Verifying an Arbiter Circuit

Chao Yan and Mark R. Greenstreet

Department of Computer Science
University of British Columbia
{chaoyan, mrg}@cs.ubc.ca

***Abstract*—This paper presents the verification of an asynchronous arbiter modeled at the circuit level with non-linear ordinary differential equations. We use Brockett's annulus to represent the allowed families of continuous waveforms for input and output signals and show that the metastability filter of the arbiter can be understood as a "Brockett annulus transformer." Improvements to the Coho verification tool are described that reduce the over approximation errors when working with non-convex reachable regions. The verification shows that the arbiter observes a four-phase handshake protocol with its clients and maintains mutual exclusion. We also show several liveness properties including bounded time response to uncontested requests and that grants are issued fairly.**

Keywords: Arbiters, formal verification, metastability dynamical systems.

I. Introduction

Deep submicron designs present designers with increasing challenges. On the one hand, correct circuit operation depends on properly accounting for a myriad of effects including leakage currents, crosstalk, power-supply noise and random parameter variations. On the other hand, the large device count enables the implementation of complex systems on chip, requiring designers to rely on higher levels of abstraction. Formal methods can help address these challenges by verifying proper behaviors at low-levels of abstraction and ensuring that the abstractions of these analog behaviors to discrete ones are sound.

This paper presents a case study of such verification: we verify the correct operation of an asynchronous arbiter. We start with a specification of a discrete arbiter including that it maintains mutual exclusion, handshakes properly with its clients, fairness, and bounded time response to uncontested requests. Verifying the arbiter raises interesting issues because the clients may make requests concurrently, and the arbiter may take arbitrarily long to respond to such requests due to metastability. We describe an approach for creating abstraction mappings from continuous waveforms to discrete sequences. We show how this framework allows us to describe the allowed continuous inputs to the arbiter, including signals with varying rise and fall times and arbitrary relationships between the assertions of requests by the two clients. Section II presents the specification of the discrete arbiter, abstraction mappings between continuous and discrete state spaces, and the arbiter circuit that we use as an example. We then summarize previous work on arbiters in circuit level verification in Section III.

We verified the arbiter using the Coho verification tool. Coho computes over approximations of the reachable space to guarantee the soundness of its results. We found that some of these approximations became so large that they prevented successful verification of the arbiter. This is mainly because of the non-convexity of the reachable space for the arbiter. While Coho was capable in principle of representing non-convex objects, in practice, Coho's approximations often filled-in these non-convexities. Section IV shows how we modified Coho to reduce these over approximations, and thus enable Coho to compute non-convex regions that bound the reachable space for the arbiter and thereby verify the arbiter's correctness.

Section V presents the actual verification of the arbiter, and Section VI summarizes these results and identifies areas for further research.

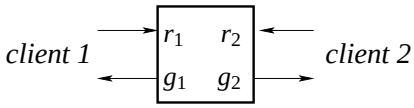


Fig. 1. An Arbiter

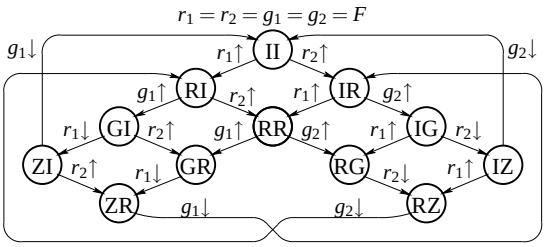


Fig. 2. Transition Diagram for a Discrete Arbiter

II. Arbiters

An arbiter provides mutual-exclusive access to a resource for some set of clients. In this paper, we consider an asynchronous arbiter with two clients as shown in Figure 1. The two clients interact with the arbiter using a four-phase handshake protocol: client i raises r_i to request the privilege; the arbiter raises g_i to grant client i the privilege; when the client is done with the privilege, it lowers r_i; and finally the arbiter lowers g_i to complete the handshake. The arbiter must guarantee mutual exclusion; in other words, signals g_1 and g_2 may not both be high at the same time.

A. Discrete Arbiters

We first present a specification for arbiters where behaviors are modeled as discrete sequences of discrete values. In particular, we specify safety properties with a transition diagram,

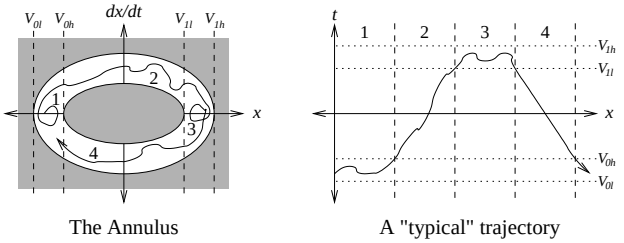


Fig. 3. Brockett's Annulus

and then state additional liveness and fairness requirements. Figure 2 shows the transition diagram. Each state has a two-letter label: the first letter indicates the state of the handshake between client 1 and the arbiter; and the second letter denotes the state of the handshake with client 2. In particular,

> "I" indicates idle: both the request and grant are false;
> "R" indicates requesting: the request signal is true, but the arbiter has not yet asserted the corresponding grant;
> "G" indicates granted: both the request and grant are true;
> "Z" indicates resetting: the request signal is false, but the arbiter has not yet lowered the corresponding grant.

Signals r_1 and r_2 are inputs. If either of these signals make a transition that is not in the diagram, that is a *failure of the environment*; in other words, the client for that input has violated the handshake protocol, and after the specification places no restriction on the behavior of the arbiter. In other words, diagram has implicit edges for all such events to a sink state for environment failures. Conversely, if either g_1 or g_2 make a transition that is not in the diagram, that is a *failure of the circuit*, and the diagram has implicit edges for all such events to a sink state for circuit failures. Thus, the arbiter verification problem includes showing that the circuit failure state is unreachable. By inspection of the transition diagram, this implies that the arbiter follows the handshake protocol and guarantees mutual exclusion.

We also want to establish liveness properties for the arbiter. In particular, we require that any uncontested request is eventually granted, and that once a client drops a request, the grant for that client will eventually be withdrawn. We do this by requiring that no trace can end in states RI, IR, ZI, ZR, IZ or RZ. Obviously, it would be desirable if the arbiter were guaranteed to eventually issue a grant when contested requests occur; i.e., that state RR cannot be terminal. It is well known that a real arbiter cannot satisfy this requirement along with the safety requirements described above (see, for example, [1]). Thus, here we do not give a liveness requirement for traces with contested requests. Finally, we require that the arbiter is fair: if client *i* has made a request, the other client can only receive a finite number of grants before the client *i* is granted. In fact, we will show that the arbiter considered in this paper issues at most one grant to one client while the other is waiting.

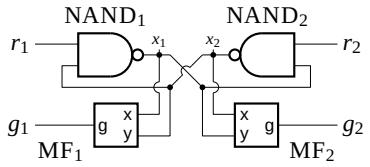
B. Continuous Arbiters

To specify the continuous behavior of the arbiter, we need an abstraction mapping from the continuous model to the discrete one. In the continuous model, signals will be modeled as continuous functions of time, i.e., as functions from $\mathbb{R}^+$ (time) to $\mathbb{R}$ (voltage or some other physical quantity). We assume that the circuit is modeled as a system of ordinary differential equations (ODEs) and that the time derivatives of all signals are defined and bounded everywhere along any trajectory.

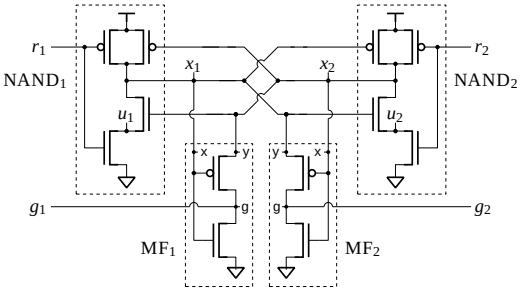
Our abstraction is based on the Brockett annulus construction [2] depicted in Figure 3. When a variable is in region 1, its value is constrained but its derivative may be either positive or negative. When the variable leaves region 1, it must be increasing; therefore, it enters region 2. Because the 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. This provides a topological basis for discrete behaviors – the hole in the middle of the annulus forces transitions to be unambiguous. A signal cannot meander to some value between the low and high values and then return to where it came from without making a complete transition. 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 3). 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. We also add constraints specifying the minimum low and high times for signals, i.e., the minimum duration of sojourns in regions 1 and 3.

Brockett annuli provide a basis for mapping continuous trajectories to discrete sequences. We will 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. Because Brockett annuli imposes minimum rise and fall times for signals, the number of rising and falling transitions is countable. To obtain a discrete sequence from a continuous trajectory, we "sample" the trajectory at the times when any signal transitions from region 1 to region 2, or from region 3 to region 4. At each such sampling time, t_s, we map a signal to "true" if its right limit at time t_s is in region 2 or 3 and to false if its right limit is in region 1 or 4. These limits exist by our assumption that the derivative function is defined everywhere along any trajectory.

Note that a Brockett annulus describes an entire family of trajectories. Given any trajectory, $x(t)$, with a trajectory 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. Of course, such an annulus also includes trajectories that will never occur during actual operation. Thus,



(a) Gate-level schematic for an arbiter.



(b) Transistor-level schematic for the arbiter above.

Fig. 4. An Arbiter Circuit

this abstraction is sound in that false positives are excluded, but false negatives could, in principle, occur. In our experience, the Brockett annulus abstraction has not been a cause for false negatives; these arise instead due to the over approximations in the reachability computation as described in Section IV.

Verification that an ODE model for a circuit satisfies a discrete specification proceeds in four steps:

1) Give a Brockett annulus specification for each continuous signal.
2) Compute an invariant set that contains all reachable trajectories of the continuous system. Here, we assume that all inputs to the circuit satisfy their Brocket annuli, and that their discrete abstractions do not produce environment failures.
3) Verify that each signal produced the circuit satisfies its Brockett annuli.
4) Verify that the discrete abstractions for all trajectories in the invariant step computed in step 2 satisfy the discrete specification.

We apply this process to an asynchronous arbiter in the remainder of this paper. First, we describe the circuit that we will verify.

C. An Arbiter Circuit

Figure 4 shows the schematic for a commonly used arbiter circuit. The arbiter consists of two parts: an R/S latch implemented by a pair of cross-coupled NAND gates, and a "metastability filter" for each grant output. When both request inputs are low, both NAND gate outputs are driven high, which in turn causes both grant outputs to be driven low. If client i makes an uncontested request, then x_i goes low in response to r_i going high which in turn causes g_i to go high. At this point the RS latch is set, and a transition on the other request input will have no effect until r_i goes low.

If both requests go high at roughly the same time, then metastability may occur. In this case, both x_1 and x_2 drop to an intermediate voltage that is the unstable equilibrium for the feedback cycle through the two NAND gates. The arbiter may remain in such a state arbitrarily long although the probability of unresolved metastability drops exponentially with time for a properly designed arbiter. Propagating such intermediate values back to the clients could cause them to malfunction. Thus, the arbiter includes *metastability filter* circuits to ensure that the grant outputs make clean transitions even if the NAND-gate outputs do not.

The metastability filter is a modified inverter. Consider circuit MF_1. The gates of the transistors in the inverter are connected to x_1. Unlike a traditional inverter, the source of the PMOS pull-up is connected to x_2. With this configuration, the pull-up transistor remains in cut-off until x_1 is at least the PMOS threshold voltage below x_2. This is to prevent g_1 from moving any significant amount above ground until client 1 has clearly won the arbitration.

In our framework for mapping continuous trajectories to discrete sequences, the metastability filter can be understood as a "Brockett annulus transformer." If the arbiter is correctly designed, then x_1 and x_2 will satisfy a Brockett-annulus specification, but the lower limit for region 3 (logically high signals) must be below the metastable voltage. This is incompatible with the input requirements of typical logic elements. The metastability filter takes as inputs signals with these shifted Brockett-annuli, and outputs signals that satisfy the requirements of typical logic circuits.

III. Related Work

Metastable behavior in digital circuits has been studied since Chaney and Molnar's original paper on synchronizer failures [3]. Hurtado [4] analyzed metastability from a dynamical systems perspective. Seitz [5] gives a nice introduction to metastability issues, and Marino [6] provides a fairly comprehensive treatment.

Arbiters have also been studied from a formal verification perspective. Kurshan and McMillan [7] use the arbiter from [5] as their main example in proposing a way to verify digital circuits modeled by differential equations. Their arbiter is the nMOS counterpart of the CMOS design presented in Section II-C. They formulated the verification problem in terms of language containment. To model the continuous behavior of the circuit, they divided the possible values for each continuous state variable into 10 to 20 intervals, and computed the set of reachable hyper-rectangles using such a grid. This is similar to the d/dt tool [8], [9]. Although the total number of possible hyper-rectangles is large, Kurshan and McMillan used COSPAN to construct the reachable space, and the next hyper-rectangle relation is only computed for reachable hyper-rectangles. Unlike our Brockett annulus approach for specifying signal transitions, Kurshan and McMillan model the inputs as making instantaneous transitions. These transitions were allowed at arbitrary times that satisfied the handshake protocol.

Mendler and Stroup [1] gave a formal specification for a continuous system to implement an arbiter. They used a dynamical systems theory argument to show that the specification is unsatisfiable by any continuous system. Mitchell

and Greenstreet [10] used measure-theory based arguments to show that Mendler and Stroup's specification is satisfiable if the liveness requirement is relaxed to an "almost-surely" version when concurrent requests are made.

Martin [11] gave a delay insensitive construction for implementing a fair arbiter from unfair ones. The "delay insensitivity" means that it functions correctly for any bounded delays in its components or wires. He uses the circuit from Section II-C as his "unfair" arbiter. We show in Section V-C that this arbiter circuit is, in fact, fair. Thus, the extra hardware and delay of Martin's construction is unnecessary if this circuit is used as an arbiter.

Formal methods for verifying analog and mixed-signal designs have received intense attention in the past five years. We summarize some of this work here noting that a comprehensive survey is in [12]. Frehse implemented *PHAVer* [13] which provides more efficient and robust implementations of the HyTech algorithms [14]. *Checkmate* [15] 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 [8] 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 [16], i.e. finite unions of full-dimensional, fixed size hyper-rectangles, and approximates the reachable state using numerical integration and polyhedral approximation. Finally, Al-Sammane *et al.* have applied symbolic rewriting and bounded-model checking techniques to several analog verification problems [17]. Each of these tools have been used to verify several analog circuits with low-dimensional state spaces including a tunnel diode oscillators, an VCOs and Sigma-Delta modulators [9], [18]–[20].

IV. Coho

We used the Coho tool to verify the arbiter circuit presented in Section II-C. This section first briefly describes how Coho computes over approximations of the reachable space for circuits modeled by ODEs. It then describes the improvements that we made to Coho in order to verify the arbiter. These changes were required to reduce approximation errors for the highly non-convex reachable regions that arise in the operation of the arbiter.

A. Coho Overview

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 by manipulating the individual polygons, avoiding any need to explicitly construct high-dimensional objects. Furthermore, projectagons can represent non-convex objects; this property is essential for the verification of the arbiter.

Coho's reachability algorithm has been described previously [21]–[24]; we summarize it here. Coho computes reachability through a sequence of time steps. In each time step, Coho computes a projectagon that contains all reachable points at the end of the time 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 first computes the LP for the intersection of the convex hulls of all of the projection polygons and then expands it outward by an amount Δ. Let LP_Δ denote this LP. Then, for each edge, Coho constructs a linear program (LP) whose feasible region contains the face corresponding to the edge. This LP is simply the intersection of LP_Δ with the linear constraints that describe the edge. Coho then computes an LP for the "bloated face" whose feasible region contains a small neighborhood around the face. Let $\mathrm{LP}_{neighborhood}$ denote this LP. The circuit model then produces a *linear differential inclusion* that is valid in the feasible region of $\mathrm{LP}_{neighborhood}$; these inclusions have the form:

$$Ax + b - u \quad \leq \quad \dot{x} \quad \leq \quad Ax + b + u \quad . \tag{1}$$

Where A is a matrix; b is a vector; $Ax + b$ is a linear approximation of the circuit model for the neighborhood of the face; and u is vector that upper bounds the approximation error. Coho finds the maximum magnitude of the derivative, $\dot{x}$, in the feasible region of $\mathrm{LP}_{neighborhood}$. Let D denote this magnitude. Then Δ/D is an upper bound on the valid size for a time step using this bloat. Coho uses the differential inclusion computed above and the LP for the face to construct an LP that contains all points reachable from the face at the end of the time step. This reachable space for this LP is then projected back to the subspace for the original projection polygon. The union of these projections for each edge of the original polygon gives the boundary of the polygon at the end of the time step.

Coho's method of turning an edge into an LP, advancing the LP by the differential inclusion, and projecting it back down tends to produce a dramatic increase in the number of projectagon edges at each time step. Thus, Coho performs polygon simplification at the end of each time step. This simplification step produces a polygon with a smaller number of vertices that contains the polygon obtained from the projection operations.

B. Improvements to Polygon Simplification

Coho has two methods for simplifying polygons: it can delete a vertex at a concave corner, and it can replace a pair of vertices at consecutive convex corners with a single vertex. Both transformations produce a polygon that strictly contains the original. In earlier versions of Coho, the cost of these operations were measured in terms of the increase in the area of the polygon and the increase in the area of its convex hull. This metric took into account that over approximating the convex hull can introduce over approximations in the reachability calculations for all edges, and not just for part

of a particular polygon. Thus, Coho preferentially eliminated concavities in the projection polygons.

When verifying the arbiter, we observed that reachability computations often started with convex regions that should evolve into concave ones due to the dynamics of the ODE model. However, these concavities tend to be initially very small, and Coho encounters a very small area cost to fill them in. Unfortunately, this prevented the projectagons from evolving highly non-convex shapes and prevented verification of the arbiter. We also noted that reducing the number of polygon vertices in the convex hull is more important for performance than eliminating concave vertices. The reason for this is that Coho spends a large percentage of its time solving LPs. Simplifying the convex hulls reduces the number of constraints in these LPs and allows them to be solved more rapidly.

Our solution was to simplify the polygon and its convex hull separately. The earlier version of Coho simplified each polygon and then computed their convex hulls. Now, Coho computes the hulls before simplification. This allows the polygons and the hulls to be simplified using objective criteria that are appropriate for each case. With this change, we obtained a small speed-up in Coho and, more importantly, projectagons can now evolve into concave shapes.

C. Interval Closure

As described above, earlier version of Coho derived an LP for each projectagon face by intersecting the constraints for the face's edge with the bloated convex hulls for each of the other projection polygons. If these polygons are non-convex, then this can produce a large over approximation of the face. Our solution is to perform an interval closure calculation. We can view each projection polygon edge as defining interval bounds for the two variables of the projection. The closure algorithm then applies these intervals to other polygons that include one of these variables in their basis to obtain bounds for other variables. This process continues until no further tightening of the interval bounds is possible. The algorithm is simple, fast and greatly reduces approximation error when the projection polygons are non-convex. However, some care must be taken to preserve the soundness of the reachability algorithm.

The problem is that although all extremal trajectories originate from faces of the projectagon, an extremal trajectory for one projection may emanate from a face whose edge only appears in another projection. Figure 5 shows an example. When edge AB of the (x,y) projection polygon is moved forward, the edge itself gives intervals of $[x_1,x_2]$ and $[y_1,y_2]$ for x and y respectively. The (x,z) projection then yields the interval $[z_1,z_2]$ for z. Now, consider what happens if point A (equivalently P) moves to the left by distance Δ in the current time step and point Q moves to the right by Δ. If interval closure were applied naïvely, then at the end of the time step, trajectories from point Q could reach points with larger x values that any point admitted by the end-of-timestep (x,y) projection polygon.

Figure 6 shows pseudocode for our interval closure calculation. The key idea is to bloat the bounding rectangle

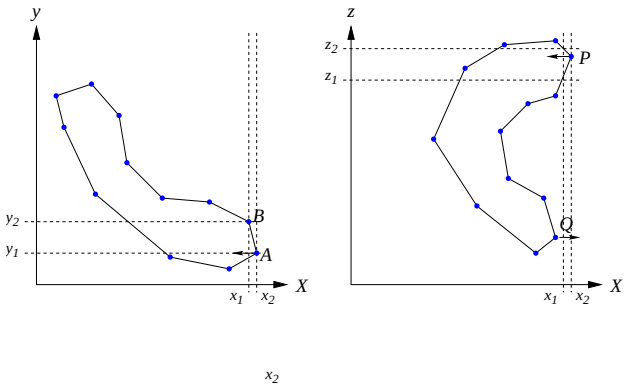


Fig. 5. Interval closure example.

```
HyperRectangle IntervalClosure(Edge e, ProjectionPolygon p, Real Δ) {
    /* e is an edge of polygon p.
        Δ is the bloat amount for the current time step.
        Return the hyper-rectangle of interval closure bounds
            for e for the current time step.
    */
    Let r be the oriented rectangle that contains all points within
    distance 2Δ of e by the ℓ∞ metric.
    Let q be the intersection or r and p.
    Let b be the bounding box of q.
    Let h0 be the hyper-rectangle obtained by interval closure
        starting with b and using all of the other projection polygons.
    Let h = bloat(h0, 2Δ). return(h).
}
```

Fig. 6. Computing interval closure for edges.

for each edge by Δ, intersect the resulting rectangle with the edge's projection polygon, and use the bounding box for this intersection as the starting point for the interval closure calculation. Coho uses the intersection of $\mathrm{LP}_{neighborhood}$ and h to describe region where the differential inclusion for the current edge must hold. Likewise, Coho uses the intersection of h and the current edge to obtain an LP that contains all points on the current face.

To establish the soundness of the interval closure method from Figure 6, we consider a projectagon, and assume that one of the projections has the basis (x,y). We show that all points reachable from the projectagon by the end of the timestep are contained in the (x,y) projection polygon at the end of the timestep. Let p be an arbitrary point of the projection polygon at the beginning of the timestep. If p is further than 2Δ from the boundary of the (x,y) projection polygon at the beginning of the timestep, then any points reachable from p at the end of the timestep will be inside the time advanced polygon, because trajectories from p can move at most Δ units outward and points on the boundary of projection polygon can move at most p units inward during the timestep.

Otherwise, let e be an edge of the (x,y) projection polygon that is within distance 2Δ of p, and let h be the hyper-rectangle computed by the interval closure algorithm for e. By construction, h contains p. Accordingly, the LPs that Coho constructs for the faces of the projection polygon have feasible regions that extend by 2Δ in all of the other dimensions beyond the nearby (i.e. within 2Δ) points of projectagon.

These extensions create a "parapet" to ensure that trajectories from faces for other projection polygons can't "escape" this polygon. In particular, the face may shrink by at most Δ along any dimension, and any point can only reach other points that are within distance Δ (by the ℓ_∞) metric of itself. Thus, to reach a point outside of the (x,y) polygon, a trajectory from p would have to touch the feasible region for one of the faces arising from the (x,y) polygon. This means that points reachable from p are also reachable from the face that it touched, and therefore project to points on the (x,y) plane that are inside the time advanced projection polygon.

V. Verifying the Arbiter

This section describes how we modeled the clients of the arbiter and then describes the verification of the arbiter circuit that was described in Section II.

A. Modeling Concurrent Input Transitions

Rising transitions of the request signals for the two clients can occur concurrently. These requests can start at different times and have different rise-times. For example, r_1 can start a slow rising at some time, and r_2 can start a fast rising signal at slightly later and overtake r_1. In these scenarios, the arbiter may issue a grant to either client, and the grant may be delayed due to metastability in the arbiter. Verifying correct operation of the arbiter requires accounting for all allowed transitions of the inputs, including overlapping ones.

We sub-divided the Brocket annulus for each request signal into sixteen disjoint regions: one region for logical low values (region 1 from Section II-B, henceforth referred to as B_1); one for a logical high values (region 3, henceforth B_3); seven for rising transitions (dividing region 2 into $B_{2,1}$ through $B_{2,7}$); and seven for falling signals (henceforth $B_{4,1} \ldots B_{4,7}$). We modeled the concurrent behaviors of the two request signals using cross-product of these partitions. Applied directly, this would require Coho to solve 256 reachability problems. We reduced this to 136 problems by exploiting the symmetry of the arbiter and its clients and further down to 108 regions by noting that there must be a failure of the arbiter or its clients if both requests are falling at the same time – this would imply that either the arbiter had violated the mutual-exclusion requirement or that at least one client had violated the handshake protocol.

We divided the verification into three phases: the first phase starts with both requests in region B_1 and ends with at least one in B_3. The second phase starts with at least one request in region B_3 and ends with that request back in B_1. The final phase starts with one request having just entered region B_1 and the other in region B_3, and ends when the second request returns to region B_1. We use an assume-guarantee approach for verification [25]: for each phase, we posit bounding boxes for entering trajectories and verify bounding boxes for exiting trajectories. We show that the boxes bounding boxes for exiting trajectories for each phase are contained in the corresponding bounding boxes boxes for entering trajectories for the other phases. This establishes an invariant set that we use to verify the arbiter.

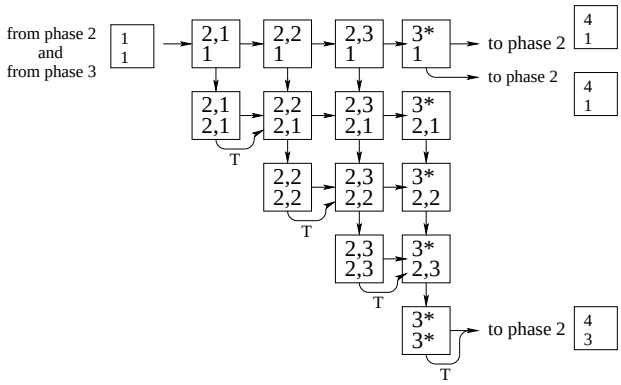


Fig. 7. First Verification Phase.

Figure 7 shows the first phase of this verification process; the other two phases are similar. Boxes are labeled with the regions for the r_1 (the upper label) and r_2, the lower label. A directed edge between boxes indicates that trajectories can leave that source box and enter the destination. We exploit the symmetry of the arbiter and only consider states where r_1 is further along in its handshake cycle than r_2. We take the reachable regions computed that would flow out the bottom of squares along the main diagonal, swap their (r_1, x_1, g_1) and (r_2, x_2, g_2) components, and include the result in the input to the cell to the right; these are the arrows labeled with T to indicate this "transposition" of the state. Finally, states labeled with $1*$ (resp. $3*$), indicate that the signal is low (resp. high) and has not completed the minimum high (resp. low) time requirement. For simplicity, we assume that these minimum dwell times are greater than the maximum rise and fall times. Thus, the boxes where r_1 enters B_3 while r_2 is in $B_{2,1}$ has an outgoing edge for r_2 entering $B_{2,2}$ but no outgoing edge for r_1 entering $B_4 - r_2$ is guaranteed to complete its rising while r_1 remains high.

We used Coho to compute reachable sets for trajectories exiting each box based on reachable sets for trajectories entering the box. For each phase, Coho computes these sets in topological order, using the "assume" bounding boxes for trajectories entering the phase, and verifying that trajectories that exit the phase satisfy the "guarantee" bounding boxes. This establishes an invariant set of trajectories that we use to verify safety and liveness properties of the arbiter in Section V-C.

B. Modeling the Circuit

For the most part, modeling the circuit is straightforward. Given the LP for a face, the modeling code computes a linear inclusion for the drain-source current of each transistors. For each node, summing the inclusions for the currents flowing into the node via transistors gives the total current to charge the node capacitance. Dividing this current by the node capacitance yields a linear differential inclusion for the node voltage.

The problem that we encountered is associated with nodes u_1 and u_2. These nodes have much smaller capacitances than the other nodes in the circuit; in fact, many designers would

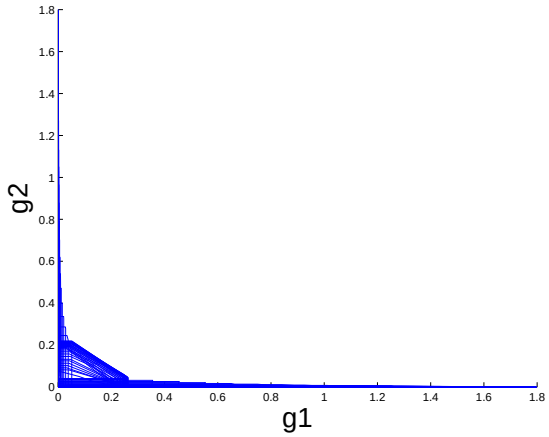


Fig. 8. Mutual Exclusion

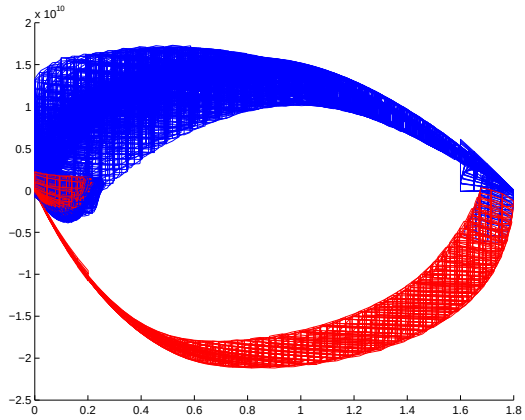


Fig. 9. Brockett Annulus for g

instinctively ignore the contributions of these tiny capacitors. Unfortunately for our verification work, these outlier node capacitances produce a stiff system of differential equations which exacerbate Coho's over approximation of the reachable space.

Coho has two principle causes of over approximation. First, there is an over approximation when producing a linear differential inclusion for a non-linear ODE. Second, over approximations are introduced when projecting the reachable region for a face back down to the basis for the projection polygon. For the arbiter, large time steps result in large linearization over approximation for nodes u_1 and u_2. Small time steps, on the other hand, lead to large over approximations for the other nodes due to the large number of projection operations. Either way, we were unable to verify the arbiter. Our solution was to follow the example of typical designers and treat nodes u_1 and u_2 as if they had no capacitance. We did this by creating a model for a nMOS tetrode with source connected to ground, gates connected to r_1 and x_2, and drain connected to u_1, and another such tetrode for the two pull-down transistors for u_2.

C. Results

Using the methods described above, Coho computed an invariant region for the arbiter. This set allows us to establish the correct operation of the arbiter as described below. Our circuit parameters correspond to the TSMC 1.8V, 180nm bulk CMOS process. Our Brockett annuli for r_1 and r_2 have inner and outer boundaries that are ellipses. Logically low signals are constrained to lie in $[0.0V, 0.2V]$ and logically high signals must be in $[1.6V, 1.8V]$. At the midpoint of rising and falling transitions ($0.9V$), the time derivative must be in $[1.5,2]*10^{10}$ volts/sec for rising transitions and $-[1.5,2]*10^{10}$ volts/sec for falling ones.

Safety Properties: Mutual Exclusion:
Figure 8 shows the reachable space projected onto the signals g_1 and g_2. Everywhere in the reachable space, either g_1 or g_2 is in $[0,0.2]$ region 1 of their Brockett annuli. Thus, at least one is false in the discrete abstraction, and the arbiter satisfies the mutual exclusion requirement.

Handshake Protocol:
In a similar fashion, projecting the reachable space is projected onto the signals g_1 and r_1 shows that g_1 enters B_2 only when r_1 is in B_3. Likewise g_1 enters B_4 when $r_1 \in [0.0, 0.22]$. When $r_1 \in [0.2, 1.6]$, $\dot{r_1} < 0$, thus $r_1 \in B_4$. This shows that g_1 starts to fall only when the discrete abstraction of r_1 is a logically low signal. Thus, the grants both rise and fall in accordance with the handshake protocol.

Brockett Annuli:
Figure 9 shows the Brockett annulus for g_1; it is slightly larger than the one that we used for the requests. We plan to re-run the verification using this slightly larger ring for the requests to show that the outputs of the arbiter satisfy the constraints that we imposed on the inputs, and thus show that this circuit could be part of a library of circuits sharing a common specification for signal transitions. The Brockett annulus for the grants shows an artifact of the over approximations used by Coho. Note that the left edge of the annulus is vertical. This is because over approximations that arise when linearizing the ODE model produce a differential inclusion that allows the grant signals to go to negative voltages. However, the model has an invariant that all signals must have values in $[0, V_{dd}]$ (we model all capacitances as being to ground). Coho clips the projectagon at the end of each time step to satisfy this invariant. This clipping produces the vertical left edge seen in the figure.

As shown in Figure 10, signal x_1 only satisfies a much less restrictive Brockett annulus than the ones we used for the request or established for the grants. The bulge in the lower right part of the annulus is a consequence of metastability in the arbiter: x_1 may drop to a value near the metastable voltage; remain their for an arbitrarily long time; and then either return to a high value (if client 1 loses the arbitration), or resolve to a low value (if client 1 wins). Comparing the Brockett annuli for x_1 and g_1, we clearly see the role of the metastability filter as a Brockett annulus transformer.

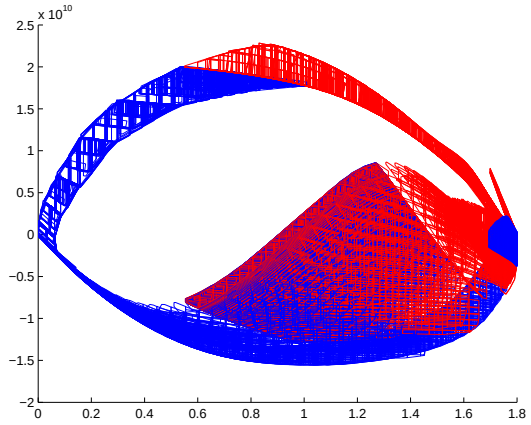


Fig. 10. Brockett Annulus for x

Liveness Properties: Initialization:
We used Coho to compute the reachable space when r_1 and r_2 were both low (i.e. in region B_1) starting from a state where x_1, x_2, g_1 and g_2 could be anywhere in $[0, 1.8]$. Coho establishes that within 200ps, x_1 and x_2 enter $[1.6V, 1.8V]$ (i.e. B_3), and g_1 and g_2 enter $[0.0V, 0.2V]$ (i.e. B_2). Thus, the arbiter can be initialized simply by not asserting any requests for a short time – no additional reset signal is needed.

Uncontested Requests:
We consider the reachable space with the additional restriction that r_2 remains within region B_1 (i.e. a logically low value). Coho shows that g_1 is asserted within 343 ps of r_1 rising (i.e. entering B_2). This shows that the arbiter is guaranteed to respond to contested requests within a bounded amount of time.

Contested Requests:.
If r_1 and r_2 are asserted at nearly the same time, the arbiter may exhibit metastable behavior and the arbiter may remain in a metastable state for an arbitrarily long period of time. Coho can show that metastability can only occur in the hyper-rectangle where:

$$\begin{array}{lll} r_1 \in B_3 & x_1 \in [0.55, 1.3] & g_1 \in B_1 \\ r_2 \in B_3 & x_2 \in [0.55, 1.3] & g_2 \in B_1 \end{array} \quad (2)$$

Outside of this region, the dot product of the derivative vector with the final stable state of granting client 1 or granting client 2 is unambiguous.

Reset:
Coho shows that if client i has a grant and lowers its request signal then the arbiter lowers the grant for client i within 270 ps. This shows that the arbiter satisfies the liveness requirement for withdrawing grants.

Fairness:
If client i wins a grant while the other client is making a request and subsequently client i drops its request, then Coho shows that the other client receives a grant within 420 ps. This shows that the other client receives at most one grant while the current client has a pending request; therefore, this simple arbiter is fair.

Other Observations:
As noted in Section V-A, we divided the rising and falling transitions for the arbiter inputs into seven regions each. After verifying the arbiter, we tried again using 1, 2, 3, 5, 9, 11, and 14 regions. For one region (i.e. no partitioning of B_2 or B_4, the approximation errors are significantly larger). For two or more regions, the approximation error decreases slightly with an increasing number of partitions.

VI. Conclusions

We have presented the verification of an asynchronous arbiter circuit. Our specification is a discrete transition diagram with additional liveness requirements. The circuit is modeled as a system of non-linear differential equations using foundry provided SPICE models for the transistors. The abstraction mapping from the continuous trajectories of the circuit model to the discrete sequences of the specification use Brockett's annulus construction. Our model allows the inputs to transition at arbitrary times. Unlike [7], we also allow the input transitions to have arbitrary waveforms that satisfy the rise and fall time and monotonicity requirements of the Brockett annulus.

We presented improvements to the Coho tool that made this verification possible. In particular, we presented an interval closure algorithm for obtaining tighter bounds for projectagon faces, and we sketched a proof that using these bounds preserves the soundness of Coho. We described how we modified the polygon simplification step to allow concavities to evolve in the projectagons. Verification of the arbiter requires representing highly non-convex regions (e.g., see Figure 8). The changes that we made to Coho enabled the successful verification of the arbiter, and we believe that similar non-convexities will be important when verifying other circuits.

The safety properties that we verified establish that the arbiter observes the handshake protocol with its clients and maintains mutual exclusion. For liveness, we showed that the arbiter can be initialized simply by leaving both requests low for at least 186 ps; an uncontested request is granted within 343 ps; a grant is lowered within 270 ps of lowering the request; and that a waiting request is granted with 420 ps of the other client dropping its request. The last of these establishes that the arbiter is strongly fair: no client will receive two or more grants while the other client is waiting.

This verification points to further work for verifying that circuits with continuous models satisfy discrete specifications. Here we mention possible areas for future research, both for the discrete and continuous aspects of the verification problem. Our discrete abstraction uses Brockett's annulus construction, and we verified safety properties by manually checking that the reachable trajectories were contained in the specified annulus. This was a relatively easy and straightforward task that could be readily automated. More generally, we would like to develop a tool to automatically translate "data sheet" style specifications to the corresponding topological properties to check by reachability analysis. For circuits more complicated than the arbiter, the discrete behavior could be specified using standard methods such as state-machine descriptions, assume-guarantee techniques or guarded command programs. For the

arbiter example in this paper, we used the Brockett annulus directly for the visual intuition that it provides, and because we believe that the space to introduce an additional notation for such a simple problem would only serve to distract from the key issues in circuit-level verification.

The arbiter provides an important generalization over our prior verification of a toggle flip-flop [23] in that the arbiter has two-inputs, and the relationship between these inputs and the arbiter outputs must be specified to capture the hand-shake protocol. We extended the Brockett annulus abstraction to capture the allowed behaviors of these inputs. We plan to apply similar techniques to handle synchronous circuits, adding the extra timing constraints that input signal transitions must satisfy timing requirements such as set-up and hold times. We believe that our specification framework can be readily adapted to synchronous design, and that the verification tasks may actually be simpler. Extending our techniques to a high-performance, synchronous design family such as domino circuits [26] would make these techniques of use to a large range of designers.

We would like to be able to show an "almost-surely" result for liveness when the two clients make concurrent requests. The proof in [10] was based on arguments about the eigenvectors of the Jacobian matrix for the time derivative function. The A matrix in our differential inclusions (see Equation 1) is roughly the same as this Jacobian matrix. However, the u term in the differential inclusion prevents making a direct correspondence. Verifying the almost-sure liveness is a topic for future work.

The other key area that we see for further research is developing methods to cope with stiff ODE models. As described in Section V-B, the small capacitance on the internal nodes of the NAND gates have associated time constants that are much smaller than the other nodes in the circuit. This is a classic example of a stiff system of ODEs. For Coho, this stiff system makes it impossible to choose a time step that produces an acceptably small over approximation of the reachable space. We are currently exploring methods for handling stiffness and expect that they will be useful for other circuits and for verification problems for hybrid systems.

REFERENCES

[1] M. Mendler and T. Stroup, "Newtonian arbiters cannot be proven correct," in *Proceedings of the 1992 Workshop on Designing Correct Circuits*, Jan. 1992.

[2] 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.

[3] T. Chaney and C. Molnar, "Anomalous behavior of synchronizer and arbiter circuits," *IEEE Transactions on Computers*, vol. C-22, no. 4, pp. 421–422, Apr. 1973.

[4] M. Hurtado, "Structure and performance of asymptotically bistable dynamical systems," Ph.D. dissertation, Sever Institute, Washington University, Saint Louis, MO, 1975.

[5] C. L. Seitz, "System timing," in *Introduction to VLSI Systems* (Carver Mead and Lynn Conway). Addison Wesley, 1979, ch. 7, pp. 218–262.

[6] L. Marino, "General theory of metastable operation," *IEEE Transactions on Computers*, vol. C-30, no. 2, pp. 107–115, Feb. 1981.

[7] 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.

[8] E. Asarin, T. Dang, and O. Maler, "The d/dt tool for verification of hybrid systems," in *Proceedings of the* 14^{th} *Conference on Computer Aided Verification*. Copenhagen: Springer, July 2002, pp. 365–370.

[9] T. Dang, A. Donzé, and O. Maler, "Verification of analog and mixed-signal circuits using hybrid system techniques," in *Proceedings of the* 5^{th} *International Conference on Formal Methods in Computer Aided Design*, Nov. 2004, pp. 21–36.

[10] I. Mitchell and M. Greenstreet, "Proving Newtonian arbiters correct, almost surely," in *Proceedings of the Third Workshop on Designing Correct Circuits*, Båstad, Sweden, Sept. 1996.

[11] A. J. Martin, "A delay insensitive fair arbiter," Computer Science Department, California Institute of Technology, Tech. Rep. 5193-TR-85, 1985.

[12] M. H. Zaki, S. Tahar, , and G. Bois, "Formal verification of analog and mixed signal designs: A survey," *Microelectronics Journal*, 2008.

[13] 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.

[14] 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.

[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] 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.

[17] G. Al-Sammane, M. H. Zaki, and S. Tahar, "A symbolic methodology for the verification of analog and mixed-signal designs," in *Proceedings of the 10th Design Automation and Test Europe Conference (DATE'2007)*, Apr. 2007, pp. 249–254.

[18] 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.

[19] G. Frehse, B. H. Krogh, and R. A. Rutenbar, "Verifying analog oscillator circuits using forward/backward abstraction refinement," in *Proceedings of Design Automation and Test Europe*, Mar. 2006, pp. 257–262.

[20] M. H. Zaki, G. Al-Sammane, S. Tahar, and G. Bois, "Combining symbolic simulation and interval arithmetic for the verication of AMS designs," in *Proceedings of the* 7^{th} *Conference on Formal Methods in Computer Aided Design (FMCAD'2007)*, Nov. 2007, pp. 207–215.

[21] 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.

[22] ——, "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.

[23] C. Yan and M. R. Greenstreet, "Circuit level verification of a high-speed toggle," in *Proceedings of the* 7^{th} *Conference on Formal Methods in Computer Aided Design (FMCAD'2007)*, Nov. 2007.

[24] ——, "Faster projection based methods for circuit-level verification," in *Proceedings of the 2008 Asia and South Pacific design automation conference (ASPDAC'08)*, Jan. 2008, pp. 410–415.

[25] T. A. Henzinger, S. Qadeer, and S. K. Rajamani, "You assume, we guarantee: Methodology and case studies," in *Proceedings of the* 10^{th} *International Conference on Computer Aided Verification*, 1998, pp. 440–451, LNCS 1427.

[26] R. Krambeck, C. Lee, and H. Law, "High-speed compact circuits with CMOS," *IEEE Journal of Solid-State Circuits*, vol. SC-17, pp. 614–619, June 1982.