Comparing Energy and Latency of Asynchronous and Synchronous NoCs for Embedded SoCs

Daniel Gebhardt   Junbok You   Kenneth Stevens

University of Utah
Salt Lake City, U.S.A.

5th May, 2010
Domain of work

- Embedded power-constrained SoCs
- Heterogeneous cores
- Specialized functionality
- Known inter-core traffic properties/requirements
Demonstrate latency and energy of a simplified, asynchronous NoC

- Design of a simple, energy-efficient asynchronous router
- Optimization algorithms for router topology and placement
- Favorable comparison with baseline synchronous NoC
- Portable SystemC bursty traffic generator
Asynchronous introduction

No clock within the network
Sender and receiver use handshaking protocol

Advantages
- No dynamic energy when idle (fine-grained clock gating)
- No clock period to meet on long wires
- No global clock tree, saving power
Design goals - a better complexity-effective solution

- Throw out complex features; will it have better power-performance?
- Low energy, simple and efficient circuits
- High area-scaled throughput
  - Sacrifice of general-purpose use and guaranteed-service
### Asynchronous Network Design

#### Design choices: circuits
- Bundled-data encoding instead of delay-insensitive (fewer link wires)
- Two-phase link protocol external to router (high throughput)
- Four-phase protocol internal to router (simple circuits)
- Individually-controlled latches for buffering (power/area savings)

#### Design choices: network
- Three bi-directional router ports (simple MUX for crossbar)
- One-flit per packet (no blocking from many-flit wormhole packets)
- No virtual channels (reduced buffer area/power)
- Source-route path addressing (top-bit controls MUX – simple)
## Asynchronous network

### Design choices: circuits
- Bundled-data encoding instead of delay-insensitive (fewer link wires)
- Two-phase link protocol external to router (high throughput)
- Four-phase protocol internal to router (simple circuits)
- Individually-controlled latches for buffering (power/area savings)

### Design choices: network
- Three bi-directional router ports (simple MUX for crossbar)
- One-flit per packet (no blocking from many-flit wormhole packets)
- No virtual channels (reduced buffer area/power)
- Source-route path addressing (top-bit controls MUX – simple)
Circuit design

Toolflow

Specification: CCS format

Implementation: Petrify & manual

Relative-Timing Constraint (RTC) generation and verification: Analyze/ARTIST tools

Convert RTC to Design Comp. Constr.: manually (for now)

Timing-driven synthesis: Design Compiler

Place & Route: SoC Encounter

Static Timing Anyls: PrimeTime

Functional Validation: ModelSim

Parasitics (Calibre)
Power (HSPICE)
Circuit design

Process technology

- IBM 65 nm 10sf process
- Artisan, regular $V_{th}$ standard cell library (now owned by ARM)
- Custom cell library for the mutex element.
Router sub-modules

**Switch**: Steers data from an incoming channel to outgoing channel
Packet source-routing bits are used for decision

**Merge**: Arbitrates between incoming channels for an output channel
- First-to-arrive request signal is granted access
- Effectively alternates equally between two competing streams
Asynchronous Network Design

Switch module

Din → Latch (MSB) → Dout

Ir → FF → 2-to-4 phase converter

reset

Data Latch

address_swizzling

DEMUX

rr1

rr2

ra1

ra2

linear controller
Merge module

from switch modules to output channel

custom mutex cell

merge controller

Daniel Gebhardt, Junbok You, Kenneth Stevens

University of Utah
Circuit evaluation

Parameters and Tools
- Router: 32-bit data, 8-bit route, 1-flit I/O buffers
- HSPICE for energy measurement with parasitic extraction from Mentor Calibre
- Latency measured using Modelsim with back-annotated delays

Results

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
<td>2740 (\mu m^2)</td>
</tr>
<tr>
<td>Energy/flit</td>
<td>1.66 (pJ)</td>
</tr>
<tr>
<td>Leakage Power</td>
<td>9 (\mu W)</td>
</tr>
<tr>
<td>Throughput</td>
<td>2.38 Gflits/s</td>
</tr>
<tr>
<td>Forward Latency</td>
<td>460 ps</td>
</tr>
<tr>
<td>Backward Latency</td>
<td>250 ps</td>
</tr>
</tbody>
</table>
Synchronous network generation with **COSI**

**COMmunication Synthesis Infrastructure** from U.C. Berkeley [A. Pinto]*

For a specific SoC, generates a NoC optimized for power and latency

**Input**
- Core dimensions
- Average inter-core bandwidth
- Router & link energy models

**Output**
- Floorplan (using Parquet)
- Topology (e.g. hier. star)
- SystemC simulator

*for more information on COSI, see: [http://embedded.eecs.berkeley.edu/cosi/](http://embedded.eecs.berkeley.edu/cosi/)*
COSI modifications

- Added Orion 2.0* energy models for routers and links
- Improved simulation result reporting
- Bursty traffic generator used for simulation

*for information on ORION, see: http://www.princeton.edu/~peh/orion.html
Bursty traffic model

$b$-model *

- Self-similar, not Poisson
- Burstiness determined by bias parameter or $b$, [0.5,1.0)
- Recursively divide traffic volume between two time periods

**Bursty traffic model**

*b-model*

- Self-similar, not Poisson
- Burstiness determined by bias parameter or $b$, $[0.5, 1.0)$
- Recursively divide traffic volume between two time periods


\[ V \times b \]

\[ V \times (1 - b) \]

\[ \frac{t}{2} \]

\[ \frac{t}{4} \]
Bursty traffic model

*b-model*

- Self-similar, not Poisson
- Burstiness determined by bias parameter or \( b \), \([0.5,1.0)\)
- Recursively divide traffic volume between two time periods

\[
V \times b
\]

\[
V \times (1 - b)
\]

Bursty traffic model

*b-model *

- Self-similar, not Poisson
- Burstiness determined by bias parameter or $b$, [0.5, 1.0)
- Recursively divide traffic volume between two time periods

Bursty traffic model implementation

- SystemC, transaction-level-modeling (TLM) library
- For both COSI and asynchronous simulators
- Generates traffic at a message granularity: how many packets to try to send at once.
The ANetGen tool

- Goals similar to COSI, but for our async network
- Input: core floorplan, inter-core bandwidth specification
- Output: router locations, topology
- Objective: minimize wirelength and hop count of high-traffic paths
Asynchronous Network Generation

Solution space exploration

Topology

- Tree topology (for now) – minimum # of 3-port routers
- Simulated-annealing perturbs topology to minimize a fitness
- **congestion**-fitness: \( F_c = \sum_{\text{path}} \text{bandwidth} \times \text{hops} \)
- **wirelength**-fitness: \( F_w = \sum_{\text{path}} \text{bandwidth} \times \text{wirelength} \)
- **full**-fitness = \( F_c k_1 + F_w k_2 \) where \( k_1 \) and \( k_2 \) are normalizing weights

Input: cores & bandwidth

Initial topology & placement

Perturb topology. Calculate congestion fitness

Did congestion fitness improve?

Place routers. Calculate full-fitness

Did full-fitness improve?

iterate until system "cools"

Record solution of highest fitness
Solution space exploration

Force-directed router placement

- Determines wirelength used in Fitness
- "Force" moves routers to shorten lengths of high-traffic paths
- Short paths $\equiv$ lower energy & latency

Path A–B has high traffic

force moves R1, R2 to reduce A-B wirelength

Daniel Gebhardt, Junbok You, Kenneth Stevens

University of Utah
Asynchronous Network Generation

Asynchronous simulator

- Built using SystemC TLM
- Wire delay, $f(length)$, from interpolation of SPICE simulations
- Wire energy estimated by Orion 2.0
- Traffic generation from aforementioned $b$-model
Evaluation Methodology and Setup

Evaluation overview

- Use existing SoC abstractions as benchmarks
- Generate NoCs using COSI and ANetGen
- Hand-design asynchronous network using COSI’s output (for one SoC)
- Simulate resulting NoCs and compare power and latency

Measuring message latency

Messages = 256 bytes, sent sequentially as quickly as possible
Latency = head packet enters network → tail packet leaves network
SoC designs used as benchmarks

- Two benchmarks: ADSTB and MPEG4 decoder
- Given core dimensions and average bandwidth between cores
- Floorplan areas scaled to: $35.7 \text{ mm}^2$ and $78.7 \text{ mm}^2$ respectively

### ADSTB core-to-core average bandwidths

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>MBytes/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>AudioDec</td>
<td>1</td>
</tr>
<tr>
<td>CPU</td>
<td>Demux</td>
<td>1</td>
</tr>
<tr>
<td>DDR</td>
<td>CPU</td>
<td>3</td>
</tr>
<tr>
<td>DDR</td>
<td>MPEG2</td>
<td>593</td>
</tr>
<tr>
<td>Dem2</td>
<td>Demux</td>
<td>31</td>
</tr>
<tr>
<td>Demux</td>
<td>MPEG2</td>
<td>7</td>
</tr>
<tr>
<td>MPEG2</td>
<td>DDR</td>
<td>424</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sender</th>
<th>Receiver</th>
<th>MBytes/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>CPU</td>
<td>DDR</td>
<td>3</td>
</tr>
<tr>
<td>CPU</td>
<td>MPEG2</td>
<td>1</td>
</tr>
<tr>
<td>DDR</td>
<td>HDTVEnc</td>
<td>314</td>
</tr>
<tr>
<td>Dem1</td>
<td>Demux</td>
<td>31</td>
</tr>
<tr>
<td>Demux</td>
<td>AudioDec</td>
<td>5</td>
</tr>
<tr>
<td>HDTVEnc</td>
<td>DDR</td>
<td>148</td>
</tr>
</tbody>
</table>
SoC designs used as benchmarks

- Two benchmarks: ADSTB and MPEG4 decoder
- Given core dimensions and average bandwidth between cores
- Floorplan areas scaled to: 35.7 $mm^2$ and 78.7 $mm^2$ respectively

MPEG4 Communication Trace Graph (CTG). BW in MBytes/s
COSI setup

- Router energy/area models generated by Orion 2.0 (see below)
- Configured to generate a hierarchical star of $\frac{N}{3} + 1$ partitions

### Router model parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Router Freq.</td>
<td>2 GHz</td>
</tr>
<tr>
<td>Tech. Library</td>
<td>65 nm $NV_T$</td>
</tr>
<tr>
<td>Voltage</td>
<td>1.0 V</td>
</tr>
<tr>
<td>No clock-tree power</td>
<td></td>
</tr>
<tr>
<td>Router I/O buff’s</td>
<td>2 / 1 flit</td>
</tr>
<tr>
<td>Crossbar</td>
<td>Multitree</td>
</tr>
<tr>
<td>Flit width</td>
<td>32 bits</td>
</tr>
<tr>
<td>Packet length</td>
<td>4 flits</td>
</tr>
</tbody>
</table>
ADSTB topologies

COSI’s topology

- CPU
- AudioDec
- R:8
- Dem1
- R:9
- Dem2
- Demux
- R:10
- MPEG2
- HDTVEnc
- DDR

ANetGen’s topology

- CPU
- AudioDec
- R:0
- Dem1
- R:2
- Demux
- R:5
- Dem2
- HDTVEnc
- R:3
- DDR
- R:4
- MPEG2
ADSTB topologies

Evaluation Methodology and Setup

Manual async topology

Radix 4 & 5 router replacements

Daniel Gebhardt, Junbok You, Kenneth Stevens

University of Utah
Manual async topology

Radix 4 & 5 router replacements

Manually map high traffic paths through fewer ports, i.e. $A \leftrightarrow B$
MPEG4 topologies

**COSI’s topology**

- risccpu
- sram2
- idct
- sdram
- au
- sram1
- upsamp
- rast
- vu
- babcalc

**ANetGen’s topology**

- risccpu
- sram2
- idct
- sdram
- au
- sram1
- upsamp
- rast
- vu
- babcalc

Daniel Gebhardt, Junbok You, Kenneth Stevens
University of Utah
## Experimental Results

**Power consumption and area**

### Power consumption of routers and wires (mW)

<table>
<thead>
<tr>
<th></th>
<th>rtr dyn</th>
<th>rtr leak</th>
<th>wire dyn</th>
<th>wire leak</th>
<th>TOTAL</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ADSTB</strong></td>
<td>5.5</td>
<td>5</td>
<td>7.86</td>
<td>4.6</td>
<td>23</td>
</tr>
<tr>
<td>Sync</td>
<td>5.5</td>
<td>5</td>
<td>7.86</td>
<td>4.6</td>
<td>23</td>
</tr>
<tr>
<td>Manual async</td>
<td>1.01</td>
<td>0.072</td>
<td>12</td>
<td>8</td>
<td>21</td>
</tr>
<tr>
<td>AnetGen</td>
<td>1.01</td>
<td>0.054</td>
<td>6.3</td>
<td>4.5</td>
<td>11</td>
</tr>
<tr>
<td><strong>MPEG4</strong></td>
<td>12.3</td>
<td>15.7</td>
<td>28</td>
<td>15</td>
<td>71</td>
</tr>
<tr>
<td>Sync</td>
<td>12.3</td>
<td>15.7</td>
<td>28</td>
<td>15</td>
<td>71</td>
</tr>
<tr>
<td>AnetGen</td>
<td>2.4</td>
<td>0.09</td>
<td>20.5</td>
<td>16.7</td>
<td>40</td>
</tr>
</tbody>
</table>

### Router area (µm²)

<table>
<thead>
<tr>
<th></th>
<th>COSI</th>
<th>AnetGen</th>
<th>%reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSTB</td>
<td>99704</td>
<td>15630</td>
<td>84%</td>
</tr>
<tr>
<td>MPEG4</td>
<td>138822</td>
<td>26050</td>
<td>81%</td>
</tr>
</tbody>
</table>

Daniel Gebhardt, Junbok You, Kenneth Stevens
University of Utah
Async routers use much less energy per bit
Manual async NoC has more wires, thus more wire energy, **negating savings in router energy**
ANetGen optimizes wirelength, and is competitive in wire energy
Overhead of routing-bit wires (more evident on MPEG4)
Experimental Results

Message latency - ADSTB

- Histogram of latencies from all messages
- Three bursty values: \{0.5, 0.65, 0.8\}
### Message latency

**Table:** Message latencies (ns); absolute max and bound of 99%.

<table>
<thead>
<tr>
<th>99% less than</th>
<th>Network</th>
<th>Burstiness</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>0.5</td>
</tr>
<tr>
<td>ADSTB sync.</td>
<td></td>
<td>158</td>
</tr>
<tr>
<td>manual async</td>
<td></td>
<td>188</td>
</tr>
<tr>
<td>ANetGen</td>
<td></td>
<td>192</td>
</tr>
<tr>
<td>MPEG4 sync.</td>
<td></td>
<td>838</td>
</tr>
<tr>
<td>ANetGen</td>
<td></td>
<td>275</td>
</tr>
<tr>
<td>Maximum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADSTB sync.</td>
<td></td>
<td>1130</td>
</tr>
<tr>
<td>manual async</td>
<td></td>
<td>510</td>
</tr>
<tr>
<td>ANetGen</td>
<td></td>
<td>510</td>
</tr>
<tr>
<td>MPEG4 sync.</td>
<td></td>
<td>11722</td>
</tr>
<tr>
<td>ANetGen</td>
<td></td>
<td>704</td>
</tr>
</tbody>
</table>
Experimental Results

Message latency per path

Maximum latencies (ns) for ADSTB

![Graph showing message latencies per path for ADSTB with maximum latencies at 2000 ns. The graph compares asynchronous and synchronous communication with two different values of b: 0.50 and 0.80. The CPU is connected to the AudioDec.]
Experimental Results

Message latency per path

ANetGen yields lower latencies at high burstiness

**Median** latencies (ns) for MPEG4

![Graph showing message latency per path for ANetGen with asynchronous and synchronous transmission with different burstiness values (b=0.50 and b=0.80).]
## Source’s output buffer delay

**Table:** Source output buffer packet delay (ns).

<table>
<thead>
<tr>
<th>Network</th>
<th>Burstiness</th>
<th>Median</th>
<th>0.5</th>
<th>0.65</th>
<th>0.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADSTB</td>
<td>sync.</td>
<td>96</td>
<td>730</td>
<td>390065</td>
<td></td>
</tr>
<tr>
<td></td>
<td>manual async.</td>
<td>90</td>
<td>508</td>
<td>320018</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANetGen</td>
<td>90</td>
<td>500</td>
<td>319818</td>
<td></td>
</tr>
<tr>
<td>MPEG4</td>
<td>sync.</td>
<td>274</td>
<td>170336</td>
<td>1.1e6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANetGen</td>
<td>84</td>
<td>250</td>
<td>171496</td>
<td></td>
</tr>
<tr>
<td>Maximum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ADSTB</td>
<td>sync.</td>
<td>1274</td>
<td>261112</td>
<td>1.3e6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>manual async.</td>
<td>952</td>
<td>215908</td>
<td>1.2e6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANetGen</td>
<td>912</td>
<td>231242</td>
<td>1.2e6</td>
<td></td>
</tr>
<tr>
<td>MPEG4</td>
<td>sync.</td>
<td>11683</td>
<td>1.2e6</td>
<td>2.99e6</td>
<td></td>
</tr>
<tr>
<td></td>
<td>ANetGen</td>
<td>1036</td>
<td>171358</td>
<td>1.1e6</td>
<td></td>
</tr>
</tbody>
</table>
Conclusion

- Simple routers + ANetGen yields **less power, area, latency**
- Async. network helped by **single-flit packets** under bursty traffic
- Higher traffic burstiness can dramatically change results, but ANetGen showed less latency increase
- Overhead of source-route wires may be too high for larger SoCs
- Need to investigate energy-efficient ways to improve scalability
- **Wire power is the killer** when routers are efficient
Conclusion

- Simple routers + ANetGen yields **less power, area, latency**
- Async. network helped by **single-flit packets** under bursty traffic
- Higher traffic burstiness can dramatically change results, but ANetGen showed less latency increase
- Overhead of source-route wires may be too high for larger SoCs
- Need to investigate energy-efficient ways to improve scalability
- **Wire power is the killer** when routers are efficient
Questions
Linear controller design

- CCS Specification

LEFT lr.'c1.'la.'c2.lr.'la.
RIGHT c1.'rr. c2.ra.'rr.ra.RIGHT
SPEC (LEFT | RIGHT) {c1, c2}

- Petri-net

- Circuit Implementation

- Relative Timing Constraint

- rtc1: lr ↑ ⇒ la ↑ < y_↓
- rtc2: lr ↑ ⇒ rr ↑ < y_↓
- rtc3: lr ↑ ⇒ y_↓ < la ↓
- rtc4: lr ↑ ⇒ y_↓ < rr ↓
Backup

Merge controller design

- CCS Specification

LEFT lr.‘c1.‘la.‘c2.lr.‘la. LEFT
RIGHT c1.‘rr. c2.ra.RIGHT
SPEC (LEFT | RIGHT) {c1, c2}

- Petri-net

- RT constraints

- Circuit Implementation

* Total 23 RT constraints.
2-4 phase converter design

- Manual design

- Timing Diagram

- Relative Timing Constraint
  - rtc1: lr2 ⊢ → la4 ↓ < lr2 ↓
  - rtc2: lr2 ⊢ → la4 ↓ < lr2 ↑
COSI weighted vs. non-weighted arbitration