Towards A Formal Theory of On Chip Communications in the ACL2 Logic

Julien Schmaltz
Saarland University - Computer Science Department
Saarbrücken, Germany

Dominique Borrione
TIMA Laboratory - VDS Group
Grenoble, France
A Motivation Example

- **eCall**
  - Automatic emergency call system
  - A phone call is automatically emitted when car sensors detect an accident
FlexRay Bus

- Basic protocol
  - Idle units send 1, to start send 0
  - “Sync edges” at each byte (from 1 to 0)
- Deterministic scheduling
  - Time is divided into rounds
  - Each unit has one slot per round
Verification

- Proof of each component
- Proof of their interconnection
Global Objective

One model for all architectures
Contribution

A functional formalism for communications: \textit{GeNoC} (Generic Network on Chip)

- Identifies the essential constituents and their properties
- Formalizes the interactions between them
- Correctness of the system is a consequence of the essential properties of the constituents
- \textbf{Mechanized support in ACL2}
  - Encapsulation allows abstraction
  - Functional instantiation generates proof obligations automatically
Outline

- Communication Principles
- GeNoC Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of GeNoC
A Unifying Model

Communication Architecture

- Navigation Interface
- eCALL Interface
- Phone Interface
- Sensors Interface
A Unifying Model
System = \mathcal{F}(Routing, Scheduling, recv, send)
Proof Obligations
System Theorem

Thm \sim \text{messages reach their destination}
System Theorem

Thm \sim \text{messages reach their destination}
Outline

- Communication Principles
- *GeNoC* Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of *GeNoC*
Overall Modeling Principles

- **Function** $GeNoC$
  - takes the list of pending communications
  - returns the list of results and the list of aborted communications

- **Transactions**
  - A transaction represents a pending communication, *i.e.* the intention of $A$ of sending $msg$ to $B$
  - It is a 4-tuple $(id \ A \ msg \ B)$
Function \textit{GeNoC}

Transactions:

(id\_1 A msg\_1 B)
(id\_2 D msg\_2 T)
(id\_3 F msg\_3 E)
(id\_4 R msg\_4 Z)

Results:

Aborted Missives:

(Node A Sending)

(Node B Sending)

(Application A Sending)

(Application B Sending)

(Message A Sending)

(Message B Sending)

(Frames A Sending)

(Frames B Sending)

(Messages A Sending)

(Messages B Sending)

(Scheduling A Sending)

(Scheduling B Sending)

(Routing A Sending)

(Routing B Sending)

(c) Julien Schmaltz, ACL2 2006, San Josè August 15-16 – p. 14/37
From transactions to missives

Transactions:
(id₁ A msg₁ B)
(id₂ D msg₂ T)
(id₃ F msg₃ E)
(id₄ R msg₄ Z)

Results:
Aborted Missives:
(id₁ A msg₁ B)
From transactions to missives

Result:
- Aborted Missives

Missives:
- \((id_1 A \text{ frm}_1 B)\)
- \((id_2 D \text{ frm}_2 T)\)
- \((id_3 F \text{ frm}_3 E)\)
- \((id_4 R \text{ frm}_4 Z)\)

Routing:

Scheduling:

Application

Node A

Interface A

recv
send

Frames

Interface B

recv
send

Frames

Node B

Application B

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 15/37
Routing Algorithm

(id₁ frm₁ Routes₁)

(id₂ frm₂ Routes₂)

(id₃ frm₃ Routes₃)

(id₄ frm₄ Routes₄)

Aborted Missives

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 16/37
Scheduling Policy

Application A

Interface A

Node A

Scheduling

Router (id1 frm1 Routes1)

Router (id3 frm3 Routes3)

Interface B

Application B

Node B

Results

Delayed

Abort Missives

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 17/37
Results

(id₂ frm₂ Routes₂)
(id₄ frm₄ Routes₄)

(id₁ B msg₁)
(id₃ E msg₃)

Aborted Missives

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 18/37
Aborted Missives

Missives

(id₂ D frm₂ T)
(id₄ R frm₄ Z)

Results

(id₁ B msg₁)
(id₃ E msg₃)

Aborted Missives
Aborted Missives

Node A

Application A

recv
send

Scheduling

Interface A

Node B

Application B

recv
send

Routing

Messages

Frames

Results

(id₁ B msg₁)
(id₃ E msg₃)
(id₂ T msg₂)

(id₄ R frm₄ Z)

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 19/37
Correctness Criterion

Transactions

(id₁ A msg₁ B)
(id₂ D msg₂ T)
(id₃ F msg₃ E)
(id₄ R msg₄ Z)

Results

(id₁ B msg₁)
(id₂ T msg₂)
(id₃ E msg₃)
(id₄ R frm₄ Z)

Aborted Missives

(c) Julien Schmaltz, ACL2 2006, San José August 15-16 – p. 20/37
Termination

Function $GeNoC$ is a recursive function and must be proved to terminate because:

- it is a prerequisite for mechanized reasoning (here ACL2)
- it is necessary to ensure liveness

To ensure the termination, we associate to every node a finite number of attempts. At every recursive call of $GeNoC$, every node with a pending transaction consumes one attempt.
Formal Definition

From a list of transactions, $\mathcal{T}$, the set of nodes $NodeSet$ and a list of attempt numbers $att$, function $GeNoC$ produces:

- The list $\mathcal{R}$ of results
- The list $\mathcal{A}$ for aborted missives

$$GeNoC : \mathcal{D}_\mathcal{T} \times GenNodeSet \times AttLst \rightarrow \mathcal{D}_\mathcal{R} \times \mathcal{D}_\mathcal{M}$$

$$(\mathcal{T}, NodeSet, att) \mapsto (\mathcal{R}, \mathcal{A})$$
Correctness Criterion

\[ \forall res \in \mathcal{R}, \exists! trans \in \mathcal{T}, \left\{ \begin{array}{l} \text{Id}_\mathcal{R}(res) = \text{Id}_\mathcal{T}(trans) \\ \land \text{Msg}_\mathcal{R}(res) = \text{Msg}_\mathcal{T}(trans) \\ \land \text{Dest}_\mathcal{R}(res) = \text{Dest}_\mathcal{T}(trans) \end{array} \right. \]

For any result \( res \), there exists a unique transaction \( trans \) such that \( trans \) and \( res \) have the same identifier, message, and destination.
Correctness Criterion

\[ \forall res \in R, \exists! trans \in T, \begin{cases} 
Id_R(res) = Id_T(trans) \\
\land \quad Msg_R(res) = Msg_T(trans) \\
\land \quad Dest_R(res) = Dest_T(trans)
\end{cases} \]

- Typical formula scheme
- Always check for \( Id \) equality
  - In ACL2, the idea is filtering according to \( Id \)'s
Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$
ACL2 Correctness Predicate

(defun genoc-thm (R T / R_ids)
  (and (equal (R-msgs R)
              (T-msgs T / R_ids))
       (equal (R-dests R)
              (T-dests T / R_ids))))

- \(T / R_{ids} = T\) filtered according to the ids of \(R\)
- Check that the messages and the destinations of \(T / R_{ids}\) and \(R\) are equal.
ACL2 Theorem

(defthm GeNoC-is-correct
  (mv-let
    (R A)
    (GeNoC T NodeSet att)
    (declare (ignore A))
    (implies (T lstp T)
      (GeNoC-thm R)
      (filters T
        (R-ids R))))))
Proof Obligations

- Interfaces
  - The composition $\text{recv} \circ \text{send}$ is an identity

- Routing $(\text{id A frm B}) \mapsto (\text{id frm Routes})$
  - Missive/Travel matching
    - Same frame and identifier
    - Routes effectively go from the correct origin to the correct destination

- Scheduling
  - Mutual exclusion between Scheduled and Delayed
  - No addition of new identifiers
  - Preserve frames and route correctness
Proof of the theorem

- Routing correctness + preserved by scheduling
  - $\rightarrow$ right destination
- No modification on frames
  - $\rightarrow$ every result is obtained by $recv \circ send$
- Interfaces correctness
  - $\rightarrow$ received message = sent message
- Mutual exclusion between $Scheduled$ and $Delayed$ + no new identifiers
  - $\rightarrow$ cut the proof in two parts
Outline

- Communication Principles
- GeNoC Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of GeNoC
Encapsulation: Interfaces

- Function \textit{send} builds a frame from a message:
  \[
  (\ (send \ *) \ \Rightarrow \ *)
  \]
- Function \textit{recv} recovers a message from a frame:
  \[
  (\ (recv \ *) \ \Rightarrow \ *)
  \]
- Their composition is an identity:
  \[
  (\text{ldefthm InterfaceCorrectness} ; ; \textit{recv} \circ \textit{send}(msg) = msg \\
  \text{equal} \ (\textit{recv} \ (\textit{send} \ \text{msg})) \ \text{msg}))
  \]
- Some additional constraints
(encapsulate
  (((send *) ⇒ *)
   ((recv *) ⇒ *)))

;; local witnesses
(local (defun send (msg) msg))
(local (defun recv (frm) frm))

;; proof obligations
(defthm InterfaceCorrectness
  (equal (recv (send msg)) msg))
(defthm send-nil
  (not (send nil)))
(defthm send-not-nil
  (implies msg (send msg)))
(defthm check-instance-interface
  \(t\) ;; we prove true
  :rule-classes nil ;; no rule
  :hints (("GOAL"
    ;; we use InterfaceCorrectness
    ;; with \(recv_{flexray}\) for \(recv\)
    ;; and \(send_{flexray}\) for \(send\)
    :use
      (:functional-instance
        InterfaceCorrectness
        (recv recv_{flexray})
        (send send_{flexray}))))
Outline

- Communication Principles
- $GeNoC$ Definition and Correctness
- ACL2 Theorem/Removing Quantifiers
- Abstraction using Encapsulation
- Applications of $GeNoC$
Applications of GeNoC

OSI Layer 1
- Bi-Φ-M
OSI Layer 2
- Ethernet

Scheduling on networks
- Circuit switching
- Packet switching
Bus arbitration
- AMBA AHB arbiter

Deterministic routing
- Octagon
- XY algorithm
Adaptive routing
- Double Y channel
Conclusion

- A generic model: \textit{GeNoC}
  - Identifies the essential constituents and their properties
  - Formalizes the global property as a consequence of proof obligations
- Its expression in ACL2
  - 1864 lines, 71 functions and 119 theorems
  - One fourth is dedicated to the modules
  - Abstraction using encapsulation
  - Automatic generation of proof obligations using \textit{functional instantiation}
Future Work

- Master/Slave protocols
- Deadlocks (structural and protocol level)
- Adding queues and channels
  - wormhole routing in Hermes (TIMA, Grenoble, France)
- Verified Distributed Stacks
  - “Verisoft” Stack (O.S., compiler, assembly, gates)
  - Interconnected Stacks through a time triggered FlexRay bus
    - Show that FlexRay matches $GeNoC$!

...
THANK YOU !!