Design Space Exploration of Stream-based Dataflow Architectures

Dr. ir. Bart Kienhuis,
Delft University of Technology

In cooperation with
Philips Research Laboratories

E-mail: kienhuis@cas.et.tudelft.nl

29 January, 1999
System Level Design

Application

Architecture

Integrated Circuit

(Real-time)

Programmability
Multi-functional
Multi-standard

Cost Effective

IC
New High-performance DSP Architectures

- Operations: 1 Gops - 10 Gops
- Bandwidth: 1000 Mbytes - 10.000 Mbytes per second

Focus is on **Stream-based Dataflow Architectures**
Outline

- Architecture Template & Video Applications
- The Y-chart Approach
- The Y-chart Environment for Stream-based Dataflow Architectures
  - Retargetable Simulator
  - Mapping
- Design Space Exploration
- Example
- Conclusions
Architecture Template

Design Choices
- Function Repertoire
- Grain Size
- Controller Protocol
- Buffer Capacity
- Packet Length

Metrics Constraints
- Throughput (real-time)
- Utilization
Video Algorithms

Specified as Kahn Process Networks \( (\text{Lee}&\text{Parks}'95) \)

- Dynamic Dataflow
- Deterministic execution trace \( (\text{Kahn}'74) \)

Assumptions

- Coarse-grained Functions
- Sample Based
Problem Statement

The Designer’s Problem

- Many design choices
- Need to evaluate different design alternatives

General and structured design approaches are lacking
The Y-chart Approach
(The Methodology)

Architecture Instance

Mapper

Performance Analysis

Performance Numbers

Applications
The Y-chart Approach (cont’d)

(Abstraction Pyramid)

Diagram showing the relationship between abstraction level and cost of modeling/evaluation. The diagram is a triangle with three levels:
- High: Back-of-the-envelope models
- Low: Estimation models
- Middle: Abstract executable models, cycle-accurate models, synthesizable VHDL models

The diagram illustrates the design space exploration of stream-based dataflow architectures.
The Y-chart Approach (cont’d)

(Stack of Y-Charts)
The Y-chart Approach (cont’d)

(Design Trajectory)
Y-Chart Environment

(For Stream-based Dataflow Architectures)
Retargetable Architecture Simulator

Required:
- Retargetable simulator
- Cycle accurate
- Fast Simulator
- Functional Correct

<table>
<thead>
<tr>
<th>Simulator (Architecture)</th>
<th>Lang.</th>
<th>Accuracy</th>
<th>Sim. Speed Instr./sec</th>
<th>1 Video Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>SPIM (MIPS 3000)</td>
<td>C</td>
<td>instruction</td>
<td>200.000</td>
<td>10 min.</td>
</tr>
<tr>
<td>tmsim (TriMedia)</td>
<td>C</td>
<td>clock-cycle</td>
<td>40.000</td>
<td>54 min.</td>
</tr>
<tr>
<td>DLX (DLX)</td>
<td>VHDL</td>
<td>RTL</td>
<td>500</td>
<td>1.2 day.</td>
</tr>
<tr>
<td>ORAS</td>
<td>C++</td>
<td>cycle</td>
<td>10.000</td>
<td>3.6 hours</td>
</tr>
</tbody>
</table>
Object Oriented Retargetable Architecture Simulator

1. Structure
   Architecture Template
   Grammar

2. Execution Model
   Processes
   Function Library (SBF)
   Executable
   Arch. Inst.

3. Measuring
   Routing Program
   Instrumenting
   Simulator
   Metric Collectors

Building Blocks

Architecture Description

Architecture Elements
Defined as Classes

PAMELA constructs
Run-Time Library

Object Oriented Principles

PAMELA

Design Space Exploration of Stream-based Dataflow Architectures / 29 January, 1999
Step 1: Architecture Description

```plaintext
FunctionalUnit {
  Type: Packet;
  FunctionalElement FilterA(1,1) {
    Type: Synchrone;
    Function { Type:
                 HighPass(throughput=1,latency=18); }
    Binding { Input ( 0->0 ); Output ( 0->0 ); }
  }

  FunctionalElement FilterB(1,1) {
    Type: Synchrone;
    Function { Type:
                 LowPass(throughput=1,latency=15); }
    Binding { Input ( 0->1 ); Output ( 0->1 ); }
  }
}
```

Diagram showing the architectural elements with base classes, abstract classes, and derived classes.
Step 2: Execution Model

FIFO::read
{
    pam_P (data);
    aSample = queue[readfifo];
    readfifo = (++readfifo)%cap;
    pam_delay (1);
    pam_V (room);
}

FIFO::write( aSample )
{
    pam_P (room);
    queue[writefifo] = aSample;
    writefifo = (++writefifo)%cap;
    pam_delay (1);
    pam_V (data);
}

Performance Modeling PAMELA
- Mutual Exclusivity
- Condition Synchronization

- Processes
- Synchronization Primitives
- Time
Step 3: Metric Collectors

<table>
<thead>
<tr>
<th>Element Type</th>
<th>Performance Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Comm. Structure</td>
<td>Utilization</td>
</tr>
<tr>
<td>Controller</td>
<td>Utilization</td>
</tr>
<tr>
<td>Buffer</td>
<td>Filling distribution</td>
</tr>
<tr>
<td>Routers</td>
<td>Response Time Controller</td>
</tr>
<tr>
<td>Functional Unit</td>
<td>Utilization, Number of Context Switches</td>
</tr>
<tr>
<td>Functional Element</td>
<td>Utilization, Pipeline Stalls</td>
</tr>
<tr>
<td></td>
<td>Throughput, Number of Operations</td>
</tr>
<tr>
<td>Architecture</td>
<td>Number of Operations, Total execution time</td>
</tr>
</tbody>
</table>
Mapping is ignored when too difficult.
Mapping (cont’d)

Mapping Approach:
- Explicit description of both Architecture and Applications
- Description Formalisms and Data Types should correspond

![Diagram showing the mapping approach]

- Architecture
- Algorithms
- Model of Architecture
- Model of Computation
- Functional Elements
- Stream Based Functions
- Ordering of Samples
- Time
- Data Type
- Streams
- Samples
- Ordering of Samples
Stream-based Functions

Basis:

1. Describe a network as a Kahn Process Network (Kahn '74)

2. Structure the nodes based on the AST model of Backus (Backus '78)
   - Controller
   - State
   - Set of Functions
Example of an SBF Object

**Binding Function**

\[
\mu(s) = \begin{cases} 
  f_a, & \text{if } s = 0 \\
  f_a, & \text{if } s = 1 \\
  f_b, & \text{if } s = 2 \\
  f_c, & \text{if } s = 3,
\end{cases}
\]

**Transition Function**

\[
\omega(s) = s + 1 \pmod{3}.
\]

<table>
<thead>
<tr>
<th>Current State</th>
<th>Function</th>
<th>Buffer0</th>
<th>Buffer1</th>
<th>Buffer2</th>
</tr>
</thead>
<tbody>
<tr>
<td>(s_0)</td>
<td>(f_a)</td>
<td>R</td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td>(s_1)</td>
<td>(f_a)</td>
<td>R</td>
<td>R</td>
<td>W</td>
</tr>
<tr>
<td>(s_2)</td>
<td>(f_b)</td>
<td>R</td>
<td>W</td>
<td>W</td>
</tr>
<tr>
<td>(s_3)</td>
<td>(f_c)</td>
<td>W</td>
<td>W</td>
<td>W</td>
</tr>
</tbody>
</table>
Mapping of an SBF Object

Edge (maps to) SBF Object (maps to) Stream

Output Buffer → Router → Input Buffer

Functional Unit

header data

packet
Mapping (cont’d)

The Mapping approach results in an Application/Architecture interface

- No need to rewrite applications
- Capture the correct timing behavior of applications on arbitrary architecture instances
  - Pipeline and Throughput

Example:

Function { Type: LowPass(throughput=1, latency=15); }
Design Space Exploration

Inverse Transformation

The Acquisition of Insight
Design Space Exploration (cont’d)
Design Space Exploration (cont’d)

Script Language PERL

Input Text

Architecture description

Mapping Table

Parameters

Results

ORAS

Nelsis Box

Simulatie

Output Text

Results from the simulations
Experiment

- Packet Length: \{ 4 \ldots 256 \} Samples per Packet
- Service Time: \{ 1 \ldots 20 \} Cycles per Request
Results

Parallelism

Result of simulating 25 different Architecture Instances
Results (cont’d)

Utilization
Conclusions

Presented a Method and Tools for exploring Stream-based Dataflow Architectures.

Better Engineered Architectures in less Time

Method

- Y-chart approach to quantify design choices
- Y-chart environment for Stream-based Dataflow Architectures
- Systematic exploration of the design space

Tools

- Object Oriented Retargetable Architecture Simulator (ORAS)
- The SBF Model
- Design Space Exploration environment based on Nelsis
Future Work

- Generalize presented methods and tools for heterogeneous system design at different levels of abstraction (Lieverse et al.)
- Compiling Matlab applications into descriptions in terms of the SBF Model (Rypkema et al.)

For more information look at:

http://cas.et.tudelft.nl/research/hse.html