Formal Verification of an Arbiter

Chao Yan   Mark Greenstreet   Jochen Eisinger

Department of Computer Science, University of British Columbia

{chaoyan,mrg,eisinger}@cs.ubc.ca

We present the circuit-level verification of a common arbiter circuit. To perform this verification, we address three issues. First, we present a specification for the arbiter and show how this specification amounts to a set of topological constraints on trajectories of the continuous model. Second, we show that computing bounding sets for these trajectories is complicated by stiffness of the differential equation model and present novel techniques for handling stiff equations in a formal verification context. Finally, we note that while no arbiter can be guaranteed to always grant a pending request, we can show liveness in the presence of concurrent requests in an "almost surely" sense.

I. INTRODUCTION

Unlike synchronous circuits where all discrete-state can be held in flip-flops, asynchronous circuits use a wider variety of state-holding elements including arbiters, (asymmetric) C-elements, toggles, etc. [1]. Here, we present the verification of a common arbiter circuit modeled at the level of non-linear differential equations like those used in simulators such as HSPICE [2]. There are many properties of circuit-level design that make it attractive for formal approaches. Key circuits tend to be small; thus, unlike system-level verification, the problems are not primarily ones of scale. Instead, the challenge is to correctly specify, model and verify circuit behaviour. As fabrication technologies progress to ever smaller feature sizes, transistors become less-and-less ideal, and phenomena that were insignificant in an older technology may cause design failures when a circuit is translated to a more advanced process. Finally, circuit-level design for key library cells is the domain of design experts who expect to spend a substantial amount of time on each cell designed. Thus, they can consider working with a verification expert if that interaction leads to a reduction in design time or risk.

This paper builds on earlier work verifying an asynchronous arbiter circuit [3] that used a simplified circuit model where pairs of series-connected transistors were treated as a single, four-terminal device. Doing so avoided the problems of "stiffness" described in Section IV-A. The underlying issue is that some nodes in a circuit can change values much faster than others. This creates a situation where there is no good time scale to perform the approximations that are required to obtain safe over approximations of the reachable space. Stiffness is common in circuits and control systems and is a longstanding topic of numerical analysis research. However, we are not aware of any earlier work exposing the challenges that stiffness poses for the reachability computations or that has presented any solutions to these problems.

This paper presents two solutions to the stiffness problem for circuit verification and verifies the "liveness" of the arbiter under metastable operation. In particular:

- We present a solution to the stiffness problem based on identifying simple invariants of the circuit that can be statically checked, and then use these invariants to reduce the approximation errors of the reachability computations. To the best of our knowledge, this work is the first to combine static analysis with reachability computations for formal verification of systems modeled by differential equations.
- We introduce second method for handling stiffness based on using implicit integration techniques implemented with a satisfiability checker. We believe that this is the first work to adapt implicit integration methods for formal verification.
- Finally, we establish the liveness of the arbiter. Of course, no arbiter can guarantee bounded time response in the presence of concurrent requests [4]. Instead, we employ the "almost-surely" approach from [5]. We combine the main analytical results from [5] with the reachability results described in the rest of this paper to establish almost-surely liveness for a real arbiter circuit modeled with device models obtained from HSPICE.

After surveying related work in Section II, Section III provides both discrete and continuous specifications for a two-client, asynchronous arbiter using a four-phase hand-shake protocol. The discrete specification is written in LTL, and the continuous specification is obtained by using an abstraction based on Brockett's annulus construction. With this formulation, the verification problem becomes one of computing reachable regions in the continuous state space of the circuit model. Section IV describes one approach to this reachability computation using the Coho tool and how we refined Coho to handle the stiff differential equations that model the arbiter circuit. Section VI presents an alternative approach to handling stiffness by formulating the reachability computation as a constraint satisfaction problem using HySat. In this constraint based approach, implicit integration methods are nearly as easy to describe as explicit ones, and we use an implicit integrator to solve the problem of stiffness. The other major issue in verifying arbiters is handling metastability. While both the Coho and HySat approaches verified the safety of the arbiter during metastability, neither can directly establish liveness. In Section V, we extend the approach first presented in [5] to verify liveness of the arbiter circuit in an "almost-surely" sense. This means that we show that the

probability that the arbiter grants a request goes to one in the limit that the time since making the request goes to infinity. Section VIII concludes the paper.

a. A two-input arbiter

Initially:
$\quad \forall i \in \{1,2\}.\ \neg r_i \wedge \neg g_i$

Assume (environment controls r_1 and r_2):
$\quad \forall i \in \{1,2\}. \qquad \Box(r_i\,\widehat{\mathbf{W}}\,g_i) \wedge \Box(\neg r_i\,\widehat{\mathbf{W}}\,\neg g_i)$
$\qquad\qquad \wedge \quad \Box(g_i\,\widehat{\mathbf{U}}\,\neg r_i)$

Guarantee (arbiter controls g_1 and g_2):
$\quad$ Handshake:
$\qquad \forall i \in \{1,2\}.\ \Box(\neg g_i\,\widehat{\mathbf{W}}\,r_i) \wedge \Box(g_i\,\widehat{\mathbf{W}}\,\neg r_i)$
$\quad$ Mutual Exclusion:
$\qquad \Box\neg(g_1 \wedge g_2)$
$\quad$ Liveness:
$\qquad \forall i \in \{1,2\}.\ (\Box(r_i\,\widehat{\mathbf{U}}\,g_i)) \wedge (\Box(\neg r_i\,\widehat{\mathbf{U}}\,\neg g_i))$

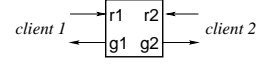b. Discrete specification

Fig. 1. An Arbiter

II. RELATED WORK

Metastable behaviour in digital circuits has been studied since Chaney and Molnar's original paper on synchronizer failures [6]. Hurtado [7] analyzed metastability from a dynamical systems perspective. Seitz [8] gives an introduction to metastability issues, and Marino [9] provides a comprehensive treatment.

Arbiters have also been studied from a formal verification perspective. Kurshan and McMillan [10] use the arbiter from [8] 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 III. They formulated the verification problem in terms of language containment. To model the continuous behaviour 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. 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 modeled the inputs as making instantaneous transitions. These transitions were allowed at arbitrary times that satisfied the handshake protocol.

Mendler and Stroup [4] 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 [5] 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.

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 presented in [11]. Frehse implemented *PHAVer* [12] which provides more efficient and robust implementations of the HyTech algorithms [13]. *Checkmate* [14] 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 [15] 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 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]. where $\mathbb{R}^+$ denotes the non-negative reals. 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 [18]–[21].

III. ARBITERS

An arbiter is an circuit that provides mutually exclusive access to a resource for some set of clients. We consider an asynchronous arbiter with two clients as shown in Figure 1.a.

A. Discrete Specification

Figure 1.b gives an LTL [22] specification for a discrete arbiter using an assume-guarantee approach [23] for separating the assumptions made about the clients, from the requirements for the arbiter. The "assume" clause describes what the environment can do: it can only modify r_1 and r_2, and it must do so in a way that satisfies the formulas in the assume clause. Conversely, the "guarantee" clause describes what the arbiter must do: it can only modify g_1 and g_2, and it must do so in a way that satisfies the formulas in the guarantee clause. We write $p\,\widehat{\mathbf{U}}\,q$ as a shorthand for $p \Rightarrow (p\,\mathbf{U}\,q)$. The LTL operators have the following interpretations:

$\quad p$: p holds in the current state.
$\quad \mathbf{N}\,p$: p holds in the next state.
$\quad \Box\,p$: shorthand for $p \wedge (\mathbf{N}\,\Box\,p)$; p holds this and all subsequent states.
$\quad \Diamond\,p$: shorthand for $\neg(\Box\neg p)$; p holds in this or some future state.
$\quad p\,\mathbf{U}\,q$: shorthand for $p \wedge (q \vee \mathbf{N}\,(p\,\mathbf{U}\,q))$; p holds in this state and continues to hold until a state in which q holds.

$p \, \widehat{\mathbf{U}} \, q$: shorthand for $p \Rightarrow (p \, \mathbf{U} \, q)$; if p holds in the current state, p will continue to hold until a state in which q holds.

$p \, \mathbf{W} \, q$: shorthand for $(p \, \mathbf{U} \, q) \vee \Box p$; p holds in the current state and continues to hold forever or until a state in which q holds.

$p \, \widehat{\mathbf{W}} \, q$: shorthand for $p \Rightarrow (p \, \mathbf{W} \, q)$; if p holds in the current state, p will continue to hold forever or until a state in which q holds.

In English, the specification says that if the clients observe the four-phase handshake protocol, then the arbiter will observe the protocol and ensure that grants are mutually exclusive. While we don't require clients to make requests, the clause

$$\Box (\mathsf{g}_i \, \widehat{\mathbf{U}} \, \neg\mathsf{r}_i)$$

states that a granted client must eventually withdraw its request – without this requirement, we would not be able to require that all requests are eventually granted, as one client could hold the grant forever. If this assumption is removed, the verification results that we present in Sections IV and VI can still be established. The liveness condition says that all requests must eventually be be granted. Of course, no physical arbiter can guarantee bounded response time to concurrent requests due to metastable behaviours. We address these issues in our specification of the continuous arbiter below.

B. A Continuous-Time Logic

This section describes how we define LTL-like formulas for continuous trajectories and how we introduce a few basic concepts from probability theory into the logic.

The state of a circuit is represented by a d-dimensional vector of real numbers; we say that d is the *dimension* of the model. In our arbiter verification, d corresponds to the number of nodes (other than power and ground) in the circuit. We write $\mathcal{V}$ to denote this d-dimensional state space of the circuit, and $d\mathcal{V}$ to denote the d-dimension time-derivative of the state. The circuit is modeled as a *differential inclusion*: if $A \subseteq \mathcal{V}$ and $x \in A$ then

$$\dot{x} \quad \in \quad F(A) \tag{1}$$

where $\dot{x}$ denotes the derivative of x with respect to time. In other words, $F : \mathcal{V} \to d\mathcal{V}$ maps regions of the state space to regions of the derivative space. By using an inclusion rather than an equation, the time derivative of the circuit is not fully determined. This non-determinism allows the differential inclusion model to include non-deterministic behaviour of the environment, such as the ordering and timing of input transitions. A behaviour of the circuit is a function from time (the non-negative reals) to states that starts in the initial region and satisfies the derivative relation. Such a behaviour is called a *trajectory*, and a circuit is characterized by the (infinite) set of trajectories allowed by its model:

$$\Phi(Q_0, F) \ = \ \{\phi : \mathbb{R}^+ \to \mathcal{V} \mid (\phi(0) \in Q_0) \wedge \\ (\forall t \in \mathbb{R}^+. \, \dot{\phi}(t) \in F(\{\phi(t)\}))\} \tag{2}$$

where $\mathbb{R}^+$ denotes the non-negative reals. Observe that if $\phi \in \Phi(Q_0, F)$, then $\dot{\phi}(t)$ is defined for all $t \geq 0$. Our continuous model for circuits is a tuple, (Q_0, F), where $Q_0 \subseteq \mathcal{V}$ is the initial region for the model, and $F : 2^{\mathcal{V}} \to 2^{d\mathcal{V}}$ is the time-derivative relation.

We extend LTL to specify continuous behaviours. Let $\phi : \mathbb{R}^+ \to \mathcal{V}$ be a trajectory, and define

$$shift(\phi, t_0)(t) \quad = \quad \phi(t + t_0) \tag{3}$$

If ϕ is a trajectory and S is a continuous LTL formula (defined below), we write $\phi \models S$ iff S is satisfied by ϕ. For a model $M = (Q_0, F)$, we write $M \models S$ iff $\forall \phi \in \Phi(Q_0, F). \, \phi \models S$.

A continuous LTL formula has a set of atomic propositions $\mathscr{P}$; these propositions correspond to subsets of $\mathcal{V} \cup (\mathcal{V} \times d\mathcal{V})$. For $P \in \mathcal{V}$, a trajectory $\phi \models P$ iff $\phi(0) \in P$; and for $P \in \mathcal{V} \times d\mathcal{V}$, $\phi \models P$ iff $(\phi(0), \dot{\phi}(0)) \in P$. Our continuous-time logic has no equivalent to the next-state operator; instead, we define $\Box$ and $\mathbf{U}$ directly on trajectories:

$$\begin{aligned}
\phi \models \Box S \quad &\equiv \quad \forall t \geq 0. \, shift(\phi, t) \models S \\
\phi \models S_1 \, \mathbf{U} \, S_2 \quad &\equiv \quad (\phi \models S_1) \wedge \\
&\quad \exists t_2 \geq 0. \ (shift(\phi, t_2) \models S_2) \wedge \\
&\quad (\forall 0 \leq t_1 < t_2. \, shift(\phi, t_1) \models S_1)
\end{aligned} \tag{4}$$

The other operators, $\Diamond$, $\widehat{\mathbf{U}}$, $\mathbf{W}$ and $\widehat{\mathbf{W}}$ can be defined from these in a manor analagous to the definitions of their continuous counterparts.

Due to the possiblity of metastability, any arbiter described by a continuous model must have input conditions that result in an unbounded delay between asserting a request and the corresponding grant. For a well-designed aribeter, the probability of a request being ungranted should go to zero in the limit that time goes to infinity. To specify such a property, we need to introduce a few ideas from probability theory. For this purpose, we let $\Phi(Q_0, M)$ is the set of events, we assume the existence of a Σ-algebra, $\Sigma_{(Q_0, M)}$, for $\Phi(Q_0, M)$ and a probability measure, $\mu : \Sigma_{(Q_0, M)} \to \mathbb{R}^+$.[1] In less technical terms, if $X \in \Sigma_{(Q_0, M)}$, then $\mu(X)$ is the probabity that the behaviour of the arbiter and its environment is given by a trajectory in X. Because $\Phi(Q_0, M)$ may contain an uncountable infinity of trajectories, there can be elements, Z, of $\Sigma_{(Q_0, M)}$, such that $\mu(Z) = 0$. Such a set of trajectories is called *negligible*; it corresponds to a set of events that occurs with probability zero. We define an *almost surely* version of the LTL "always" operator as shown below:

$$\phi \models \Box_Z S \quad \equiv \quad (\phi \models (\Box S)) \vee ((\phi \in Z) \wedge (\mu(Z) = 0)) \tag{5}$$

In other words, ϕ satisfies $\Box_Z S$ iff S holds everywhere along ϕ, or if ϕ is in a negligible set, Z. This means that the probability of S holding everywhere along ϕ is equal to 1. Note that the term $((\phi \in Z) \wedge (\mu(Z) = 0))$ is treated as an atomic proposition in the formula; verification of this assertion is outside of the continuous LTL framework.

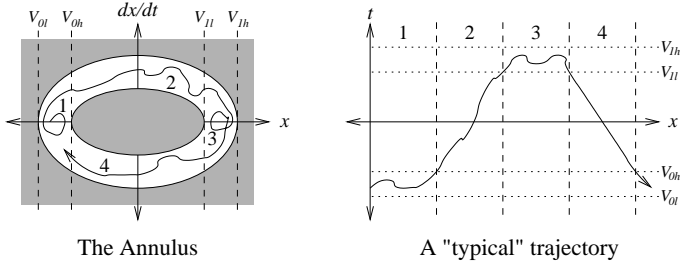[1] [24] provides an introduction to measure theory.

Fig. 2. Brockett's Annulus

C. Specifying a Continuous Arbiter

To specify a continuous arbiter, we need to identify regions of the state space, $\mathcal{V} \times d\mathcal{V}$ that correspond to true or false values of the atomic propositions (such as r_1) from the discrete specification, and we need to modify the liveness conditions to use an almost-surely formulation for situations with contested requests. Of course, uncontested requests and releases of requests should receive responses for all trajectories and not just a subset with probability 1.

Our abstraction from continuous trajectories to discrete values is based on the Brockett annulus construction [25] depicted in Figure 2. 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 rising and falling transitions to be unambiguous – regions 2 and 4 of the annulus admit signals with the same levels, but are distinguished by the value of the signal's time derivative. This construction forbids a signal from making a partial transition to some value between the low and high values and then returning to where it came from without making a complete transition. Furthermore, the horizontal radii of the annulus define the minimum and maximum low and high levels of the signal (i.e. V_{0l}, V_{0h}, V_{1l}, and V_{1h} in Figure 2). 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 impose 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.

A Brockett annulus provides the mapping from continuous trajectories to discrete traces. We write $B_i(x)$ to indicate variable x is in region i of the annulus, and $B_{i,j}$ to indicate that it is in region i or region j. If a trajectory is in region B_1 for variable x, then its discrete abstraction is unambiguously low (i.e. false); likewise if it is in region B_3, then its clearly high. If an input to the arbiter is in region B_2 (resp. B_4), then the arbiter *may* treat it as high, but it is not required to do so until the signal enters region B_3 (resp. B_1). Likewise for how the clients interpret the outputs of the arbiter.

To show that contested requests are granted (almost-surely), we follow the approach of [5]. Their approach requires demonstrating that the arbiter's clients do not act as feedback controllers. To do so, they introduced a concept they called "α − insensitivity." We omit the technical details here, and simply note that "accidently" designing clients that act as feedback controllers for a real arbiter is extremely far-fetched, and we're much better off worrying about the approximations used in HSPICE models and other more probable causes of failure. Thus, in our continuous specification, we assume that α-insensitivity holds and write α-ins to denote this α-insensitivity assumption. The reader is refered to [5] for the mathematical details of α-insensitivity.

Figure 3 shows our specification for the behaviour of an arbiter with a continuous model. Here, we wrote $r_{\sim i}$ to denote the "other" request. For the most part, this is a direct translation of the discrete specification from Figure 1.b to a continuous one using the Brockett annulus construction to provide the required atomic propositions for the continuous version. The only other change was that we rewrote the first clause of the liveness condition from the discrete specification with two clauses. The first liveness clause for the continuous specification,

$$\alpha\text{-ins} \Rightarrow (\Box_Z(B_3(r_i) \, \widehat{\mathbf{U}} \, B_{2,3}(g_i))),$$

says that if the clients satisfy the α-insensitivity requirement described above, then all requests are eventually granted except for those in a set of trajectories, Z, where Z has

Initially:
  $\forall i \in \{1,2\}.\ B_1(r_i) \wedge B_1(g_i)$
Assume (environment controls r_1 and r_2):
  $\forall i \in \{1,2\}.\quad \Box(B_3(r_i)\,\widehat{\mathbf{W}}\,B_{2,3}(g_i)) \wedge \Box(B_1(r_i)\,\widehat{\mathbf{W}}\,B_{4,1}(g_i))$
    $\wedge\quad \Box(B_3(g_i)\,\widehat{\mathbf{U}}\,B_4(r_i))$
Guarantee (arbiter controls g_1 and g_2):
  Handshake:
    $\forall i \in \{1,2\}.\ \Box(B_1(g_i)\,\widehat{\mathbf{W}}\,B_{2,3}(r_i)) \wedge \Box(B_3(g_i)\,\widehat{\mathbf{W}}\,B_{4,1}(r_i))$
  Mutual Exclusion:
    $\Box\neg(B_{2,3}(g_1) \wedge B_{2,3}(g_2))$
  Liveness:
    $\forall i \in \{1,2\}.$
        $\alpha\text{-ins} \Rightarrow (\Box_Z(B_3(r_i)\,\widehat{\mathbf{U}}\,B_{2,3}(g_i)))$
      $\wedge\quad (B_3(r_i)\,\widehat{\mathbf{U}}\,(B_{2,3}(g_i) \vee B_3(r_{\sim i})))$
      $\wedge\quad (\Box(B_1(r_i)\,\widehat{\mathbf{U}}\,B_4(g_i)))$

Fig. 3. Specification for a continuous arbiter

Fig. 4. A transistor-level implementation of an arbiter

IV. VERIFICATION USING COHO

zero probability. To verify this condition, we don't have to explicitly construct the set Z, we simply have to prove that such a set exists. Section V describes this proof process. The second liveness clause,

$$(B_3(r_i)\,\widehat{\mathbf{U}}\,(B_{2,3}(g_i) \vee B_3(r_{\sim i}))),$$

states that all uncontested requests must eventually be granted.

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 the circuit produced satisfies its Brockett annulus.
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.

D. An Arbiter Circuit

Figure 4 shows an implementation of an arbiter based on a SR-latch using a pair of cross-coupled NAND gates (see [26, Fig. 5]). The "metastability filters" MF_1 and MF_2 ensure that neither grant signal is driven high until the corresponding NAND-gate output is at least one p-channel threshold voltage below the other. This ensures that metastability has clearly resolved before a grant is asserted. The output inverters that produce $\overline{g_1}$ and $\overline{g_2}$ provide additional signal conditioning as described further in Section IV.
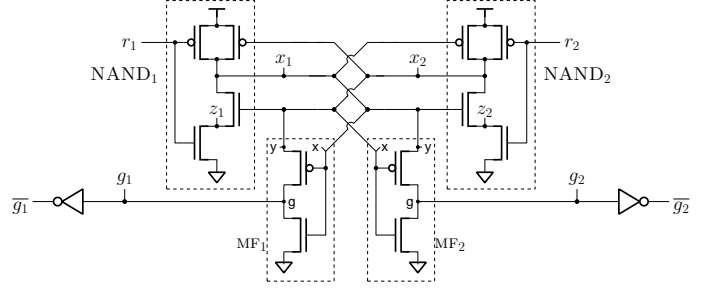
Our verification is based on the Coho reachability tool for systems modeled by ordinary differential equations (ODEs). Coho uses *projectagons* to represent reachable regions in continuous spaces. A projectagon represents an object by its projection onto two-dimensional subspaces. Conversely, given a set of projection polygons, the projectagon is the object obtained by intersecting the prisms obtained by inverse-projecting each projection polygon back into the full-dimensional space. Most operations on projectagons can be performed efficiently by manipulating the individual polygons, avoiding explicit operations on high-dimensional objects. Intuitively, projectagons allow relationships between pairs of signals to be represented with a high-degree of accuracy. These relationships capture the input-output relationship of gates such as the NAND gates and metastability filters in the arbiter. Projectagons can represent non-convex objects; this property is essential for the verification of the arbiter.

Coho's reachability algorithm [27], [28] computes reachability through a sequence of time steps. In each step, Coho computes a projectagon that contains all reachable points at the end of the step. The key to Coho's approach is that extremal trajectories originate from the boundary of the faces of the projectagon, and these faces correspond to edges of the projection polygons. This allows Coho to compute reachability by working on one edge at a time: Coho moves each edge forward, projects it back down to its polygon's subspace, and uses these projections to compute a bounding polygon for each projection polygon at the end of each time-step.

The non-linear differential equations that arise in circuit analysis cannot be solved analytically; thus, reachability computations must be based on approximate, numerical methods. Coho uses linear, differential inclusions to bound the solutions of non-linear circuit models. All of Coho's approximations over approximate the reachable space. This ensures soundness for verifying LTL properties.

Coho computes bounds on the points reachable from a projectagon face during a time step by approximating ODEs by *linear differential inclusions:*

$$Ax + b - u \quad \leq \quad \dot{x} \quad \leq \quad Ax + b + u \tag{6}$$

where x is a vector giving the current, continuous state of the circuit; $\dot{x}$ is the time derivative of x; 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.

To obtain a linear differential inclusion for the circuit, we use standard nodal analysis techniques based on Kirchoff's current law (KCL). For simplicity, we model transistors as voltage-controlled current sources, and treat all capacitances as having constant values to ground. Let n be the number of nodes of the circuit and m be the number of transistors. Let $I_{ds} : \mathbb{R}^n \to \mathbb{R}^m$ be a function from vectors of node voltages to vectors of transistor currents. Let $M \in \mathbb{R}^{n \times m}$ be the connectivity matrix for transistors to nodes:

$$
\begin{aligned}
M(i,j) \quad = \quad &+1, \quad \text{if the source of transistor } j \\
&\qquad \text{is connected to node } i \\
= \quad &-1, \quad \text{if the drain of transistor } j \qquad (7) \\
&\qquad \text{is connected to node } i \\
= \quad &0, \quad \text{otherwise}
\end{aligned}
$$

Let C be the diagonal matrix where $C(i,i)$ is the capacitance from node i to ground. We now have:

$$
\tfrac{d}{dt}V \quad = \quad C^{-1}MI_{ds}(V) \qquad (8)
$$

We impose the physically realistic requirement that all circuit nodes have some non-zero capacitance to ground, although this would preclude some "equivalent-circuit" models for devices. This restriction has not been problematic in our experience, and it ensures that C is positive definite and therefore invertible.

We use HSPICE to obtain tables of I_{ds} data for a particular process on relatively fine grid and use this data in our reachability calculations. Because I_{ds} is monotonic in transistor node voltages, it is straightforward to obtain linear inclusions that contain the HSPICE model based on the tabulated data. Substituting these linear inclusions into Equation 8 provides the linear differential inclusion required for Coho. This approach is simple and general: we can generate accurate models for any process with vendor provided SPICE models.

The inputs to the arbiter are described using a Brockett annulus. To construct an inclusion for $\dot{r}_i$ in the neighbourhood of a face, Coho computes upper and lower bounds for r_i in this neighbourhood, and then uses the Brockett annulus to determine a linear inclusion for $\dot{r}_i$ for this interval.

A. Stiffness

As described above, Coho uses over approximations to ensure soundness. To avoid false-negatives (failure to verify a correct circuit), it is important to minimize these approximation errors. A critical issue is the choice of the time-step size in the reachability computation. Some error is introduced at the end of each time-step when Coho projects the reachable space onto two-dimensional sub-spaces. To prevent a rapid growth in the number of polygon vertices, Coho replaces each polygon at the end of a projection step with an inscribing

polygon with fewer vertices. Thus, time-steps should be large to reduce the projection and simplification errors. However, the error term, u, in the linear inclusion approximation of the ODE grows rapidly with step-size for typical circuit models, suggesting that time steps should be small to reduce linearization errors. Thus, the goals of minimizing the approximation errors due to linearizing the model and minimizing the errors arising from projection and polygon simplification are in tension with each other.

A system of ordinary differential equations, $\dot{x} = f(x)$ is said to be *stiff* if the Jacobian of f is an ill-conditioned matrix. For circuits, stiffness occurs when nodes have vastly different time-constants. This occurs in the arbiter where nodes z_1 and z_2 have much smaller capacitances than the other nodes in the circuit. If Coho chooses a large time-step (suitable for the nodes other than z_1 and z_2), then the linearization errors for z_1 and z_2 will be large, creating large over approximations for the voltages of these nodes. As the currents flowing through the n-channel devices driving x_1 and x_2 are quite sensitive to z_1 and z_2, this leads to large over approximations for x_1 and x_2 and a failure to verify the arbiter. Conversely, if Coho uses times steps that are small enough to obtain tight bounds for z_1 and z_2, the accumulated projection and simplification errors for the other nodes causes a failure to verify the arbiter. Circuit simulators such as HSPICE handle stiffness by using implicit integration algorithms. However, we are unaware of any formulation of an implicit algorithm that is compatible with a forward reachability computation such as used in Coho. As other reachability tools use similar methods, we expect that they will have similar problems if they attempt to verify common CMOS designs. This conjecture is supported by the absence of published results for formal verification of stiff systems.

A simplified model of the arbiter circuit was verified in [28]. That analysis side stepped the problems of stiffness by treating nodes z_1 and z_2 as if they had no capacitance. With this assumption, the voltage on these nodes is always exactly the value that balances the currents flowing through the upper and lower n-channel transistors of each NAND gate. With that assumption, each such pair of transistors can be modeled as a single, four-terminal device, i.e., a transistor with two gates. This simplification reduced the ODE model from six dimension down to four and eliminated the stiffness issues. However, the verification is incomplete. For example, we note that with optical proximity rules, the spacing between series-connected transistors is growing relative to other circuit dimensions for sub-100nm processes. If the capacitances of these nodes are ignored, it is impossible to determine when they have become large enough to cause a circuit failure.

We present two solutions to the problem of stiffness for circuit verification. The remainder of this section presents modifications to Coho's reachability computation that allow it to compute tight overapproximations The Coho solution is based on a change of variables and constraining the reachability computation with an externally verified invariant. Section VI presents an alternative approach based on the

satisfiability tool, HySat.

B. Changing Variables

When either transistor connected to node z_1 is conducting, z_1 tends to converge very quickly to a small neighborhood near its equilibrium value; however, the precise value of equilibrium varies widely according to the values of r_1, x_1 and x_2. For any choice of values for the voltages of r_1, x_1 and x_2, there is a unique voltage for z_1 such that $\dot{z}_1 = 0$. This is because the current from x_1 to z_1 through the upper transistor is determined by the voltages of nodes x_1, x_2 and z_1 and is negative monotonic in the voltage of node z_1. Likewise, the current from z_1 to ground through the lower transistor is determined by the voltages of nodes z_1 and r_1 and is positive monotonic in the voltage of node z_1. These properties hold for any realistic transistor model. Thus, given values for the voltages on nodes r_1, x_1 and x_2, there is a unique voltage for node z_1 such that these two currents are equal. This is the voltage at which $\dot{z}_1 = 0$, and we call this voltage the *equilibrium voltage* and denote it by $q_1(r_1,x_1,x_2)$. We define $q_2(r_2,x_2,x_1)$ in the analogous manner.

We replace z_1 and z_2 in the circuit's ODE with $u_1 = z_1 - q_1(r_1,x_1,x_2)$ and $u_2 = z_2 - q_2(r_2,x_2,x_1)$ respectively. Whenever a transistor driving z_1 is conducting, u_1 tends rapidly to zero, and it is much easier to show that u_1 converges to zero than to show that z_1 converges to a moving target. Likewise for u_2 and z_2. This change of variables formalizes the designer's intuition that the capacitance of nodes z_1 and z_2 "usually won't matter." The chain rule yields:

$$\dot{u}_1 \quad = \quad \dot{z}_1 - \left(\frac{\partial q_1}{\partial r_1} \dot{r}_1 + \frac{\partial q_1}{\partial x_1} \dot{x}_1 + \frac{\partial q_1}{\partial x_2} \dot{x}_2 \right) \quad (9)$$

Although z_1 is not a state variable of the modified ODE, it can be reconstructed by noting that $z_1 = u_1 + q_1(r_1,x_1,x_2)$, and then $\dot{z}_1$ can be determined based on the values for r_1, x_1, z_1 and x_2. In our implementation, we use a four-dimensional table, indexed by the values of u_1, r_1, x_1 and x_2 to compute values for $\dot{x}_1$ and $\dot{u}_1$. This table accounts for all four transistors of the NAND gate that produces x_1 and the capacitance on nodes x_1 and z_1. By including $\dot{x}_1$ in the table, we eliminate the need to reconstruct z_1 or calculate q_1. By directly computing $\dot{u}_1$ and $\dot{x}_1$, we avoid reconstructing z_1 with large error bound intervals that would then propagate to the other quantities. The same construction applies for computing $\dot{u}_2$ and $\dot{x}_2$.

C. An additional invariant

With the change of variables described above, Coho still encountered a problem at the rising edge of r_1. If r_1 is low and r_2 is high, then x_2 will be low, and both transistors connected to node z_1 will be in cut-off. In this case, the equilibrium voltage for z_1 is determined by balancing the small leakage currents of the two transistors. Thus, u_1 can have a large value when r_1 is low. Following a rising edge of r_1, u_1 should go quickly to zero. However, Coho's over approximations from when r_1 was low led it to a region from which it could not

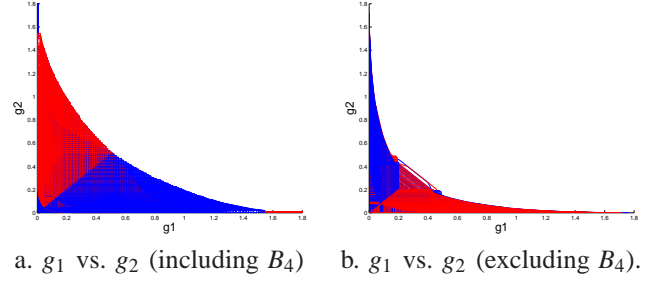a. g_1 vs. g_2 (including B_4)   b. g_1 vs. g_2 (excluding B_4).

Fig. 5. Verification with Coho: Mutual Exclusion

establish this contraction. In fact, the range for u_1 blew up to cover the entire interval from 0 to V_{dd}, and led to continuous states that violated the specification from Figure 3.

We solved this problem by manually establishing a simple invariant. The intuition behind this invariant is that based on the leakage currents for tour implementation in the TSMC 180nm, 1.8V CMOS process, we can determine that node z_1 eventually settles to 1.45V if $r_1 = 0$ and $x_2 = V_{dd}$, and this is an upper bound for z_1. By symmetry the same bound applies to z_1, from which we postulated the invariants $-1.45 \le u_1 \le 1.45V$ and $0 \le z_1 \le 1.45V$. It is straightforward to establish this invariant by computing the values of $\dot{u}_1$, $\dot{u}_2$, $\dot{z}_1$, and $\dot{z}_2$ on the boundary faces of this region to show that trajectories on these faces flow inward. We constrained the projectagons computed by Coho to satisfy these simple, externally verified invariants.

D. Verifying the Arbiter

We implemented the methods described above and applied them to the arbiter circuit from Figure 4 using device models for the TSMS 180nm, 1.8V CMOS process. The Brockett annulus for the request inputs was described with two ellipses. The inner ellipse has a diameter along the voltage axis from 0.4V to 1.6V, and a diameter along the voltage-derivative axis with endpoint at $\pm 1.5 \times 10^{10}$ volts/sec. The outer ellipse had a voltage diameter from 0V to 1.8V and a voltage-derivative diameter spanning $\pm 2 \times 10^{10}$ volts/sec. To model concurrent requests, we partitioned the input annuli into 16 regions each: one region for each of B_1 and B_3, and seven for each of B_2 and B_4. Subdividing the rising and falling regions allows Coho to work with smaller regions and reduce the approximation errors. This creates 256 possible pairs of regions, with a simple transition relation between them because both signals are constrained to going around their annuli. We used Coho to compute a fixpoint such that the starting projectagon for each region contained the union of the projectagons that could enter from neighbouring regions. Noting the symmetry of r_1 and r_2 we reduced the number of regions to consider to 136, and excluding regions where r_1 and r_2 were concurrently falling (implying that a failure of mutual-exclusion would have already been reported) further reduced this to 108 regions.
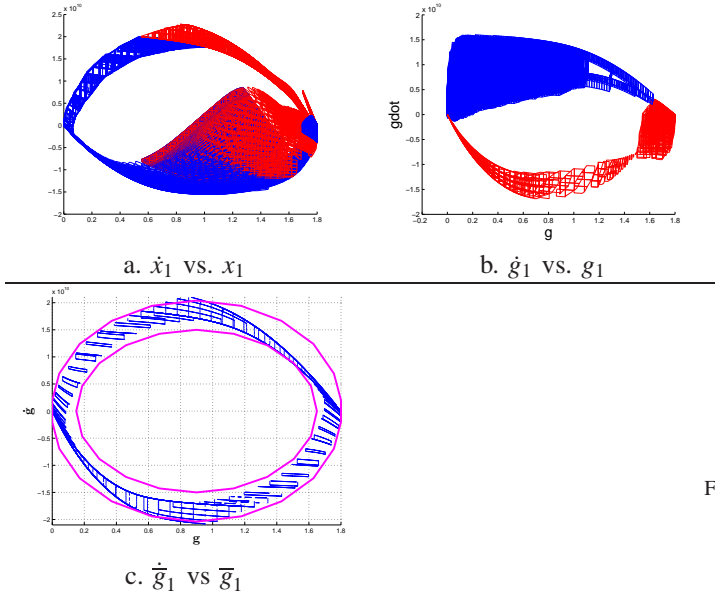
a. $\dot{x}_1$ vs. x_1 b. $\dot{g}_1$ vs. g_1

c. $\dot{\overline{g}}_1$ vs $\overline{g}_1$

Fig. 6. Verification with Coho: Brockett Annuli

Fig. 7. Reachable regions when r_1 and r_2 are both high

Figure 5 shows the verification of mutual exclusion. Part (a) of the figure shows all reachable states; clearly g_1 and g_2 are never both high. In fact, the region where they both reach values near 0.5V only occurs when one grant is falling and the other is rising as the arbiter transfers a grant that one client released and the other has requested. Figure 5.b shows the reachable space for g_1 and g_2 when when falling transitions of the grant signals are excluded. This shows that the separation of grants is very distinct.

Figure 6 shows the derivatives of the x, g and $\overline{g}$ signals versus their voltages. The grant signal, g_1 satisifies a Brockett annulus, but it is less restrictive than the one that we used for the request signals. In contrast, the time-derivative of signal x_1 has a much different shape. The lobe in the lower right shows the metastable behaviours: x_1 can start to fall, and then return to its nominal high value if x_2 wins the contest. The contrast between the plots for x_1 and g_1 shows the effectiveness of metastability filter as a Brockett annulus transformer. The output inverters that produce $\overline{g}_1$ and $\overline{g}_2$ further improve the transitions to produce the plot shown in part c where the reachable space computed by Coho is indicated by the blue polygons, and the pink ellipses show the Brockett annulus used for the request signals. With this output buffering, the output signals come extremely close to satisfying the original input annulus. We believe that we could repeat the verification with a slightly less restrictive input annulus and have the output satisfy the same requirements.

Similar plots show that the arbiter satisfies the handshake requirements. Coho's reachability analysis shows that an uncontested request is always granted within 343ps, and that grants are always withdrawn within 270ps of dropping the corresponding request. Finally, Coho shows that if one client holds a grant while the other client is making a request,
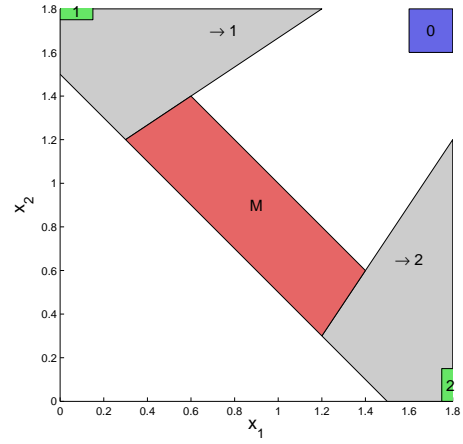
then the waiting client is granted within 420ps of the first client dropping its request. Thus, the arbiter is extremely fair, giving alternating grants when presented with heavily contested requests.

Using Coho, we have verified all properties of the arbiter from the specification given in Figure 3 except for the clause

$$\alpha\text{-ins} \Rightarrow (\Box_Z(B_3(r_i)\,\widehat{\mathbf{U}}\,B_{2,3}(g_i))),$$

This property concerns the behaviour of the arbiter under metastable conditions, and we describe our technique for verifying this property in the next section.

V. LIVENESS (ALMOST-SURELY)

When both requests are asserted concurrently, the arbiter may enter a metastable condition that can persist for an arbitrarily long time. Thus, it is not possible to prove that all behaviours when both requests are simultaneously asserted will eventually lead to granting a client. On the other hand, with a properly designed arbiter, the probability of no grant being issued when both requests are asserted should decrease exponentially with time. This implies that the probability of a liveness failure should go to zero as the settling time goes to zero. If the arbiter works for all situations except for some set with a probability measure of zero, then we say that the circuit works *almost surely* [24, Chapter 2.6]. Our approach is to use Coho to bound the reachable space when both requests are asserted. Most of this space can be shown to quickly resolve to granting one client or the other. For a small region near the metastable point, such progress can't be demonstrated, and we use the method from [5] to show that this metastable region is exited with probability one. The remainder of this section describes these steps in greater detail.

To verify liveness when both requests are asserted, we first use Coho to compute the reachable state after both requests are asserted. Figure 7 shows the outcome of this analysis. The square marked **0** indicates the initial region for x_1 and x_2

when r_1 and r_2 are both in region B_1. Coho then determines the reachable space for all low-to-high transitions of r_1 and r_2 allowed by their Brockett rings. After both r_1 and r_2 have been in region B_3 for a while, Coho shows that the state is in the union of the regions labeled **1**, **2**, $\rightarrow$**1**, $\rightarrow$**2**, and **M**. Region **1** is where x_1 has gone low, and grant g_1 will be asserted. Coho shows that all trajectories in region $\rightarrow$**1** converge to region **1** in bounded time and thus lead to asserting grant g_1. Likewise, from regions **2** and $\rightarrow$**2** lead to asserting g_2. Region **M** contains the metastable point. Because trajectories on the stable manifold for the metastable point remain in region **M** indefinitely, Coho cannot verify that all such trajectories eventually lead to issuing a grant.

We now show that the arbiter is live in the almost-sure sense. We first make a change of variables. Let

$$w^- = x_1 - x_2, \qquad w^+ = x_1 + x_2 \qquad (10)$$

We'll write $\mathbf{M}_w$ to denote the region M in (w^-, w^+) coordinates. As shown in [5], almost-all trajectories diverge from region **M** if we can find constants $c, d > 0$ such that for all points (w_1^-, w_1^+) and (w_2^-, w_2^+) with $w_1^- < w_2^-$,

$$
\begin{array}{lcl}
((w_2^+ - w_1^+) \le c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_2^- - \dot{w}_1^- \ge d(w_2^- - w_1^-) \\
(w_2^+ > w_1^+) \wedge ((w_2^+ - w_1^+) = c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_2^+ - \dot{w}_1^+ \le -d(w_2^- - w_1^-) \\
(w_2^+ < w_1^+) \wedge ((w_2^+ - w_1^+) = c(w_2^- - w_1^-)) & \Rightarrow & \dot{w}_1^+ - \dot{w}_2^+ \le -d(w_1^- - w_2^-)
\end{array}
\qquad (11)
$$

Basically, this condition ensures that if at some time, t_0 two trajectories differ only by their w^- components, then they must exponentially diverge; this ensures that one of them must leave region **M**. This implies that the set of trajectories that remain in **M** indefinitely must have a Lesbesgue measure of zero, which yields the desired almost-surely result.

We now present a simple test that ensures that the conditions from Equation 11 hold. For any point $w \in \mathbf{M}_w$, let $J(w)$ be the Jacobian operator for the ODE model projected onto the (w^-, w^+) space:

$$
J(w) = \begin{bmatrix} \frac{\partial \dot{w}^-}{\partial w^-} & \frac{\partial \dot{w}^-}{\partial w^+} \\ \frac{\partial \dot{w}^+}{\partial w^-} & \frac{\partial \dot{w}^+}{\partial w^+} \end{bmatrix} \qquad (12)
$$

Now, define:

$$
\begin{array}{lcl}
\mu & = & \min_{w \in M} J(w)(1,1), -J(w)(2,2) \\
h_1 & = & \left(\max_{w \in M} |J(w)(1,2)| / J(w)(1,1) \right)^{-1} \\
h_2 & = & \max_{w \in M} -|J(w)(2,1)| / J(w)(2,2)
\end{array}
\qquad (13)
$$

The conditions of Equation 11 are satisfied if

$$(\mu > 0) \quad \wedge \quad (h_1 > h_2) \qquad (14)$$

where μ, h_1 and h_2 are defined as in Equation 13. This is straightforward to show by integration along the line segment from point (w_1^-, w_1^+) to point (w_2^-, w_2^+) from Equation 11. We implemented these tests using the interval arithmetic toolbox, INTLAB [29] for Matlab.

We note that [5] includes an additional constraint for the environment called α-insensitivity, which is ensures that the clients do not act as feedback controllers to stabilize the saddle-point of the arbiter. We do not know of a way to automatically verify this condition. On the other hand, we believe that accidently constructing a feedback controller for a circuit like the arbiter is an extremely far-fetched scenario. Thus, we consider this omission in our verification to be insignificant from a practical perspective.

VI. IMPLICIT INTEGRATION BY CONSTRAINT SOLVING WITH HYSAT

When verifying the arbiter using Coho, the biggest challenge was the stiffness of the circuit models. We note that circuit simulators such as HSPICE handle stiffness using implicit (a.k.a. "backwards") integrators. Coho is based on a forward integration method, as are all other reachability tools we have seen. Implicit algorithms are more robust when integrating stiff models, which motivates exploring hte possibility of using an implicit integrator for reahcability computations.

The difference between forward and backward algorithms can be seen by considering a simple Euler integrator. If the differential equation model is $\dot{x} = f(x)$, then the forward algorithm repeatedly applies the computation:

$$x(t+h) \approx x(t) + hf(x(t)) \qquad (15)$$

whereas the backward algorithm is based on the formula

$$x(t+h) \approx x(t) + hf(x(t+h)) \qquad (16)$$

The difference is in whether the derivative is estimated from its value at the beginning of the time step, $f(x(t))$ in Equation 15 or at the end, $f(x(t+h))$ in Equation 16. The backward computation is more stable if $f(x)$ changes rapidly with x, but it requires solving a non-linear system of equations at each time step. This section describes how we used the HySat tool to implement a backward integrator for reachability computations.

HySat [30] is a program that given a boolean combination of non-linear constraints will either prove it to be unsatisfiable, or produce a set of hyperrectangles that each *may* include one or more satisfying assignments. We now describe how we used HySat to verify the mutual-exclusion property of the arbiter. Describing an implicit, Euler integrator in HySat is very simple: we just write the constraints that $x(t+h)$ must satisfy according to Equation 16 given $x(t)$ – we don't have to write an algorithm to solve the system on non-linear inequalities; this is handled by HySat's constraint solver. HySat requires a closed-form formula for the circuit model, and for this initial investigation, we used a simple, first-order transistor model chosing the model parameters to approximate somewhat roughly the i_{ds} curves of the HSPICE BSIM3 models. Likewise, the Brockett annulus defining the transitions of the inputs as well as assumptions that the inputs obey the handshake protocol can be encoded as constraints on the state space. Having described the circuit and environment models and the integration method using constraints defines a transition relation of pairs (v, v') that satisfy Equation 15 for a given time-step, h. It should be noted that Equation 16

approximates the derivative throughout the integration step as being equal to the value at the end of the time-step. Thus, this computation can underapproximate the reachable space.

HySat also includes a bounded model checking [31] front end for performing bounded verification of non-linear transition systems. Given formulas describing the initial states, the transition relation, and a set of unsafe states, HySat looks for a trace from the initial states to an unsafe state by iteratively unrolling the transition relation a fixed number of times, and checking that the resulting formula that describes *unsafe* states is unsatisfiable. If an unsafe state is found, the error is reported. Bounded model checking is fully automatic and can quickly find errors in a design, but does not provide a proof that the property holds for all time.

We use k-induction to show that mutual exclusion holds in all states. The k-induction proof proceeds as follows. Our initial verification shows that the arbiter is safe for any sequence of k steps from the initial region. We then ask HySat to find *any* x in the state space such that x and any state reachable from x in k steps satisfies the mutual-exclusion property, but there is some state reachable from x in $k+1$ steps that violates the property. If HySat shows this formula to be unsatisfiable, then that establishes, by k-induction, that the arbiter satisfies mutual exclusion for all reachable states.

Using the methods described above, we used HySat to verify the mutual exclusion problem. From the specification shown in Figure 3, we used the region $0V \leq g_1, g_2, r_1, r_2 \leq 0.15V$ for the initial state, and all internal nodes were chosen to be in small intervals that contained their steady state voltage levels. We first used bounded model checking to show that $(g_1 < 0.6V) \vee (g_2 \leq 0.6V)$ holds when unfolding the transition relation up to 100 time steps of 5ps each. This is a weaker version of mutual-exclusion (wider bounds for g_1 and g_2) than established by Coho, but the verification is fully automatic. We then used the k-induction approach with $k = 4$. We first used HySat to establish the base case: $(g_1 < 0.6V) \vee (g_2 \leq 0.6V)$ holds for the first four time steps, i.e. 20ps. Note that this base case is implied by the bounded-model checking result above. We then showed used HySat to establish the induction step: from any continuous state for which $(g_1 < 0.6V) \vee (g_2 \leq 0.6V)$ holds for the four steps, then the property olds for the fifth time step. This establishes by induction that this weaker formulation of the mutual-exclusion property holds for all time.

VII. COMPARISON

The infrastructure for verifying circuits with Coho is much more thoroughly developed than that for HySat. Thus, any comparison is comparing a well-developed tool against an initial prototype. With this in mind, we observe that we could use Coho to verify the complete specification whereas we only used HySat to show that the mutual-exclusion property could be verified. By using differential inclusions in Coho's forward integration algorithm, Coho is guaranteed to find an over-approximation of the reachable space. As noted above,

the implicit integration method with HySat does not provide the same guarantee. While guaranteeing over-approximations, the bounds computed by Coho were much tighter than those computed by HySat.

The HySat approach was an initial experiment, and we report it because we believe it shows several promising areas for further research. The HySat verification was much more automated than the Coho method: HySat did not require the manual implementing a change of variables for the model and introduction of an extra invariant that were needed for the Coho verification. Furthermore, the HySat verification was much faster, requiring a few minutes of CPU time rather than several days.

We see the two approaches as complementary. In particular, we are interested in exploring way to use tools such as HySat to derive bounds automatically and quickly that can be used to reduce the over-approximations of Coho and reduce the time required for verification. For example, HySat can automatically verify the extra invariant that we had to add manually in the Coho verification process. We are also interested in further exploring the applications of implicit integration methods to formal verification of systems modeled by stiff ODEs, and the constraint-based formulation in HySat provides an attractive basis for such experiments.

VIII. CONCLUSIONS

We presented the verification of an asynchronous arbiter circuit. In particular, we presented two solutions to the problems of stiffness in the ODE model for the circuit and showed how liveness can be verified in an almost-surely sense (i.e. with probability one) in the presence of metastable behaviours.

Our first approach to stiffness built on earlier work with the reachability tool Coho. We implemented a change of variables so that the "fast" variables of the circuit would converge quickly to zero. We showed how this allowed a forward reachability computation like that in Coho to handle stiffness without incurring large over-approximation errors. We noted that the time-constants associated with circuit nodes can vary drastically as nodes transition from "floating" to "driven" conditions and showed how simple, static invariants can improve the accuracy of the reachable space computation under these conditions. While we tested these techniques with Coho, we believe that they should be applicable to other reachability tools (e.g. [12], [14], [15], [32]).

We also described an experiment in addressing stiffness by using an implicit integration algorithm described as a system of non-linear constraints. In this case, our implementation was based on the HySat [30] tool. This approach is less developed than the Coho method, was only used to verify a single property of the arbiter, and used simpler circuit models. Nevertheless, this experiment demonstrated the promise of implicit methods. We believe that it is likely that future circuit verification work will combine static and symbolic techniques where tools like HySat excel with numerical based

reachability computations that are necessitated by the non-existence of closed form solutions for circuit modeled by non-linear differential equations.

We verified the almost-surely liveness of the arbiter by extending the methods first presented in [5]. In particular, we presented an easily verified condition that implies the liveness condition from the earlier work. We implemented our new test for liveness in Matlab and used it to verify the almost-surely liveness of the arbiter circuit.

Asynchronous design is a promising area for circuit verification. Asynchronous use a moderate number of types of state-holding cells, and the control of state transitions is distributed through the system. While this creates opportunities for optimizations that are not possible in synchronous design, it also creates additional verification questions. For example, does the circuit have stable states that were not intended by the designer? Are the dynamics of the circuit preserved when shrinking to smaller process sizes? Addressing these questions requires clear specifications of the intended behaviour and sound methods for verifying that the actual circuits satisfy their specifications. In this paper, we have shown how this specification and verification process can be applied to an asynchronous arbiter, and we believe that similar techniques can be used to verify other asynchronous components as well.

REFERENCES

[1] J. Ebergen, "A formal approach to designing delay-insensitive circuits," *Distribued Computing*, vol. 5, no. 3, pp. 107–119, July 1991.

[2] I. Synopsis, "HSPICE: The gold standard for circuit simulation," http://www.synopsis.com/Tools/Verification/AMSVerification/CircitSimulation/HSPICE/Pages/default.aspx, 2009.

[3] C. Yan and M. R. Greenstreet, "Faster projection based methods for circuit level verification," in *Proceedings of Thirteenth Asia South Pacific Design Automation Conference (ASPDAC)*, Seoul, Jan. 2008.

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

[5] 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.

[6] 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.

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

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

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

[10] 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.

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

[12] 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.

[13] 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.

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

[15] 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.

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

[19] 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.

[20] 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.

[21] 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 7th Conference on Formal Methods in Computer Aided Design (FMCAD'2007)*, Nov. 2007, pp. 207–215.

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

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

[24] D. Pollard, *A User's Guide to Measure Theoretic Probability*. Cambridge University Press, 2001.

[25] 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.

[26] A. J. Martin, "Programming in VLSI: From communicating processes to delay insensitive circuits," in *University of Texas Year of Programming Institute on Concurrent Programming*, C. Hoare, Ed. Addison-Wesley, 1989.

[27] M. R. Greenstreet and I. Mitchell, "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.

[28] C. Yan and M. R. Greenstreet, "Verifying an arbiter circuit," in *Proceedings of the 8th Conference on Formal Methods in Computer Aided Design (FMCAD'2008)*, Nov. 2008.

[29] S. Rump, "INTLAB – INTerval LABoratory," in *Developments in Reliable Computing*, T. Csendes *et al.*, Eds. Kluwer, 1999, pp. 77–104, http://www.ti3.tu-harburg.de/rump/.

[30] M. Fränzle, C. Herde, S. Ratschan, T. Schubert, and T. Teige, "Efficient solving of large non-linear arithmetic constraint systems with complex boolean structure," *JSAT Special Issue on Constraint Programming and SAT*, vol. 1, pp. 209–236, 2007.

[31] A. Biere, A. Cimatti, E. M. Clarke, and Y. Zhu, "Symbolic model checking without BDDs," in *TACAS '99: Proceedings of the 5th*

International Conference on Tools and Algorithms for Construction and Analysis of Systems.   Springer-Verlag, 1999, pp. 193–207.

[32] W. Hartong, L. Hedrich, and E. Barke, "Model checking algorithms for analog verification," in *Proceedings of the 39th ACM/IEEE Design Automation Conference*, June 2002, pp. 542–547.