Formal Verification of CHP Specifications with CADP, Illustration on an Asynchronous Network-on-Chip

Gwen Salaün, Wendelin Serwe (INRIA / VASY)  
Yvain Thonnart, Pascal Vivet (CEA / LETI)

ASYNC’07 Symposium, Berkeley, USA

March 12th, 2007
Outline of the Presentation

- Introduction
- Translation from CHP to LOTOS
- CADP toolbox overview
- Verification of ANOC protocol
- Conclusion & Future Work
Context & Objective

Designing of complex asynchronous designs:
- existing tool support for: simulation and synthesis
- verification is needed!

=> Translate CHP to LOTOS by using CADP toolbox
CHP language

- Abstract descriptions of asynchronous circuits?
  - Model asynchronous handshaking by asynchronous VLSI programming language seen as a Process Algebra

- Several existing languages:
  - High-level languages to describe processes communicating by message-passing along wires
  - CHP, Balsa, Haste/Tangram, Verilog channel extension, SystemC extensions, ...

- CHP (Communicating Hardware Processes):
  - Compilation to VLSI circuits [Martin-86]
  - Inspired by guarded commands and CSP
  - Tool support: TAST tools (TIMA Lab., Grenoble)

- Specific *Probe* operator:
  - Probe allows to observe a pending communication
  - Used to exploit low-level aspects of hardware implementation of communication channels
Outline of the Presentation

• Introduction
• Translation from CHP to LOTOS
• CADP toolbox overview
• Verification of ANOC protocol
• Conclusion & Future Work
CHP to LOTOS translation principle

- CHP and LOTOS are based on CSP
- Main differences between CHP and LOTOS
  - looping guarded commands vs recursive processes
  - symmetrical vs asymmetrical sequential composition
  - implicit vs explicit (exit/accept) variable passing
  - implicit vs explicit termination
  - internal vs external choice
  - p2p HW type vs multi-dir abstract typed channel
  - no LOTOS equivalent for CHP probe operation!

  - in CHP: probed channel?
    - Corresponds to a shared variable/resource
  - in LOTOS: probed channel?
    - Requires additional processes
The Probe Operator: c#, c#V

- Used in the passive side only
- Boolean Operation:
  \[ c#V = \begin{cases} 
  \text{true} & \text{if active side waits (for sending V) on c} \\
  \text{false} & \text{otherwise} 
  \end{cases} \]

- Active side is blocked in case of a successful probe:
  - Cannot change V before synchronisation / acknowledge
  - Cannot emit a different value on c

- Thus: Channels are "particular shared variables"
  - Written only by active side
  - Read only by passive side
  - Between two writes, a synchronisation is required
Probe operator : Example

Two-way arbiter example :

- client 1: @[ $c_1!; \text{loop } ]$
- client 2: @[ $c_2!; \text{loop } ]$
- arbiter:
  $@$[@[ [ $c_1\# \Rightarrow (c_1?, c!1); \text{loop }$
  $c_2\# \Rightarrow (c_2?, c!2); \text{loop } ]$

Interaction with client 1 only

Definition of a SOS semantics for CHP :
$\Rightarrow$ to guarantee translation correctness

Channel translation

• Translation of a channel \( c \):
  – Depends whether a probe occurs on \( c \)
  – Perform pre-processing before the translation task
    ▪ This optimizes the generated state-graph

• Three cases:
  – Un-probed channels \( \rightarrow \) direct translation
  – Single probe in guards \( \rightarrow \) simplified translation
    \[ @[ c_1# \Rightarrow (c_1?, c!1); \text{loop} \ldots \]
  – Probe in expression \( \rightarrow \) generic translation
    \[ @[ c_2# \text{ and } \neg(c_1#\text{true})) \Rightarrow (c!2, c_2?); \text{loop} \ldots \]
Channel translation: for Un-Probed channel

- For un-probed channels: Direct translation

CHP Model

```plaintext
PROCESS SimpleBuffer
  PORT( E : IN DI passive DR[32];
       S : OUT DI active DR[32] )
  VARIABLE data: DR[32];
BEGIN
  [ E?data ;
    S!data ;
    loop ];
END;
```

LOTOS model

```plaintext
PROCESS SimpleBuffer
  [E, S :T] :
  noexit :=
  E?data:T ;
  S!data ;
  SimpleBuffer[E,S]
ENDPROC
```
Channel translation: for Single Probe in Guards

Simplified grammar for guards:

- Guard ::= V | c# | c#V
- No probe in expressions V

⇒ Avoid additional channel process and gates

- Send c!V

- Probe c#V

- Receive c?x

value matching
• Translation Schema

• Tool Implementation
  – code specialization for probes
    ▪ (reduction up to a factor of 156)
  – 19,300 lines of SYNTAX, LOTOS NT, and C
  – test base of more than 500 CHP specifications
Outline of the Presentation

- Introduction
- Translation from CHP to LOTOS
- **CADP toolbox overview**
- Verification of ANOC protocol
- Conclusion & Future Work
CADP : Key Concepts

• CADP takes roots in concurrency theory

• Process algebra
  – Modular value-passing languages
  – Equivalences (Bisimulation)
  – Compositionality

• Explicit-state verification
  – As opposed to symbolic methods (BDDs, etc.)
  – Action-based models (Labeled Transition Systems)
  – μ-calculus, temporal logics
  – Model checking + Equivalence checking

http://www.inrialpes.fr/vasy/cadp

(Google: CADP Toolbox)
Outline of the Presentation

• Introduction
• Translation from CHP to LOTOS
• CADP toolbox overview
• Verification of ANOC protocol
  – ANOC presentation
  – state space generation techniques
  – verification techniques

• Conclusion & Future Work
Asynchronous Network-on-Chip Architecture

- **ANOC architecture**
  - 2D-mesh based
  - Provide Quality-of-Service
  - Implemented in QDI logic

- **ANOC network protocol**
  - Packet Switching
  - Source Routing

- **ANOC Communication node**
  - Composed of:
    - 5 input controllers
    - 5 output controllers
  - Handle Virtual Channel policy
ANOC Node Input Controller

- For each node input:
  - Routes flits of a packet to the corresponding output
  - Direction determined by the header flit
  - 4 possible directions
  - 2 virtual channels

- Complex arrangement of 14 Asynchronous Processes
Verification Approach: Overview

- Simplifications on the CHP level
- Compositional state space generation
- Verification of properties
  - absence of deadlocks
  - correct stimulus-response protocol
  - NOC data integrity
  - NOC data routing
- Simplified via SVL scripts
Simplifications on the CHP level

- **Data Independence**
  - fix part of the flits
  - reduction from $10^{25}$ down to $5 \times 10^{16}$ states

- **Traffic Generator**
  - emulate a “realistic environment”
  - check correctness (“observer” processes)

- **Verification Scenarios**
  - cut a large verification into several smaller ones
  - several sequences of inputs
  - a generic SVL script for all scenarios
Compositional State Space Generation

- Principle: “Divide and conquer”
- Alternate the steps of
  - generation
  - hiding internal transitions
  - minimization
  - combination
- Order following the data path
  - Use inputs to restrict behaviors
  - Use SVL scripts (41 steps to generate the state graph)

- Results
  - The SVL script generates in about 4’ the corresponding LTS
    - 1300 states, 3116 transitions
  - Largest intermediate LTS observed:
    - 295 000 states, 812 000 transitions
Verified Properties: Deadlocks

• Deadlock freedom:
  – check for states without successor

• Infinite Occurrence:
  – check for cyclic behavior

• no issue detected
Verified Properties: Stim.-Resp. protocol

- Correct Stimulus-Response Protocol:
  stimuli $\{ S_1, \ldots, S_m \}$ trigger responses $\{ R_1, \ldots, R_n \}$
- A single check $\left( (S_1 \parallel \ldots \parallel S_m) ; (R_1 \parallel \ldots \parallel R_n) \right)^*$
  is insufficient!
  (overlapping stimuli and responses)
- Three steps of equivalence checking
  - cyclic occurrence of all stimuli: $\left( (S_1 \parallel \ldots \parallel S_m) \right)^*$
  - cyclic occurrence of all responses: $\left( (R_1 \parallel \ldots \parallel R_n) \right)^*$
  - stimuli generate responses: $\left( S_i ; R_j \right)^*$

- no issue detected
Verified Properties: NOC Data Integrity

• Observer processes:
  – Compare responses with the expected results
  – Use special error channels

• Check for absence of error signals

• no issue detected
Verified Properties: NOC Data Routing

- NoC DATA routing: expresses as a μ-calculus formula:
  \[
  [true * . on\_channel(0) . to\_dest(1)] \\
  \langle no\_Data0\_toD() * . 'Data0\_to1' \rangle \\
  true
  \]

- CHP model check: **a routing issue is detected**
  - Tool generate a counter example:
    - Occurs if a new packet is admitted in the input controller before last flit of the previous packet was routed

- NoC node design?
  - correct in simulation on Verilog netlist: **no routing error**

- So ... a real **routing issue**?
  - due to CHP model under-specification:
    - CHP model does not account for handshake expansion
    - asynchronous processes actually have a 1/2 capacity (half-buffers)
  - If we explicit in the CHP model the real design slack, **corresponding to the chosen HSE reshuffling, the routing issue is fixed.**
Conclusion

• Translation of CHP into LOTOS
  – Formal definition (including a SOS semantics)
  – Implementation of a translator tool

• Verification strategy using CADP toolbox:
  – Compositional state graph generation
  – Verification of various properties

• Case studies on CHP models of:
  – Asynchronous DES
  – ANOC communication node
    ▪ Verification revealed a routing issue in the CHP model
due to absence of the real system slack modeling

=> Positive feedback from realistic case studies