

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

C. Llanos<sup>2,4</sup>, M. Ayala-Rincón<sup>1,4</sup>, R. Nogueira<sup>2,4</sup>, R. Jacobi<sup>3,4</sup>, R. W. Hartenstein<sup>5,6</sup>

<sup>1</sup>Departamentos de Matemática, <sup>2</sup>Engenharia Mecânica e <sup>3</sup>Ciência da Computação

<sup>4</sup>Universidade de Brasília

<sup>5</sup>Fachbereich Informatik, <sup>6</sup>Kaiserslautern University of Technology

2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Architectures

Dagstuhl, Germany, July 20-25, 2003

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems

## 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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems

## Characteristic of Arvind's approach

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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems

## 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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems

## Overview (Kapur Approach)

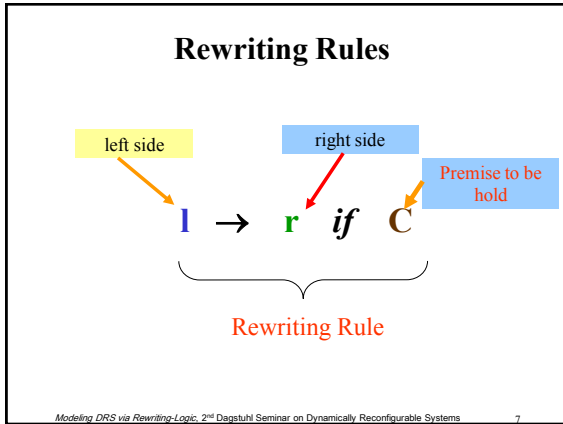
- Kapur has used his well-known *Rewriting Rule Laboratory - RRL* for verifying arithmetic circuits
- RRL is used to verify automatically properties of arithmetic hardware circuits (adders, multipliers, SRT division circuits)

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems

## 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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems



### Rewriting

➤ Rewriting rules:

$$l \rightarrow r \text{ if } C$$

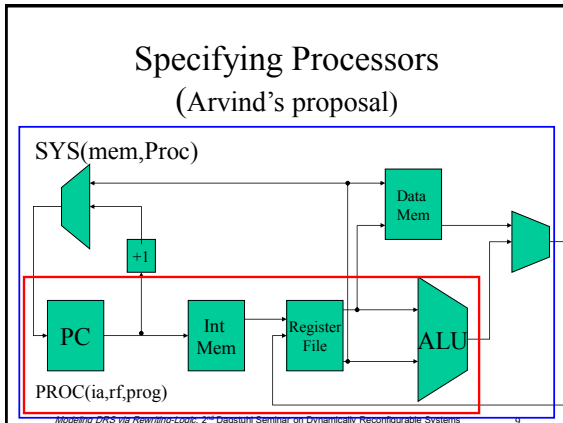
Premise to be hold

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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 8



### Specifying Processors

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

Instruction set:

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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 10

### Defining Instruction of the processor by rewriting rules

Jz-jump-rule: Conditional jump

$$Jz(r_1, r_2)$$

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

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

Rewriting rules can implement state transitions in the processor

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 11

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

GCD Mod Rule

$$Gcd(a, b) \rightarrow Gcd(a-b, b) \text{ if } (a \geq b) \wedge (b \neq 0)$$

GCD Flip Rule

$$Gcd(a, b) \rightarrow Gcd(b, a) \text{ if } a < b$$

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

$$Gcd(6,15) \xrightarrow{Flip} Gcd(15,6) \xrightarrow{Mod} Gcd(9,6) \xrightarrow{Mod} Gcd(3,6) \xrightarrow{Mod} Gcd(6,3) \xrightarrow{Mod} Gcd(3,3) \xrightarrow{Mod} Gcd(3,0) \xrightarrow{Result!}$$

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 12



### How the Execution Process is described in the ELAN system

```
< [0,port(cPort1),port(cPort2),reg(cReg1),reg(cReg2)# addr1,addr2,const(cConst1),op1,op2]
>
[0,port(cPort1),port(cPort2),reg(cRegRes1),reg(cRegRes2) #
addr1,addr2,const(cConst1),op1,op2]
where cRegRes1 := () operate(cPort1,cPort2,op1)
where cRegRes2 := ()
operate(cReg1,cConst1,op2)
```

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 19

### How the Reconfiguration Process is described in ELAN system

```
[] reconfigure(MACsArray( [ fix0 # rec0 ] ) →
MACsArray([ fix0 # getRecMAC(MACConfig0) ])
```

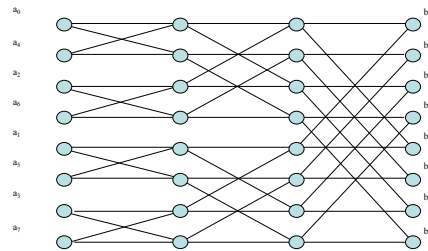
Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 20

### 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)
- Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 21

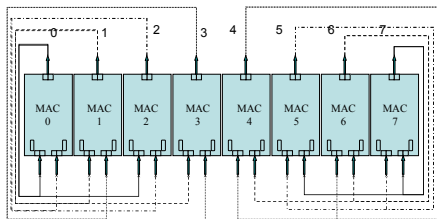
### Reconfiguration for FFT



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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 22

### FFT in Optimal Space



Interconnections in reconfiguration step 3

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 23

### An Execution Rule for a pair of MACs

```
[MAC01]
< [0,port(cPort1),port(cPort2),reg(cReg1),reg(cReg2)#
addr1,addr2,const(cConst1),op1,cP2]
[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( cPort1,cConst1,op2 )
where cRegRes3 := () operate( cPort3,cPort4,op3 )
where cRegRes4 := () operate( cPort3,cConst2,op4 )
```

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 24

### A Reconfiguration Rule for a pair of MACs

```
[reconfiguration]
<
  [ fix0 # rec0 ]
  [ fix1 # rec1 ]
>
```

```
<
  [ fix0 # addr(0),addr(2),const( < 1,0000  0,0000 > ), < + > , < + > ]
  [ fix1 # addr(1),addr(3),const( < 1,0000  0,0000 > ), < + > , < + > ]
>
```

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 25

### A Pipelined Reconfigurable FFT (eliminating the reconfiguration overhead)

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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 26

### 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.

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 27

### 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

Modeling DRS via Rewriting-Logic, 2<sup>nd</sup> Dagstuhl Seminar on Dynamically Reconfigurable Systems 28