Verifying an Arbiter Circuit

Chao Yan & Mark Greenstreet

University of British Columbia

Outline

- Circuit-Level Verification
  - Why verify at the circuit level?
  - Why verify an arbiter?
  - A specification for an arbiter.
- Coho
  - Overview
  - Enhancements
- Verifying the Arbiter Circuit
  - Arbiter Circuit
  - Properties Verified
- Conclusion and Future Work

Circuit-Level Verification

- What is circuit-level verification?
  - Analog circuit verification: verify stability, gain, noise-figure, etc.
  - Mixed-signal circuit verifcation: verify interactions between analog and digital circuits.
  - Digital circuit verification: Show that a circuit in an analog-model implements the desired discrete behavior.
- Properties that we verify:
  - Show correct operation for all valid input waveforms.
  - Use real, industry standard circuit and device models.
- Properties that we would like to verify:
  - Show correct operation for all process parameters.
  - Include crosstalk, power supply noise, etc. in our circuit models.

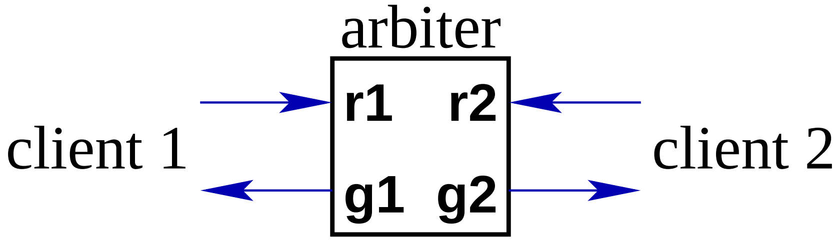
Why do Circuit-Level Verification?

- Digital design has become relatively low error:
  - Systematic design flows.
  - Lots of simulation.
  - Equivalence checking.
  - Model checking.
- Circuit-level bugs remain a problem:
  - SPICE is still the main validation tool, and it doesn't scale.
  - Deep-submicron circuit effects undermine digital abstractions.
  - Hard/impossible to simulate bugs.

Related Work

- Kurshan & McMillan (IEEE TCAD 1991) verified an nMOS arbiter.
  - Assumed inputs make instantaneous transitions.
  - This assumption greatly reduces the size of the reachable space.
- Many have formulated proof with various similar assumptions that no perfect arbiters can be built:
  - Hurtado 1975, Marino 1981, Chapiro 1984, Mendler & Stroup 1993.
- We are aware of no previous verification of an arbiter or any other multi-input, digital circuit with state that
  - uses a realistic model for the inputs applied;
  - uses realistic device models.
- We present such a verification in the current work.

Arbiters

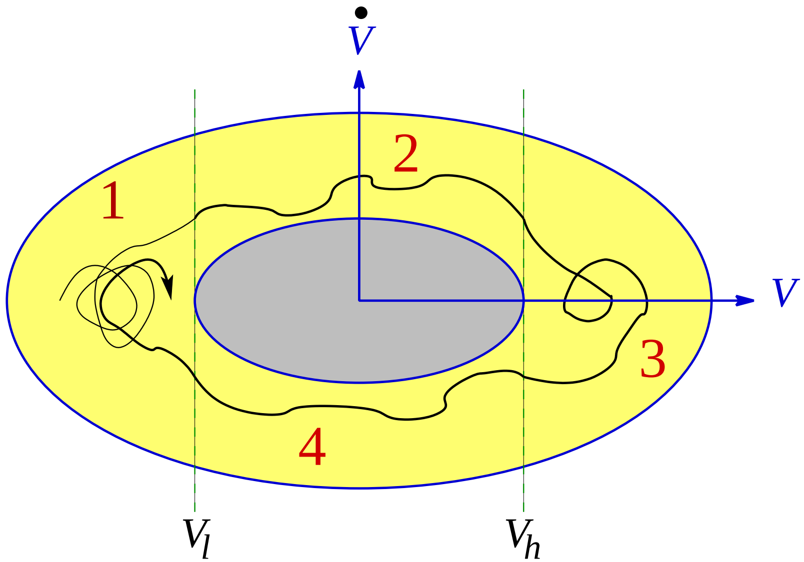


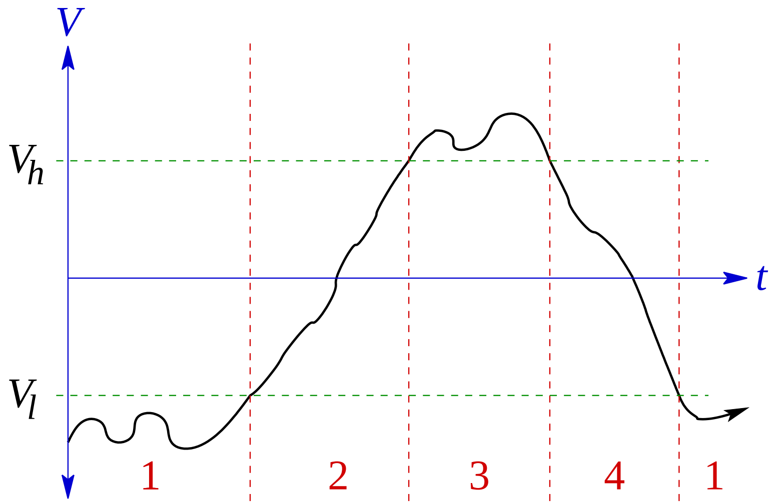
- Specification
  - Initially: $\neg r_1 \wedge \neg r_2 \wedge \neg g_1 \wedge \neg g_2$.
  - Assume: $\Box r_i \, U \, g_i$, $\Box \neg r_i \, U \, \neg g_i$.
  - Guarantee:
    - Handshake: $\Box \neg g_i \, U \, r_i$, $\Box g_i \, U \, r_i$.
    - Mutual Exclusion: $\Box \neg (g_1 \wedge g_2)$.
    - Liveness: $\Box (r_1 \oplus r_2) \Rightarrow \Diamond (g_1 \oplus g_2) \vee (r_1 \wedge r_2)$, $\Box \neg r_i \Rightarrow \Diamond \neg g_i$.
      Note: because metastability is unavoidable, no arbiter can guarantee $\Box (r_1 \wedge r_2) \Rightarrow \Diamond (g_1 \vee g_2)$.
- Why Verify an Arbiter?
  - Exercise in modeling concurrent events from the environment.
  - Requires handling a non-trivial circuit behavior: metastability.

Specifying an Arbiter

- Specifying signal behavior – Brockett's annulus.
- Specifying an arbiter.

Specifying an Arbiter





- Specifying signal behavior – Brockett's annulus:
  - Region 1 represents a logical low signal. The signal may wander in a small interval.
  - Region 2 represents a monotonically rising signal.
  - Region 3 represents a logical high signal.
  - Region 4 represents a monotonically falling signal.
  - Brockett's annulus allows entire families of signals to be specified.

Specifying an Arbiter

- Specifying signal behavior – Brockett's annulus. . .
- Specifying an arbiter.
  - Handshake:

$$
\begin{array}{llcl}
Discrete: & \Box\neg g_i \; U \; r_i & \wedge & \Box g_i \; U \; r_i \\
Continuous: & \Box g_i \in B_1 \; U \; r_i \in B_{23}) & \wedge & \Box g_i \in B_3 \; U \; r_i \in (B_{14})
\end{array}
$$

  - Mutual Exclusion:

$$
\begin{array}{ll}
Discrete: & \Box\neg(g_1 \wedge g_2) \\
Continuous: & \Box\neg((g_1 \in B_{23}) \wedge (g_2 \in B_{23}))
\end{array}
$$

  - Liveness:

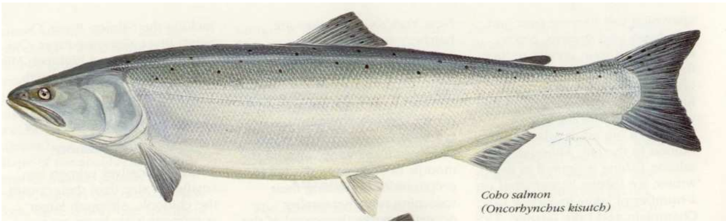
$$
\begin{array}{lcll}
Discrete: & & \Box\neg r_i & \Rightarrow \Diamond\neg g_i \\
 & \wedge & \Box(r_1 \oplus r_2) & \Rightarrow \Diamond(g_1 \oplus g_2) \vee (r_1 \wedge r_2) \\
Continuous: & & \Box(r_i \in B_{14}) & \Rightarrow \Diamond(g_i \in B_{14}) \\
 & \wedge & \Box(r_1 \in B_{23}) \oplus (r_2 \in B_{23}) & \Rightarrow \\
 & & \Diamond & (g_1 \in B_{23}) \oplus (g_2 \in B_{23}) \\
 & & \vee & (r_1 \in B_{23}) \wedge (r_2 \in B_{23})
\end{array}
$$

Verification by Reachability

- For all input (i.e. request) waveforms that satisfy:
  - Handshake protocol
  - Brockett annulus specification
- Find an invariant set that contains all trajectories, and
- Verify that everywhere in this set, the outputs (i.e. grants) satisfy:
  - Handshake protocol
  - Mutual exclusion
  - Liveness (for uncontested requests)
  - Brockett annulus specification

Coho: Reachability Using Projections

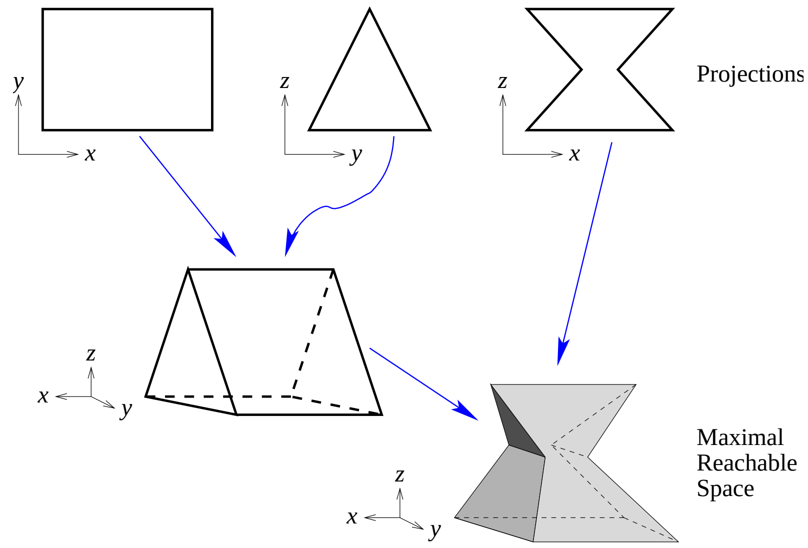
- Coho represents the reachable space by its projection onto two dimensional subspaces.
  - Provides a tractable representation.
  - Exploits extensive algorithms for 2D computational geometry.
- Coho models circuits using linear differential inclusions.
  - Inclusions computed for neighborhood of each projectagon.
  - Each inclusion of the form: $\dot{v} = Av + b \pm u$, where u is an error term.
- All approximations overapproximate the reachable space:
  - Coho is sound for verifying safety properties.
  - False negatives are possible.
  - Not useful for verifying unbounded liveness properties, but that doesn't seem to be an issue for circuit-level verification.



from: http://pond.dnr.cornell.edu/nyfish/Salmonidae/coho_salmon.jpg

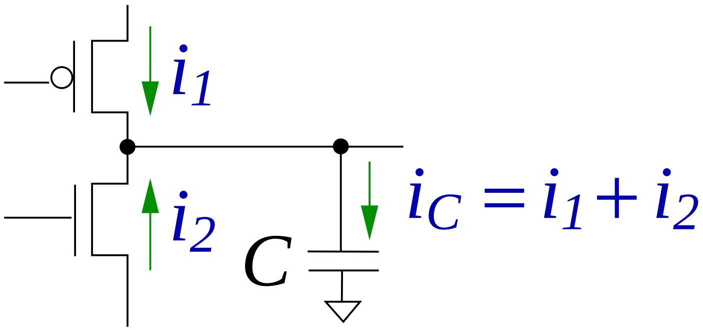
Projectagons

- Coho projects high dimensional polyhedron onto two-dimensional subspaces.
- A projectagon is the intersection of a collection of prisms, back-projected from the projection polygons.
- Coho computes reachable sets by integrating over a series of timesteps:
  - A bounding projectagon is obtained by moving each face forward in time.
  - Projectagon faces correspond to projection polygon edges; thus, Coho works on one edge at a time.



Circuit Model

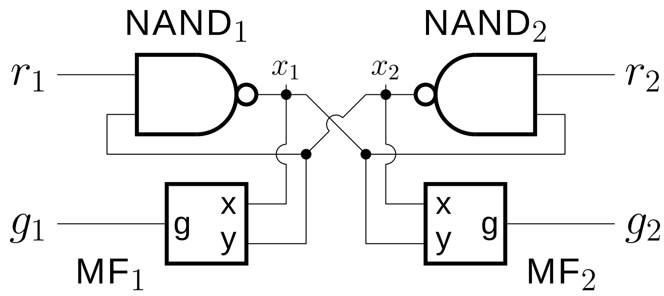
- Transistors models as voltage controlled current sources.
- The I_{ds} function is obtained by tabulated data from HSPICE simulations.



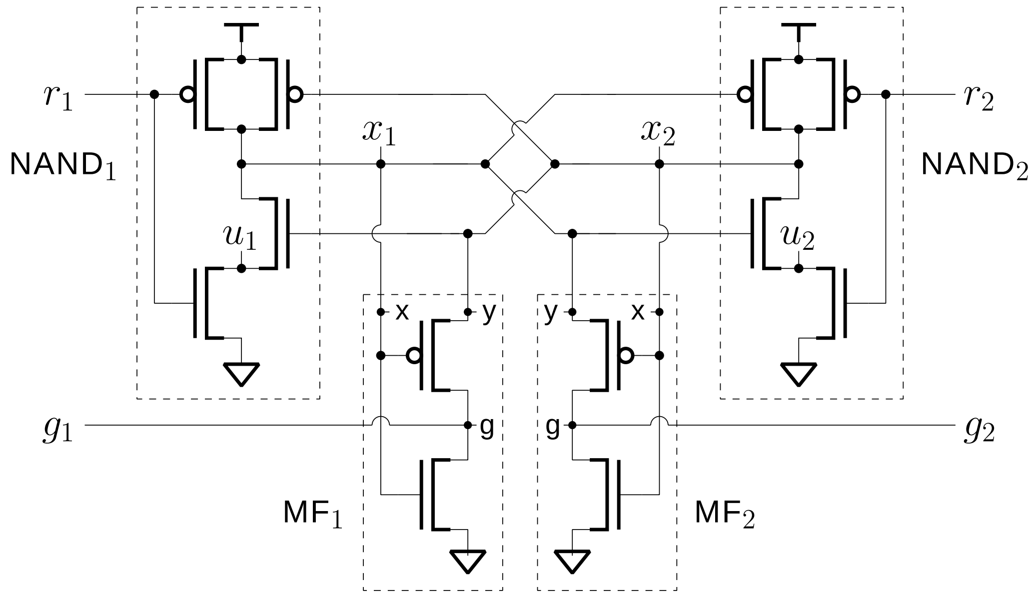
- At each time step, and for each projection polygon edge, Coho:
  - computes a bounding box for node voltages of each transistor.
  - computes a model of the form $i_1 = A_1 v + b_1 \pm u_1$ where u_1 is an error bound. Likewise for i_2.
  - bounds $i_c = (A_2 + A_2)v + (b_1 + b_2) \pm (u_1 + u_2)$. This produces a worst-case error bound.
- Approximate the ODEs by *linear differential inclusions*:

$$\boldsymbol{A} \left[\begin{array}{c} \boldsymbol{v} \\ \boldsymbol{in} \end{array} \right] + \boldsymbol{b} - \boldsymbol{u} \;\leq\; \dot{\boldsymbol{v}} \;\leq\; \boldsymbol{A} \left[\begin{array}{c} \boldsymbol{v} \\ \boldsymbol{in} \end{array} \right] + \boldsymbol{b} + \boldsymbol{u}$$

Arbiter Circuit

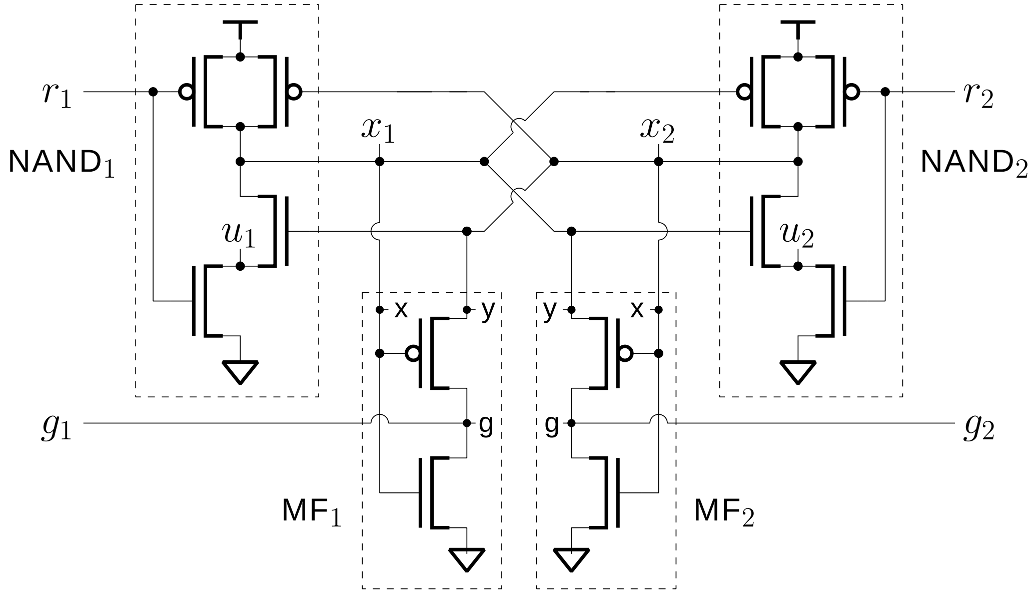


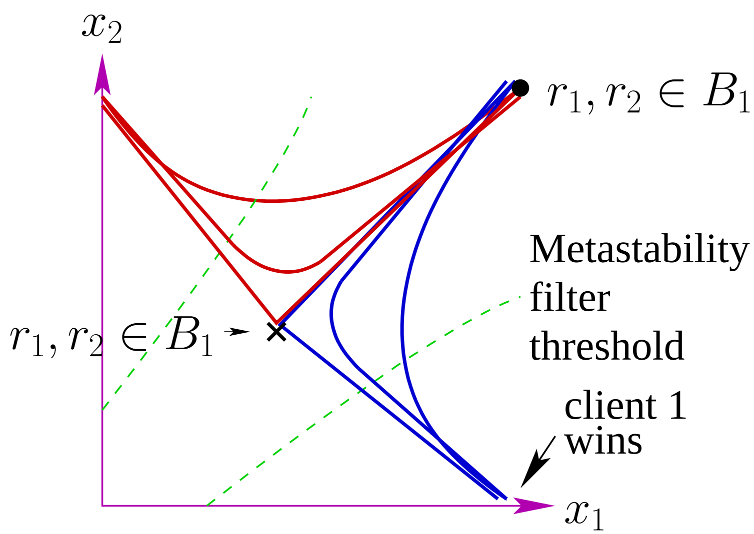
Gate-level schematic of the arbiter circuit.



Transistor-level schematic of the arbiter circuit.

Metastable Operation





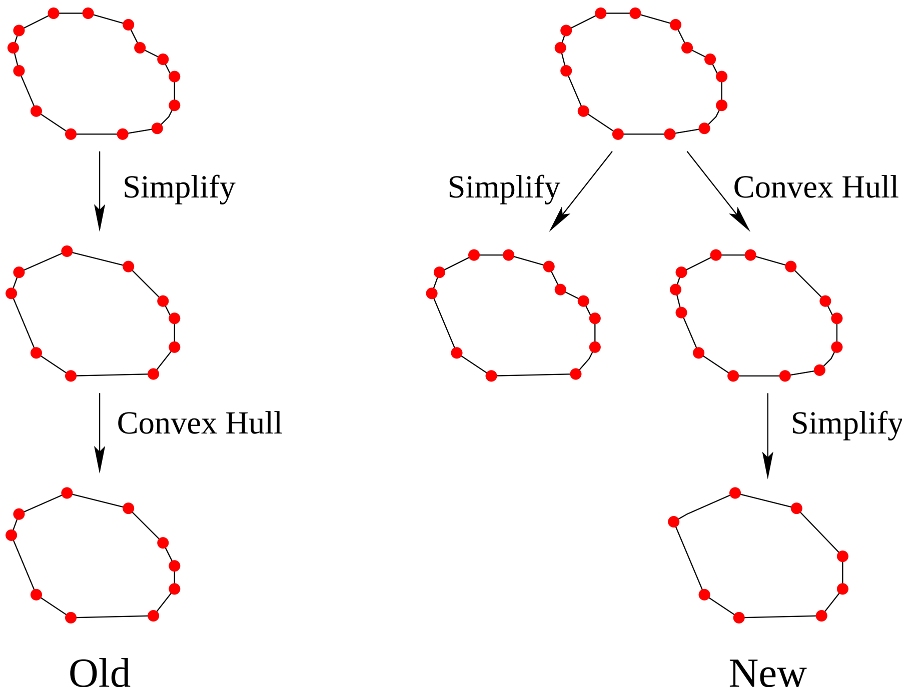
- Metastable operation leads to a highly non-convex reachable set.
- Coho can represent non-convex objects because projection polygons can be arbitrary, simple polygons.
- But, we had to improve some of Coho's approximations to get acceptable bounds on the reachable set.

Coho Improvements

- Polygon Simplification.
- Interval Closure.

Coho Improvements

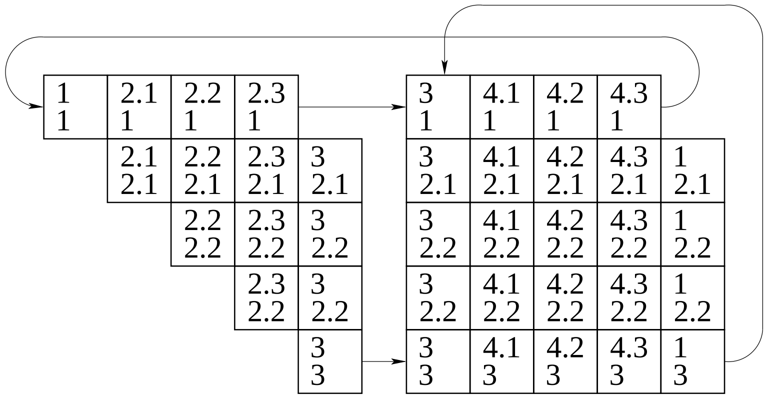
- Polygon Simplification:
  - Simplify convex hull and full polygon separately.
  - This allows more detailed projection polygons and slightly faster computation.



Coho Improvements

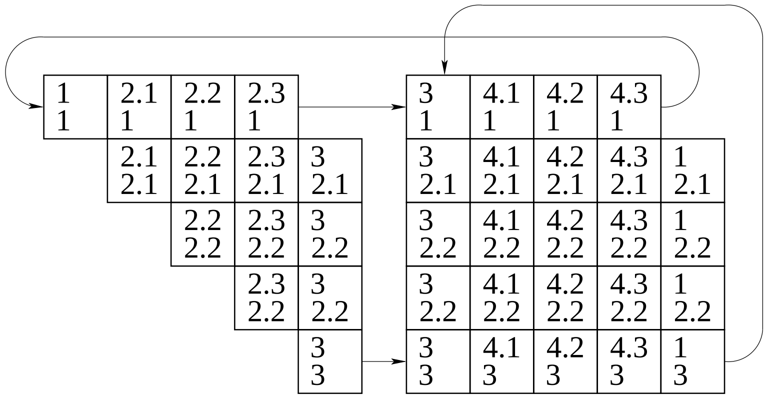
- Interval Closure (new):
  - Convex hull may badly overapproximate a projection polygon.
  - Starting from an edge of a projection polygon, compute the “interval closure” of all variable:
    - Use bounds from edge to restrict other polygons and find bounds for other variables.
    - Continue until no further improvement realised.
    - Example:
      - Edge for $x - y$ polygon gives interval bounds for x and y.
      - Using y bounds with $y - z$ polygon provides an interval bound for z.
      - Using z bounds with $w - z$ polygon provides an interval bound for w.
      - Using x bounds with $w - x$ polygon provides another interval bound for w – use the intersection.
      - ...
  - The paper includes a proof of soundness for the algorithm.

Verifying the Arbiter



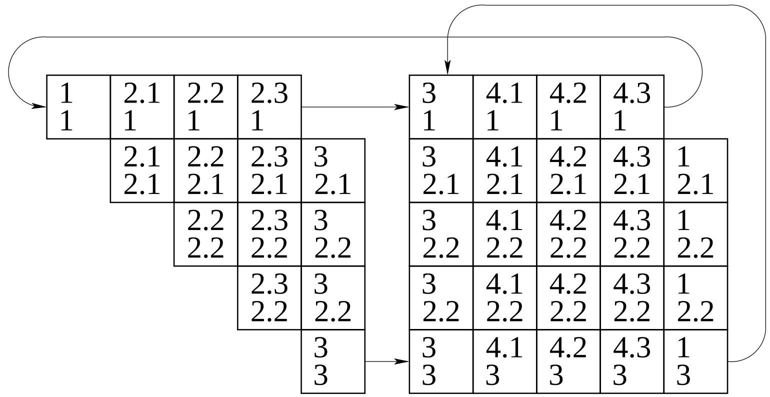
- Modeling concurrent input transitions.
- Use Brocket annulus to represent request signals.
- Divide the verfication into three phases.

Verifying the Arbiter



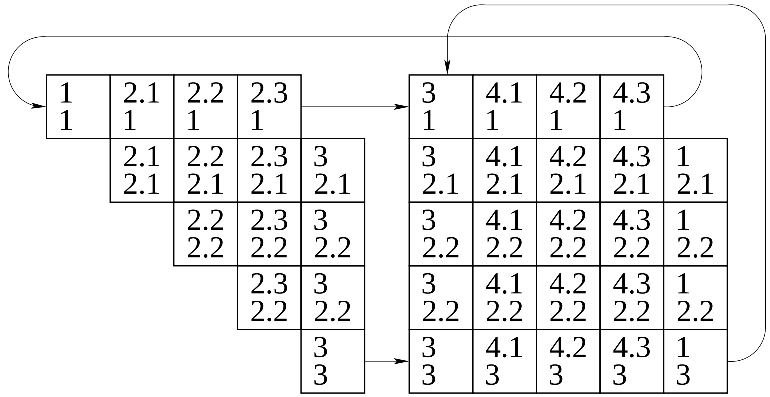
- Modeling concurrent input transitions:
  - Rising transistions of the two request signals can occur concurrently.
  - Requests can start at different times and have different rise-time.
  - Verification must account for all transitions allowed by handshake protocol and the Brockett annuli.
- Use Brocket annulus to represent request signals.
- Divide the verfication into three phases.

Verifying the Arbiter



- Modeling concurrent input transitions. . .
- Use Brocket annulus to represent request signals:
  - Sub-divide the annulus to reduce over approximation
    - B_2 (request rising) and B_4 (falling request) are divided into seven regions.
    - Subdivision needed, but the degree of subdivision isn't critical.
  - Reduce the number of reachability problems by exploiting the symmetry of arbiter:
    - If both requests are asserted, only consider states with $r_1 > r_2$.
    - Note that this can still lead to a grant of client 2!
- Divide the verfication into three phases.

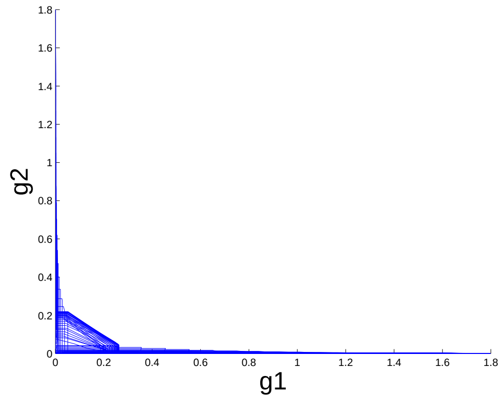
Verifying the Arbiter



- Modeling concurrent input transitions...
- Use Brocket annulus to represent request signals...
- Divide the verfication into three phases:
  - Asserting requests: $(r_1, r_2 \in B_1) \rightarrow (r_1 \in B_3) \wedge (r_2 \in B_{123})$
  - Falling phase, uncontested request:
    $(r_1 \in B_3) \wedge (r_2 \in B_1) \rightarrow (r_1 \in B_1) \wedge (r_2 \in B_{123})$
  - Falling phase, contested requests: $(r_1, r_2 \in B3) \rightarrow (r_1, r_2 \in B1)$
  - Use assume-guarantee reasoning
    - For each phase, assume an initial hyperrectangle, guarantee a final hyperrectangle.
    - Inclusion of final $\subset$ initial across all phases establishes invariant set.

Results

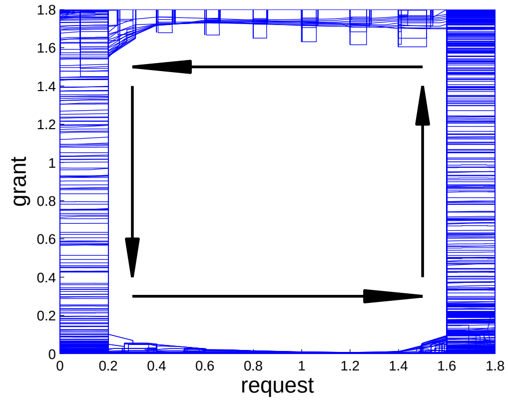
- Safety Properties
  - Mutual Exclusion
  - Handshake Protocol
  - Brockett Annuli
- Liveness Properties



Mutual Exclusion

Results

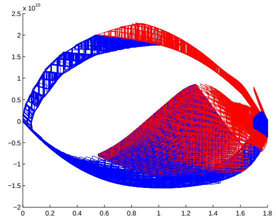
- Safety Properties
  - Mutual Exclusion
  - Handshake Protocol
  - Brockett Annuli
- Liveness Properties

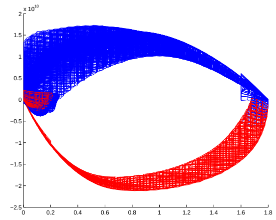


Handshake

Results

- Safety Properties
  - Mutual Exclusion
  - Handshake Protocol
  - Brockett Annuli
- Liveness Properties



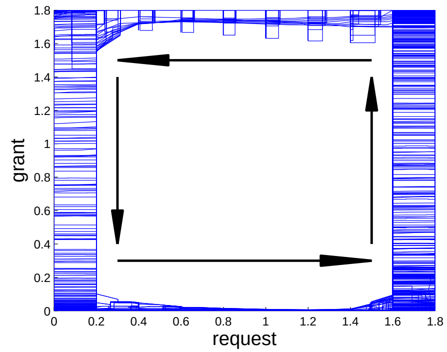


$\dot{x}$ vs. x $\dot{g}$ vs. g.

Brockett Annuli

Results

- Safety Properties
  - Mutual Exclusion
  - Handshake Protocol
  - Brockett Annuli



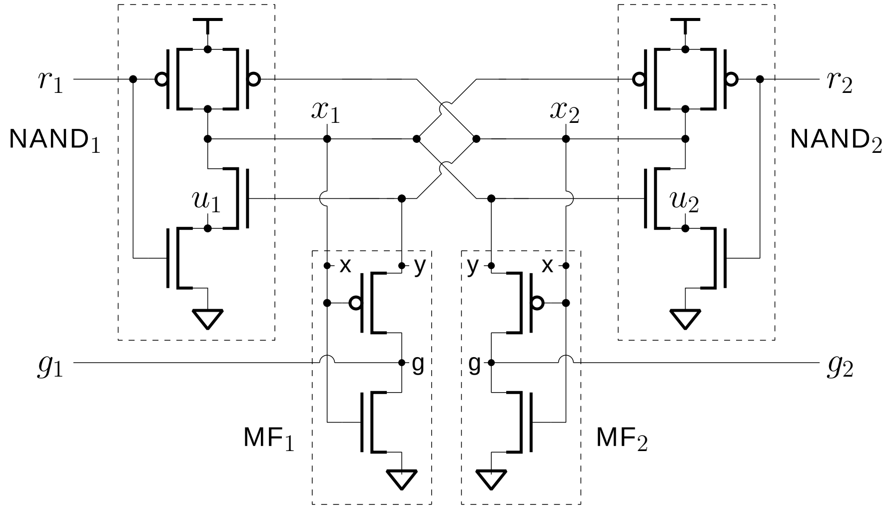
Handshake

- Liveness Properties:
  - Initialization: stable within 200ps
  - Uncontested Requests: grant the client within 350ps
  - Contested Requests: metastability within hyper-rectangle

$$r_1 \in B_3 \quad x_1 \in [0.55, 1.3] \quad g_1 \in B_1$$
$$r_2 \in B_3 \quad x_2 \in [0.55, 1.3] \quad g_2 \in B_1$$

  - Reset: withdraw grants within 270ps
  - Fairness: grant the other client within 420ps

But we're not completely satisfied



- Internal nodes u_1 and u_2 have much smaller capacitances
- Produce a stiff system
  - Large time steps results in large linearization over approximation
  - Small time steps lead to large number of projection operations
- Ignore the capacitance
  - Create a model for the nMOS tetrode
  - Use similar method to compute linear differential inclusion

Conclusion and Future Work

- Conclusion
  - Verify safety and liveness properties of an arbiter circuit.
  - Improved Coho to verify more complicated circuits.
  - The metastability filter transforms Brocket annuli.
- Future Work
  - Solve the stiffness problem.
  - Verify more properties and more circuits.
  - Formally describes the specification and translate it automatically.
  - Combine simulation and formal verification.

Conclusion and Future Work

- Conclusion
    - Verify safety and liveness properties of an arbiter circuit.
    - Improved Coho to verify more complicated circuits.
    - The metastability filter transforms Brocket annuli.
- Future Work
    - Solve the stiffness problem.
    - Verify more properties and more circuits.
    - Formally describes the specification and translate it automatically.
    - Combine simulation and formal verification.
- Questions?

Thank You!