Components, Interfaces and Compositions: from SoCs to SOCs

Partha S. Roop

University of Auckland
Organization

- Significance of components and interfaces.
- Two recent frontiers – SoCs and SOCs.
- Key problems:
  - Component matching – refinement based and DES control based.
  - Component composition – converter / choreographer synthesis.
- Conclusions.
Acknowledgements

- Forced Simulation is joint work with A. Sowmya (UNSW), S. Ramesh (General Motors R&D) and the link to DES is with Robi Malik (Waikato).
- Local module checking and converter synthesis is joint work with Roopak Sinha (Postdoc) and Samik Basu (Iowa State).
- Web Services composition is joint work with Adeel Ali (PhD student), Ian Warren (Soft. Eng., Auckland) and Zeeshan Bhatti (PhD student)
“Simple? Yet, not a single person on the face of this earth knows how to make me.”

- Making of lead (graphite + clay)
- Making of body (cedar + lacquer)
- Eraser (rubber + factice + ...)
- Label (carbon + resin + ...)
- Ferrule (brass + zinc + ...)

Mass manufacturing

Structural assembly

Mechanical Assembly

Electronics

Quality control
A System-on-a-chip (SoC) Example

Consumer electronics revolution fuelled by SoCs

Embedded Systems

- Compliance to strict safety standards [IEC 61508, DO 178]

Timing/Functionality requirements

Service Oriented Computing

Internet Service Composition
Featuring The Future ...!
Related work

- Abstract Interfaces [Parnas’77]
- OO methodologies and UML
- Formal techniques:
  - IO Automata
  - Interface Automata
  - Interface Theories
  - Discrete controller synthesis
  - Module checking
  - Converter synthesis
Two key questions

- Question 1: specification matching / component adaptation (the “what” question).
- Question 2: component composition (the “how” question).
First Question:

Specification Matching –
“Can a given device automatically be adapted to implement a new function?”

Two Answers:
- Forced Simulation
- Supervisory Control
Coffee Brewer Example

Assume Given:
- Coffee Brewer device
- Can brew 4 or 8 cups of coffee
- Medium or strong

### Specification $F$

- States: $0, 1, 2, 3$
- Transitions:
  - $0 \xrightarrow{\text{error}} 1$
  - $0 \xrightarrow{\text{ready4m}} 2$
  - $2 \xrightarrow{\text{reset}} 0$
  - $0 \xrightarrow{\text{ready8m}} 3$

### Device $D$

- States: $0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10$
- Transitions:
  - $0 \xrightarrow{\text{strong} \land 8\text{cups}} 4$
  - $4 \xrightarrow{\text{breat}} 0$
  - $0 \xrightarrow{\text{default}} 1$
  - $1 \xrightarrow{\text{ready8m}} 2$
  - $2 \xrightarrow{\text{reset}} 0$
  - $0 \xrightarrow{\text{8cups}} 3$
  - $3 \xrightarrow{\text{reset}} 0$
  - $0 \xrightarrow{\text{error}} 5$
  - $5 \xrightarrow{\text{ready4m}} 6$
  - $6 \xrightarrow{\text{ready4s}} 7$
  - $7 \xrightarrow{\text{reset}} 0$
  - $0 \xrightarrow{\text{error}} 8$
  - $8 \xrightarrow{\text{ready8m}} 9$
  - $9 \xrightarrow{\text{ready8s}} 10$
  - $10 \xrightarrow{\text{replenish}} 0$
Disabling and Forcing

Assume Given:
- Coffee Brewer device
- Can brew 4 or 8 cups of coffee
- Medium or strong strength

Device $D$
- Brew:
  - $8_{cups} \land \neg 8_{cups}$
  - $\neg 8_{cups} \land 8_{cups}$
  - $\neg 8_{cups} \land \neg 8_{cups}$

Specifications $F$
- Brew:
  - $8_{cups}$
- Error:
  - $\neg 8_{cups}$
- Reset:
  - $\neg 8_{cups}$

Force Brew Switch

Disable Strength Switch

Brew Switch
- $[brew]$
An Adapter for the Coffee Brewer

Specification $F$

Adapter $A$

- States:
  - 0: Ready 0
  - 1: Error
  - 2: Ready 8
  - 3: Default

- Transitions:
  - From 0 to 1: Error
  - From 0 to 2: Ready 8
  - From 0 to 3: Default
  - From 1 to 1: Default
  - From 1 to 2: Ready 8
  - From 2 to 0: Reset
  - From 2 to 1: Error
  - From 2 to 2: Ready 8
  - From 3 to 0: Reset
  - From 3 to 1: Brew
  - From 3 to 2: Brew
  - From 3 to 3: Default
  - From 5 to 0: Error
  - From 5 to 1: Brew
  - From 5 to 2: Ready 4
  - From 5 to 3: Reset
  - From 7 to 0: Error
  - From 7 to 1: Brew
  - From 7 to 2: Ready 8
  - From 7 to 3: Reset
  - From 9 to 0: Error
Forced Composition

Let $A$ be an adapter and $D$ be a device. Define the **forced composition** $A \parallel D$ by

\[
(q_A) \xrightarrow{\alpha} (q'_A) \quad (q_D) \xrightarrow{\alpha} (q'_D)
\]

\[
(q_A, q_D) \xrightarrow{\tau} (q'_A, q'_D)
\]

\[
(q_A) \xrightarrow{\sigma} (q'_A) \quad (q_D) \xrightarrow{\sigma} (q'_D)
\]

\[
(q_A, q_D) \xrightarrow{\sigma} (q'_A, q'_D)
\]
Specification Matching Problem

Let $F$ be a specification and $D$ be a device. We say that

"$D$ can implement the function $F$", if there exists a well-formed and deterministic adapter $A$ such that

$$A // D \approx F$$
Theorem
There exists a well-formed and deterministic adapter $A$ such that

\[ A \parallel D \approx F \]

if and only if

\[ F \leq_{\text{fsim}} D \]
Condition for the existence of $A$

$R \subseteq Q_F \times Q_D \times \Sigma^*$ is a forced simulation relation between $F$ and $D$ provided:

1. $q_F^0 R^s q_D^0$ for some $s \in \Sigma^*$;
2. If $q_F R^\sigma q_D$ for $\sigma \in \Sigma$ and $s \in \Sigma^*$, then there exists $q_D' \in Q_D$ such that $q_D \xrightarrow{\sigma} q_D'$ and $q_F R^s q_D'$;
3. If $q_F R^\varepsilon q_D$ for all $\sigma \in \Sigma$ and all $q_F' \in Q_F$ such that $q_F \xrightarrow{\sigma} q_F'$, there exists $q_D' \in Q_D$ and $s \in \Sigma^*$ such that $q_D \xrightarrow{\sigma} q_D'$ and $q_F R^s q_D'$.

Start states must be related

Directly related

States related by a forcing sequence
Example

\[ R = \{(f_0, d_0, \alpha), (f_0, d_1, \varepsilon), (f_2, d_2, \varepsilon)\} \]
Another Solution

\[ R = \{(f_0, d_0, \varepsilon), (f_0, d_1, \alpha), (f_2, d_2, \varepsilon)\} \]
Supervisory Control Problem

Let $F$ be a specification and $P$ be a plant. We say that

“$F$ can be achieved by control of $P$”
“$F$ is controllable with respect to $P$”

if there exists a supervisor $S$ such that

$$L(S \parallel P) = L(F)$$
Creating a Plant from the Device

Assume Given:
- Coffee Brewer device that can
  - brew 4 or 8 cups of coffee
  - medium or strong

\[ A \parallel D = (A \parallel [D]) \setminus [\Sigma] \]
Least Restrictive vs. Well-Formed

Device $D$

Function $F$

Adapter $A_{fsim,2}$

Adapter $A_{supcon}$
## Comparison and Summary

<table>
<thead>
<tr>
<th>Feature</th>
<th>Forced simulation</th>
<th>Supervisory control</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relationship between $A$ and $F$</td>
<td>$A \parallel D \approx F$</td>
<td>$L(A \parallel D) \subseteq L(F)$</td>
</tr>
<tr>
<td>Well-formedness</td>
<td>guaranteed</td>
<td>requires additional steps</td>
</tr>
<tr>
<td>Forced cycles</td>
<td>not possible</td>
<td>may occur</td>
</tr>
<tr>
<td>Nonblocking</td>
<td>guaranteed</td>
<td>can be guaranteed</td>
</tr>
<tr>
<td>Uniqueness</td>
<td>solutions weakly bisimilar</td>
<td>unique least restrictive solution</td>
</tr>
<tr>
<td>Controllability</td>
<td>not considered</td>
<td>handled</td>
</tr>
<tr>
<td>Complexity</td>
<td>$O(</td>
<td>Q_F</td>
</tr>
</tbody>
</table>
Second Question:

- **Composition** – Design and develop systems from multiple independently developed components

  How to effectively address protocol-mismatches during composition?

**Answer:**
Relationship to convertibility verification.
Motivation


- “verifying functionality and timing at the system-level is probably the most difficult and important aspect of SoC design. .. For many teams, verification takes 50%-80% of the overall design effort"

- “the low-level interfaces do not work; for example, a handshake signal inverted”
Suggested design flow

1. Requirements and Specification
   - System Specification
   - Initial Requirements (Boiler Plates)

2. Behavioural Model

3. Refine & Test

4. Hardware/Software Partitioning
   - Hardware architecture model
   - Prototype software

5. Prototype software

6. Co-simulation and protocol compatibility checking

7. Hardware (HW) specification
   - Block 1 Spec
   - Block n Spec

7. Software (SW) specification

8. System Level Formal Verification
   - SW IP Interfaces
   - HW IP Interfaces
Solution Mechanism

- Converter-based protocol conversion
  - Develop a converter: acts as a mediator between two components with mismatched protocols

Goal: Compose $P_1$ and $P_2$ to realize the Specification
Solution Mechanism

Converter-based protocol conversion
- Develop a converter: acts as a mediator between two components with mismatched protocols

Goal: Compose $P_1$ and $P_2$ to realize the Specification
Solution Mechanism

- Converter-based protocol conversion
  - Develop a converter: acts as a mediator between two components with mismatched protocols

**Solution:** Converter addresses mismatches
Set-top box

Challenges:
• Multi-clock
• Differing data-widths
• Control signals mismatch
Converter

Key control

Video decoder

PAL/NTSC Encoder

Converter

SoC

(Uncontrollable Inputs)
How about service composition?
Composition Framework

- WSDL Generator
- LTS Encapsulated Service Models
- User Guided Data Connections
- Goal Specification
- Composite Service

Parallel Composition
### Related Work

<table>
<thead>
<tr>
<th>Approach</th>
<th>Model</th>
<th>Spec</th>
<th>Multiple Protocols</th>
<th>Algorithm</th>
<th>UE</th>
<th>Buffering</th>
<th>Data</th>
<th>Multiclock</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avnit et al.’08</td>
<td>SPA</td>
<td>nil</td>
<td>no</td>
<td>refinement</td>
<td>no</td>
<td>yes</td>
<td>limited</td>
<td>yes</td>
</tr>
<tr>
<td>D’Silva et al.’04</td>
<td>SPA</td>
<td>Nil</td>
<td>no</td>
<td>Refinement</td>
<td>no</td>
<td>yes</td>
<td>limited</td>
<td>yes</td>
</tr>
<tr>
<td>Passerone et al.’02</td>
<td>LTS</td>
<td>LTS</td>
<td>no</td>
<td>Game-theoretic</td>
<td>no</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Kumar et al.’97</td>
<td>LTS</td>
<td>LTS</td>
<td>no</td>
<td>Supervisory Control</td>
<td>yes</td>
<td>no</td>
<td>no</td>
<td>no</td>
</tr>
<tr>
<td>Tivoli et al.’08</td>
<td>LTS</td>
<td>Nil</td>
<td>Yes</td>
<td>Coverability-analysis</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>Our Approach</td>
<td>LTS</td>
<td>CTL</td>
<td>yes</td>
<td>Model checking</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
</tr>
</tbody>
</table>
## Related Work – Service Composition

<table>
<thead>
<tr>
<th>Approach</th>
<th>Input Services</th>
<th>Data</th>
<th>Behaviour</th>
<th>Composition Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Type</td>
<td>Auto Model</td>
<td>Model</td>
<td>Flow</td>
</tr>
<tr>
<td>Mitra et al.</td>
<td>Syntactic</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>ASTRO</td>
<td>Syntactic</td>
<td>-</td>
<td>DataNet</td>
<td>+</td>
</tr>
<tr>
<td>Berardi et al.</td>
<td>Syntactic</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Lecue et al.</td>
<td>Semantic</td>
<td>-</td>
<td>Schema Graph</td>
<td>+</td>
</tr>
<tr>
<td>Proposed</td>
<td>Syntactic</td>
<td>+</td>
<td>Schema Graph</td>
<td>+</td>
</tr>
</tbody>
</table>
Types of Protocol Mismatch

- **Control-signal mismatch**
- Data mismatch
- Clock mismatch
Ref: Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin
Passerone, Luca de Alfaro, Thomas A. Henzinger and Alberto L. Sangiovanni-Vincentelli, ICCAD’02

**Protocol Model**

- **KS** = (AP, S, s₀, Σ, R, L)
  - AP: atomic propositions, S: set of states,
  - s₀ ∈ S: start state, Σ: transition labels,
  - R: Transitions, L: labels states to propositions.

---

Handshake Producer

Serial Consumer

primed: input signal
unprimed: output signal
Protocol Model Composition

Ref: Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin
Passerone, Luca de Alfaro, Thomas A. Henzinger and Alberto L. Sangiovanni-Vincentelli, ICCAD’02
Specification Language

- **CTL Syntax**

\[ \Phi \rightarrow \text{tt} \mid P \mid \neg P \mid \Phi \lor \Phi \]

- All/some successors satisfy \( \Phi \)

- All/some reachable states satisfy \( \Phi \)

- Along all paths \( \Phi \) is satisfied until \( \Psi \)

- Along some path \( \Phi \) is satisfied until \( \Psi \)
Protocol Model Properties

Input cannot be made before corresponding output:

1. $AG[s_0,t_0 \Rightarrow AX(\neg s_1,t_1)]$
2. $AG[s_1,t_1 \Rightarrow AX(\neg s_0,t_0)]$

$(s_0,t_0)$: for a action
$(s_1,t_1)$: for b action
Protocol Model Properties

Input cannot be made before corresponding output:

1. $\text{AG}[s_0, t_0 \Rightarrow \text{AX} \neg(\neg s_1, t_1)]$
2. $\text{AG}[s_1, t_1 \Rightarrow \text{AX} \neg(\neg s_0, t_0)]$

Output of $b/a$ is not allowed before $a/b$ is received:

1. $\text{AG}[s_1, t_0 \Rightarrow \text{AX} \neg(s_0t_0)]$
2. $\text{AG}[s_0, t_1 \Rightarrow \text{AX} \neg(s_1t_1)]$

Ref: *Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin*
Passerone, Luca de Alfaro, Thomas A. Henzinger and Alberto L. Sangiovanni-Vincentelli, ICCAD’02
Protocol Model Properties

**Input cannot be made before corresponding output:**

1. \( \text{AG}[s_0, t_0 \Rightarrow \text{AX}\neg(\neg s_1, t_1)] \)
2. \( \text{AG}[s_1, t_1 \Rightarrow \text{AX}\neg(\neg s_0, t_0)] \)

**Output of b/a is not allowed before a/b is received:**

1. \( \text{AG}[s_1, t_0 \Rightarrow \text{AX}\neg(s_0 t_0)] \)
2. \( \text{AG}[s_0, t_1 \Rightarrow \text{AX}\neg(s_1 t_1)] \)

Ref: *Convertibility Verification and Converter Synthesis: Two Faces of the Same Coin*
Passerone, Luca de Alfaro, Thomas A. Henzinger and Alberto L. Sangiovanni-Vincentelli, ICCAD’02
Lock-Step Composition

- Converter-based solution
  - Protocol-models move if and only if the converter allows that move
  - Converter cannot block any outputs

- Let \( c_i \) be composed with \( (s_i, t_i) \) then \( (s_i, t_i) \xrightarrow{a} (s_j, t_j) \) is allowed if and only if \( c_i \) can move on \( (a') \)
Converter Synthesis

\[(s,t)/c \models \Phi\]

\[(s_1,t_1)/c_1 \models \Phi_1 \quad (s_2,t_2)/c_2 \models \Phi_2 \ldots \quad (s_k,t_k)/c_k \models \Phi_k\]

- The antecedent holds if and only if the consequents hold
- Local, top-down approach similar to tableau-based CTL model checking
### Tableau Rules

\[ (s,t) // c \models \Psi \]

\[ \exists \pi \subseteq \Pi : \forall \sigma \in \pi : (s_\sigma, t_\sigma) // c_\sigma \models \Psi_{AX} \]

\[
\begin{aligned}
\Psi_{AX} &= \{ \Phi \mid AX\Phi \in \Psi \} \\
\Pi &= \{ \sigma \mid (s,t) \overset{\sigma}{\rightarrow} (s_\sigma, t_\sigma) \} \\
c_\sigma : c \overset{\sigma'}{\rightarrow} c_\sigma \quad \text{and} \quad D(\sigma, \sigma')
\end{aligned}
\]

\( \Psi \) only contains formulas of the form \( AX\Phi \)
1. Identify the set of possible transitions from \((s,t)\): \( \Pi \)
2. Enable a subset of possible transitions using converter:
   \[ c \rightarrow c_\sigma \]
3. All enabled transition leads to states satisfying \( \Phi \)'s

**Enabled transition set must include all possible output transitions. Also, resulting machine has to be responsive to \( T \) input.**
Example

\(\Phi_1 = s_0t_0 \Rightarrow A\neg (\neg s_1, t_1)\)
\(\Phi_2 = s_1t_1 \Rightarrow A\neg (\neg s_0, t_0)\)
\(\Phi_3 = s_1t_0 \Rightarrow A\neg (s_0t_0)\)

\(s_0t_0//c_0 \vdash \{ AG[s_0t_0\Rightarrow A\neg (\neg s_1, t_1)], AG[s_1t_1\Rightarrow A\neg (\neg s_0, t_0)], AG[s_1t_0\Rightarrow A\neg (s_0t_0)] \} \)
Example

$\phi_1 = s_0t_0 \Rightarrow AX \neg(s_1t_1)$

$\phi_2 = s_1t_1 \Rightarrow AX \neg(s_0t_0)$

$\phi_3 = s_1t_0 \Rightarrow AX \neg(s_0t_0)$

$s_0t_0/c_0 \vdash \{AG[\phi_1], AG[\phi_2], AG[\phi_3] \}$
Example

\[ \Phi_1 = s_0t_0 \Rightarrow AX(\neg s_1t_1) \]
\[ \Phi_2 = s_1t_1 \Rightarrow AX(\neg s_0t_0) \]
\[ \Phi_3 = s_1t_0 \Rightarrow AX(\neg s_0t_0) \]

\[ s_0t_0//c_0 \models \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \]

\[ s_0t_0//c_0 \models \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \]
Example

\( \Phi_1 = s_0t_0 \Rightarrow AX\neg(s_1,t_1) \)
\( \Phi_2 = s_1t_1 \Rightarrow AX\neg(s_0,t_0) \)
\( \Phi_3 = s_1t_0 \Rightarrow AX\neg(s_0t_0) \)

\[
\begin{align*}
s_0t_0/c_0 & \models \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \\
s_0t_0/c_0 & \models \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \\
s_0t_0/c_0 & \models \{ AXAG[\Phi_1], AX\neg(s_1,t_1), AXAG[\Phi_2], AXAG[\Phi_3] \}
\end{align*}
\]
Example

\[ \Phi_1 = s_0 t_0 \Rightarrow AX \neg (\neg s_1, t_1) \]
\[ \Phi_2 = s_1 t_1 \Rightarrow AX \neg (\neg s_0, t_0) \]
\[ \Phi_3 = s_1 t_0 \Rightarrow AX \neg (s_0 t_0) \]

\[
\begin{align*}
s_0 t_0 \sslash c_0 & \models \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \\
s_0 t_0 \sslash c_0 & \models \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \\
s_0 t_0 \sslash c_0 & \models \{ AXAG[\Phi_1], AX \neg (\neg s_1, t_1), AXAG[\Phi_2], AXAG[\Phi_3] \} \\
s_0 t_0 \sslash c_0' & \models \{ AG[\Phi_1], (\neg s_1, t_1), AG[\Phi_2], AG[\Phi_3] \} \\
s_0 t_1 \sslash c_1 & \models \{ AG[\Phi_1], (\neg s_1, t_1), AG[\Phi_2], AG[\Phi_3] \} \\
s_1 t_1 \sslash c_2 & \models \{ AG[\Phi_1], (\neg s_1, t_1), AG[\Phi_2], AG[\Phi_3] \} \\
s_1 t_0 \sslash c_3 & \models \{ AG[\Phi_1], (\neg s_1, t_1), AG[\Phi_2], AG[\Phi_3] \}
\end{align*}
\]
\( \Phi_1 = s_0t_0 \Rightarrow AX(\neg s_1,t_1) \)
\( \Phi_2 = s_1t_1 \Rightarrow AX(\neg s_0,t_0) \)
\( \Phi_3 = s_1t_0 \Rightarrow AX(\neg s_0t_0) \)

**Example**

\[
\begin{align*}
s_0t_0/c_0 & \models \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \\
\{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \\
\{ AXAG[\Phi_1], AX(\neg s_1t_1), AXAG[\Phi_2], AXAG[\Phi_3] \} \\
\{ AG[\Phi_1], \neg (\neg s_1t_1), AG[\Phi_2], AG[\Phi_3] \}
\end{align*}
\]
\[ \Phi_1 = s_0t_0 \Rightarrow AX\neg(s_1t_1) \]
\[ \Phi_2 = s_1t_1 \Rightarrow AX\neg(s_0t_0) \]
\[ \Phi_3 = s_1t_0 \Rightarrow AX\neg(s_0t_0) \]

**Example**

\[
\begin{align*}
s_{00}/c_0 & \vdash \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \\
s_{00}/c_0' & \vdash \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \\
s_{00}/c_0 & \vdash \{ AXAG[\Phi_1], AX\neg(s_1t_1), AXAG[\Phi_2], AXAG[\Phi_3] \} \\
s_{00}/c_0' & \vdash \{ AG[\Phi_1], \neg(s_1t_1), AG[\Phi_2], AG[\Phi_3] \} \\
s_{00}/c_0' & \vdash \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \}
\end{align*}
\]

Same formula state pair
Example

\[ \Phi_1 = s0t0 \Rightarrow AX(\neg s1,t1) \]
\[ \Phi_2 = s1t1 \Rightarrow AX(\neg s0,t0) \]
\[ \Phi_3 = s1t0 \Rightarrow AX(\neg s0t0) \]

\[ s0t0//c0 \equiv \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \]
\[ s0t0//c0 \equiv \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \]
\[ s0t0//c0 \equiv \{ AXAG[\Phi_1], AX(\neg s1,t1), AXAG[\Phi_2], AXAG[\Phi_3] \} \]
\[ s0t0//c0' \equiv \{ AG[\Phi_1], \neg(\neg s1,t1), AG[\Phi_2], AG[\Phi_3] \} \]
\[ s0t0//c0' \equiv \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \]
Example

\[ \Phi_1 = s_0t_0 \Rightarrow AX(\neg s_1,t_1) \]
\[ \Phi_2 = s_1t_1 \Rightarrow AX(\neg s_0,t_0) \]
\[ \Phi_3 = s_1t_0 \Rightarrow AX(\neg s_0t_0) \]

<table>
<thead>
<tr>
<th>State</th>
<th>Guard Condition</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>s0t0</td>
<td>{AG[\Phi_1], AG[\Phi_2], AG[\Phi_3]}</td>
<td></td>
</tr>
<tr>
<td>s0t0</td>
<td>{AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3}</td>
<td></td>
</tr>
<tr>
<td>s0t0</td>
<td>{AXAG[\Phi_1], AX(\neg s_1,t_1), AXAG[\Phi_2], AXAG[\Phi_3]}</td>
<td></td>
</tr>
<tr>
<td>s0t0/c0'</td>
<td>{AG[\Phi_1], \neg(\neg s_1,t_1), AG[\Phi_2], AG[\Phi_3]}</td>
<td>SUCCESS T in producer allowed</td>
</tr>
<tr>
<td>s0t1</td>
<td>{AG[\Phi_1], \neg(\neg s_1,t_1), AG[\Phi_2], AG[\Phi_3]}</td>
<td></td>
</tr>
<tr>
<td>s1t1</td>
<td>{AG[\Phi_1], \neg(\neg s_1,t_1), AG[\Phi_2], AG[\Phi_3]}</td>
<td></td>
</tr>
<tr>
<td>s1t0</td>
<td>{AG[\Phi_1], \neg(\neg s_1,t_1), AG[\Phi_2], AG[\Phi_3]}</td>
<td></td>
</tr>
</tbody>
</table>
Example

\(\Phi_1 = s_0t_0 \Rightarrow AX(\neg s_1,t_1)\)
\(\Phi_2 = s_1t_1 \Rightarrow AX(\neg s_0,t_0)\)
\(\Phi_3 = s_1t_0 \Rightarrow AX(\neg s_0t_0)\)

\[s_0t_0//c_0 \vdash \{AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \}\]
\[s_0t_0//c_0 \vdash \{AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \}\]
\[s_0t_0//c_0 \vdash \{AXAG[\Phi_1], AX(\neg s_1,t_1), AXAG[\Phi_2], AXAG[\Phi_3] \}\]
\[s_0t_1//c_1 \vdash \{AG[\Phi_1], \neg(\neg s_1,t_1), AG[\Phi_2], AG[\Phi_3] \}\]
Example

\[ \Phi_1 = s_0 t_0 \Rightarrow AX \neg(s_1 t_1) \]
\[ \Phi_2 = s_1 t_1 \Rightarrow AX \neg(s_0 t_0) \]
\[ \Phi_3 = s_1 t_0 \Rightarrow AX \neg(s_0 t_0) \]

s0t0//c0 ⊨ \{AG[Φ1], AG[Φ2], AG[Φ3] \}

s0t0//c0 ⊨ \{AXAG[Φ1], Φ1, AXAG[Φ2], Φ2, AXAG[Φ3], Φ3 \}

s0t0//c0 ⊨ \{AXAG[Φ1], AX\neg(s_1 t_1), AXAG[Φ2], AXAG[Φ3] \}

s0t1//c1 ⊨ \{AG[Φ1], \neg(s_1 t_1), AG[Φ2], AG[Φ3] \}

FAIL
Example

\[ \Phi_1 = \text{s0t0} \Rightarrow \text{AX} \neg (\neg \text{s1,t1}) \]
\[ \Phi_2 = \text{s1t1} \Rightarrow \text{AX} \neg (\neg \text{s0,t0}) \]
\[ \Phi_3 = \text{s1t0} \Rightarrow \text{AX} \neg (\text{s0t0}) \]

\[ \text{s0t0//c0} \models \{ \text{AG}[\Phi_1], \text{AG}[\Phi_2], \text{AG}[\Phi_3] \} \]
\[ \text{s0t0//c0} \models \{ \text{AXAG}[\Phi_1], \Phi_1, \text{AXAG}[\Phi_2], \Phi_2, \text{AXAG}[\Phi_3], \Phi_3 \} \]
\[ \text{s0t0//c0} \models \{ \text{AXAG}[\Phi_1], \text{AX} \neg (\neg \text{s1,t1}), \text{AXAG}[\Phi_2], \text{AXAG}[\Phi_3] \} \]
\[ \text{s0t0//c0'} \models \{ \text{AG}[\Phi_1], \neg (\neg \text{s1,t1}), \text{AG}[\Phi_2], \text{AG}[\Phi_3] \} \quad \text{SUCCESS T in producer allowed} \]
\[ \text{s0t1//c1} \models \{ \text{AG}[\Phi_1], \neg (\neg \text{s1,t1}), \text{AG}[\Phi_2], \text{AG}[\Phi_3] \} \quad \text{FAIL T in producer blocked} \]
\[ \text{s1t1//c2} \models \{ \text{AG}[\Phi_1], \neg (\neg \text{s1,t1}), \text{AG}[\Phi_2], \text{AG}[\Phi_3] \} \]
\[ \text{s1t0//c3} \models \{ \text{AG}[\Phi_1], \neg (\neg \text{s1,t1}), \text{AG}[\Phi_2], \text{AG}[\Phi_3] \} \]
Example

\[ \Phi_1 = s_{0t0} \Rightarrow AX(\neg s_{1t1}) \]
\[ \Phi_2 = s_{1t1} \Rightarrow AX(\neg s_{0t0}) \]
\[ \Phi_3 = s_{1t0} \Rightarrow AX(\neg s_{0t0}) \]

\[
\begin{align*}
s_{0t0}//c_0 & \vdash \{ AG[\Phi_1], AG[\Phi_2], AG[\Phi_3] \} \\
s_{0t0}//c_0 & \vdash \{ AXAG[\Phi_1], \Phi_1, AXAG[\Phi_2], \Phi_2, AXAG[\Phi_3], \Phi_3 \} \\
s_{0t0}//c_0 & \vdash \{ AXAG[\Phi_1], AX(\neg s_{1t1}), AXAG[\Phi_2], AXAG[\Phi_3] \} \\
s_{1t1}//c_2 & \vdash \{ AG[\Phi_1], \neg(\neg s_{1t1}), AG[\Phi_2], AG[\Phi_3] \} \\
\end{align*}
\]
Example

\[ \Phi_1 = s_0t_0 \Rightarrow AX\neg(s_1,t_1) \]
\[ \Phi_2 = s_1t_1 \Rightarrow AX\neg(s_0,t_0) \]
\[ \Phi_3 = s_1t_0 \Rightarrow AX\neg(s_0t_0) \]
Example
Example
Example

T' → t0
a' → t1

s0 → s0
T' → T0
b' → s1

aT' → bT'
a' → t0
bb' → s1
t0 → t1

Ta' → Tb'
ba' → ab'
as1 → as0

T' T → c0
T' b → c34
a' T b'a → a' b

Converter
Types of Protocol Mismatch

- Control mismatches
- Data Mismatches
- Clock mismatch
Conclusion

- Two key problems
  - Component selection / matching
  - Component composition
  - Both problems solved in the context of SoCs and SOC.
- Key issues considered:
  - Control mismatches
  - Data-width / types
  - Clock
- Future work: Incremental design
References

The following slides discuss tableau construction to deal with data-width mismatches in SoCs followed by data-type mismatches in SOCs.
Complexity

\[ O(|I| \times 2^{|S|} \times 2^{|\Phi|} \times 2^{|E|}) \]

- \(|I|\) is the size of the set of all counter valuations:
  - For 1 counter \(C\) with range \([0,R]\), there are \(R+3\) valuations (\(R+1\) valid values, 2 invalid)
  - For \(n\) counters where each counter \(C_i\)’s range is \([0,R_i]\), \(|I| = (R_1+3) \times \ldots \times (R_n+3)\).
Complexity

\[ O(|I| \times 2^{|S|} \times 2^{|\Psi|} \times 2^{|E|}) \]

- \(|S|\) is the size of the synchronous parallel composition of all IPs.
- \(|\Psi|\) is the size of the formula set \(\Psi\).
- \(|E|\) is the size of the set of signals that can be buffered by the converter.
Introducing Data Counters

- $P_1$ and $P_2$ communicate using a 32-bit data buffer.
- $P_1$ writes 16-bit data ($DOut_{16}$) while $P_2$ reads 32-bit data ($DIn_{32}$).
Introducing Data Counters

- Data mismatches are possible:
  - P1 may write data when buffer is full (overflow).
  - P2 may read data when buffer is empty (underflow).

- Converter must ensure that the above situations are avoided.
Introducing Data Counters

- We introduce a **data counter** $C$, which is used by the converter to keep track of the number of bits contained in the data buffer after each transition in the system. $C$ is initialized to 0.

- Whenever a $\text{DOut}_{16}$ is encountered, $C$ is incremented by 16.

- Whenever a $\text{DIn}_{32}$ is encountered, $C$ is decremented by 32.
Introducing Data Counters

- The following CTL specification is used to ensure that counter remains within bounds

\[ AG \ (0 \leq c \leq 32) \]
Processing Data Counters

- **Init** → **DOut_{16}**
  - \( c = 0 \)

- **DOut_{16}** → **Wait**
  - \( c = 16 \)
  - \( c = 32 \)

- **Wait** → **DIn_{32}**
  - \( c = 32 \)
  - \( c = 0 \)

- **DOut_{16}** (loop)
  - \( c = 48 \)
STEP-4: CTL Specifications

- **AG EF DOut\(_{16}\), AG EF DIn\(_{32}\):** There must always exist a reachable state in the system where \(P_1\) can write data (\(P_2\) can read data).

- **AG AF (Idle\(_S\) \& Idle\(_T\) \& C=0):** The protocols must always eventually reset to a state where the data buffer is empty.
STEP 5 – Model Checking

- Given the protocol composition and a set of properties, we can use tableau-construction as before to generate a converter.
Example

$\Phi_1 = AG (0 \leq c \leq 32)$
$\Phi_2 = AG EF DOut_{16}$
$\Phi_3 = AG EF DIn_{32}$
$\Phi_4 = AG AF (Idle_S \land Idle_T \land c=0)$

$C0/s0t0ca0 \vdash \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$
Example

\[ \Phi_1 = AG (0 \leq c \leq 32) \]
\[ \Phi_2 = AG EF DOut_{16} \]
\[ \Phi_3 = AG EF DIn_{32} \]
\[ \Phi_4 = AG AF (Idle_S \land Idle_T \land c=0) \]

C0//s0t0ca0 \models \{ \Phi_1, \Phi_2, \Phi_3, \Phi_4 \}

C0//s0t0ca0 \models \{(0 \leq c \leq 32), AX \Phi_1, EF DOut_{16}, AX \Phi_2, AX \Phi_3, EF DIn_{32}, AX \Phi_4, AF (Idle_S \land Idle_T \land c=0)\}
Example

\[ \Phi_1 = AG (0 \leq C \leq 32) \]
\[ \Phi_2 = AG EF DOut_{16} \]
\[ \Phi_3 = AG EF DIn_{32} \]
\[ \Phi_4 = AG AF (Idle_S \land Idle_T \land C=0) \]

C0//s0t0ca0 ⊨ {Φ1, Φ2, Φ3, Φ4}

C0//s0t0ca0 ⊨ { (0 ≤ C ≤ 32), AX Φ1, EF DOut_{16}, AX Φ2, AX Φ3, EF DIn_{32}, AX Φ4, AF (Idle_S \land Idle_T \land C=0) }  

UNR tableau rule

C0//s0t0ca0 ⊨ { AX Φ1, DOut_{16} \lor EXEF DOut_{16}, AX Φ2, AX Φ3, DIn_{32} \lor EXEF DIn_{32}, AX Φ4, (Idle_S \land Idle_T \land C=0) \lor AX AF (Idle_S \land Idle_T \land C=0) }
Example

\[ \Phi_1 = AG \{0 \leq C \leq 32\} \]
\[ \Phi_2 = AG \text{EF} \text{DOut}_{16} \]
\[ \Phi_3 = AG \text{EF} \text{DIn}_{32} \]
\[ \Phi_4 = AG \text{AF} (\text{Idle}_S \land \text{Idle}_T \land C=0) \]

\[ C_0 = 0 \]
\[ \text{Buf} = {} \]

\[ C_0/s_0/t_0/c_0 = \{ \Phi_1, \Phi_2, \Phi_3, \Phi_4 \} \]

\[ C_0/s_0/t_0/c_0 \models \{(0 \leq C \leq 32), AX \Phi_1, EF \text{DOut}_{16}, AX \Phi_2, AX \Phi_3, EF \text{DIn}_{32}, AX \Phi_4, AF (\text{Idle}_S \land \text{Idle}_T \land C=0)\} \]

\[ C_0/s_0/t_0/c_0 \models \{AX \Phi_1, \text{DOut}_{16} \lor EXEF \text{DOut}_{16}, AX \Phi_2, AX \Phi_3, \text{DIn}_{32} \lor EX \text{EF \ DIn}_{32}, AX \Phi_4, (\text{Idle}_S \land \text{Idle}_T \land C=0) \lor AX AF (\text{Idle}_S \land \text{Idle}_T \land C=0)\} \]

\[ C_0/s_0/t_0/c_0 \models \{AX \Phi_1, EXEF \text{DOut}_{16}, AX \Phi_2, AX \Phi_3, EX \text{EF \ DIn}_{32}, AX \Phi_4, (\text{Idle}_S \land \text{Idle}_T \land C=0)\} \]
Example

$\Phi_1 = \text{AG } (0 \leq c \leq 32)$
$\Phi_2 = \text{AG EF DOut}_{16}$
$\Phi_3 = \text{AG EF DIn}_{32}$
$\Phi_4 = \text{AG AF (Idle}_S \land \text{Idle}_T \land c=0)$

C0//s0t0ca0 $\vdash \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$

C0//s0t0ca0 $\vdash \{(0 \leq c \leq 32), \text{AX } \Phi_1, \text{EF DOut}_{16}, \text{AX } \Phi_2, \text{AX } \Phi_3, \text{EF DIn}_{32}, \text{AX } \Phi_4,$
$\text{AF (Idle}_S \land \text{Idle}_T \land c=0)\}$

C0//s0t0ca0 $\vdash \{\text{AX } \Phi_1, \text{DOut}_{16} \lor \text{EXEF DOut}_{16}, \text{AX } \Phi_2, \text{AX } \Phi_3, \text{DIn}_{32} \lor \text{EX EF DIn}_{32}, \text{AX } \Phi_4,$
$\text{(Idle}_S \land \text{Idle}_T \land c=0) \lor \text{AX AF (Idle}_S \land \text{Idle}_T \land c=0)\}$

C0//s0t0ca0 $\vdash \{\text{AX } \Phi_1, \text{EXEF DOut}_{16}, \text{AX } \Phi_2, \text{AX } \Phi_3, \text{EX EF DIn}_{32}, \text{AX } \Phi_4,$
$\text{(Idle}_S \land \text{Idle}_T \land c=0)\}$

C0//s0t0ca0 $\vdash \{\text{AX } \Phi_1, \text{EXEF DOut}_{16}, \text{AX } \Phi_2, \text{AX } \Phi_3, \text{EX EF DIn}_{32}, \text{AX } \Phi_4\}$
Example

$\Phi_1 = AG \left(0 \leq c \leq 32\right)$
$\Phi_2 = AG \ EF \ DOut_{16}$
$\Phi_3 = AG \ EF \ DIn_{32}$
$\Phi_4 = AG \ AF \ (Idle_S \land Idle_T \land c=0)$

$\Psi_{AX} = \{ \Phi_1, \Phi_2, \Phi_3, \Phi_4 \}$
$\Psi_{EX} = \{ EF \ DOut_{16}, EF \ DIn_{32} \}$

$C0//s0t0ca0 \vdash \{ AX \ \Phi_1, \ EXEF \ DOut_{16}, AX \ \Phi_2, AX \ \Phi_3, EX \ EF \ DIn_{32}, AX \ \Phi_4 \}$
Example

$\Phi_1 = AG (0 \leq c \leq 32)$
$\Phi_2 = AG EF DOut_{16}$
$\Phi_3 = AG EF DIn_{32}$
$\Phi_4 = AG AF (Idle_S \land Idle_T \land c=0)$

$C_0//s_0t_0ca_0 \models \{AX \ \Phi_1 , \ EXEF \ DOut_{16} , \ AX \ \Phi_2 , \ AX \ \Phi_3 , \ EX \ EF \ DIn_{32} , \ AX \ \Phi_4\}$

$\Psi_{AX} = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$
$\Psi_{EX} = \{EF \ DOut_{16}, \ EF \ DIn_{32}\}$

$\Pi = \{(s_0,t_0,ca_1), (s_0,t_1,ca_1)\}$

$\pi \subseteq \Pi = \{(s_0,t_0,ca_1), (s_0,t_1,ca_1)\}$
Example

$\Phi_1 = \text{AG } (0 \leq c \leq 32)$
$\Phi_2 = \text{AG } EF \text{ DO}_{16}$
$\Phi_3 = \text{AG } EF \text{ DI}_{32}$
$\Phi_4 = \text{AG } AF (\text{Idle}_S \land \text{Idle}_T \land c=0)$

$C_0//s_0t_0ca_0 \models \{ \text{AX } \Phi_1, \text{EX } EF \text{ DO}_{16}, \text{AX } \Phi_2, \text{AX } \Phi_3, \text{EX } EF \text{ DI}_{32}, \text{AX } \Phi_4 \}$

$\Psi_{AX} = \{ \Phi_1, \Phi_2, \Phi_3, \Phi_4 \}$
$\Psi_{EX} = \{ \text{EF } \text{ DO}_{16}, \text{EF } \text{ DI}_{32} \}$

$\Pi = \{ (s_0,t_0,ca_1), (s_0,t_1,ca_1) \}$
$\pi \subseteq \Pi = \{ (s_0,t_0,ca_1), (s_0, \text{X}) \}$

• Signal a is not present in buffers.
• Transition to $(s_0,t_1,ca_1)$ will lead to counter value to become negative.
Example

$\Phi_1 = AG \ (0 \leq c \leq 32)$
$\Phi_2 = AG \ EF \ DOut_{16}$
$\Phi_3 = AG \ EF \ DIn_{32}$
$\Phi_4 = AG \ AF \ (Idle_S \land Idle_T \land c=0)$

$C_0/s_0 t_0 c_{a_0}$ \implies \{AX \ \Phi_1, \ EXEF \ DOut_{16}, \ AX \ \Phi_2, \ AX \ \Phi_3, \ EX \ EF \ DIn_{32}, \ AX \ \Phi_4\}$

$\Psi_{AX} = \{\Phi_1, \Phi_2, \Phi_3, \Phi_4\}$
$\Psi_{EX} = \{EF \ DOut_{16}, \ EF \ DIn_{32}\}$

$\Pi = \{(s_0,t_0,c_{a_1}), (s_0,t_1,c_{a_1})\}$

$\pi \subseteq \Pi = \{(s_0,t_0,c_{a_1})\}$
Example

\( \Phi_1 = AG (0 \leq c \leq 32) \)
\( \Phi_2 = AG EF DOut_{16} \)
\( \Phi_3 = AG EF DIn_{32} \)
\( \Phi_4 = AG AF (Idle_S \land Idle_T \land c=0) \)

\( \Psi_{AX} = \{ \Phi_1, \Phi_2, \Phi_3, \Phi_4 \} \)
\( \Psi_{EX} = \{ EF DOut_{16}, EF DIn_{32} \} \)
\( \Pi = \{(s_0,t_0,ca_1), (s_0,t_1,ca_1)\} \)
\( \pi \subseteq \Pi = \{(s_0,t_0,ca_1)\} \)

\( C_0/s_0t_0ca_0 \models \{ AX \Phi_1, EX EF DOut_{16}, AX \Phi_2, AX \Phi_3, EX EF DIn_{32}, AX \Phi_4 \} \)

\( C_1/s_0t_0ca_1 \models \{ \Phi_1, EF DOut_{16}, \Phi_2, \Phi_3, EF DIn_{32}, \Phi_4 \} \)
Example

φ₁ = AG (0 ≤ C ≤ 32)
φ₂ = AG EF DOut₁₆
φ₃ = AG EF DIn₃₂
φ₄ = AG AF (Idleₛ ∧ Idleₜ ∧ C=0)

C₁/s₀t₀ca₁ ⊨ {φ₁, EF DOut₁₆, φ₂, φ₃, EF DIn₃₂, φ₄}

C₁/s₀t₀ca₁ ⊨ {(0 ≤ C ≤ 32), AXφ₁, DOut₁₆ ∨ EX EF DOut₁₆, AX φ₂, EF DIn₃₂, AX φ₃, AF (Idleₛ ∧ Idleₜ ∧ C=0), AX φ₄}
Example

$\Phi_1 = \text{AG (} 0 \leq C \leq 32 \text{)}$
$\Phi_2 = \text{AG EF DOut}_{16}$
$\Phi_3 = \text{AG EF DIn}_{32}$
$\Phi_4 = \text{AG AF (Idle}_S \land \text{Idle}_T \land C=0)$

$C_1/s0t0ca1 \vdash \{ \Phi_1, \text{EF DOut}_{16}, \Phi_2, \Phi_3, \text{EF DIn}_{32}, \Phi_4 \}$

$C_1/s0t0ca1 \vdash \{ (0 \leq C \leq 32), \text{AX}\Phi_1, \text{DOut}_{16} \lor \text{EX EF DOut}_{16}, \text{AX} \Phi_2, \text{EF DIn}_{32}, \text{AX} \Phi_3, \text{AF (Idle}_S \land \text{Idle}_T \land C=0), \text{AX} \Phi_4 \}$

and so on....
A Too for SoC Composition
The currency converter revisited

Click for Demo
Auto-FSM via WSDL

Currency Conv Example

GeoIP Service - http://www.webservicex.net/geoipservice.asmx?WSDL
Item Service – localhost:80
Auto Connect
Auto+Manual Connect
Redundant Connections
Goal specifications

- The price must not be calculated until destination country is known.
- Conversion should be made from item’s currency to user’s currency.
- There must exist a path to a state where the converted rate can be obtained.
Specifying the Goal

**GOAL**: Obtain the converted rate

**CTL**: EF(Label=calc.multiply)
Specifying the Goal

Constraint 1: The price must not be calculated until destination country is known.

CTL: ~\((\text{Label}=\text{item.price}) \land \text{AU}(\text{Label}=\text{item.CountryToShip})\)