Faster Projection Based Methods for Circuit Level Verification

Chao Yan, Mark Greenstreet
CS Department, University of British Columbia
Vancouver, BC, V6T1Z4
{chaoyan,mrg}@cs.ubc.ca

Abstract— As VLSI fabrication technology progresses to 65nm feature sizes and smaller, transistors no longer operate as ideal switches. This motivates the verification of digital circuits using continuous models. Recently, we showed how such verification can be performed using projection based methods.However, the verification was slow, requiring nearly four CPU days to verify a nine-transistor toggle flip-flop. Here, we describe improvements to the reachability algorithms and optimizations of the software architecture. These produce a $15\times$ reduction in computation time and significant reductions in the overapproximation errors. With these changes, the same toggle flip-flop can be verified in a few hours, making formal verification a viable alternative to circuit simulation.

I. Introduction

Deep-submicron technologies simultaneously confront designers with transistor behaviors that require circuit-level models to produce working designs and 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. 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.

Reachability analysis plays a central role in formal methods. Forward reachability computes all states that are reachable from an initial state or set of states. Conversely, backward reachability computes all states that can eventually reach a specified state or set of states. Efficient reachability algorithms for systems with discrete models have enabled the practical application of model checking in digital design.

Circuit-level verification requires reachability with continuous state spaces, and practical verification requires efficient representation and manipulation of regions in these spaces. COHO [], the tool described in this paper, uses *projectagons* to represent reachable regions by their projections onto two-dimensional subspaces. As nearly all properties of systems with continuous models are formally undecidable [1, 7], COHO computes approximations of forward reachable sets. COHO guarantees that all approximations are over approximations. Thus, the analysis is sound but not complete for safety properties: COHO can fail to verify a correct circuit, but it will never claim to verify an incorrect design. Section II describes COHO in more detail.

Recently,we described the verification of a toggle flip-flop circuit using COHO. While this demonstrated the feasibility of using projectagon techniques for circuit verification, the verification of this simple circuit required nearly four CPU days. Section III describes improvements to COHO that make it much more practical tool:

- COHO make extensive use of linear programs. We present improvements to COHO's LP solver and related algorithms to dramatically speed-up COHO.
- We present improvements to the reachability algorithm that allow larger time steps, improved accuracy and fewer evaluations of the circuit model.
- We show how these changes fit within COHO's software architecture. COHO runs within MATLAB providing flexible and interactive verification and debugging of circuits. Java is used to implement complex data structures, and C is used to maximize performance of the linear programming codes.

In Section IV, we evaluate the impact of these changes. The new version of COHO runs $15\times$ faster than the original version with smaller approximation errors. In Section V, we conclude the paper and examine the practicality of formal techniques for circuit-level verification.

II. Coho

COHO represents reachable sets with projectagons. A projectagon is the high-dimensional 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, projectagons can represent non-convex objects. The high-dimensional object represented by a projectagon is the set of all points that satisfy the constraints of each projection. To ensure that the projectagon is bounded, each dimension of the full-dimensional projectagon must be in the basis of the two-dimensional subspace for at least one projection polygon. As an example, Fig. 1 shows how a three-dimensional object (the "anvil") can be represented by its projection onto the xy, yz, and xz planes.

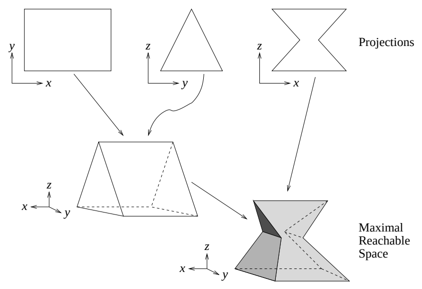


Fig. 1. A Three-Dimensional "Projectagon"

COHO utilizes the following properties of projectagons: the intersection of two projectagons is represented exactly by the intersection of their projection polygons; the union of two projectagons is over approximated by the union of their projection polygons; the convex hull of a projectagon is over approximated by the convex hulls of its projection polygons; and, in the absence of degeneracies, faces of a projectagon correspond to edges of its projection polygons.

To verify a circuit, we represent its dynamics as an ordinary differential equation:

$$\dot{v} \quad = \quad f(v, in) \tag{1}$$

where v is the voltage state vector; $\dot{v}$ is the time derivative of v; and in is a vector of inputs to the circuit. COHO computes reachability in a sequence of time-steps similar to those taken by a numerical integrator. At each time step, COHO computes an over approximation of the points reachable from each face of the current projectagon. To do so, it computes a linear program, $Av \leq b$, that contains the face, and a differential inclusion:

$$Fv + g - u \quad \leq \quad \dot{v} \quad \leq \quad Fv + g + u \tag{2}$$

where $Fv + g$ is a linear approximation of $f(v)$ in a neighborhood of the face, and u accounts for uncertainties in the model and the inputs and errors introduced by linearizing f. From these, COHO constructs a linear program (LP) whose feasible region contains all points reachable at the end of the time step by trajectories starting on the face. COHO the computes the projection of this feasible region back onto the subspace for the projection polygon with the edge that corresponds to this face. The union of all of the projected faces for a projection polygon gives the boundary of the polygon at the end of the time step. More details are given in [].

To ensure soundness, the differential inclusion for each face (see Eq. 2) must be valid anywhere that a trajectory starting on that face may reach within the timestep. In this section, we sketch the original algorithm, and Section III describes our refinements to this algorithm. In the original version, COHO constructed an LP for the convex hull of the current projectagon, and bloated it by relaxing every constraint by a predetermined amount, Δv to produce a LP LP_{bloat}. For each projection polygon edge, COHO intersected LP_{bloat} with constraints for an oriented rectangle with bloat Δv centered on the edge. The resulting LP, LP_{face} contained a neighborhood of all points within distance Δv of the face. COHO called the user provided model to obtain a differential inclusion that is valid in LP_{face}. COHO then solved two LPs for each variable in the model to determine bounds on $\dot{v}$ and used $(\Delta v)/\|\dot{v}\|_\infty$ as an upper bound for the time-step for the current step. The smallest such bound over all faces determined the size of the time-step that COHO took. COHO then used this time-step to compute the time-advanced projection polygons as described in the previous paragraph.

The original version of COHO successfully verified the toggle circuit. The program was divided into two pieces, a MATLAB process that provided an interactive interface. The top-level loops to iterate over each edge of each polygon to process each face was written in MATLAB as were the circuit models. To describe the circuits, we used the MSPICEpackage that provides a circuit simulation environment within MATLAB. This allowed us to use the same circuit description for simulation and verification. The Java process provided geometric operations and a robust LP solver. We had experienced that the LPs arising in COHO are sometimes very ill-conditioned. Our LP solver exploits the special structure of COHO's LPs, and uses arbitrary precision rational arithmetic when double-precision methods fail. This partition into a MATLAB component and a Java component provided a convenient point for logging and checkpointing, which is a great aid for debugging.

Although COHO verified the toggle, it was too slow, and we realized that we would have to improve the performance of COHO before applying it to other circuits. As described in the sketch above, COHO makes extensive use of LPs. To obtain acceptable performance we focused on improving the performance of the linear program solver. We also noted that the time-steps taken by COHO were often extremely small. By revising the bloating and time-step calculations, we reduced the amount of computation required at each step, increased the typical time-step size and reduced the amount of approximation error. We describe these changes in the next section.

III. Performance Improvements

This section first describes how we improved the performance of COHO's LP solver. We then describe how we revised the bloating and time-step calculation.

A. Faster LP Solves

COHO solves LPs when creating linear inclusion models for each face and when projecting faces back onto their projection planes at the end of each time step. As we will describe in Section B, we eliminated the LP solves from the time-step calculation. Although some LP solves can still be performed by the user-provide model code to determine linear inclusions with small errors, most LPs in the new version of COHO occur when projecting faces back onto their projection place. Thus, we will focus on this case.

All of the constraints in COHO's LPs are inequality constraints corresponding to the convex hulls of the projection polygons and constraints to describe the oriented rectangles of bloated edges. Thus, a COHO LP has the form: $Av \leq b$. Let $m, n \in \mathbb{Z}^+$ such that $A \in \mathbb{R}^{m \times n}$. The basic idea behind the projection algorithm is to solve LPs of the form

$$\max_{v \in \mathbb{R}^n} (\hat{x}\cos\theta + \hat{y}\sin\theta) \cdot v \text{ s.t. } Av \leq b \tag{3}$$

for all θ from 0 to 2π. where $\hat{x}$ and $\hat{y}$ are unit vectors for the basis variables of the projection. Of course, COHO does not need to solve Eq. 3 for every possible θ, it only needs to solve it for one θ for each edge of the projection polygon. COHO uses Simplex which works on the dual of Eq. 3:

$$\min_{u \in \mathbb{R}^{+m}} b \cdot u \text{ s.t. } A^T u = \hat{x}\cos\theta + \hat{y}\sin\theta \tag{4}$$

When COHO finds a solution to Eq. 4, it finds an optimal basis, $\mathcal{B}$, and $u = A_{\mathcal{B}}^{-T}(\hat{x}\cos\theta + \hat{y}\sin\theta)$. The critical value of θ is the one at which u acquires a negative element. Successive values for θ can be determined by a single linear system solve each. This is how COHO computes the projection of each face at the end of each time step.

When θ in Eqs. 4 and 3 is increased to force a pivot to the next edge of the projection, the standard form LP becomes infeasible. Traditional formulations of Simplex assume a feasible basis; thus, the original COHOrestarted the LP solver to establish feasibility for each edge of the projection of each face. In the absence of degeneracies, only a single pivot is required to re-establish feasibility. Accordingly, we modified our projection algorithm to try each column of A^T to determine if its introduction into the basis achieves optimality. This requires a single linear-system solve for each column tried which can be performed in $O(n)$ time due to the special structure of COHO's LPs.We found that this optimization works about 80% of the time which resulted in a significant improvement in performance. The rather high failure rate is because the prisms whose intersection forms the projectagon are orthogonal to each other, leading to a higher rate of degeneracy than for typical LPs.

The projection of a face at the end of a time step can have clusters of very closely spaced vertices separated by much larger gaps. These clusters arise from near degeneracies in the COHO LPs. To avoid a rapid growth in the number of vertices in the projection polygon, COHO performs a simplification step where the projection polygon is replaces by an enclosing polygon of smaller degree. Consequently, every vertex but one in a cluster will be discarded by the simplification process, but the projection algorithm expended a significant amount of computation time to determine these vertices. We avoided this extra work by enforcing a lower bound on the change of θ at each step of the projection algorithm.

Our new projection algorithm can skip over vertices if the normals of the consecutive polygon edges are nearly parallel. Thus, the polygon obtained from these the revised projection algorithm could be an under approximation which would violate the soundness requirement for COHO. Conversely, we can use each vertex from the projection algorithm to define a half plane, and construct the polygon defined by the intersection of these half-planes. The resulting polygon is an over approximation. COHO computes both polygons. If their areas differ by more than a preset tolerance (2%), COHO reverts to computing the exact projection polygon. Otherwise it uses the overapproximation.

Another consequence of the degeneracies in COHO LPs is that it can be numerically difficult to determine a favorable pivot. This can lead to pivots to infeasible vertices and cause Simplex to fail. The original COHO solved this problem by verifying each pivot using arbitrary precision rational arithmetic (APR). With the special structure of COHO LPs, this could be done with a single linear system solve requiring $O(n)$ arithmetic operations. Nevertheless, the APR calculations dominated the execution time of the solver.

The revised COHO uses ordinary double-precision arithmetic for each pivot. It then verifies that each pivot succeeded in reducing the cost function first by using interval arithmetic, and in the infrequent event that this fails, COHO uses APR. If the pivot failed to reduce the cost, COHO repeats the pivot step with APR. Likewise, at the end of the algorithm, COHO tests the optimality of the solution by verifying that it is feasible in both the primal and dual LPs, again using interval arithmetic first and APR if the result from the interval calculation is inconclusive. In this way, we obtain the certainty of APR while performing nearly all calculations using ordinary, double-precision arithmetic.

Finally, we note that while Java is an ideal language for prototyping, it does not achieve the performance of code written in C. Thus, once we had developed our linear programming codes in Java, we re-implemented the final versions of the LP solver and projection algorithm in C using the GNU MP package [5] for rational computation and Profil/BIAS [9] for interval computation. BIAS uses the directed rounding modes of IEEE 754 [4] compliant floating point units which make the switching of rounding modes quite efficient. We used Java's JNI interface. Thus, we preserved the MATLAB $\leftrightarrow$ Java interface of COHO, and this change had only a small, local impact on the software architecture.

B. Bloating and Time-Step Calculations

Choosing a good bloat amount at each time step is important to obtaining good performance. If the bloat is too small, then COHO will take very small time steps resulting in in long execution times and larger over approximations from the projection errors. Conversely, if the bloat amount is too large, then the non-linearity errors (u in Eq. 2) will be large, causing another kind of over approximation and small time steps. In the original COHO algorithm, the time steps were nearly always much smaller that what would actually be safe for the given bloat. This is because the ℓ_∞ norm for the derivative used in calculating the time step (see Section II) is usually very pessimistic. When edges were advanced, their projections at the end of a time step would lie well inside the bloat region. In the original COHO, we compensated for this by observing how much of the bloat had been used at the end of a time step and repeating the time advance operation with a correspondingly larger step. In spite of doubling the number of projection computations, this sped-up the original COHO.

The present version of COHO completely discards the phase of computing a time step for a given bloat. Instead, at the end of each time step, COHO computes the bloat and time step that it should have used. It uses these to set the time step and bloat for the next step. COHO also checks at the end of each step that the estimated bloat was sufficient for the estimated time step. If not, COHO updates the bloat amount and/or step size and repeats the time step. In addition to enabling larger time steps, COHO computes each step in less time because it has eliminated the step-size calculation phase and the projection phase is only performed once.

In the original COHO, each variable is bloated equally in both the positive and negative directions and all variables were bloated equally. In digital circuits, a few signals will be in transition at any given time and the others will be relatively stable. This results in excess bloating. To achieve an acceptable step size, the bloat for fast changing signal must be relatively large. When the same bloat is used for all variables, the bloat for slow changing signals is excessive, leading to much larger error terms in the differential inclusion than needed. Likewise, when a signal is changing, it is generally either clearly rising or clearly falling. Thus, a large bloat is only needed in one direction, allowing the total bloat for these variables to be reduced by nearly a factor of two.

We implemented asymmetric and anisotropic bloating in the new version of COHO. Asymmetric bloating allows the positive and negative bloats for a variable to be different. Thus, bloating can adapt to the direction in which a signal is making a transition. Anisotropic bloating allows each variable to have its own bloat amount. Thus, bloating can adapt according to which variables are changing and which are stable. COHO determines these bloat amounts at the end of each time step for use in the next time step. This approach allowed a significant increase in the typical step size. As an added benefit, the smaller total bloat reduced the error terms in the differential inclusion, allowing COHO to compute much tighter bounds on the reachable regions.

C. Summary

In summary, we modified the COHO's LP solver and its bloating and time-step calculations to address issues of runtime and approximation errors. We eliminated the need to restart the LP solver for each vertex computed by the projection algorithm; we avoid the computation of closely spaced vertices that would just be eliminated by the simplification step later; we allowed most calculations to be performed using native double precision arithmetic while retaining the robustness and soundness of arbitrary precision, rational arithmetic; and we wrote a C implementation of the linear program routine that utilizes hardware supported directed rounding to obtain an efficient LP solver. Many of these optimizations are present in

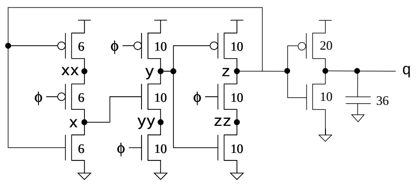
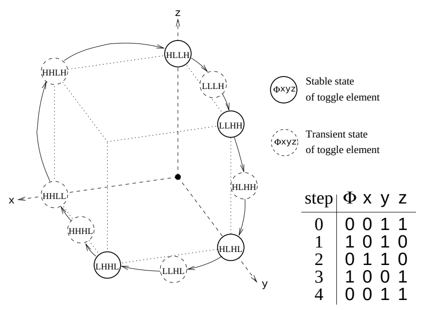


Fig. 2. Yuan and Svensson's Toggle Circuit



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

commercial LP solvers such as cplex [8]. However, such off-the shelf codes do not provide the operation of projecting the feasible region of an LP onto a subspace which is the dominant use of LPs in COHO.

We increased the typical step size and reduced bloating errors by using the results of each time step to estimate the step-size and bloat amounts for the next time step. We implemented asymmetric and anisotropic bloating to account for the direction of signal transitions and the fact that at any given time, many signals are relatively stable. The next Section evaluates these optimizations and quantifies their benefits.

IV. EVALUATION

To evaluate the impact of the algorithmic changes described in the previous section, we repeated the verification of the toggle circuit. This section describes the toggle verification and then evaluates the new version of COHO showing how the new version achieves a $15\times$ improvement in performance compared with the original COHO.

A. Verifying the Toggle

Fig. 2 shows the toggle circuit from [10] and Fig. 3 shows the state transition diagram starting from the state where z is high when ϕ is low. 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

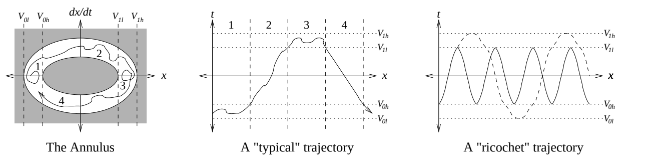


Fig. 4. Brockett's Annulus

one toggle can drive the clock input of another to implement a ripple counter. The toggle has seven nodes (we verify the the output inverter separately); thus, the state space of this system is $\mathbb{R}^7$.

To specify the desired continuous behavior of the toggle circuit, we use the Brockett annulus construction [2] shown in Fig. 4. Region 1 specifies a logical low signal: the level of the signal is constrained but its derivative may be either positive or negative. 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.

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. We add constraints for the minimum time that ϕ must remain in region 1 before entering region 2, and likewise for region 3; these minimum low- and high-times are readily satisfied by real circuits and necessary for successful verification. This Brockett annulus 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 specify a hyper-rectangle for the initial region and regions for the end of each state transition from Fig. 3. The final hyper-rectangle is contained in the initial hyper-rectangle, demonstrating the desired invariant. The regions at the end of each transition are disjoint, which shows that the toggle has a period twice that of the clock. Finally, we then show that for all reachable states, q, the output of the inverter, satisfies the same Brockett annulus constraints as we required for the clock input, ϕ. This shows that toggles can be composed to form a ripple-counter.

B. Performance

We verified the toggle circuit using the original version of COHO and COHO with the modification described in Section III. We ran both versions on a 3 GHz Xeon dual-core processor with 2Gbytes of memory. Although COHO is partitioned into separate MATLAB and Java processes, in both implementations, only one process is active at a time. Thus, only one core was used. We used the time function of the bash shell and the Java's nanoTime to measure runtimes. We acknowledge that nanoTime measures elapsed time, rather than CPU time. We used the Unix utility vmstat to verify that our processes had negligible paging activity and always had use of the CPU. Thus, elapsed time and CPU time should be very nearly the same.

C. The Linear Program Solver

We ran both versions of the face projection algorithm on 1219 example projections randomly selected from the toggle verification. The original algorithm had a run time of 570 seconds. When we modified the algorithm to skip over clusters of closely spaced vertices, the run-time dropped to 340 seconds. The number of LPs solved dropped from 18305 to 8923, roughly a 50% decrease which corresponds the decrease in run time. We set the error tolerance (maximum difference in area between the over- and under-approximated projections) to two percent. We note that more that 90% of the projections had errors less than 0.1%. Fewer than 2% of the problems violated the error bound and required running the exact algorithm.

We then modified the LP solver in the projection algorithm to check for the case that the new optimal basis is a single pivot from the old, now infeasible, one. Of the remaining 8923 LPs, 1219 are for finding the first vertex of the projection polygon. Of the remaining 7619 LPs, COHO solves 3862 in a single pivot and finds a feasible basis in one pivot for 1773 more. Only 1984 require restarting the LP solver from the beginning. With this modification, the time drops to 127 seconds, and additional 62% reduction in the run time.

Finally, using the C version of the LP solver reduces the time to 88 seconds, a savings of just over 30%. This disappointing result is due to overhead in the JNI and problems with incompatibilities between the Profil/BIAS interval arithmetic package and the JVM. We hope to solve these problems soon. Overall, the LP solver is now 6.4% times faster than the original. In the original implementation of COHO, the LP solver accounted for 75% of the total runtime; thus, we've achieved a net speed-up of $2.4\times$.

D. Improved Bloating and Time-Step

We used part of the third state transition of the toggle to evaluate the new bloating and time-step calculations. This segment took 570 minutes in the original COHO, and the new version completed the same segment in 39 minutes. As described above, the improvements to the LP solver account for a factor of 2.4 of this speed up. The remaining factor of 6 comes from the changes to the bloating and time-step calculations.

The original version of COHO required 186 time steps to complete the verification of this phase. With the new algorithm, this decreases to 67 steps. Thus, the average time-step is a factor of 2.8 larger in the new version which gives a corresponding speed-up for verifying this phase. As shown in Fig. 5 show the improvement in step-size in more detail. At the beginning of the phase, all three of x, y and z are stable. When the clock input, ϕ is sufficiently high, nodes y and yy start to fall, triggering the rising transition on z which then triggers a falling transition on x. Thus, there is a large amount activity

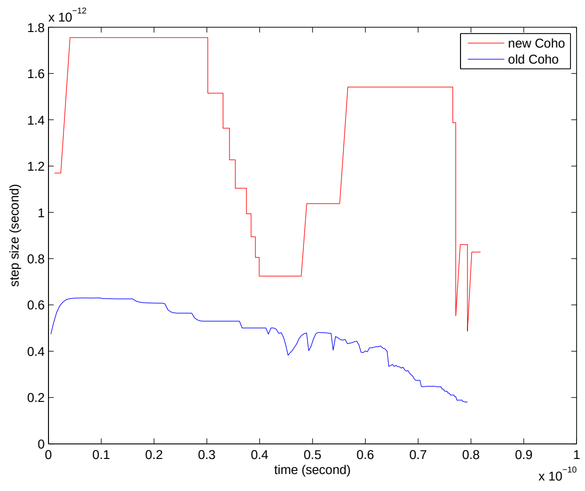


Fig. 5. Step size of new/original COHO

during the middle of this phase, and Fig. 5 shows how the new version of COHO adapts and decreases its step size during this activity. At the end of the cycle, the nodes of the toggle are converging to their limit values, and the new version of COHO returns to taking larger steps. In contrast, the step size for the original COHO remains small throughout its verification of this phase.

The computation of each step is also more efficient because the new COHO only performs the face projections once, and the bloat amount and step-size calculations are a fast calculation following the projections. On the other hand, the new COHO may have to repeat a time step if the estimated bloat is too small (or, equivalently, the step-size is too large). We observed that of the 67 time-steps performed to verify the third phase of the toggle operation, 38 succeeded and 29 failed and required revising the time step and for 11 of those 29 revising the bloat amounts. This failure rate, 43%, seems high. However, when we revised the algorithm parameters to reduce the number of failures, we found that the step-size decreased even faster. The overall effect was to increase the total run-time. We plan to further investigate the trade-offs in making step-size and bloat amount estimates in our future work.

Finally, the new algorithm significantly reduces the over-approximation errors. The average length of an edge of the bounding hyper-rectangle at the end of the phase decreased by 48%.

V. CONCLUSIONS AND FUTURE WORK

We have presented performance improvements to COHO that have enabled a $15\times$ reduction in run-time. These reductions came from improvements projection algorithm that computes the image of the feasible region of a linear program onto a projection plane and better calculation of the step-size for the integration algorithm and the bloat amounts that are required to ensure that the reachability calculation is sound. This allows the verification of the toggle circuit to be completed in under 10 hours.

There are many areas for future work. First, we would like to compare our verification result and run-time with those from other verification tools. Unfortunately, the other tools that we are aware of have not verified a seven-dimensional circuit such as the toggle. CheckMate [6] and PHAVer [3] have been used to verify a simple tunnel-diode oscillator and an idealized Sigma-Delta modulator. Much earlier, Kurhan and MacMillan verified an arbiter circuit with four nodes. COHO appears to be able to verify significantly larger designs than these other tools. An important topic for further research is to apply COHO to more complicated circuits such as flip-flops and pre-charged logic gates. We would like to be able to automatically verify typical cells in a standard cell library.

We also know that there are further areas of improvement possible for COHO. We are investigating why the step-size and bloat estimates often fail, and we believe that we can improve our circuit models to further reduce the over-approximations. We note that the computations performed by COHO for each projectagon face are independent. This suggests that additional performance gains should be possible with by exploiting this parallelism.

ACKNOWLEDGEMENTS

REFERENCES

[1] Eugene Asarin and Oded Maler. On the analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–65, 1995.

[2] R.W. Brockett. Smooth dynamical systems which realize arithmetical and logical operations. In Hendrik Nijmeijer and Johannes M. Schumacher, editors, *Three Decades of Mathematical Systems Theory: A Collection of Surveys at the Occasion of the 50th Birthday of J. C. Willems*, volume 135 of *Lecture Notes in Control and Information Sciences*, pages 19–30. Springer, 1989.

[3] Goran Frehse. PHAVer: Algorithmic verification of hybrid systems past HyTech. In *Proceedings of the Fifth International Workshop on Hybrid Systems: Computation and Control*, pages 258–273. Springer-Verlag, 2005. LNCS 3414.

[4] David Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–48, March 1991.

[5] T. Granlund. *GNU MP: The GNU Multiple Precision Library*. The Free Software Foundation, Inc.: Boston, 2002.

[6] Smriti Gupta, Bruce H. Krogh, and Rob A. Rutenbar. Towards formal verification of analog designs. In *Proceedings of 2004 IEEE/ACM International Conference on Computer Aided Design*, pages 210–217, November 2004.

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

[8] ILOG CPLEX Inc. Cplex. http://www.ilog.com/products/cplex/.

[9] O. Knuppel. Profil/bias: a fast interval library. *Computing*, 53:277–287, 1994.

[10] Jiren Yuan and Christer Svensson. High-speed CMOS circuit technique. *IEEE Journal of Solid-State Circuits*, 24(1):62–70, February 1989.