Relative Timing Based Verification of Concurrent Systems

Marco A. Peña
Department of Computer Architecture
Technical University of Catalonia

Advisors: Jordi Cortadella and Enric Pastor

Outline

- PART I: Motivation and background
- PART II: Formal verification with relative timing
- PART III: Compositional verification
- PART IV: Conclusions and future work
Motivation: the problem

- System’s complexity: continuous growth in scale and functionality
- Probability to introduce design errors increases
- System failures are unacceptable:
  - 1994: Floating point divider unit of Pentium µP
  - 1996: Launch failure of Ariane 5 rocket
- **Simulation**: General bug-finding techniques
- **Formal verification**: Exhaustive coverage

Simulation

- Predominant verification method: intuitive idea
- Construction of test-cases: manually, randomly, etc.
- Cannot be exhaustive: “Heisenbug” paradigm, corner-cases, infinite cases, …

Simulation methodology:

```
System       simulator     (observe output)
Input patterns  01011...    10101...
```
Formal verification

- **Mathematical/Algorithmic** methods to determine correctness:
  - Formal model of the system
  - Formal specification language (*e.g.* some logic)
  - Exhaustive reasoning method
- Ensures consistency with specification for **all** possible input patterns

Formal verification methodology...

\[
\text{System model} \quad \rightarrow \quad \text{verifier} \quad \rightarrow \quad \text{yes/no/?}
\]

Model checking

- The checker enumerates **all** the states of the system
- Combinatorial **state explosion**: 32 bit register has \(2^{32}\) states
- **Symbolic** methods, partial orders, abstractions, etc.
- Many automatic tools and success stories exist
State space exploration

- Example:
  
  \[
  \begin{array}{c}
  \text{Environment} \\
  \begin{array}{c}
  a \quad b \\
  c \quad d
  \end{array}
  \end{array}
  \]

  - Initial state:
    - \( a = 1, \ b = c = d = 0 \)
    - \( a \) is about to fall

- FSM-like model: Transition system
  - State, transitions, events
- State space exploration
  - Initial state + transition relation
  - Iterative until fix-point
- Combinatorial state explosion problem
Symbolic methods

- Using $v_1, v_2, \ldots, v_n$ Boolean variables, a state is a minterm in $B^n$.
- Sets of states and transitions are Boolean functions in $B^n$.
- Boolean functions represented as Binary Decision Diagrams (Bryant): worst-case exponential, but efficient in practice.
Symbolic methods

SR(a-) = \{0000, 0100, 0010, 0110, 0011, 0111\}

SR(a-) = a'·d' + a'·c

EnR(c+) = (0000, 0100)

EnR(c+) = a'·c'·d'

Appropriate for untimed systems...

Timed systems

- Symbolic methods are mature for untimed systems
- Timed systems: correct behavior depends on timing characteristics of the system:
  - RTOSs: *e.g.* reaction to interrupts within a time frame
  - Communication protocols: *e.g.* time-out conditions
  - Aggressive circuit designs for performance
  - GALS-type systems: *e.g.* bridge of clock domains
- Time introduces new sources of exponentiality
- Conventional verification techniques do not apply
Timed systems

- Example:

![Timed transition system diagram]

- **Timed transition system** (Manna, Pnueli)
  - Transition system
  - Min/Max delay bounds
    - \( d(a-) \in [1, 2] \)
    - \( d(b+) \in [1, 2] \)
    - \( d(c+) \in [1, 3] \)
    - \( d(d+) \in [2, 3] \)
    - \( d(a+) \in [1, 2] \)
    - ...
Exact timed state space

- State $s_6$ cannot exist
- Infiniteness:
  - Time assignments
  - Instances

Timed automata (Dill, Alur, Courcoubetis, ...):
- Finite automaton to describe the control
- Real-valued clocks to model quantitative timing constraints
- COSPAN, KRONOS, UPAAL, MOCHA, ...

Clock regions and region automata (Dill, Alur, ...):
- Equivalence class of clock valuations
- Sensible to the number of clocks and the delay bounds

Zone automata:
- Collapse groups of regions into convex geometric regions
- Difference Bound Matrices (Dill)

Symbolic methods not easily applicable:
- Difference Bound Matrices and BDDs (Dill, Balarin, ...)
- Discrete counters and NDDs: OpenKRONOS (Bozga, Maler, ...)

Exact timed state space
Relative timing

- Delays impose:
  - \(d^+\) cannot fire in \(s_2\)
  - \(d^+\) is only firable in \(s_4\)
  - \(b^+\) always fires before \(d^+\)
- Concurrency reduction by imposing relative orderings
Relative timing

- Lazy Transition System (Cortadella, et al.)
  - $d^+$ is lazy w.r.t. $b^+$
  - Explicit distinction between enabling and firing
  - Conservative overestimation of the timed state space

GOAL of the thesis
GOAL of the thesis

“Model-checking-like” approach for the verification of safety properties in timed systems
- Safety properties (invariants): “an undesired behavior never occurs”
- Satisfiability of the properties depends on timing characteristics of the system
- Example: no spurious/undesired events in a circuit

Some problems:
- State explosion
- Time dimension
- Algorithmic complexity

Some solutions:
- Symbolic representation
- Relative timing
- Incremental approach

System model: Timed Transition Systems (TTS)
- Safety properties (modeled as Boolean invariants)
- Relative timing: Lazy Transition Systems (LzTS)
- LzTSs can be represented using BDDs \(\Rightarrow\) large systems

Iterative incremental refinement of the untimed domain:
- Localized analysis of property violations,
- Off-line absolute timing analysis, and
- Incorporation of Relative Timing constraints

Back-annotation: counterexample trace or sufficient relative timing constraints for correctness are reported
PART II

Formal verification with relative timing

- System model
- From absolute to relative timing
- State space refinement by timing constraints
- Overall verification approach
- Experimental results

System model

- Timed Transition system
  - Transition system
  - Min/Max delay bounds

\[
\begin{align*}
    d(a) &\in [1,2] \\
    d(b) &\in [1,2] \\
    d(c) &\in [2.5,3] \\
    d(g) &\in [0.5,0.5] \\
    d(d,x,y) &\in [0,\infty) \\
\end{align*}
\]
Safety properties

- Many properties can be expressed as relative orderings of events
- Example:
  - g must fire before d after having fired x
- "Good traces"
- "Bad traces"

System model

- Timed Transition system
  - Transition system
  - Min/Max delay bounds
    - d(a) ∈ [1,2]
    - d(b) ∈ [1,2]
    - d(c) ∈ [2.5,3]
    - d(g) ∈ [0.5,0.5]
    - d(d,x,y) ∈ [0,∞)
- Lazy Transition system
From absolute to relative timing

- Firing times depend on the enabling and the delays
- Enabling information can be extracted from a trace:
  - Event Structure (ES)
  - An ES covers other traces *modulo* concurrency
  - Key notion: *enabling compatibility*
- Timing analysis on an ES:
  - Relative timing relations $\Rightarrow$ Timed ES
- Main result: *only traces enabling compatible with the timed ES exist in the timed domain*
Enabling information

An event $e$ can only become enabled at the time another event $e'$ fires ($e'$ triggers $e$).

The order of the enabling implies the firing times of the events.
Timing consistent trace

Event structure from a trace

Time assignment to event firings such that:

\[ d_{\text{min}}(g) \leq t_6 - t_2 \leq d_{\text{max}}(g) \]

\[ \vdots \]
Enabling compatibility

Trace and event structure are enabling compatible.

Enabling compatibility
Timing analysis (time separation)

- McMillan, Dill (1992): min/max constraints in acyclic graphs
- Hulgaard, Burns (1994): max constraints for cyclic graphs with choice
- Chakraborty (1997): linear constraints and approximate methods

\[
\text{max } \tau(g) - \tau(d) = -2
\]
Enabling compatibility & Timing consistency

Theorem:
A trace is timing consistent iff it is an enabling compatible trace of the timed event structure.
State space refinement by timing constraints: enabling compatible product
Enabling compatibility

Not enabling compatible

State space partition
State space partition

Property: \( g \) must fire before \( d \) after having fired \( x \)

State space partition & Timing analysis

Timing analysis
State space partition & Timing analysis

Timing analysis

State space partition & Timing analysis

Timing analysis
State space partition & Timing analysis

Property: g must fire before d after having fired x

State space partition & Timing analysis

Timing analysis
Property: \( g \) must fire before \( d \) after having fired \( x \)
Overall verification approach

Verification approach

Symbolic state space exploration and failure detection
Verification approach: iterative refinement

- Failure trace
- Event structure
- Timing analysis
- Composition
Verification approach: iterative refinement
Verification approach: iterative refinement
Verification approach: back-annotation

Back-annotation: sufficient timing constraints

Verification approach: flow

Diagram:
- TTS
- TS + delay
- LzTS
- Failure search
  - no
  - yes
    - Correct
    - Counter example
- Timing consistent?
  - yes
  - no
- Causality extraction
- Timing analysis
- Timed ES
- Backannotation
- Timing constraint composition
Verification approach: convergence

Nodal points

- All cycles cut by nodal points
- Finite number of traces between nodal points
- Convergence and exact results guaranteed

Experimental results
Experiments: gate decompositions

- Complex-gate decompositions in *Speed-Independent* asynchronous circuits:
  - Robust hazard-free operation regardless of gate delays

Experiments: set-up

- Verification of asynchronous control circuits
- Library delays assigned to gates
- Closed system: circuit vs. specification

- Verification:
  - Input-output conformance w.r.t. the specification
  - Absence of internal hazards
## Experiments: results

<table>
<thead>
<tr>
<th>name</th>
<th>gates</th>
<th>un timed</th>
<th>failure</th>
<th>iters</th>
<th>correct</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbuf-read-cll</td>
<td>10</td>
<td>74</td>
<td>16</td>
<td>3</td>
<td>Y</td>
<td>1</td>
</tr>
<tr>
<td>rcv-setup</td>
<td>6</td>
<td>78</td>
<td>34</td>
<td>2</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>alloc-outbound</td>
<td>11</td>
<td>82</td>
<td>20</td>
<td>4</td>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>ebergen</td>
<td>8</td>
<td>83</td>
<td>22</td>
<td>1</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>mp-forward-pkt</td>
<td>8</td>
<td>186</td>
<td>70</td>
<td>8</td>
<td>Y</td>
<td>5</td>
</tr>
<tr>
<td>def</td>
<td>6</td>
<td>255</td>
<td>164</td>
<td>6</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>half</td>
<td>7</td>
<td>227</td>
<td>133</td>
<td>1</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>chu133</td>
<td>9</td>
<td>288</td>
<td>204</td>
<td>2</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>connect</td>
<td>12</td>
<td>408</td>
<td>244</td>
<td>10</td>
<td>N</td>
<td>12</td>
</tr>
<tr>
<td>nowick</td>
<td>10</td>
<td>510</td>
<td>292</td>
<td>4</td>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>chu150</td>
<td>8</td>
<td>520</td>
<td>339</td>
<td>3</td>
<td>N</td>
<td>1</td>
</tr>
<tr>
<td>sbuf-send-cll</td>
<td>13</td>
<td>1592</td>
<td>1081</td>
<td>18</td>
<td>N</td>
<td>54</td>
</tr>
<tr>
<td>ptt</td>
<td>8</td>
<td>2812</td>
<td>1841</td>
<td>2</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>sbuf-send-pkt2</td>
<td>13</td>
<td>4544</td>
<td>4044</td>
<td>13</td>
<td>N</td>
<td>103</td>
</tr>
<tr>
<td>vme</td>
<td>15</td>
<td>10664</td>
<td>8665</td>
<td>20</td>
<td>N</td>
<td>30</td>
</tr>
<tr>
<td>sbuf-ram-write</td>
<td>15</td>
<td>14016</td>
<td>12362</td>
<td>415</td>
<td>N</td>
<td>550</td>
</tr>
<tr>
<td>ram-read-sbuf</td>
<td>16</td>
<td>19328</td>
<td>17488</td>
<td>550</td>
<td>N</td>
<td>317</td>
</tr>
<tr>
<td>m1</td>
<td>16</td>
<td>11574</td>
<td>11574</td>
<td>317</td>
<td>N</td>
<td>127</td>
</tr>
<tr>
<td>mto</td>
<td>20</td>
<td>642291</td>
<td>2</td>
<td>N</td>
<td>48</td>
<td></td>
</tr>
<tr>
<td>send-bm</td>
<td>12</td>
<td>717961</td>
<td>3</td>
<td>N</td>
<td>185</td>
<td></td>
</tr>
<tr>
<td>trimos-send</td>
<td>24</td>
<td>1.8E6</td>
<td>1</td>
<td>N</td>
<td>127</td>
<td></td>
</tr>
<tr>
<td>mmu</td>
<td>22</td>
<td>5.2E6</td>
<td>3</td>
<td>N</td>
<td>480</td>
<td></td>
</tr>
</tbody>
</table>

## Experiments: results comparison

<table>
<thead>
<tr>
<th>name</th>
<th>un timed</th>
<th>iters</th>
<th>CPU</th>
</tr>
</thead>
<tbody>
<tr>
<td>sbuf-read-cll</td>
<td>74</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>rcv-setup</td>
<td>78</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>alloc-outbound</td>
<td>82</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>ebergen</td>
<td>83</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>mp-forward-pkt</td>
<td>186</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>def</td>
<td>255</td>
<td>6</td>
<td>2</td>
</tr>
<tr>
<td>half</td>
<td>227</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>chu133</td>
<td>288</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>connect</td>
<td>408</td>
<td>10</td>
<td>12</td>
</tr>
<tr>
<td>nowick</td>
<td>510</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>chu150</td>
<td>520</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>sbuf-send-cll</td>
<td>1592</td>
<td>18</td>
<td>54</td>
</tr>
<tr>
<td>ptt</td>
<td>2912</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>sbuf-send-pkt2</td>
<td>4544</td>
<td>19</td>
<td>103</td>
</tr>
<tr>
<td>vme</td>
<td>10568</td>
<td>21</td>
<td>30</td>
</tr>
<tr>
<td>sbuf-ram-write</td>
<td>14016</td>
<td>34</td>
<td>415</td>
</tr>
<tr>
<td>ram-read-sbuf</td>
<td>19328</td>
<td>36</td>
<td>550</td>
</tr>
<tr>
<td>m1</td>
<td>21026</td>
<td>29</td>
<td>317</td>
</tr>
<tr>
<td>mto</td>
<td>722304</td>
<td>2</td>
<td>46</td>
</tr>
<tr>
<td>send-bm</td>
<td>763208</td>
<td>3</td>
<td>185</td>
</tr>
<tr>
<td>trimos-send</td>
<td>2.1E6</td>
<td>1</td>
<td>127</td>
</tr>
<tr>
<td>mmu</td>
<td>5.6E6</td>
<td>3</td>
<td>480</td>
</tr>
</tbody>
</table>

### TranSyT
- PIII 866MHz
- 1GB RAM
- 256MB peak
- No specific strategy
- Back-annotation

### OpenKronos
(Bozga et al. 2002):
- Timed automata
- Discrete clocks as counters
- NDDs
- UltraSparc
- 2GB RAM !
- No back-annotation
PART III

Compositional verification

- IPCMOS architecture
- Verification of IPCMOS pipelines
  - Strategy
  - Abstractions
  - Assume-Guarantee and induction
  - Results

Goal

GOAL: Formal verification of a complex timed design (IPCMOS architecture)
IPCMOS architecture

General IPCMOS architecture

- **Pulse-based** asynchronous clocking technique for large devices operating at GHz frequencies
- Block-level interlocking scheme ⇒ scalable
- Schuster, et al. (IBM) ISSCC 2000
Linear IPCMOS pipeline architecture

- 2-phase request (VALID) and acknowledge (ACK) protocol
- **VALID**: data is available to the next block, mimics delay of the logic
- **ACK**: data received by the next block
- Performance up to 4GHz (year 2000 technology)
- 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**

- Transition relations:
  - $\text{En}(Y^+) : \neg Y \cdot \neg Z$
  - $\text{En}(Y^-) : Y \cdot \text{ACK}$
  - . . .

- Failure conditions:
  - Shortcut at $Y : \neg Z \cdot \text{ACK}$
  - . . .

**Reset circuit**

- Capture a positive pulse at $\text{ACK}$ from the next stage and produce a positive pulse at $\text{CLKR}$ and a negative pulse at $\text{CLKRN}$ 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 mimics the delay of the functional block controlled by the stage.

Verification of IPCMOS pipelines
Verification goal

Assuming data-path is correct, the pipeline is correct (spec.):

\[ S = \text{“Every data fed into the pipeline is acknowledged once and only once at every stage”} \]

\[ S \text{ is modeled by a deadlock condition plus correctness of CMOS circuits: no short-circuits, etc.} \]

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

Assume the abstractions are correct

Prove that the abstractions are correct to guarantee a sound analysis

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

A\textsubscript{in} and A\textsubscript{out} are untimed abstractions that hide the pulse-based behavior

**Assume**: pose verification in terms of: A\textsubscript{in} || A\textsubscript{out} \leq S

**Guarantee** soundness of the abstractions:
\[
\text{IN} \parallel I_1 \parallel \ldots \parallel I_n \parallel \text{OUT} \leq A\textsubscript{in} || A\textsubscript{out}
\]

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: results

<table>
<thead>
<tr>
<th>Expression</th>
<th>CPU time</th>
<th>Refinements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_{in} \parallel A_{out} \leq S$</td>
<td>1m</td>
<td>--</td>
</tr>
<tr>
<td>$A_{in} \parallel I \parallel OUT \leq A_{in} \parallel A_{out}$</td>
<td>28m</td>
<td>7</td>
</tr>
<tr>
<td>$IN \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out}$</td>
<td>9m</td>
<td>3</td>
</tr>
<tr>
<td>$A_{in} \parallel I \parallel A_{out} \leq A_{in} \parallel A_{out}$</td>
<td>10m</td>
<td>3</td>
</tr>
<tr>
<td>$IN \parallel I \parallel OUT \leq S$</td>
<td>35m</td>
<td>40</td>
</tr>
</tbody>
</table>

**CPU time** is measured in minutes (m).

**PART IV**

Conclusions and future work
Conclusions

- New technique for the verification of timed systems
  - Relative timing
  - Avoids generating the exact timed state space
  - Symbolic methods ⇒ bigger systems
- Key notion: enabling compatibility
  - Timing added on-demand: Incremental refinement
  - Back-annotation: sufficient conditions for correctness
- Combined with compositional reasoning to overcome complexity: IPCMOS case study

Publications


Future work

- Improvements:
  - Study sensibility to selection of traces, ESs, …
  - Consider OR-causality in ESs, timing analysis, …
  - Faster convergence and better back-annotation
  - Enabling compatible product: critical for performance

- Long-term:
  - Beyond safety properties, temporal logic, …
  - Automatic derivation of symbolic timing constraints
  - Hierarchical verification and compositional reasoning

Questions?