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

C. Llanos[2,4] ,M. Ayala-Rincón[1,4], R. B. Nogueira[2,4], R. P. Jacobi[3,4] and R. W. Hartenstein[5,6]

[1] Departamentos de Matemática, [2] Engenharia Mecânica e [3] Ciência da Computação
[4] Universidade de Brasília
[5] Fachbereich Informatik, [6] Kaiserslautern University of Technology

2nd Dagsthul Seminar on Dynamically Reconfigurable Architectures

Dagstuhl, Germany, July 20-25, 2003

Overview(Arvind aproach)

- 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

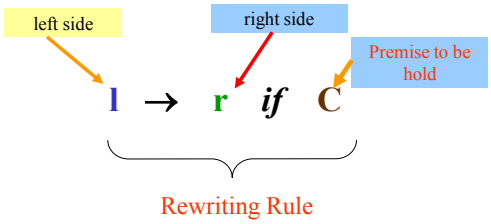
Overview (Kapur Approach)

- 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:

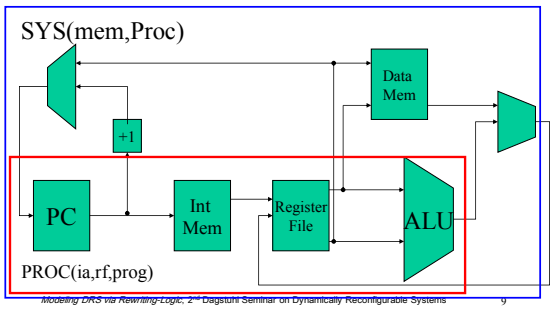
  $l \rightarrow r \ \textit{if} \ C$ (Premise to be hold)

  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, replacing the matched sub-term with the corresponding right-side of the rule. All that, whenever the premise C of the rule holds.

Specifying Processors (Arvind's proposal)



Specifying Processors

- Basic Processor
  - Single cycle, non pipelined, in-order execution
    - **SYS := Sys(MEM,PROC)**
    - **PROC := Proc(ia, rf, prog)**
- AX Architecture

  Instruction set:

r:=Loadc(v)	r:=Loadpc
r:=Op(r_1,r_2)	Jz(r_1,r_2)
r:=Load(r_1)	Store(r_1,r_2)

Defining Instruction of the processor by rewriting rules

Jz-jump-rule: (Conditional jump)

[Jz(r_1, r_2)]

Proc(ia, rf, prog) → 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) → Gcd(a-b, b) if (a ≥ b) ∧ (b ≠ 0)

GCD ***Flip Rule***

Gcd(a, b) → Gcd (b, a) if a < b

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

Gcd(6,15) → Flip Gcd(15,6) → Mod Gcd (9, 6) → Mod Gcd(3, 6) → Flip Gcd (6, 3) → Flip Gcd(3,3) → Mod Gcd(3,0) ← **Result!**

Characteristic of Rewriting

➢Rules are applied non-deterministically

➢Controlling the execution of rules can be accomplished by logic

Rewriting-Logic = Rewriting Rules + Strategies

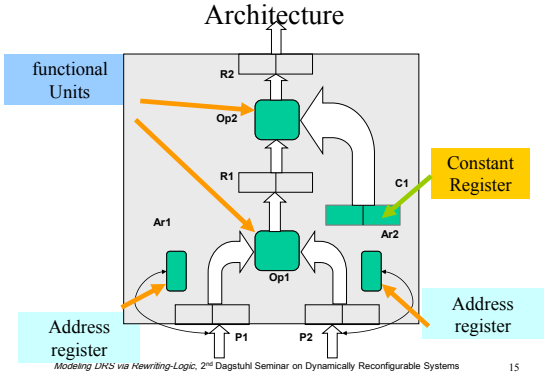
Examples of Rewriting Oriented Environments

➢ELAN: it has great flexibility for defining types and ease manipulation of strategies

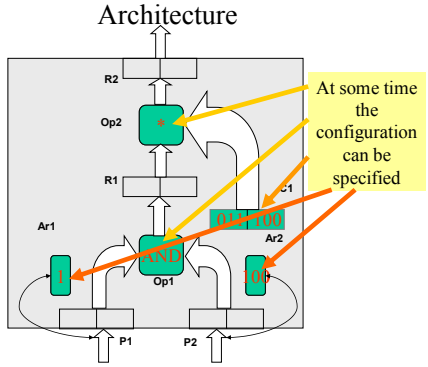
➢Maude: •It's necessary to do more effort for description

•it provides model checking useful for hardware verification

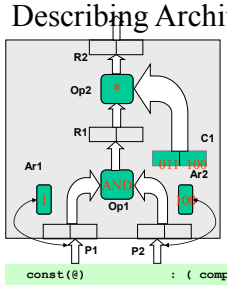
Example of a Reconfigurable Architecture



Example of Reconfigurable Architecture



Describing Architectures in ELAN

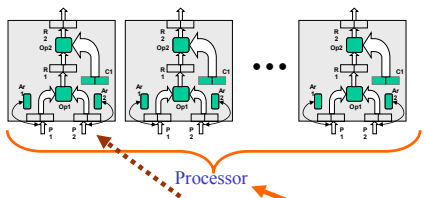


Problem: how can this architecture be described in ELAN

Using and defining types It's possible to describe fixed parts and reconfigurable ones

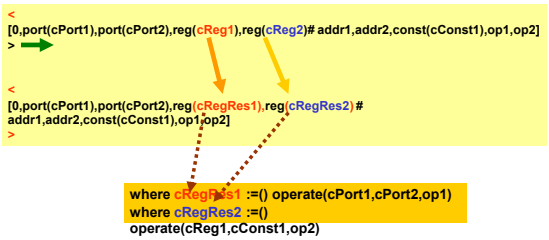
```
const(@)          : ( complexUnit ) Const;
port(@)           : ( complexUnit ) Port;
reg(@)            : ( complexUnit ) Reg;
addr(@)           : ( int ) Addr;
@,@,@,@,@         : ( int Port Port Reg Reg ) fixMAC;
@,@,@,@,@         : ( Addr Addr Const OpUnit OpUnit ) recMAC;
[  @ # @  ]       : ( fixMAC recMAC ) MAC;
```


Describing more Complex Architectures

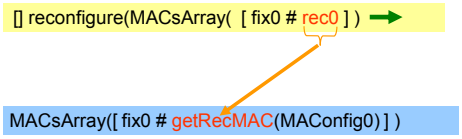



```
@,@,@,@,@         : ( int Port Port Reg Reg ) fixMAC;
@,@,@,@,@         : ( Addr Addr Const OpUnit OpUnit ) recMAC;
[ @ # @ ]         : ( fixMAC recMAC ) MAC;
< @ @ @ @ @ @ @ @ @ @ > : ( int rArrayStruct MAC MAC MAC MAC MAC MAC MAC MAC )Proc;
```

How the Execution Process is described in the ELAN system



How the Reconfiguration Process is described in ELAN system

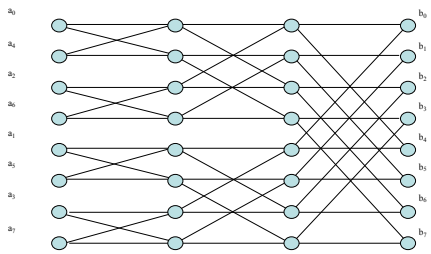


Using Strategies in ELAN

```
strategies for Proc
implicit
  [] process
     input;
          repeat*(reconfiguration;propagation;execution);
     output
  end
end
```

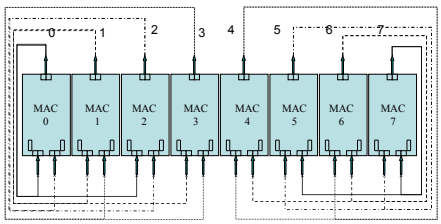
- Using strategies for guiding the application of the rules
- Strategies in ELAN allow to separate execution and reconfiguration steps
- This approach allows a closer specification to transference register description (RTL Description)

Reconfiguration for FFT



Number of reconfiguration = $ln(n) + 1$

FFT in Optimal Space

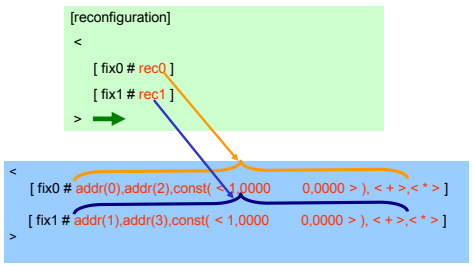


Interconnections in reconfiguration step 3

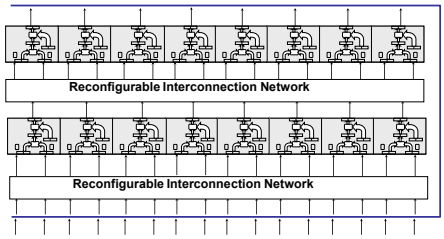
An Execution Rule for a pair of MACs

```
[MAC01]
< [0,port(cPort1),port(cPort2),reg(cReg1),reg(cReg2)#
addr1,addr2,const(cConst1),op1,op2]
  [1,port(cPort3),port(cPort4),reg(cReg3),reg(cReg4)#
addr3,addr4,const(cConst2),op3,op4]
< >
[0,port(cPort1),port(cPort2),reg(cRegRes1),reg(cRegRes2)#
addr1,addr2,const(cConst1),op1,op2]
[1,port(cPort3),port(cPort4),reg(cRegRes3),reg(cRegRes4)#
addr3,addr4,const(cConst2),op3,op4]
>
   where cRegRes1 := () operate( cPort1,cPort2,op1 )
   where cRegRes2 := () operate( cRegRes1,cConst1,op2 )
   where cRegRes3 := () operate( cPort3,cPort4,op3 )
   where cRegRes4 := () operate( cRegRes3,cConst2,op4 )
```

A Reconfiguration Rule for a pair of MACs



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