Modeling Dynamically Reconfigurable Systems via Rewriting-Logic (modeling and simulation of the FFT in Optimal Space)


Overview (Arvind approach)

- Applying rewriting techniques in hardware design [Arvind et al]
  - specification of correct processors
  - Cache protocols over memory systems
  - Specification of digital circuits
  - Specification and verification of network protocols

Characteristic of Arvind’s approach

- rewriting is neither applied for simulation nor for verification
- Proposal ⇒ Translate to Verilog!

Overview (using Haskell)

- Bejesse et al use Haskell (a functional language) for circuit design, specification and verification
- These ideas are implemented in LAVA system
- This approach shows how the high level abstraction of functional languages is suitable for hardware design

Lava approach takes advantage of high level abstraction provided by functional languages

Overview (Kapur Approach)

- Kapur has used his well-known Rewriting Rule Laboratory - RRL for verifying arithmetic circuits
- RRL is used to verify automatically properties of arithmetic hardware circuits (adders, multipliers, SRT division circuits)

Why Rewriting?

- Rewriting is the formal framework of all functional languages
- This fact allows us to work in more abstract levels
- Rewriting assistant environments help in the task of formal verification of hardware
Rewriting Rules

Rewriting

- Rewriting rules:
  \[ l \rightarrow r \text{ if } C \]
  - Premise to be hold
  - Semantic: “l is replaced by r if C is true”

- Operational semantics:
  a rule is applied to a term, when its left-side matches a sub-term with the corresponding right-side of the rule. All that, whenever the premise C of the rule holds.

Specifying Processors

Specifying Processors (Arvind’s proposal)

SYS(mem,Proc)

PROC(ia, rf, prog)

Defining Instruction of the processor by rewriting rules

Jz-jump-rule:

[ Jz(r_1, r_2) ]

Proc(ia, rf, prog) \rightarrow Proc( rf[r_2], rf, prog)

if im[ia] = jz(r_1, r_2) and rf[r_1] = 0

Rewriting rules can implement state transitions in the processor

Example: Euclid’s Algorithm for greatest common divisor (GCD)

GCD Mod Rule

Gcd(a, b) \rightarrow Gcd(a-b, b) if (a \geq b) \land (b \neq 0)

GCD Flip Rule

Gcd(a, b) \rightarrow Gcd(b, a) if a < b

The term Gcd(6,15) can be reduced by applying the Mod and Flip rules

Gcd(6,15) \rightarrow \text{flip} Gcd(15,6) \rightarrow \text{flip} Gcd(9,6) \rightarrow \text{flip} Gcd(3,6) \rightarrow Gcd(6,3) \rightarrow \text{flip} Gcd(3,3) \rightarrow Gcd(0,0) \text{ Result!}
Characteristic of Rewriting

- Rules are applied non-deterministically
- Controlling the execution of rules can be accomplished by logic

Rewriting-Logic = Rewriting Rules + Strategies

Examples of Rewriting Oriented Environments

- ELAN: it has great flexibility for defining types and ease manipulation of strategies
- Maude: it’s necessary to do more effort for description
  - it provides model checking useful for hardware verification

Example of a Reconfigurable Architecture

![Reconfigurable Architecture Diagram]

Describing Architectures in ELAN

Problem: how can this architecture be described in ELAN
Using and defining types It’s possible to describe fixed parts and reconfigurable ones

Describing more Complex Architectures

![Complex Architecture Diagram]
How the Execution Process is described in the ELAN system

How the Reconfiguration Process is described in ELAN system

Using Strategies in ELAN

Reconfiguration for FFT

FFT in Optimal Space

An Execution Rule for a pair of MACs
A Reconfiguration Rule for a pair of MACs

\[
\text{[reconfiguration]}
<
\begin{align*}
\text{fix0} & \rightarrow \text{recl} \\
\text{fix1} & \rightarrow \text{rec1}
\end{align*}
>
\[
\begin{align*}
\text{fix0} & \rightarrow \text{addr(0),addr(2),const( < 1,0000 \ 0,0000 > ), < + > , < * > } \\
\text{fix1} & \rightarrow \text{addr(1),addr(3),const( < 1,0000 \ 0,0000 > ), < + > , < * > }
\end{align*}
\]

A Pipelined Reconfigurable FFT (eliminating the reconfiguration overhead)

While one Mac array is being reconfigured the other array is computing one step of FFT

Advantages of ELAN Environment

- ELAN has the advantage of an embedded inference engine
- A flexible type definition mechanism (data and operators)
- A powerful manipulation of typed expressions through rules and meta-rules
- The availability of logical strategies to control their application.

Conclusions

- The high abstraction of Rewriting Environments makes design exploration easier
- Using ELAN is possible to simulate the description of the architecture
- Descriptions in ELAN are close to the physical architecture