide intervals for ϕ. Consider the scenario when Coho is working on a slice with $\phi \in [lo, hi]$ and let P_0 be the projectagon at the end of a time step. Coho divides the P_0 into a portion, P_{lo}, with $\phi \leq hi$ and a portion, P_{hi} with $\phi \geq hi$. To obtain P_{lo}, Coho intersects each projection polygon that includes ϕ in its basis with the $\phi \leq hi$ half-plane. Projection polygons in other bases are left unchanged. This ensures that $P_{lo} \subseteq P_0 \cap \{u \,|\, u_\phi \leq lo\}$. Coho computes P_{hi} for the timestep in the same manner. Because $\dot{\phi} > 0$ when ϕ is in region 2 of the Brockett ring, Coho will eventually reach a time step when P_{lo} is empty.

For each projection polygon basis, Coho computes the union of the polygons that it computed for this baseis for P_{hi} to obtain the projection polygon that it will use for that basis in the next slice of ϕ. It is straightforward to show that the union of two projectagons is contained in the projectagon obtained from the union of their projection polygons. Occasionally, the polygons for two consecutive time steps will be disjoint – this can occur when the projectagon barely crosses into the $\phi \geq hi$ half-space. In this case, Coho uses the bounding box of the union as a simple, over approximation.

The transistor model from Section V-A can produce large error bounds that include currents that flow against the drain-to-source voltage. These non-physical behaviours allowed by the model caused Coho to fail to verify the toggle. It is simple to show that the circuit model has an invariant that all node voltages are between 0 volts (i.e. ground) and 1.8 volts (i.e. V_{dd}) and that $xx \geq x$, $yy \leq y$ and $zz \leq z$. We modified Coho to allow the user-supplied model to provide such invariants. With these user-supplied invariants, Coho successfully computed the tight bounds on the reachable region at the end of the phase described above.

Phase 2: $(\phi = 1, x = 0, y = 1, z = 0) \rightarrow (\phi = 0, x = 1, y = 1, z = 0)$*:* In this phase, ϕ makes a high-to-low transition, and x goes from low-to-high. The verification proceeded in the same manner as for Phase 1.

Phase 3: $(\phi = 0, x = 1, y = 1, z = 0) \rightarrow (\phi = 1, x = 0, y = 0, z = 1)$*:* In this phase, ϕ goes from low-to-high, and all three of x, y, and z change their values, in the order $y \downarrow \rightarrow\ z \uparrow \rightarrow\ x \downarrow$. This was the most challenging phase to verify. As seen in Table I, we used ten projection polygons for this phase instead of six as were used in the other phases.

The greatest challenge arose because z can start its rising transition while y is still falling. To show that output, q of the

Start and end hyper-rectangle for each phase							
Phase	ϕ	x	y	z	xx	yy	zz
1, start	0.2	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 1.0]$	$[0.000, 0.100]$	$[0.000, 0.100]$
1, end	1.6	$[0.000, 0.002]$	$[1.790, 1.800]$	$[0.000, 0.014]$	$[1.788, 1.8]$	$[0.000, 0.004]$	$[0.000, 0.001]$
2, start	1.6	$[0.000, 0.100]$	$[1.700, 1.800]$	$[0.000, 0.100]$	$[1.700, 1.8]$	$[0.000, 0.100]$	$[0.000, 0.100]$
2, end	0.2	$[1.795, 1.800]$	$[1.758, 1.800]$	$[0.000, 0.043]$	$[1.795, 1.8]$	$[1.152, 1.736]$	$[0.000, 0.003]$
3, start	0.2	$[1.750, 1.800]$	$[1.750, 1.800]$	$[0.000, 0.050]$	$[1.750, 1.8]$	$[0.800, 1.800]$	$[0.000, 0.040]$
3, end	1.6	$[0.000, 0.001]$	$[0.000, 0.005]$	$[1.703, 1.800]$	$[1.785, 1.8]$	$[0.000, 0.002]$	$[0.843, 1.740]$
4, start	1.6	$[0.000, 0.100]$	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.8]$	$[0.000, 0.100]$	$[0.800, 1.800]$
4, end	0.2	$[0.000, 0.100]$	$[1.700, 1.800]$	$[1.700, 1.800]$	$[0.000, 1.0]$	$[0.000, 0.100]$	$[0.000, 0.100]$

Projection Polygons for each phase	
Phase	Polygons
1	x vs. xx, x vs. z, z vs. zz, z vs. xx, ϕ vs. y, ϕ vs. yy
2	x vs. xx, x vs. y, x vs. yy, y vs. yy, ϕ vs. z, ϕ vs. zz
3	x vs. xx, x vs. y, x vs. yy, y vs. yy, y vs. z, y vs. zz, z vs. zz, z vs. z, z vs. xx, ϕ vs. z
4	x vs. xx, y vs. yy, y vs. z, y vs. zz, z vs. zz, ϕ vs. z

TABLE I
REACHABILITY SUMMARY

toggle satisfies the same Brockett annulus as used for ϕ (see Section V-C), the transitions of z need to have relatively small rise and fall times. The time derivative of z depends on the values of ϕ, y, z and zz. We found that once ϕ was high (i.e. greater than 1.6 volts), it was helpful to slice on the value of y. We used 0.1 volt wide slides for y as it fell from 1.3 volts to 0.1 volts. We sliced z in the same manner but found that it was unnecessary to slice x.

For z to make its rising transition, zz must also rise. Otherwise, the current through the transistor between z and zz can dominate the current through the pull-up transistor for z. We found it surprisingly challenging to show that zz rises at an acceptable rate. The problem occurs when both z and zz have wide bounds including values close to ground for each. With wide bounds, the error term for the current through the transistor between z and zz can be large. At one extreme of this error bound, the current from z to zz is negative which keeps the lower bound for zz at ground. At the other extreme, the current form z to zz is large and positive which keeps z from rising.

We solved the negative current problem by adding a transistor model that simply determines the minimum and maximum drain-to-source current for the region around the current face. While this model has a large error-term, it never predicts a current of the wrong sign. We modified Coho to each compute forward time step twice: first using the least-squares current models and then using the min/max models. Coho then computes the intersection of the two projectagons that it obtains. Because each of the projectagons contains the actual reachable space, their intersection does as well. This preserves the soundness of Coho while significantly reducing the errors.

Finally, we introduced the improvements in the bloat calculation described in Section IV-B and polygon simplification described in Section IV-C. With these changes, we were able to verify this phase (and all four phases) of the toggle's operation.

Phase 4: $(\phi = 1, x = 0, y = 0, z = 1) \rightarrow (\phi = 0, x = 0, y = 1, z = 1)$*:* In this phase, ϕ makes a high-to-low transition, and y goes from low-to-high. The verification proceeded in the same manner as for Phases 1 and 2.

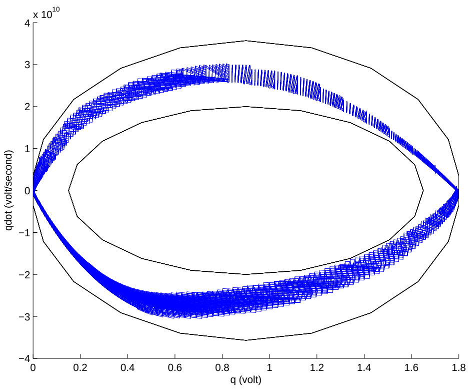


Fig. 6. The Brockett Annulus for q and $\dot{q}$

C. Verifying the Output Brockett Annulus

Thus far, we have ignored the q output of the toggle in our analysis – we simply included a load on z equal to the gate capacitance of transistors that drive q. We verified the operation of the inverter separately. To do so, we first constructed the Brockett annulus for the z output.

At each time step of the verification described above, we determined the reachable combinations of z and $\dot{z}$. We note that $\dot{z}$ is negative monotonic in z and positive monotonic in zz. Thus, the extremal values of z vs. $\dot{z}$ occur on the boundary of the z vs. zz projection. For each edge of the z vs. zz projection, Coho computes the linearized circuit model and uses this model to find the reachable combinations of z and $\dot{z}$. From these, we constructed a Brockett annulus that is satisfied by z and $\dot{z}$.

We then perform a separate reachability analysis for the output inverter. The input to this circuit is modeled by the Brockett annulus derived above, and we compute the reachable region of q vs. $\dot{q}$ as described above for z. Figure 6 shows the result; q clearly satisfies the constraints that we used for ϕ. Thus, these toggles can be composed to form an arbitarily large ripple-counter as desired.

VI. Conclusions

We have implemented a working version of the Coho algorithm for performing reachability analysis of circuits modeled by non-linear differential equations. We used Coho to verify a toggle flip-flop using a model that exposed all seven nodes of the circuit and used accurate (vendor provided BSIM-3) transistor models. We believe that our verification of a seven-dimensional, non-linear system is the largest such verification to date and shows the feasibility of using formal methods to verify digital designs at the circuit level.

Coho is slow, and our immediate work will be to address issues of performance. We are aware of obvious opportunities for improving the performance including more efficient algorithms for some of the critical operations, and re-implementing critical pieces of code in C instead of the current Matlab and Java implementation. Furthermore, there is abundant parallelism available in Coho, as each edge of each projection polygon can be processed independently during each time step. The multithreading capabilities of the Java part of the code should make this parallelism readily accessible and we will explore parallel implementations. We note that the memory requirements for Coho are relatively modest, and unlike many reachability tools, we don't expect memory to be a critical limitation in the near future.

In the process of verifying the toggle, we have identified numerous places where the over approximations of our models and algorithms could be reduced at a relatively low computational cost. We will explore these refinements. This should enable the verification of more complex circuits. While climbing up the ladder to higher-dimensional models is challenging, we note that it is also very rewarding. The fraction of interesting cell designs that can be modeled grows rapidly with dimension, and we expect that a tool that could verify circuits with ten to fifteen nodes would be adequate for most cells in a cell-library or specialized logic functions in a full-custom design. Once cells are verified with specifications that allow compositional reasoning, more traditional tools for discrete equivalence and model checking could be used to verify larger designs.

This first demonstration of verifying a circuit with a detailed, non-linear circuit model was far from automatic. Significant human expertise was required in using Coho, improving the reachability algorithms, modeling the circuits, and writing the specification. This is typical for the first demonstration of a new verification technology. We expect that as we improve Coho and apply it to a wider range of examples, we will also develop a more automated verification flow that will allow these verification techniques to be used by typical circuit designers.

References

[1] J. Yuan and C. Svensson, "High-speed CMOS circuit technique," *IEEE Journal of Solid-State Circuits*, vol. 24, no. 1, pp. 62–70, Feb. 1989.

[2] M. R. Greenstreet and I. Mitchell, "Integrating projections," in *Proceedings of the First International Workshop on Hybrid Systems: Computation and Control*, T. A. Henzinger and S. Sastry, Eds., Berkeley, California, Apr. 1998, pp. 159–174.

[3] ——, "Reachability analysis using polygonal projections," in *Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*. Berg en Dal, The Netherlands: Springer, Mar. 1999, pp. 103–116, LNCS 1569.

[4] R. Alur, T. Henzinger, and P.-H. Ho, "Automatic symbolic verification of embedded systems," *IEEE Transactions on Software Engineering*, vol. 22, no. 3, pp. 181–201, 1996.

[5] T. Henzinger, J. Preussig, and H. Wong-Toi, "Some lessons from the HyTech experience," in *Proceedings of the 40th Annual Conference on Decision and Control*. IEEE Press, 2001, pp. 2887–2892.

[6] T. A. Henzinger, "The theory of hybrid automata," in *Proceeding of the 11^{th} Annual IEEE Symposium on Logic in Computer Science (LICS'96)*, 1996, pp. 278–292.

[7] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi, "Algorithmic analysis of nonlinear hybrid systems," *IEEE Transactions on Automatic Control*, vol. 43, no. 4, pp. 540–554, Apr. 1998.

[8] E. Asarin and O. Maler, "On the analysis of dynamical systems having piecewise-constant derivatives," *Theoretical Computer Science*, vol. 138, pp. 35–65, 1995.

[9] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya, "What's decidable about hybrid automata?" *Journal of Computer and System Sciences*, vol. 57, pp. 94–124, 1999.

[10] G. Frehse, "PHAVer: Algorithmic verification of hybrid systems past HyTech," in *Proceedings of the Fifth International Workshop on Hybrid Systems: Computation and Control*. Springer-Verlag, 2005, pp. 258–273, LNCS 3414.

[11] R. Bagnara *et al.*, "Possibly not closed convex polyhedra and the parma polyhedra library," in *Proceedings of the International Symposium on Static Analysis*, 2002, pp. 213–229, LNCS 2477.

[12] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *Proceedings of the Fourteenth Conference on Computer Aided Verification*. Copenhagen: Springer, July 2002, pp. 365–370.

[13] O. Bournez, O. Maler, and A. Pneuli, "Orthogonal polyhedra: Representation and computation," in *Proceedings of the Second International Workshop on Hybrid Systems: Computation and Control*. Springer, 1999, pp. 46–60, LNCS 1569.

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

[15] B. Silva, K. Richeson, *et al.*, "Modeling and verifying hybrid dynamical systems using *checkmate*," in *Proceedings of the 4^{th} International Conference on Automation of Mixed Processes (ADPM 2000)*, 2000, pp. 323–328.

[16] 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.

[17] R. 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*, ser. Lecture Notes in Control and Information Sciences, H. Nijmeijer and J. M. Schumacher, Eds. Springer, 1989, vol. 135, pp. 19–30.

[18] T. S. Parker and L. O. Chua, *Practical Numerical Algorithms for Chaotic Systems*. New York: Springer, 1989.

[19] M. D. Laza, "A robust linear program solver for projectahedra," Master's thesis, Department of Computer Science, University of British Columbia, Vancouver, BC, Dec. 2001.

[20] M. R. Greenstreet, "Verifying safety properties of differential equations," in *Proceedings of the 1996 Conference on Computer Aided Verification*, New Brunswick, NJ, July 1996, pp. 277–287.