Formal Validation of Deadlock prevention in Networks-on-Chips

F. Verbeek  J. Schmaltz

Radboud University, Nijmegen

12-05-2009 / ACL2 Workshop
1. Introduction

2. GeNoC and deadlock

3. A deadlock-related proof
The GeNoC Stack

Specification

Routing: src to dest
Scheduling: src to dest

Routing: src to dest
Scheduling: current to dest

Routing: current to dest
Scheduling: current to dest

ACL2 / RTL

Implementation

Schmaltz/Borrione [FAC08]

Borrione/Helmy/Pierre/Schmaltz [JES09]

Van den Broek/Schmaltz [Previous talk]

Ouchet/Borrione/Morin-Allory/Pierre
Introduction

Contribution

Specification

Routing: src to dest
Scheduling: src to dest

Routing: src to dest
Scheduling: current to dest

Routing: current to dest
Scheduling: current to dest

Deadlock verification

Implementation

ACL2 / RTL
The HERMES Network-on-Chip

Moreas et al. 2004

Constituents

Nodeset Injection Routing Scheduling

F. Verbeek, J. Schmaltz (RU)
GeNoC for HERMES

GeNoC

- Nodeset
- Injection
- Routing
- Scheduling

F. Verbeek, J. Schmaltz (RU)

12-05-2009 / ACL2 Workshop
GeNoC for HERMES

GeNoC
- Nodeset
- Injection
- Routing
- Scheduling

Proof Obligations

HERMES

F. Verbeek, J. Schmaltz (RU)

NoCs and Deadlock

Radboud University Nijmegen
Introduction

GeNoC for HERMES

Nodeset
Injection
Routing
Scheduling

Proof Obligations

2D Mesh

HERMES

Radboud University Nijmegen

F. Verbeek, J. Schmaltz (RU)
GeNoC for HERMES

- Nodeset
- Injection
- Routing
- Scheduling
- Proof Obligations

HERMES
- 2D Mesh
- Immediate
GeNoC for HERMES

Nodeset Injection
Routing Scheduling

Proof Obligations

2D Mesh Immediate

XY
GeNoC for HERMES

GeNoC
- Nodeset
- Injection
- Routing
- Scheduling
- Proof Obligations

HERMES
- 2D Mesh
- XY
- Immediate
- Circuit Switching

F. Verbeek, J. Schmaltz (RU)

NoCs and Deadlock

12-05-2009 / ACL2 Workshop
GeNoC for HERMES

GeNoC

Nodeset
Routing
Injection
Scheduling

Proof Obligations

HERMES

2D Mesh
XY
Immediate
Circuit Switching

F. Verbeek, J. Schmaltz (RU)

NoCs and Deadlock

Radboud University Nijmegen

12-05-2009 / ACL2 Workshop
GeNoC for HERMES

GeNoC

Nodeset
Injection
Routing
Scheduling

Proof Obligations

HERMES

2D Mesh
XY
Immediate
Circuit Switching

F. Verbeek, J. Schmaltz (RU)

NoCs and Deadlock

Radboud University Nijmegen
GeNoC for HERMES

![Diagram of GeNoC and HERMES](image)

**GeNoC**
- Nodeset
- Injection
- Routing
- Scheduling

**HERMES**
- 2D Mesh
- XY
- Immediate
- Circuit
- Switching

Proof Obligations
GeNoC for HERMES

GeNoC
- Nodeset
- Injection
- Routing
- Scheduling

HERMES
- 2D Mesh
- XY
- Immediate
- Circuit
- Switching

Proof Obligations
**Introduction**

**GeNoC for HERMES**

- **GeNoC**
  - Nodeset
  - Injection
  - Routing
  - Scheduling
  - Proof Obligations

- **HERMES**
  - 2D Mesh
  - XY
  - Immediate
  - Circuit
  - Switching
GeNoC for HERMES

GeNoC
- Nodeset
- Injection
- Routing
- Scheduling

HERMES
- 2D Mesh
- XY
- Immediate
- Circuit
- Switching

Proof Obligations
GeNoC for HERMES

- Nodeset
- Injection
- Routing
- Scheduling
- Proof Obligations

- 2D Mesh
- XY
- Immediate
- Circuit Switching
GeNoC for HERMES

GeNoC
- Nodeset
- Injection
- Routing
- Scheduling

Proof Obligations

HERMES
- 2D Mesh
- XY
- Immediate
- Circuit
- Switching

F. Verbeek, J. Schmaltz (RU)
GeNoC

trlst denotes the list of travels
st denotes the current state of the network
arr stores arrived travels

(defun GeNoC (trlst st arr)
  (if (endp trlst)
      (list arr nil)
    (mv-let (delayed enroute)
      (injection trlst st)
      (let ((enroute' (routing enroute)))
        (mv-let (st' arr')
          (scheduling st enroute')
          (GeNoC (append delayed enroute')
                st'
                (append arr' arr))))))
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
          ))))))
\[
\begin{align*}
\text{trlst} &= (a, b, c) \\
\text{arr} &= \text{nil} \\
\text{del} &= \text{nil} \\
\text{er} &= (a, b, c) \\
\text{er'} &= \text{nil} \\
\text{arr'} &= \\
\end{align*}
\]

\[
\begin{align*}
\text{arr} &= \text{nil} \\
\text{del} &= \text{nil} \\
\text{er} &= (a, b, c) \\
\text{er'} &= \text{nil} \\
\text{arr'} &= \\
\end{align*}
\]
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
        ))))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er' =
arr' =
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st’ er’ arr’)
        (scheduling st er’)
        (GeNoC
          (append del er’)
          st’
          (append arr’ arr))
        ))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er’ = (a, b, c)
arr’ = nil
(if (endp trlst)
  (list arr nil)
  (mv-let ((del er))
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let ((st' er' arr')
        (scheduling st er'))
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er' = (a, b, c)
arr' = nil
\begin{align*}
\text{trlst} &= (a, b, c) \\
\text{arr} &= \text{nil} \\
\text{del} &= \\
\text{er} &= \\
\text{er'} &= \\
\text{arr'} &= \\
\end{align*}

\begin{verbatim}
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
          ))))
\end{verbatim}
\texttt{trlst = (a, b, c)}
\texttt{arr = nil}
\texttt{del = nil}
\texttt{er = (a, b, c)}
\texttt{er' =}
\texttt{arr' =}

\begin{verbatim}
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
          ))))))
\end{verbatim}
\[ \text{trlst} = (a, b, c) \]
\[ \text{arr} = \text{nil} \]
\[ \text{del} = \text{nil} \]
\[ \text{er} = (a, b, c) \]
\[ \text{er'} = \]
\[ \text{arr'} = \]

\[
\begin{align*}
\text{(if (endp trlst)} \hfill \\
\text{(list arr nil)} \hfill \\
\text{(mv-let (del er)} \hfill \\
\text{(injection trlst st)} \hfill \\
\text{(let ((er (routing er)))} \hfill \\
\text{(mv-let (st' er' arr')} \hfill \\
\text{(scheduling st er')} \hfill \\
\text{(GeNoC)} \hfill \\
\text{(append del er')} \hfill \\
\text{st'} \hfill \\
\text{(append arr' arr)} \hfill \\
\text{))))))
\end{align*}
\]
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
          )))))

trlst = (a, b, c)
arr  = nil
del  = nil
er   = (a, b, c)
er'  = (a, b, c)
arr' = nil
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st’ er’ arr’)
        (scheduling st er’)
        (GeNoC
          (append del er’)
          st’
          (append arr’ arr)
          ))))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er’ = (a, b, c)
arr’ = nil
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
        ))))))
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
          )))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er' =
arr' =
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st' er' arr')
        (scheduling st er')
        (GeNoC
          (append del er')
          st'
          (append arr' arr)
        ))))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er' =
arr' =
(if (endp trlst)
  (list arr nil)
  (mv-let (del er)
    (injection trlst st)
    (let ((er (routing er)))
      (mv-let (st’ er’ arr’)
        (scheduling st er’)
        (GeNoC
          (append del er’)
          st’
          (append arr’ arr)))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er’ = (a, b)
arr’ = (c)
(if (endp trlst)
    (list arr nil)
    (mv-let (del er)
      (injection trlst st)
      (let ((er (routing er)))
        (mv-let (st’ er’ arr’)
          (scheduling st er’)
          (GeNoC
            (append del er’)
            st’
            (append arr’ arr))
          ))))

trlst = (a, b, c)
arr = nil
del = nil
er = (a, b, c)
er’ = (a, b)
arr’ = (c)
Termination of GeNoC

Current situation:
- Measure is sum of attempts of each travel
- Termination of GeNoC = Attempts have been exhausted

Desired situation:
- Termination of GeNoC = 1.) All travels have arrived
  or
  2.) deadlock
A *generic measure* is added as parameter to GeNoC:

- Generic function `deadlock-statep`
- Generic function `measurep`

New proof obligation:

```lisp
(defthm measure-decreases
  (implies (and (measurep (trlst st meas))
               (not (deadlock-statep trlst st)))
           (O< (acl2-count (mv-nth 2
                             (scheduling trlst st)))
                (acl2-count meas))))
```
Measure for Circuit Switching

- Measure:
  
  \[
  \text{(defun ct-measurep \(\text{trlst st meas}\)}
  \text{(equal meas)}
  \text{(sum-of-lst \(\text{route-lengths trlst}\)))}
  \]

- Deadlock-state:

  \[
  \text{(defun ct-deadlock-statep \(\text{trlst st}\)}
  \text{(if \(\text{endp trlst}\)}
  \text{t)}
  \text{(and \(\text{not \(\text{free-nodes \(\text{route \(\text{car trlst}\)}\}}\)) st))}
  \text{(ct-deadlock-statep \(\text{cdr trlst st}\)))}
  \]
(if (endp trlst)
  (list arr trlst)
  (mv-let (del er)
    (injection trlst st)
    (let ((er’ (routing er)))
      (if (deadlock-statep er’ st)
          (list arr trlst)
          (mv-let (st’ arr’ meas’)
            (scheduling st er’)
            (GeNoC
              (append del er’)
              st’ meas’
              (append arr’ arr)
            ))))))
To prove that a NoC is deadlockfree:

1. Define set of conditions $P$, such that:

   $$ P \iff \neg \text{(deadlock-statep)} $$

2. Prove that $P$ is inductive for GeNoC
Instantiation

Network:

Nodeset: $x \times y$ 2D Mesh, $b$ buffers per node
Injection: Immediate
Routing: XY Routing
Scheduling: Circuit Switching with bookings
Theorem:

Let each node have $b$ buffers.

$$(\text{len trlst}) < 2b \implies \text{no deadlock}$$
Deadlock prevention

Theorem:

Let each node have \( b \) buffers.

\[
(len \ trlst) < 2b \implies \text{no deadlock}
\]
A deadlock-related proof

Deadlock prevention

Let each node have $b$ buffers.

$\text{(len trlsl)} < 2b \implies \text{no deadlock}$
A deadlock-related proof

Proof structure

Generic:

1. Define set of conditions $P$, such that:
   \[ P \implies \neg (\text{deadlock-statep}) \]

2. Prove that $P$ is inductive for GeNoC

Theorem:

\[ (\text{len trlst}) < 2b \implies \text{no deadlock} \]
Proof structure

Instantiated:

1 Prove:

\[(\text{len trlst}) < 2b \implies \text{at least one travel can advance}\]

2 Prove:

\[(\text{len trlst}) < 2b \implies (\text{len (scheduling trlst st).trlst)}) < 2b\]

Theorem:

\[(\text{len trlst}) < 2b \implies \text{no deadlock}\]
A deadlock-related proof

Theorem

\[(\text{len trlst}) < 2b \implies \text{at least one travel can advance}\]

Proof

- A travel can advance if
  - all nodes of the route are booked, or
  - all nodes of the route can be booked.
- A node can be booked if
  - it isn’t booked, and
  - it isn’t full.
Theorem

\[(\text{len trlst}) < 2b \implies \text{at least one travel can advance}\]

Proof

\[\text{Thm1} \quad \text{no full nodes} \quad \text{no booked nodes} \]
\[\text{Thm2} \quad \text{booked node} \]
\[\text{Thm3} \quad \text{full node} \]
Thm3
No booked nodes and a full node implies any travel currently in the full node can advance.

Proof
Let $n$ be the full node and $v$ be a travel in $n$:

Thm3.1 The bound on the number of travels implies that for all $n' \neq n$, $n'$ is not full.

Thm3.2 No cycles in routing implies $n \notin \text{route}(v)$
Conclusions

- Proven deadlock-prevention theorem by bounding the number of messages.
- Proof consists of 3157 lines: 39 functions and 321 theorems.
- Theorem is proven for any routing, any topology, any nodeset (e.g. ports), and for both circuit and packet switching.
- Defined proof pattern for deadlock-related proofs.
Future work

- Define better bound/set of restrictions.
- Define a more accurate measure than route lengths.
- Prove theorem for wormhole switching.
- Varying injection methods.
A deadlock-related proof

Questions?