BeiHang Short Course, Part 5: Pandora “Smart” IP Generators

James C. Hoe
Department of ECE
Carnegie Mellon University
Collaborator: Michael Papamichael

CONNECT NoC IP Generator
http://users.ece.cmu.edu/~mpapamic/connect/

- Highly-parameterized “push-button” framework
- Multiple interfaces for configuring the desired NoC
- Bundled output
  - synthesizable Verilog RTL
  - documentations
  - testbenches and scripts
- Public web-based release in March 2012
  - actively used by researchers
  - 4500+ unique visitors, 1200+ NoCs generated

User Breakdown
- 35% Academia
- 40% Industry
- 25% Other
- 5% Other
Generator as IP

- Powerful? **Very!**
- Easy to use? **Not Really . . . .
  - low-level cryptic domain-specific parameters
  - complexity of integrating, using, tuning and validating an instantiated IP within an enclosing context
- If you went to **CONNECT** for an NoC right now
  - which configuration would you ask for first?
  - if not good enough, how to get a better one . . . .
  - do you know what good enough is . . . .

---

Rethink the IP Paradigm

- Generator is the “21st-century” IP
- Why limit to structural view of design
- Why not offer also . . . .
  - pre-knowledge about outcome & tradeoff of parameter combinations
  - IP-specific “meaningful” parameterizations,
    that is, ask **how fast?** instead of **how many?**
  - any X where IP authors can do better than IP users

**Shift burdens from IP users to IP authors**

⇒ make knowledge and expertise reusable
Outline

- **CONNECT** Demo and Motivation
- Pandora Smart-IP Paradigm
- Applying Pandora to **CONNECT**
- Conclude

---

**IP Ecosystem and Roles**

**IP Users**

- **Application Developers**
  - Assemble, configure and integrate multiple IPs to build larger chips

**IP Authors**

- **Domain Experts**
  - Know the underlying algorithms and theory specific to the domain

- **Hardware Experts**
  - Can build HW based on a set of specs or SW implementation

---

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L5-s7
Pandora Smart-IPs

- IP enriched with domain/HW expert knowledge to simplify IP use (integration, tuning, validation)

- Bridge the gap between
  - high-level app goals+req’s
  - low-level IP-specific knowledge: algo+HW

- May save work for the IP author in the long term

Pandora Key Principles

1. Embodying Domain-Expert Knowledge
2. Raising the Level of Abstraction
3. Instrumentation and Introspection
4. Providing Supporting Collaterals
IP know thyself

1 - Embody Knowledge

- Carry quantitative and qualitative meta-data
  ⇒ facilitates faster, informed design space navigation

- Generic HW implementation characteristics
  - how IP parameters affect impl. characteristics
    e.g., area, frequency, power/energy, memory size)

- Domain-specific metrics and properties
  - how IP parameters affect higher-level domain metrics
    e.g., for NoC IP: saturation bandwidth, idle latency
  - can also include high-level properties
    e.g., for NoC IP: ordering guarantees

Varying Form and Precision

1 - Embody Knowledge

HW Characterization Libs    Design Space Mapping

Designer Knowledge    Analytical formulas
Don’t ask what I don’t know

Today’s IP generators are richly parameterized
  – needed for efficiency and design space coverage
  – BUT too many low-level parameters ⇒ complexity
• Pandora goal-oriented parameterization
  – in terms of application-level design goals
  – meaningful and intuitive to the non-expert end-user
• Coordinated changes of multiple low-level param’s
to achieve user high-level design goals
  – well-known points, heuristic driven, feedback-search
  – leverage embedded characterization data

Speaking the Right Language

What is “high-level”?
  – depending on user objectives and levels of expertise
  – varying forms and levels of abstraction

Configuration interfaces
Pick points in the design space, e.g.,

- Set of good configurations or personalities
- Objective & Constraint-Driven Queries

Tuning Interfaces
Move within the design space, e.g.,

- Tuning knobs
- Fuzzy trade-off selectors
Do you know what your IP is doing?

1 2 3 - Instrumentation & Introspection 4

- Instrumentations
  - tap into design for monitoring validation purposes
  - monitor, collect and analyze info about IP’s operation

- Introspection
  - process low-level data to reach high-level conclusions
  - diagnose & fix performance or correctness issues

Built-in and Context-Appropriate

1 2 3 - Instrumentation & Introspection 4

- Created by the IP author with design familiarity
  ⇒ easy to use, no guess work by IP user
- Feedback interpreted with IP author knowledge
  ⇒ natural & meaningful to IP user
e.g., protocol checker and deadlock detector

Greatly facilitate debug and validate

- Synthesizable?
  - monitor deployed or under-test designs
  - minimal speed overhead, full fidelity
  - capture rare or hard-to-reproduce behaviors
Providing “Full Service”

- Elementary supporting material
  e.g., testbenches, documentation, etc.

- Scripts and interfaces, e.g.,
  - for configuring the IP, or
  - processing output logs

Minimizes barrier-to-entry for IP user

Outline

- CONNECT Demo and Motivation

- Pandora Smart-IP Paradigm

- Applying Pandora to CONNECT

- Conclude
Generic HW Characterization

1 - Embody Knowledge

• HW resource usage (e.g., slices, BRAM, LUTRAM) & frequency
• At the network level, router level and subcomponents

Sample characterization targeting FPGAs

<table>
<thead>
<tr>
<th>Flit Widths</th>
<th>128 bits</th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 flits</td>
<td>1 slice</td>
<td>2 slices</td>
<td>4 slices</td>
</tr>
<tr>
<td>8 flits</td>
<td>2 slices</td>
<td>4 slices</td>
<td>8 slices</td>
</tr>
<tr>
<td>16 flits</td>
<td>4 slices</td>
<td>8 slices</td>
<td>16 slices</td>
</tr>
<tr>
<td>32 flits</td>
<td>8 slices</td>
<td>16 slices</td>
<td>32 slices</td>
</tr>
</tbody>
</table>

• Frequency

At the network level, router level and subcomponents

Sample characterization targeting FPGAs

<table>
<thead>
<tr>
<th>Flit Widths</th>
<th>128 bits</th>
<th>256 bits</th>
<th>512 bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>4 flits</td>
<td>1 slice</td>
<td>2 slices</td>
<td>4 slices</td>
</tr>
<tr>
<td>8 flits</td>
<td>2 slices</td>
<td>4 slices</td>
<td>8 slices</td>
</tr>
<tr>
<td>16 flits</td>
<td>4 slices</td>
<td>8 slices</td>
<td>16 slices</td>
</tr>
<tr>
<td>32 flits</td>
<td>8 slices</td>
<td>16 slices</td>
<td>32 slices</td>
</tr>
</tbody>
</table>

Domain-specific characterization

1 - Embody Knowledge

• Network performance metrics, e.g., load-latency across various NoCs and traffic patterns
• Capture Network properties qualitative criteria and heuristics, e.g., traffic isolation, QoS properties, buffer size based on packet size

Uniform Random Traffic

90% Neighbor Traffic
Interface at Multiple Expertise Levels

User Expertise | User Input | Resulting NoC
---|---|---
Novice | app personalities, common configs, traces | Pandora configuration and refinement engine
Intermediate | comm. graph, HW impl. trade-off, design-space queries |
Expert | low-level config. spec |

Novice / Intermediate / Expert

1. Pick NoC based on quantitative input, e.g.,
   - weighted graph representing communication pattern
   - traffic traces from application runs
2. Pick NoC based on qualitative input, e.g.,
   - “mostly neighbor request-response traffic, 3 hot spots, need traffic isolation”
3. Design space query on objectives & constraints, e.g.,
   - “lowest area NoC @ 800MHz & has bisection bandwidth of 8 Gbps”

These examples don’t help me if my IP is not NoC :(.
Runtime Monitoring and Analysis

1. Monitoring of events & gathering of statistics
   - e.g., link utilization, allocation conflicts, average load per endpoint/router/port, avg. buffer occupancy, how frequently high-priority traffic blocked low-priority traffic

2. Capturing sequence of events
   - e.g., protocol transitions triggered by flow control messages or transient temporal network behavior

3. Built into the NoC and tailored to specific instance

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L5-s23

---

Runtime Monitoring and Analysis

1. Dynamically detect configuration & usage issues
   - e.g., invalid routing configuration, packets making U-turns, allocation errors, improper implementation of flow control

2. Diagnostic tools that capture domain-specific effects
   - e.g., deadlock, livelock or starvation detection
   - offline post-processing for more sophisticated analysis, e.g., detect suboptimal endpoint mapping on NoC

Aid debugging, provide feedback, and guide tuning

J. C. Hoe, CMU/ECE/CALCM, © 2014, BHSC-L5-s24
“Generator as IP” to “IP as a Service”

- Packaging and releasing the IP as a service
  - only need internet and browser to use
  - no worry about equipment, environment and tools setup
  - transparent updates and fixes
  - usage statistics, crowd-sourcing

Recap the Last Hour

1. **Embodying Domain Knowledge**
   Design space characterization w.r.t. HW cost + domain metrics

2. **Raise the Level of Abstraction**
   High-Level Interfaces tailored to domain, meaningful to IP user

3. **Instrumentation and Introspection**
   Monitor, collect and analyze info about IP operation

4. **Providing Supporting Collaterals**
   Supporting material, toolset, release strategies
What else could we do?

- Standard interfacing schema for composing IPs
- Can composed Smart-IPs cross-tune amongst themselves?
- Frameworks for building Smart-IPs: expressing structural and meta-knowledge/expertise
- What else to encapsulate?