Trade-Offs in the Configuration of a Network on Chip for Multiple Use-Cases

Andreas Hansson¹, Kees Goossens²,³
May 8, 2007
Outline

- Introduction
- Consistency
- Implementation
- Results
- Conclusions
Introduction
Introduction

- Systems on Chip (SoC) grow **increasingly complex**
- *Multiple applications* are integrated on the same chip
- Applications are *started and stopped dynamically*
Reconfiguration requirements

- Means for *performing changes*
  - programmable architecture
  - control infrastructure

- Ability to *specify and compute configurations*
  - multiple use-case resource binding

- Facilities for *controlling change*
  - software libraries
  - hardware support
Consistency
Transactions

- Intellectual property (IP) modules interact via *transactions*
  - Typically using protocols such as DTL, APB, AHB, OCP and AXI

- A transaction is composed of *request and response*
  - Command, read data, write data, tags, etc

- Transactions take place over a network *connection*
  - Bi-directional peer-to-peer interconnection
  - Enables end-to-end services
  - Comprises *request channel and response channel*
Channels

- A channel can be seen as a set of *virtual wires*

- The mapping to physical resources is determined by
  - *a path* through the router network,
  - *a queue* in the destination network interface (NI),
  - the associated end-to-end flow control credits for the source NI, and
  - a set of *TDM time-slots*.

- Channels are modified through *three basic operations*
  - Open, modify and close
Transactions revisited

- The virtual-wire view of a channel is non-trivial to uphold
Transactions revisited

- The virtual-wire view of a channel is non-trivial to uphold
- Transactions occur on *multiple levels and locations*
  - High-level bus transactions between IP and NI protocol shell
  - Word-level handshakes from NI shell to kernel
  - Flits and link-level flow control in NI kernel and router network
  - End-to-end flow control between NI kernel and NI kernel
Transactions revisited

- The virtual-wire view of a channel is non-trivial to uphold
- Transactions occur on *multiple levels and locations*
  - High-level bus transactions between IP and NI protocol shell
  - Word-level handshakes from NI shell to kernel
  - Flits and link-level flow control in NI kernel and router network
  - End-to-end flow control between NI kernel and NI kernel

Transient state is spread across the whole interconnect
Implementation
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Hardware support

- Run-time programmability is implemented by
  - registers in the *NI kernels and shells* (possibly also in the routers), and
  - a memory-mapped configuration port on each NI

- Configuration infrastructure is provided by *the network itself*
  - A dedicated NI port is *looped back* to its configuration port
  - Configuration is done via *configuration connections*
Transaction-level configuration

- Configuration must take place in a *quiescent state*
  - Cut on *transaction boundaries*, but...
  - this might never occur (or take a very long time)

- Add functionality in NI shell to
  - finish ongoing handshakes, then *mask valid and accept*
  - enable *observability and control* of the protocol state machine in the shell
Normal operation

MNIP

SNIP

rdata

wdata

livetrns

Normal operation in between transactions

write read

write read

write read
Transaction-level stopping

Transaction-level stopping involves stopping current transactions and not starting any new transactions.

- **MNIP**: Master NI side
- **SNIP**: Slave NI side
- **rdata**: Read data
- **wdata**: Write data
- **stop**: STOP instruction
- **finish rdata**: Finish read transaction
- **no new write**: No new write transaction
- **stop_in**: Stop in transaction
- **blocked**: Blocked transactions
- **no live transactions**: No live transactions
- **finish current transactions**: Finish current transactions
- **don't start any new transactions**: Don't start any new transactions
Software support

- Configuring the network involves *individual registers* in routers and NIs
  - Programming is tedious and error prone
  - Abstraction level must be raised

- We propose the *Æthereal Run-Time library*
  - High-level (modify the topology of the application graph)
    - open_conn, close_conn, modify_conn
  - Low-level (provide hooks for e.g. a configuration manager)
    - set_flow_control, set_time_slots, get_live_transactions
Results
Setup time

- Three different implementations of the configuration master
  - Ideal processor
  - ARM7TDMI@100MHz with pure read/write calls
  - ARM7TDMI@100MHz with ART library
Setup time

- Three different implementations of the configuration master
  - Ideal processor
  - ARM7TDMI@100MHz with pure read/write calls
  - ARM7TDMI@100MHz with ART library

- Network contribution to the configuration time is negligible
Guaranteed setup time

- Configuration is traditionally done using *best-effort services*
  - Latency of configuration transactions is inherently unpredictable

- Guaranteed service configuration connections
  - enable *tight bounds* on time for opening and modifying a connection
  - at a very *limited hardware cost* (<2%)
Variation in tear-down time

- Opening and modifying connections is done *without involving the IPs*
  - Enabled by buffers in NIs together with end-to-end flow control
  - *Time required can be bound* by guaranteed service configuration connections

- Closing a connection involves finishing in flight transactions
  - The IPs inherently affect the reconfiguration time
Conclusions
Conclusions

- There is a need for a systematic approach to (re-)configuration of NoCs
  - Multiple use-case applications is a fact

- Reusing the NoC to also carry configuration data
  - avoids instantiating a dedicated configuration interconnect
  - has limited performance cost

- Reconfiguration hardware and software can be reused for debug purposes

- Guaranteed configuration connections
  - enable predictable open and modify operations
  - with a low hardware cost

- Connection closing is inherently dependent on the IP behaviour
  - Bounding the configuration time requires insertion and analysis of configuration points
Binary size

- Varying the number of NIs and the number of connections