Network On Chip Test Methodology

Hong-Sik Kim
Topics

- Introduction
- NoC basics
- NoC related research topics
- Study on reliability and testability of NoC
- Conclusions
Design Paradigm Shift

- Log # transistors
- Technology
- Design gap
- Design productivity
- Time

paradigm shifts
Advances in IC Design Methodologies

- Transistor Level
  - Transistor Model
    - Capacity Load
- Gate Level
  - Gate Level Model
    - Capacity Load
- RTL
- IP Reuse
  - IP Block Model
    - Communication Model
- Platform Reuse
  - System Model
    - APIs

- 1980대 초반
- 1980대 후반
- 1990대 초반
- 1990대 후반
- 2000대 초반

- GDSII, SPICE
- Net-list
- HDL
- SystemC
  - Standard On-chip Buses
- Transaction level Model
Interconnect Delay

- Interconnect Delay Dominates Gate Delay
  - Global interconnect delay is being continuously increasing
  - Multiple clock cycles to cross chip die is required
  - Performance limitation

*2001 International Technology Roadmap for Semiconductors*
Interconnect Delay

- Ratio between global communication delay and operational delay
  - 2 : 1 in 2002
  - 9 : 1 in 2010

<table>
<thead>
<tr>
<th>Operation</th>
<th>0.13um Delay</th>
<th>50nm Delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>32b ALU Operation</td>
<td>650 ps</td>
<td>250 ps</td>
</tr>
<tr>
<td>32b Register Read</td>
<td>325 ps</td>
<td>125 ps</td>
</tr>
<tr>
<td>Read 32 b from 8KB RAM</td>
<td>780 ps</td>
<td>300 ps</td>
</tr>
<tr>
<td>Transfer 32b across chip (10 mm)</td>
<td>1400 ps</td>
<td>2300 ps</td>
</tr>
<tr>
<td>Transfer 32b across chip (200 mm)</td>
<td>2800 ps</td>
<td>4600 ps</td>
</tr>
</tbody>
</table>

W.J. Dally presentation: Computer architecture is all about interconnect (it is now and it will be more so in 2010) HPCA Panel February 4, 2002
Noise in DSM

• DSM technology
  ▪ very subtle to noise effect
  ▪ low operating voltage, high clock frequency
  ▪ increased coupling cap

• Source of Noise
  ▪ Crosstalk
  ▪ Power supply noise
  ▪ Substrate noise
  ▪ Soft errors
  ▪ EMI
  ▪ Thermal noise
Noise in DSM

- Crosstalk fault
  - Noise on a wire is included by the switching activities on neighboring wires
  - Due to the capacitive coupling between the adjacent wires
  - Results in propagation delay or glitch

\[ C_{\text{couple}} \propto \frac{t}{s} \]

- \( C_{\text{couple}} \) depends on technology parameters and transition directions
- With technology scaling, wire spacing is shrinking faster than wire height
- Crosstalk Delay Faults

- When the transition of adjacent signals in opposite directions, the delay is longest. 
  \[ C_{\text{total}} = 2C_{\text{couple}} + 2C_{\text{couple}} + C_{\text{ground}} \]
### Noise in DSM

**• Crosstalk Analysis**

- Different transmission patterns have different $C_{\text{total}}$, and then have different delay

<table>
<thead>
<tr>
<th>Ctotal</th>
<th>Patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-- --</td>
</tr>
<tr>
<td></td>
<td>--↑</td>
</tr>
<tr>
<td></td>
<td>--↓</td>
</tr>
<tr>
<td></td>
<td>↑--</td>
</tr>
<tr>
<td></td>
<td>↓--</td>
</tr>
<tr>
<td></td>
<td>↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓</td>
</tr>
<tr>
<td></td>
<td>↑↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓↓</td>
</tr>
<tr>
<td></td>
<td>↑↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓↓</td>
</tr>
<tr>
<td></td>
<td>↑↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓↓</td>
</tr>
<tr>
<td>Cground</td>
<td>↑↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓↓</td>
</tr>
<tr>
<td>Ccouple+Cground</td>
<td>↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓</td>
</tr>
<tr>
<td>2Ccouple+Cground</td>
<td>↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓</td>
</tr>
<tr>
<td>3Ccouple+Cground</td>
<td>↑↑</td>
</tr>
<tr>
<td></td>
<td>↓↓</td>
</tr>
<tr>
<td>4Ccouple+Cground</td>
<td>↑↑</td>
</tr>
</tbody>
</table>

How to address reliability problems in DSM Tech.

• High Noise Probability in DSM Technology
  ▪ SoC will operate in the presence of noise due to crosstalk, EMI, soft error etc
  ▪ Data may be delayed or corrupted
  ▪ Malfunction is modeled as single/multiple upsets

  ▪ Present design reduces noise by physical design
  ▪ **Future design will tolerate noise** by pushing solutions to higher abstraction levels
Communication Centric Application

• Many Core Processor
  ▪ For massive multi-media application
  ▪ Need for low power consumption and reliable data communication

• Mile Stone
  ▪ 2001 : p960 Regatta by IBM
  ▪ 2005 : ARM11 MPCore, MPSoC
  ▪ 2007 : Tile64 Multi-Core Processor by Tilera
  ▪ 2009 : Larabee scheduled by INTEL

• Features in terms of interconnect
  ▪ Very communication centric core
  ▪ Need for scalability of global communication
• Global Interconnect Solutions
  ▪ Shared Bus, Segmented Bus, Crossbar Architecture

• Shared Bus
  ▪ Low cost and low scalability
  ▪ Capacitance grows as # of cores increases

• Segmented Bus
  ▪ Low cost but limited bandwidth

• Crossbar Architecture
  ▪ Many communication channels by parallel interconnects
  ▪ High Cost
  ▪ Low scalability, typically impossible for 10x10
New Solution: Network on Chip Design

- **Technology Aspect**
  - Need for reliable high performance global data transfer
  - Need for reliable global communication against DSM noise

- **Product Aspect**
  - Many-Core processor application requires communication centric design
  - Previous solutions cannot provide enough scalability and bandwidth

- **Network on Chip Design Methodology**
  - An evolution of on-chip bus interconnect technology
  - Interconnection model implemented on a chip in the form of a micro-network
    - High performance, scalability and reliability
    - Affordable cost
    - High complexity
Network on Chip Basics

• High Flexibility in Topology and Routing Policy
  ▪ Various topology
  ▪ Resource constrained router preferable

• NoC Components
  ▪ IP Core
  ▪ Network Adaptor
    - Connects between IP core and router
  ▪ Routing Node
    - Switch and routing table
  ▪ Link
    - Wires connecting nodes
• **NoC Switches**

  - NoC contains multiple point-to-point data links interconnected by switches
  - The switch controls and distributes traffic in the network (routing algorithm)
  - Switches may be asynchronous or synchronous
  - Switches may contain buffers such as FIFO’s

• **Inside the Switch**

  - Five directions – north, east, south, west, core
  - Crossbar implemented using multiplexers
  - Output buffer or Input buffer option
### On-Chip Bus VS Network on Chip

<table>
<thead>
<tr>
<th><strong>Bus Pros &amp; Cons</strong></th>
<th><strong>Network Pros &amp; Cons</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Every unit attached adds parasitic capacitance, therefore electrical performance degrades with growth.</td>
<td>+ Only point-to-point one-way wires are used, for all network sizes.</td>
</tr>
<tr>
<td><strong>+</strong> Bus timing is difficult in a deep submicron process.</td>
<td><strong>-</strong> Network wires can be pipelined because the network protocol is globally asynchronous.</td>
</tr>
<tr>
<td>Bus testability is problematic and slow.</td>
<td>+ Dedicated BIST is fast and complete.</td>
</tr>
<tr>
<td><strong>+</strong> Bus arbiter delay grows with the number of masters. The arbiter is also instance-specific.</td>
<td><strong>+</strong> Routing decisions are distributed and the same router is reinstanciated, for all network sizes.</td>
</tr>
<tr>
<td>Bandwidth is limited and shared by all units attached.</td>
<td><strong>-</strong> Aggregated bandwidth scales with the network size.</td>
</tr>
<tr>
<td>Bus latency is zero once arbiter has granted control.</td>
<td><strong>-</strong> Internal network contention causes a small latency.</td>
</tr>
<tr>
<td><strong>The silicon cost of a bus is near zero.</strong></td>
<td><strong>+</strong> The network has a significant silicon area.</td>
</tr>
<tr>
<td>Any bus is almost directly compatible with most available IPs, including software running on CPUs.</td>
<td><strong>-</strong> Bus-oriented IPs need smart wrappers. Software needs clean synchronization in multiprocessor systems.</td>
</tr>
<tr>
<td><strong>The concepts are simple and well understood.</strong></td>
<td><strong>-</strong> System designers need reeducation for new concepts.</td>
</tr>
</tbody>
</table>
NoC Topologies

- Mesh
- Octagon
- Butterfly Fat-Tree
- H-Tree
NoC Structures

• Regular topology NoC
  ▪ From multi-computer networks
  ▪ Designed for general and homogeneous systems
  ▪ NoC platform design
• Application specific NoC

- Customized NoC for each application results in significant performance improvement
- Irregular-topology and hierarchy
- For both homogeneous and heterogeneous systems
NoC Examples

• NOSTRUM
  - 2D mesh topology
  - 128 bit wide link
  - Non-minimal adaptive hot potato routing
  - No buffering
  - Best effort, guaranteed latency virtual channels
• Αthereal (Phillips)
  ▪ Unknown, probably low dimensional tree or mesh
  ▪ Deterministic source based routing
  ▪ Provides both guaranteed traffic and best effort service
  ▪ Input buffering
  ▪ 6 port router (32 bit word size)
NoC Examples

• SPIN
  ▪ Fat tree topology
  ▪ Wormhole switching
  ▪ Adaptive routing
  ▪ Input buffering
  ▪ bidirectional 32 bit links
NoC Examples

- **Octagon (UCSD)**
  - Basic 8 node architecture
  - 2 hop diameter
  - Source based routing
NoC Examples

• XPIpes (Stanford Univ.)
  ▪ Heterogeneous NoC
  ▪ Source based routing
  ▪ Wormhole switching
  ▪ Long, pipelined links
  ▪ Output buffering
  ▪ Virtual channel
  ▪ Acknowledgement based flow control with retransmission
NoC Commercial Products

• Cell Processor
  ▪ High performance heterogeneous media chip
  ▪ EIB (element interconnect bus) based on chip network

• Larabee (Intel)
  ▪ Many-core GPU
  ▪ Ring bus network based inter-processor communication
• Tile64 (Tilera)
  ▪ Based on MIT Raw Processor
  ▪ 64 identical GP core included
  ▪ 3 way VLIW pipeline
  ▪ on-chip iMesh network
  ▪ 5 2D mesh networks
  ▪ Each link consists of 2 unidirectional 32 bit link
  ▪ wormhole routing
  ▪ 31 Tbps network throughput
NoC Design Flow

NoC Parameters

System Level Modeling

Application Model

RTL Generation

HW/SW Co-Sim.

Verified?

Synthesis and Formal Verification

Verified?

Emulation and Layout

Verified?

Objective is satisfied?

To Fab.
NoC Research Issues

- Fault tolerant design

System Level Modeling

Application Model

RTL Generation

HW/SW Co-Sim.

Veriﬁed?

Synthesis and Formal Veriﬁcation

Emulator Layout

Verified?

Y

To Fab.

Performance and Power Estimation

Mapping and Scheduling

Objective is satisfied?

N

Y

**NoC Model**

**Design Objective**

- DFT methodology
- Test scheduling
Fault Tolerant Design

- Hardware Failure Taxonomy

Failure

- Permanent
  - defects, wearout, design failure, out of parameters
  - To test and tolerate

- Temporary
  - Transient
    - soft error, EMI, substrate noise
  - Intermittent
    - gradual degradation, weak part
  - To tolerate
Fault Tolerant Design

• FT for transient faults
  ▪ Software layer is responsible and recoverable
  ▪ Link-to-link or end-to-end re-transmission
  ▪ Error detection and correction (for example CRC)
  ▪ ED/EC and retransmission

• FT for permanent faults
  ▪ System should avoid using the faulty module
  ▪ On-line or production test be prerequisite
Fault Tolerant Design

- **NOC FT**
  - Yield improvement
  - Reliable data communication against noise

- **CFC based FT**
  - Crosstalk free coding algorithm
  - Reducing total cap comparable with shielded case by encoding data

- **FT Routing**
  - Flooding
  - ECC and Re-transmission
Crosstalk Free Coding

- CFC Algorithm
  - Forbidden pattern free coding (010, 101)
  - Forbidden transition free coding
  - Reduce the total cap into $2C_{\text{couple}} + C_{\text{ground}}$

C. Duan et al, “Forbidden transition free crosstalk avoidance CODEC design”, DAC, 2008

<table>
<thead>
<tr>
<th>Class</th>
<th>$C_{\text{eff}}$</th>
<th>Transition patterns</th>
</tr>
</thead>
<tbody>
<tr>
<td>0C</td>
<td>$C_L$</td>
<td>000 → 111</td>
</tr>
<tr>
<td>1C</td>
<td>$(1 + \lambda)C_L$</td>
<td>011 → 000</td>
</tr>
<tr>
<td>2C</td>
<td>$(1 + 2\lambda)C_L$</td>
<td>010 → 000</td>
</tr>
<tr>
<td>3C</td>
<td>$(1 + 3\lambda)C_L$</td>
<td>010 → 100</td>
</tr>
<tr>
<td>4C</td>
<td>$(1 + 4\lambda)C_L$</td>
<td>010 → 101</td>
</tr>
</tbody>
</table>

Diagram:
- Encoder and decoder connections showing data flow.
- Physical channel symbolizing the transmission path.
- Transition patterns illustrating the coding effectiveness.
FT Routing Algorithm

- Flooding based FT
  - Probabilistic flooding
  - Directed flooding
  - Redundant Random Walk

M. Pirretti et al, “Fault tolerant algorithms for network on chip interconnect”, IEEE annual symposium on VLSI emerging trends in VLSI systems design, 2004
FT Routing Algorithm

• ECC and Retransmission based FT 1
  ▪ FEC method
  ▪ Single error correction and double error detection
    - In case of single error, the corresponding router corrects the error
    - In case of double error, the corresponding router request retransmission


• ECC and Retransmission based FT 2
  ▪ Hamming code based one
  ▪ Target fault model: SEU and crosstalk on link wires

FT Routing Algorithm

• Retransmission Policy

- Retransmission policy
  - E2E: error check is performed at the end point
    - power consumption reduced
    - in case of destination header error, additional network traffic occurs
  - S2S
    - error check is done at every hop
    - power consumption increased
    - in case of high error rate, network latency guaranteed

![Diagram of FT Routing Algorithm]

End to End

Switch to Switch
NoC Testing

• How to test NoC
  ▪ IP core test
    - tested as SoC cores
    - NoC network resource can be reused as TAM
  ▪ Router and link test
    - Wrapper based structural test
    - On-line functional test by error checker
    - BIST (built-in self test)
    - Functional test
NoC Testing

• NoC Test Model
  - Network as TAM
  - Test wrapper to isolate cores during test mode
  - Packet based test delivery

Test channel = 2W

ATE

W bits

core

core

core

core

NoC

W bits

Wrapper

CUT

<table>
<thead>
<tr>
<th>Head</th>
<th>Test head</th>
<th>flit0</th>
<th>flit1</th>
<th>flit2</th>
<th>Tail</th>
</tr>
</thead>
<tbody>
<tr>
<td>0111</td>
<td>0110</td>
<td>000010</td>
<td>101101</td>
<td>1000xx</td>
<td></td>
</tr>
</tbody>
</table>

Flit0 = 000010
Flit1 = 101101
Flit2 = 1000xx
NoC Testing

- Test Channels and Cost

Test channel = 2W

Test channel = 4W
NoC Testing

- Test Channels and Cost

**P93791 - Test time and ATE cost**

Test ports configuration: #INPUTS / #OUTPUT PORTS
NoC Testing

• Test Compression Application for Test Cost Reduction
  ▪ Limited test channel reduce the test parallelism
  ▪ Test compression can increase the number of cores under parallel testing
  ▪ So results in test application time reduction or test cost reduction

J. Dalmasso et al, “Fitting ATE Channels with Scan Chains: a Comparison between a Test Data Compression Technique and Serial Loading of Scan Chains”, DELTA, 2006
**NoC Testing**

- Test Compression Application for Test Cost Reduction

---

<table>
<thead>
<tr>
<th>Original Test Packet</th>
<th>Compressed Test Packet</th>
<th>Decoded Test Packet</th>
</tr>
</thead>
<tbody>
<tr>
<td>W=5 bits</td>
<td>M=2 bits</td>
<td>W=5 bits</td>
</tr>
<tr>
<td>packet header</td>
<td></td>
<td>packet header</td>
</tr>
<tr>
<td>test header</td>
<td></td>
<td>test header</td>
</tr>
<tr>
<td>01101</td>
<td></td>
<td>01101</td>
</tr>
<tr>
<td>01101</td>
<td></td>
<td>01101</td>
</tr>
<tr>
<td>01110</td>
<td></td>
<td>01110</td>
</tr>
<tr>
<td>tail</td>
<td></td>
<td>tail</td>
</tr>
</tbody>
</table>

---

ATE CUT
NoC Testing

- Test Compression Application for Test Cost Reduction

  - Experimental Results
  - 33% test time reduction for d695 benchmark circuits

For d695, 32 bit test channel assumed

<table>
<thead>
<tr>
<th>System Configuration</th>
<th>Test Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 32-bit test input</td>
<td>36588 cycles</td>
</tr>
<tr>
<td>3 inputs of 12, 10, and 10 bits</td>
<td>24395 cycles</td>
</tr>
</tbody>
</table>

J. Dalmasso et al, “Fitting ATE Channels with Scan Chains: a Comparison between a Test Data Compression Technique and Serial Loading of Scan Chains”, DELTA, 2006
NoC Testing

- NoC Test Scheduling Algorithm
  - Test scheduling with dedicated path
  - Each core associated with a routing path
  - Test pipeline maintained


- Define test packet
- Define path for each core
- Sort packet according to test time
- Select schedule for minimum total test time
- Select a packet
- Find available path
- Schedule packet
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6 5 4 8 10 7 3 2 1 9

- Test Info.
  - d695, test channel = 32 bits
  - 3 inputs with 10, 10, 12 bits
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6  5  4  8  10  7  3  2  1  9

```
6       10856
6       10856
6       9869
```

```
6

```
5          10
10
2
1
3          6
4
9          8
7
```

10
12
10

In
Out
Out
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set:
6 5 4 8 10 7 3 2 1 9

Diagram of network-on-chip (NoC) with test pattern set 9869.
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>5</td>
<td>4</td>
<td>8</td>
<td>10</td>
<td>7</td>
<td>3</td>
<td>2</td>
<td>1</td>
<td>9</td>
</tr>
</tbody>
</table>

[Diagram showing the test scheduling algorithm with nodes and connections labeled with test patterns and values.]
NoC Testing

• NoC Test Scheduling Algorithm
  ◦ An simple example

Test pattern set

6  5  4  8  10  7  3  2  1  9
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6 5 4 8 10 7 3 2 1 9
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set: 6 5 4 8 10 7 3 2 1 9

Diagram:

- Nodes represent test patterns:
  - 4: 5829
  - 5: 6826
  - 6: 9869

- Connections indicate test scheduling:
  - In: 5 - 10
  - Out: 10 - 3 - 6 - 4 - Out
  - In: 9 - 5 - 10 - 2 - Out
  - Out: 10 - 7 - Out
NoC Testing

• NoC Test Scheduling Algorithm
  ▪ An simple example

Test pattern set
6 5 4 8 10 7 3 2 1 9
NoC Testing

• NoC Test Scheduling Algorithm
  ▪ An simple example

Test pattern set

5829 10434
4 8

6826 10069
5 10

9869
8
NoC Testing

• NoC Test Scheduling Algorithm
  ▪ An simple example

Test pattern set

6 5 4 8 10 7 3 2 1 9

5829 10434
4 8

6826 10069
5 10

9869 13328
6 7

5 10

3 6

4

9 8

7

1

10

12

In

Out

Out

Out

10
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

5829 10434
4 8

6826 10069 12576
5 10 3

9869 13328
6 7

Diagram showing a network with nodes and connections.
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6  5  4  8  10  7  3  2  1  9

5829  10434  11022
4  8  2

6826  10069  12576
5  10  3

9869   13328
6  7

5
10
2

3
6
4

9
8
7

10

In
Out

12

10

Out

Out
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set:

6 5 4 8 10 7 3 2 1 9

Diagram:

5829 10434 11022 11047
4 8 2 1

6826 10069 12576
5 10 3

9869 13328
6 7

5 10 2

3 6 4

9 8 7
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6 5 4 8 10 7 3 2 1 9

5829 10434 11022 11047
4 8 2 1

6826 10069 12576
5 10 3 9

9869 13328
6 7

5 10 12
3 6 4
9 8 7

10 In

Out
NoC Testing

- NoC Test Scheduling Algorithm
  - An simple example

Test pattern set

6  5  4  8  10  7  3  2  1  9

5829 10434 11022 11047
4  8  2  1
6826 10069 12576
5  10  3  9
9869 13328
6  7

10 12

10 3 5 10 2 4

9 8 7

10

5
Conclusions

- DSM technology with high noise environment
- Tolerating noise needed in addition to removing it
- Communication centric products such as many core processor requires new communication methodology

- NoC Design Methodology Requirement
  - Crosstalk free coding
  - ED/EC, retransmission
  - Multi-casting

- NoC FT Methodology
  - Structural NoC test model
  - Test compression and test scheduling for NoC test cost reduction
Thank you !!!