System adaptivity and fault-tolerance in NoC-based MPSoCs: the MADNESS project approach

leandro.fiorin@usi.ch

“Though this be madness, yet there is method in't”
Hamlet Act 2, scene 2, 193-206

MADNESS project - www.madnessproject.org
MADNESS - Methods for predictAble Design of heterogeneous Embedded Systems with adaptivity and reliability Support

Project Coordinator
Name: Luigi Raffo
Institution: University of Cagliari (Italy)
Email: raffo@unica.it
Project website:
http://www.madnessproject.org

Duration: 36 months
Start: 2010.01.01
Total Cost: 2.92 M€
EC Contribution: 1.95 M€
Contract Number: INFSO-ICT-248424

Partners:
Università degli Studi di Cagliari (Italy)
Università della Svizzera italiana (Switzerland)
Silicon Hive (The Netherlands)
Lantiq Deutschland GmbH (Germany)
Universiteit van Amsterdam (The Netherlands)
Universiteit Leiden (The Netherlands)
Informatik Centrum Dortmund (Germany).
Outline

• Motivations
• Objectives
• The MADNESS project: in brief
• Adaptivity and fault tolerance
• Architectural support for adaptivity and fault tolerance
• Polyhedral Process Network communication APIs
• Process migration mechanism
• Assessment: task migration overhead
• Assessment: remapping strategy accuracy
Motivations

- Modern embedded MPSoC architectures integrate a huge number of heterogeneous components.
- Increasing complexity in fitting different computational tasks to specific processing elements, achieving optimal use of resources and parallelism available.
- Irregular communication patterns that require ad-hoc interconnect infrastructures.
- For certain classes of applications, “static” design needs to be extended including the “dynamic” generation of processor IP.
- A certain degree of fault tolerance must be guaranteed, while avoiding the use of expensive massive redundancy.
- Adaptivity to support variable QoS/user/application requirements.
Objectives

- To define **innovative methodologies** for system-level design, able to guide designers and researchers to the optimal composition of an MPSoC architecture.
- The proposed methodologies will extend the classic design space exploration to:
  - improve **design predictability**, bridging the so called “implementation gap”.
  - consider, apart from more traditional constraints, **continued availability of service**, taking into account fault recovery as one of the optimization factors to be satisfied.
  - support **adaptive runtime management** of the architecture, considering new metrics posed by novel dynamic strategies and advanced support for communication issues.
The MADNESS framework: in brief

- **Hardware library** of IP blocks explored in varying configurations
- Design Space Exploration (DSE) engine
- FPGA-based emulation and evaluation platform
- Re-configurable compilation tool-chain and HAL
- Support for system adaptivity and fault tolerance
Adaptivity and fault tolerance

- Probability that some manufacturing defect will escape end-of-production testing or that faults will become evident during normal operation has to be taken into account.
- "Traditional" approaches (involving massive redundancy) hardly adoptable in constrained devices (e.g. embedded devices).
- Due to time-to-market requirements, solutions leading to continued availability in the presence of faults must be developed within the normal design flow.

Objectives:
- To build a system-level methodology supporting continuity of service by providing self-checking policies and system-level reconfigurations policies.
- To define and implement an optimal architecture for the monitoring and detection of faults in the behavior of the system (including a self-monitoring of elements of the NoC).
- To define reconfiguration methodologies reacting to hard faults affecting the NoC, as the ones incapacitating other IPs present in the MPSoC, with the final target of dynamically mapping the application onto the MPSoC.
Adaptivity and fault tolerance

• **Fault detection:**
  – we target detection of both temporary and permanent run-time faults
  – at architectural level:
    • NoC: use of self-checking strategies
    • Processing elements (not critical): predefined software testing routines
  – at application level:
    • Processing elements (critical): concurrent self-checking at KPN level

• **Reconfiguration strategies:**
  – at architectural level:
    • NoC: limited redundancy on specific components, whose use is activated at the detection of run-time permanent faults
  – at application level:
    • Processing elements: online remapping of KPN tasks running on faulty processors
Adaptivity and fault tolerance

- New applications can be loaded at run-time
- Power-aware application management techniques are often needed
- With deep-sub-micron technology, the possibility of experiencing faults in the circuitry is significant

- Proposed solution: allow the re-mapping of application processes onto the processing cores at run-time

- Requires support at:
  - architectural level
  - Software/Middleware stack
  - Fault detection
  - Remapping strategy
• System architecture
  – NoC-based
    • Arbitrary topology (irregular topologies allowed)
    • NI extended to support message-passing communication protocol (overlapping communication and computation)
Architectural support for adaptivity and fault tolerance

- **System architecture:**
  - Generic processing elements
    - No instruction set extension (communication and synchronization managed accessing memory-mapped registers at the NI)
  - DMA-based message passing handler
  - Dual-port Data and Instruction memories
  - Local bus to access memory mapped resources
  - Tag decoding for interrupt generation

Programming model:
- `SHMPI_send(*char msg_addr, int tag, int size, int dest_ID);`
- `SHMPI_recv(*char msg_addr, int tag, int size, int send_ID);`
- `SHMPI_nb_recv(*char msg_addr, int tag, int size, int send_ID);`
Polyhedral Process Networks communication APIs

- Implement block on read and block on write behavior on top of SHMPI primitives
- No remote memory access implies flow control
- Request Driven flow control
- Two software FIFOs
- Request messages (polling- or interrupt-based)

• On the source tile:
  • the process is stopped
  • state is saved and forwarded to the destination tile
  • migration decision is propagated to the other involved tiles (dashed arrows)

• On the destination tile:
  • The destination tile receives a specific message for process activation
  • the required software FIFOs are created
  • the replica of the migrated process is activated using OS call
  • the state of the migrated process is resumed
  • the execution starts from the beginning of the interrupted iteration

• On the predecessor and successor tile(s):
  • the middleware storing the mapping of the processes are updated
Fault tolerant support:
- Fault detection
  - Self-testing module
  - PPN-level self-checking patterns
- Task migration hardware module
  - Embedded into the Network Adapter
  - Collaborate with the resource manager for performing the migration
Assessment: task migration overhead

\[\tau_0, \tau_1\]: time required by the run-time manager to make the remapping decision
\[\tau_1, \tau_2\]: time required by the source tile (tile1) to send all the process state to the destination tile
\[\tau_2, \tau_3\]: the destination tile (tile2) copies the process state to its local memory and starts the execution

\[(\tau_3-\tau_0) = 28,934\] clock cycles

Texe(P1) may be lost due to the migration mechanism

\[(\tau_3-\tau_0) + \text{Texe}(P1) = 152,560\] clock cycles

Max Overhead is only 0.88% of the total execution time
Assessment: remapping strategy accuracy

(a) tile₁: P₁, tile₂: P₂, tile₃: P₀, tile₄: P₃

(b) tile₁: P₁, tile₂: P₂, tile₃: P₀, tile₄: P₃

(c) tile₁: P₁, tile₂: P₂, tile₃: P₀, tile₄: P₃

<table>
<thead>
<tr>
<th>Remapping</th>
<th>Measurement</th>
<th>Analytical model</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>53.42</td>
<td>56.02</td>
</tr>
<tr>
<td>R₂</td>
<td>-1.09</td>
<td>1.56</td>
</tr>
<tr>
<td>R₃</td>
<td>41.75</td>
<td>38.82</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remapping</th>
<th>Measurement</th>
<th>Analytical model</th>
</tr>
</thead>
<tbody>
<tr>
<td>R₁</td>
<td>53.42</td>
<td>56.02</td>
</tr>
<tr>
<td>R₂</td>
<td>0.04</td>
<td>0.00</td>
</tr>
<tr>
<td>R₃</td>
<td>-0.03</td>
<td>0.00</td>
</tr>
</tbody>
</table>
Thank you!

Questions?

leandro.fiorin@usi.ch