A case study for the verification of complex timed circuits: IPCMOS

Marco A. Peña
Jordi Cortadella
Enric Pastor
Alexander Smirnov
Universitat Politècnica de Catalunya

Overview

GOAL: Formal verification of the IPCMOS architecture
Outline

- Formal verification with Relative Timing
- IPCMOS architecture
- Verification of IPCMOS pipelines
  - Strategy
  - Abstractions
  - Assume-Guarantee
  - Results
- Details on the verification of a single stage pipeline
- Conclusions

Formal verification with Relative Timing
Verification approach: main features

- Approach published in ASYNC 2000
- Iterative incremental refinement of the untimed state space by:
  - Off-line timing analysis on small acyclic graphs, and
  - Incorporation of Relative Timing constraints
- Applicable to Timed Transition Systems (TS + delay bounds), with any type of causality relations
- Verification of temporal safety properties
- BDD-based symbolic representation: large untimed state spaces
- Backannotation: sufficient relative timing constraints for correctness are reported

Verification approach: system model

Timed Transition System (Manna, Pnueli)
- Transition System
- Min/Max Delays
Verification approach: system model

Timed Transition System (Manna, Pnueli)

• Transition System
• Min/Max Delays

\[ \delta(a) \in [1,2] \]
\[ \delta(b) \in [1,2] \]
\[ \delta(c) \in [2.5,3] \]
\[ \delta(g) \in [0.5,0.5] \]
\[ \delta(d,x,y) \in [0,\infty) \]

Verification approach

Symbolic state space exploration and failure detection
Verification approach

- Failure trace
- Event structure

- Timing analysis
- Composition
Verification approach: flow

IPCMOS architecture
General IPCMOS architecture

- **Pulse-based** asynchronous clocking technique for large devices operating at GHz frequencies
- Block-level interlocking scheme \(\Rightarrow\) scalable
- Schuster, et al. ISSCC 2000

Linear IPCMOS pipeline architecture

- **VALID** : worst-case performance through a block \(\Rightarrow\) data is available to the next block
- **ACK** : data received by the next block
- Performance up to 4GHz
- Correctness depends on pulse widths
Two-stage pipeline at work

Critical pulse width
Stage building blocks

- Capture a negative pulse at **VALID** from the previous stage and produce a positive pulse at **ACK** to the next stage and a negative pulse at **CLKE** to the functional unit.

---

Strobe circuit

- Capture a negative pulse at **VALID** from the previous stage and produce a positive pulse at **ACK** to the next stage and a negative pulse at **CLKE** to the functional unit.
Strobe circuit

- Transition relations:
  - En(Y+) : ¬Y · ¬Z
  - En(Y-) : Y · ACK
  - . . .

- Failure conditions:
  - Shortcut at Y : ¬Z · ACK
  - . . .

Reset circuit

- Capture a positive pulse at ACK from the next stage and produce a positive pulse at CLKR and a negative pulse at CLKRN to reset both the strobe and valid circuits
Valid circuit

- Capture negative pulses at CLKE and produce a negative pulse at VALID. The pulse is reset by a positive pulse at CLKR.
- Delay after the inverter depends on the worst-case delay of the functional unit controlled by the stage.

Verification of IPCMOS pipelines
Verification goal

- Assuming data-path is correct, the pipeline is correct (spec.):
  \( S = \text{“Every data fed into de pipeline is acknowledged once and only once at every stage”} \)

- \( S \) is modeled by a deadlock condition plus correctness of CMOS circuits: no short-circuits, etc.

- Correctness regardless of the length of the pipeline
  \[ \text{IN} \parallel I_1 \parallel \ldots \parallel I_n \parallel \text{OUT} \leq S, \ n \geq 1 \]

Assume-Guarantee verification

- Pnueli 1984, Clarke et al. 1989, etc.
- Abstractions to overcome complexity: preserve the input/output behavior and the properties of interest
- Assume the abstractions are correct
- Prove that the abstractions are correct to guarantee a sound analysis
Verification strategy

- Key observation:
  - Pulse-based communication only at the extremes
  - Internal communication is time-independent, i.e. 2-phase handshaking
- Allows:
  - Untimed abstractions
  - assume-guarantee

Verification strategy

- $A_{in}$ and $A_{out}$ are untimed abstractions that hide the pulse-based behavior
- **Assume**: pose verification in terms of: $A_{in} \parallel A_{out} \leq S$
- **Guarantee** soundness of the abstractions:
  - $IN \parallel I_1 \parallel \ldots \parallel I_n \parallel OUT \leq A_{in} \parallel A_{out}$
- Prove correctness of a one-stage pipeline
Abstractions

Assume-guarantee strategy

- Assume: $A_{in} \parallel A_{out} \leq S$
- Guarantee correctness of $A_{out}$
- Guarantee correctness of $A_{in}$
- Guarantee correctness of $A_{in}$ (induction)
- Guarantee correctness of 1-stage: $IN \parallel I \parallel OUT \leq S$
Assume-guarantee

- Assume: $A_{in} \parallel A_{out} \leq S$
- Straightforward
- Few seconds of CPU time

Assume-guarantee

- Guarantee correctness of $A_{out}$:
  $$A_{in} \parallel I \parallel OUT \leq A_{in} \parallel A_{out}$$
- Check that any output produced by $I \parallel OUT$ can also be produced by $A_{out}$ at the same time
- 28 minutes of CPU time
Assume-guarantee

- Guarantee correctness of $A_{in}$ with one stage:
  \[ \text{IN} \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out} \]
- Check that any output produced by $\text{IN} \parallel I$ can also be produced by $A_{in}$ at the same time
- 9 minutes of CPU time

Assume-guarantee

- Guarantee $A_{in}$ is a \textit{behavioral fixed point}:
  \[ A_{in} \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out} \]
- 10 minutes of CPU time
- By induction, no matter how long the pipeline is, $A_{in}$ can be used as a correct abstraction
Assume-guarantee

- Guarantee correctness of a 1-stage pipeline:
  \[ \text{IN} \parallel \text{I} \parallel \text{OUT} \leq S \]
- 35 minutes of CPU time
- Most complex verification due to pulse-based environment

Assume-guarantee: results

<table>
<thead>
<tr>
<th>CPU time</th>
<th>Refinements</th>
</tr>
</thead>
<tbody>
<tr>
<td>( A_{\text{in}} \parallel A_{\text{out}} \leq S )</td>
<td>( 1m )</td>
</tr>
<tr>
<td>( A_{\text{in}} \parallel I \parallel \text{OUT} \leq A_{\text{in}} \parallel A_{\text{out}} )</td>
<td>( 28m )</td>
</tr>
<tr>
<td>( \text{IN} \parallel I \parallel A_{\text{out}} \leq A_{\text{in}} \parallel A_{\text{out}} )</td>
<td>( 9m )</td>
</tr>
<tr>
<td>( A_{\text{in}} \parallel I \parallel A_{\text{out}} \leq A_{\text{in}} \parallel A_{\text{out}} )</td>
<td>( 10m )</td>
</tr>
<tr>
<td>( \text{IN} \parallel I \parallel \text{OUT} \leq S )</td>
<td>( 35m )</td>
</tr>
</tbody>
</table>
Details on the verification of a single stage pipeline

- Most aggressive environment: pulse-based
- Stage described at transistor level (32 transistors)
- Fail conditions:
  - Conformance of the interface
  - Absence of deadlock
  - Absence of short-circuits and signal glitches
A verification iteration

Failure trace. Is it time-feasible?

Early Z- while Y still being discharged!

A verification iteration

ES capturing causality
A verification iteration

Off-line timing analysis

Sufficient minimum width of VALID pulse

A verification iteration

Reduced Timed ES

Relative timing constraints

Sufficient minimum width of VALID pulse
Conclusions

- RT-based verification: combine absolute timing (for analysis) with relative timing (for state space calculation)
- RT crucial to prove correctness of a complex timed system at transistor level
  - Relevant feature: back-annotation of sufficient RT-constraints that guarantee correct operation
  - Abstractions, assume-guarantee reasoning and induction to overcome complexity: but still manual

- Future work:
  - Automate abstractions, AG-reasoning, etc.
  - Reduce BDD size or try other data structures