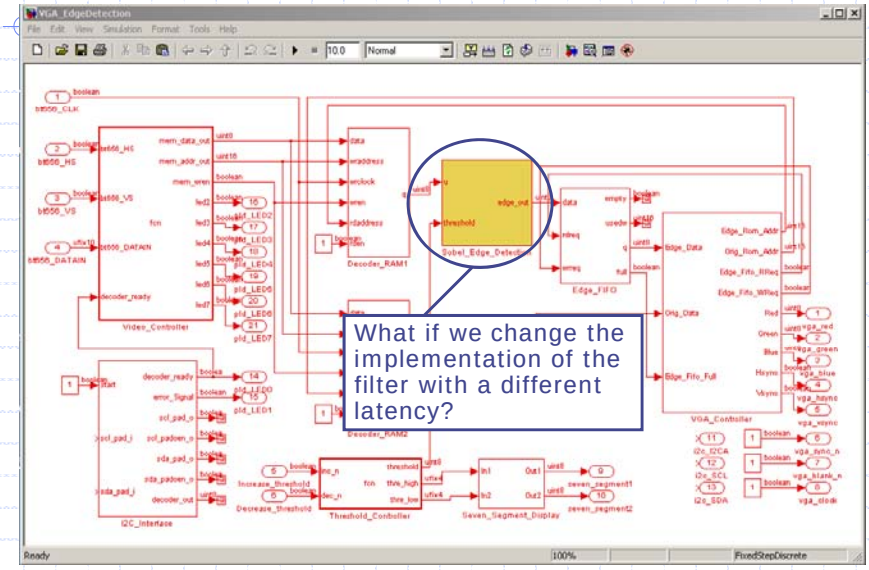
Verification of refinements in rule-based designs

Nirav Dave, Myron King, Arvind (MIT)
Michael Katelman, Jose' Meseguer (Illinois)

WG 2.8, Marble Falls, TX
March 11, 2010

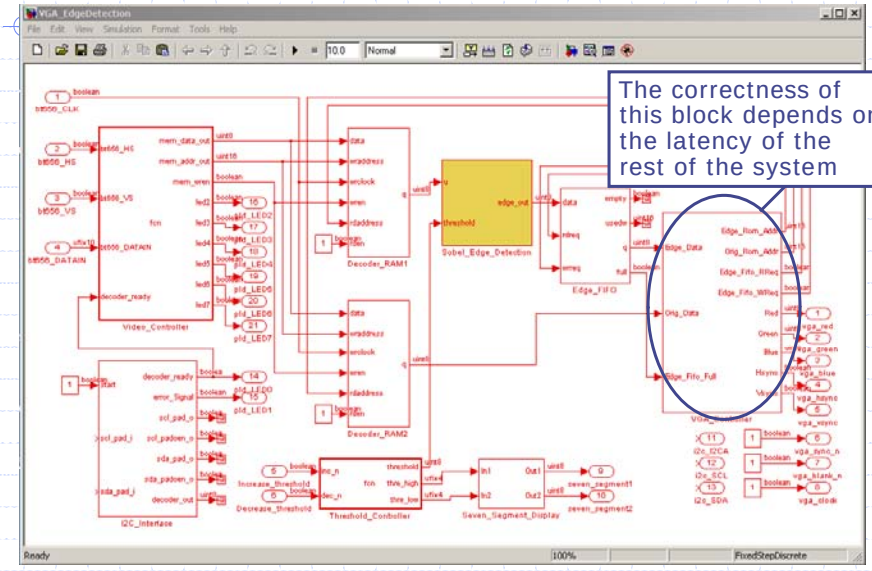
1

A typical RTL design



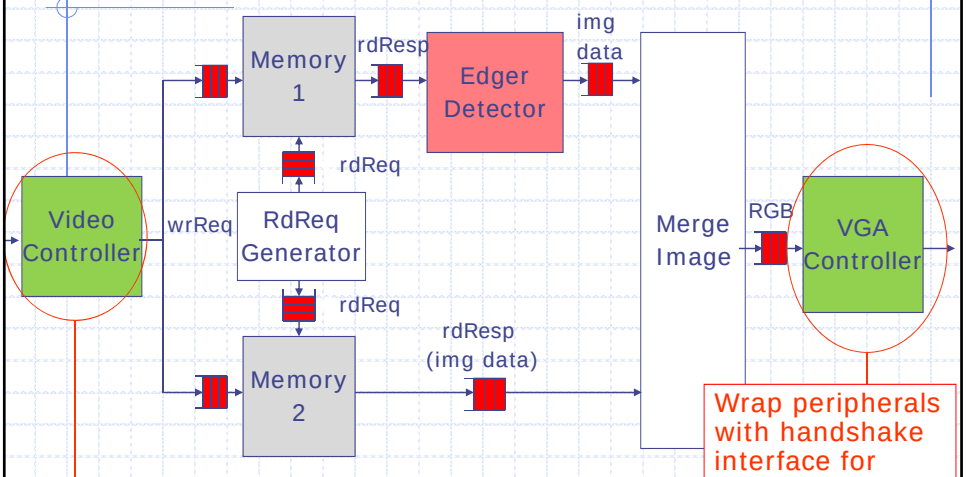
2

Problem in refinement



3

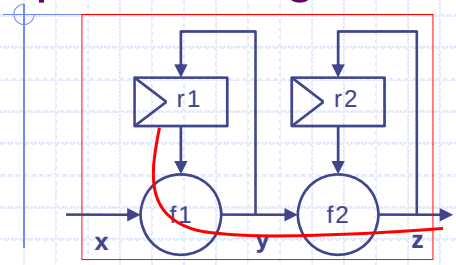
Latency-Insensitive design



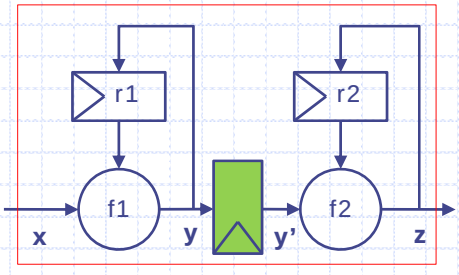
4

Refinements inside a block

Limitations of FSM-equivalence preserving refinements



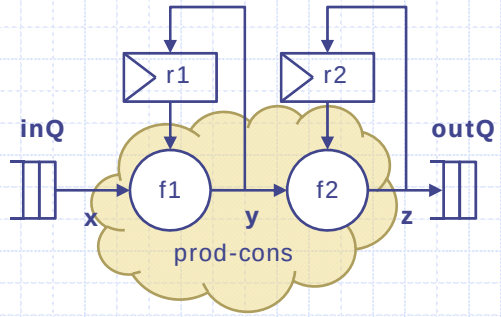
$y_i = f1(x_i; r1_i);$
$r1_0 = 0; r1_{i+1} = y_i;$
$z_i = f2(y_i; r2_i);$
$r2_0 = 0; r2_{i+1} = z_i;$



Produces the same z shifted by one clock

The two FSMs are not equal

A rule-based description

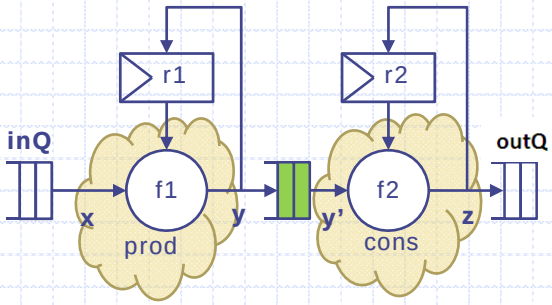


register r1 = 0, r2 = 0
inQ, outQ

```
rule producer-consumer when (!inQ.empty && !outQ.full):
    let x = inQ.first;
    let y = f1(x,r1);
    let z = f2(y,r2);
    r1 := y; r2 := z
    outQ.enq(z); inQ.deq;
```

7

Rules for the Refined System



register r1 = 0, r2 = 0
fifo q, inQ, outQ

Can be implemented by many different FSMs

```
rule produce when
        (!q.full && !inQ.empty):
    let x = inQ.first;
    let y = f1(x,r1);
    q.enq(y); inQ.deq;
    r1 := y
```

```
rule consume when
        (!q.empty && !outQ.full):
    let y = q.first;
    let z = f2(y,r2);
    outQ.enq(z); q.deq;
    r2 := z;
```

8

Schedules

- The semantics of rule-based systems only dictates that an execution must conform to some sequential execution of rules
- The compiler tries to execute in each cycle as many of the enabled rules as possible without violating the semantics
- Each schedule results in a different FSM

9

Back to our example

The one rule system has only one schedule but the refined system has many

```
rule produce when
        (!q.full && !inQ.empty):
    let x = inQ.first;
    let y = f1(x,r1);
    q.enq(y); inQ.deq;
    r1 := y
```

```
rule consume when
        (!q.empty && !outQ.full):
    let y = q.first;
    let z = f2(y,r2);
    outQ.enq(z); q.deq;
    r2 := z;
```

Some schedules

prod; cons; prod; cons; prod; cons;...

prod; prod; cons; prod; cons; prod; cons;...

prod; prod; cons; cons; prod; prod;...

prod; (prod|cons); (prod|cons); (prod|cons);...

10

In what sense are these two systems the same?

```
rule producer-consumer when (!inQ.empty && !outQ.full):
    let x = inQ.first;
    let y = f1(x,r1);
    let z = f2(y,r2);
    r1 := y; r2 := z
    outQ.enq(z); inQ.deq;
```

```
register r1 = 0, r2 = 0
inQ, outQ
```

Original System ↑↓ Refined System

same set of behaviors?

```
register r1 = 0, r2 = 0
fifo q, inQ, outQ
```

```
rule produce when
      (!q.full && !inQ.empty):
    let x = inQ.first;
    let y = f1(x,r1);
    q.enq(y); inQ.deq;
    r1 := y
```

```
rule consume when
      (!q.empty && !outQ.full):
    let y = q.first;
    let z = f2(y,r2);
    outQ.enq(z); q.deq;
    r2 := z;
```

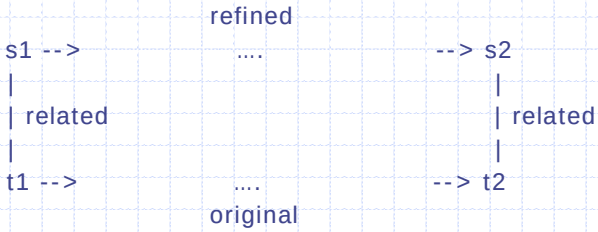
11

Same set of behaviors?

- A set of rules defines a transition system
- A behavior is the sequence of values assumed by the state variables (r1, r2, inQ, outQ, q) as a consequence of rule executions
- In order to relate two systems we have to define “related” states of the two systems
  - The state of the two system should be related when q is empty (The designer specifies this)
- Proof burden ?

12

Strong stuttering simulation



- Show that if the two systems start out in the same relatable state and the refined systems gets into a relatable state then there exists transitions in the original system that can get to an equivalent related state.

Can be done using a SMT solver

13

The tool

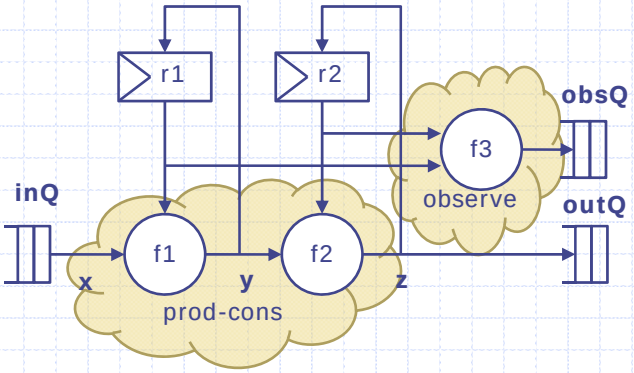
- The tool we have built shows that either the refinement is correct or produces a behavior that it is unable to reproduce in some bounded amount of time on the original system

Wonderful as a debugging aid because works in tens of seconds for many examples we have tried

Most complex example: refining a 4 stage processor pipeline into a 5 stage pipleline

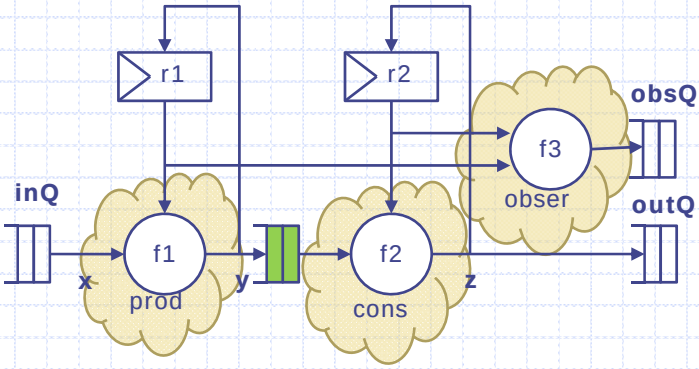
14

Non-determinism: Adding an Observer rule



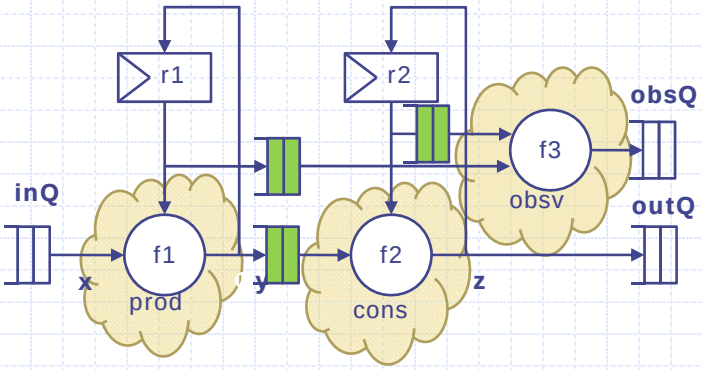
15

Wrong refinement



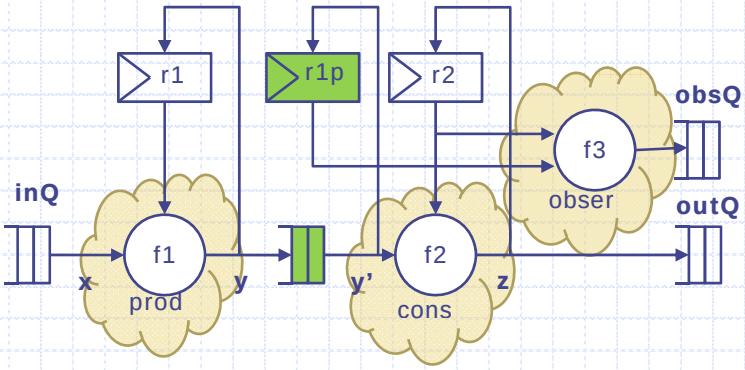
16

Correct refinement 1



17

Correct refinement 2



Thanks

18