A Lustre V6 tutorial

Verimag

December 4, 2008 - Synchron'08 - Aussois

Outline

- Lustre
- Lustre V6
- The Lustre V6 compiler

Outline

- Lustre
- Lustre V6

  P. Raymond & the Synchronous group et al.

- The Lustre V6 compiler

  P. Raymond, J. Ballet, E. Jahier

Lustre

a Data-flow Synchronous Language

- Generalised synchronous circuits: wires hold numerics
- Operators + wires structured into nodes
- Pre-defined operators
  - Boolean: and, not, ...
  - Arithmetic: +, -, ...
  - Temporal: pre, when, current

Lustre

Targetting reactive critical systems

- Time constraints

  ⟶ we want a predictable bound on execution time

- Memory constraints

  ⟶ we want a predictable bound on memory usage

  ⟶ (we want that bound to be as small as possible)

⇒ No loops, first-order

Lustre

a loop-free first-order language

But Can those limitations be overlooked ?

Lustre

a loop-free first-order language

But Can those limitations be overlooked ?

→ Yes: loops and genericity were introduced in V4

Lustre

Example of loops and genericity in V4

```
node add(const n:int; t1,t2 : int ^ n)
returns (res:int ^ n);
let
  res = t1 + t2; -- for i=0..n-1, res[i] = t1[i] + t2[i];
tel
```

Lustre

Example of loops and genericity in V4

```
node add(const n:int; t1,t2 : int ^ n)
returns (res:int ^ n);
let
  res = t1 + t2; -- for i=0..n-1, res[i] = t1[i] + t2[i];
tel
```

- **this is legal as long as n is a ground constant which value is known at compile time**

Lustre

Example of loops and genericity in V4

```
node add(const n:int; t1,t2 : int ^ n)
returns (res:int ^ n);
let
  res = t1 + t2; -- for i=0..n-1, res[i] = t1[i] + t2[i];
tel
```

- **this is legal as long as n is a ground constant which value is known at compile time → static genericity**

Lustre

Example of loops and genericity in V4

```
node add(const n:int; t1,t2 : int ^ n)
returns (res:int ^ n);
let
  res = t1 + t2; -- for i=0..n-1, res[i] = t1[i] + t2[i];
tel
```

- this is legal as long as n is a ground constant which value is known at compile time ⟶ static genericity

- Pushing that idea further ⇒ Lustre V6

Outline

- Lustre

- Lustre V6

  a statically generic (1.5-order) Lustre

- The Lustre V6 compiler

Lustre V6

What's new (compared to V4)

- Structure and enumerated types
- Package mechanism (Ada-like)

  ⟶ Name space

  ⟶ Encapsulation
- (Static) Genericity

  ⟶ Parametric packages

  ⟶ Parametric nodes (well-typed macros)

  ⟶ Static recursion

  ⟶ Array iterators (versus homomorphic extension – not new; different)

Lustre V6

Structures

```
type complex = struct {
   re : real = 0.;
   im : real = 0.
};
```

Lustre V6

Structures

```
type complex = struct {
   re : real = 0.;
   im : real = 0.
};

node plus (a, b : complex) returns (c : complex);
let
  c = complex { re = a.re+b.re ; im = a.im+b.im };
tel
```

Lustre V6

Enumerated type

```
type trival = enum { Pile, Face, Tranche };
```

Lustre V6

Enumerated clocks + merge (©$Pouzet$)

```
type trival = enum { Pile, Face, Tranche };
```

Lustre V6

Enumerated clocks + merge (©*Pouzet*)

```
type trival = enum { Pile, Face, Tranche };
node merge_node(clk: trival;
```

Lustre V6

Enumerated clocks + merge (©*Pouzet*)

```
type trival = enum { Pile, Face, Tranche };
node merge_node(clk: trival;
   i1 when Pile(clk); i2 when Face(clk);
   i3 when Tranche(clk))
returns (y: int);
```

Lustre V6

Enumerated clocks + merge (© *Pouzet*)

```
type trival = enum { Pile, Face, Tranche };
node merge_node(clk: trival;
   i1 when Pile(clk); i2 when Face(clk);
   i3 when Tranche(clk))
returns (y: int);
let
    y = merge clk
        (Pile:    i1)
        (Face:    i2)
        (Tranche: i3);
tel
```

Lustre V6

Packages

```
package complex
provides
  type t; -- Encapsulation
  const i:t;
  node re(c: t) returns (r:real);
```

Lustre V6

Packages

```
package complex
provides
  type t; -- Encapsulation
  const i:t;
  node re(c: t) returns (r:real);
body
  type t = struct { re :  real ; im :  real };
  const i:t = t { re = 0. ; im = 1. };
  node re(c: t) returns (re:real);
  let re = c.re; tel;
end
```

Lustre V6

Generic packages

```
model modSimple
  needs type t;
  provides
        node fby1(init, fb: t) returns (next: t);
```

Lustre V6

Generic packages

```
model modSimple
  needs type t;
  provides
        node fby1(init, fb: t) returns (next: t);
body
  node fby1(init, fb: t) returns (next: t);
  let next = init -> pre fb; tel
end
```

Lustre V6

Generic packages

```
model modSimple
  needs type t;
  provides
        node fby1(init, fb: t) returns (next: t);
body
  node fby1(init, fb: t) returns (next: t);
  let next = init -> pre fb; tel
end
package pint is modSimple(t=int);
```

Lustre V6

Generic nodes

```
node mk_tab<<type t; const init: t; const size: int>>
       (a:t) returns (res: t^size);
let
  res =  init ^ size;
tel
```

Lustre V6

Generic nodes

```
node mk_tab<<type t; const init: t; const size: int>>
       (a:t) returns (res: t^size);
let
  res =  init ^ size;
tel
node tab_int3 = mk_tab<<int, 0, 3>>;
```

Lustre V6

Generic nodes

```
node mk_tab<<type t; const init: t; const size: int>>
       (a:t) returns (res: t^size);
let
  res =  init ^ size;
tel
node tab_int3 = mk_tab<<int, 0, 3>>;
node tab_bool4 = mk_tab<<bool, true, 4>>;
```

Lustre V6

Generic nodes

```
node toto_n<<
   node f(a, b: int) returns (x: int);
   const n : int
  >>(a: int) returns (x: int^n);
var v : int;
let
  v = f(a, 1);
  x =  v ^ n;
tel
```

Lustre V6

Generic nodes

```
node toto_n<<
   node f(a, b: int) returns (x: int);
   const n : int
  >>(a: int) returns (x: int^n);
var v : int;
let
  v = f(a, 1);
  x =  v ^ n;
tel
node toto_3 = toto_n<<Lustre::iplus, 3>>;
```

Lustre V6

Static recursion

```
node consensus<<const n : int>>(T: bool^n)
returns (a: bool);
let
  a =  with (n = 1)  then T[0]
      else T[0] and consensus << n-1 >> (T[1 .. n-1]);
tel

node main = consensus<<8>>;
```

Lustre V6

Are parametric nodes necessary?

- Indeed, parametric nodes could be emulated with the package mechanism

⟶ but we keep them to keep the syntax ligth

⟶ we didn't really want to have recursive packages

Lustre V6

Arrays

- **As in Lustre V4**

  $\longrightarrow$ **The array size is static (`var mat23: int ^ 2 ^ 3;`)**

  $\longrightarrow$ **Array slices (`T1[3..5] = T2[0..2];`)**

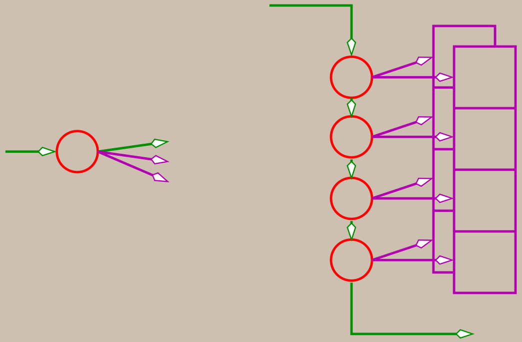
- **But no more homomorphic extension**

  where $t1 + t2$ means $\forall i \in \{0, .., size - 1\},\ t1[i] + t2[i]$

  $\Rightarrow$ **operate on arrays via iterators**

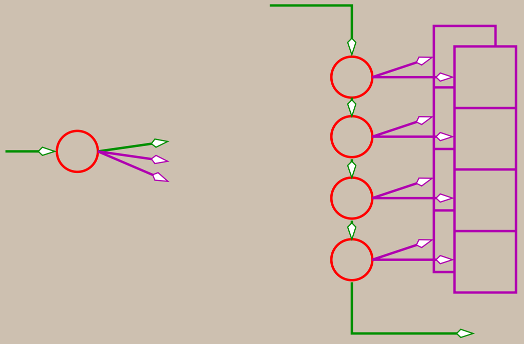
Lustre V6

The fill iterator



Lustre V6

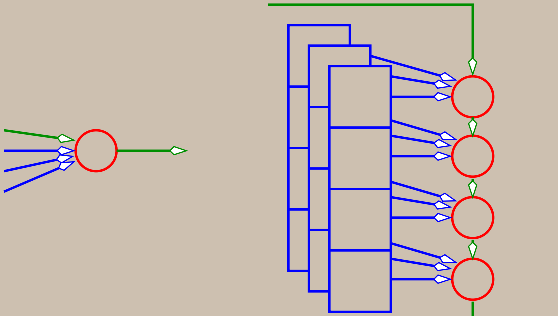
The fill iterator



```
node incr (acc :  int) returns (acc', res :  int);
fill<<incr; 4>>(0) ⇝(4, [0,1,2,3])
```

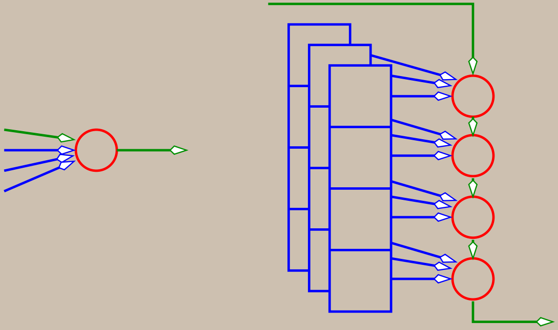
Lustre V6

The red iterator



Lustre V6

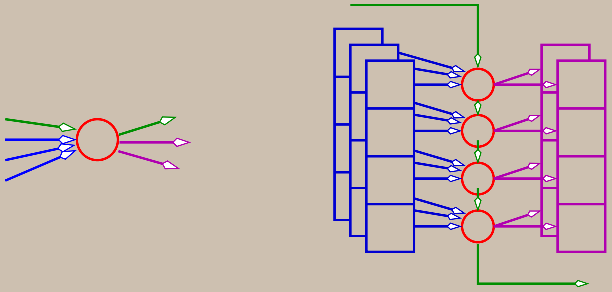
The red iterator



```
red<<+; 3>>(0, [1,2,3]) ⇝6
```

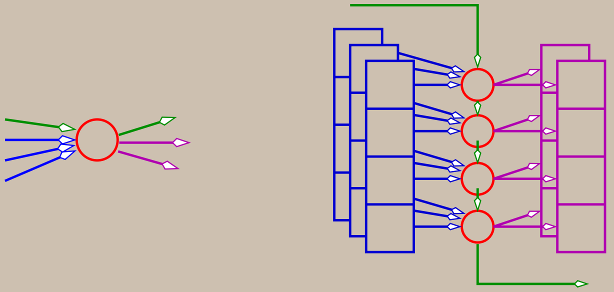
Lustre V6

fill+red=~~mapred~~, ~~fillred~~, fold



Lustre V6

fill+red=~~mapred~~, ~~fillred~~, fold

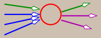


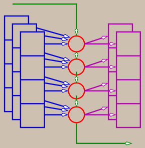
```
fill<<incr; 4>>(0) ≡ fold<<incr; 4>>(0)
red<<+; 3>>(0, [1,2,3]) ≡ fold<<+; 3>>(0, [1,2,3])
```

Lustre V6

The fold iterator

```
node cumul(acc_in,x:int) returns (acc_out,y:int)
let
   y = acc_in+x;
   acc_out = y;
tel
```



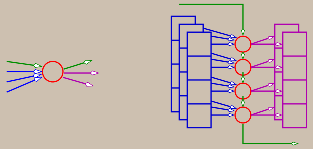


```
  fold<<cumul>>(0, [1,2,3])
```

Lustre V6

The fold iterator

```
node cumul(acc_in,x:int) returns (acc_out,y:int)
let
   y = acc_in+x;
   acc_out = y;
tel
```

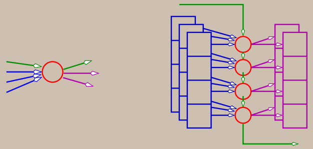


```
  fold<<cumul>>(0, [1,2,3])⇝(6, [1,3,6])
```

Lustre V6

The fold iterator

```
node cumul(acc_in,x:int) returns (acc_out,y:int)
let
   y = acc_in+x;
   acc_out = y;
tel
```

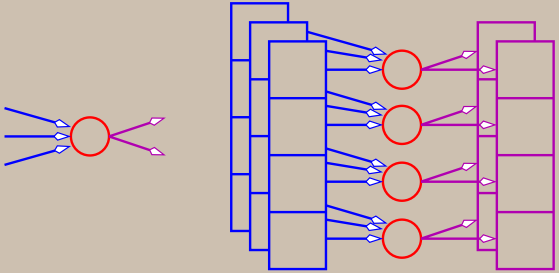


```
  fold<<cumul>>(0, [1,2,3])⇝(6, [1,3,6])

  fold<<fold<<fold<<full_adder; n>>; m>>; p>>
                          (false, x, y) ⇝(r,''x+y'')
```

Lustre V6

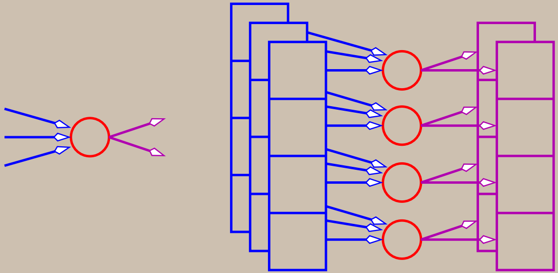
The map iterator



```
map <<+; 3>>([1,0,2],[3,6,-1]) ⤳[4,6,1]
```

Lustre V6

The map iterator



`map <<+; 3>>([1,0,2],[3,6,-1])` ⇝ [4,6,1]

Lustre V6

About Lustre V6 array iterators

More general that usual iterators:

their are of variable arity

Outline

- Lustre
- Lustre V6
- The Lustre V6 compiler
  - The front-end
  - The back-end (J. Ballet)
  - The back-back-end (J. Ballet)

The Lustre V6 compiler

The Front-end: LUS2LIC

- Perform usual checks

  ⟶ Syntax, Types, Clocks

  ⟶ Unique definition of outputs

  ⟶ Combinational cycles detection

The Lustre V6 compiler

The Front-end: LUS2LIC

- Perform usual checks
  - ⟶ Syntax, Types, Clocks
  - ⟶ Unique definition of outputs
  - ⟶ Combinational cycles detection
- Perform some static evaluation
  - ⟶ arrays size
  - ⟶ parametric packages and nodes
  - ⟶ recursive nodes

The Lustre V6 compiler

The Front-end: LUS2LIC

- Perform usual checks

  ⟶ Syntax, Types, Clocks

  ⟶ Unique definition of outputs

  ⟶ Combinational cycles detection

- Perform some static evaluation

  ⟶ arrays size

  ⟶ parametric packages and nodes

  ⟶ recursive nodes

- Generate intermediate code: LIC (Lustre internal code)

Lustre Internal Code (LIC)

was: expanded code (ec)

- LIC $\equiv$ core Lustre
  - No more packages
  - Parametric constructs are instanciated

    $\longrightarrow$ constants

    $\longrightarrow$ types

    $\longrightarrow$ nodes

Lustre Internal Code (LIC)

was: expanded code (ec)

- LIC $\equiv$ core Lustre
  - No more packages
  - Parametric constructs are instanciated

    ⟶ constants

    ⟶ types

    ⟶ nodes

Lustre Internal Code (LIC)

was: expanded code (ec) cont.

- **LIC versus ec**

  ⟶ **Nodes are not (necessarily) expanded**

  ⟶ **Arrays are not (necessarily) expanded**

Lustre Internal Code (LIC)

was: expanded code (ec) cont.

- LIC versus ec

  ⟶ Nodes are not (necessarily) expanded

  ⟶ Arrays are not (necessarily) expanded

- LIC versus Lustre v4

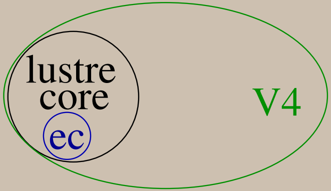
  ⟶ Structures and enums

  ⟶ array iterators

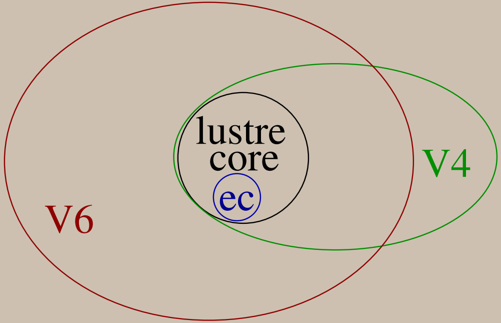
Lustre potatoes

lustre
core

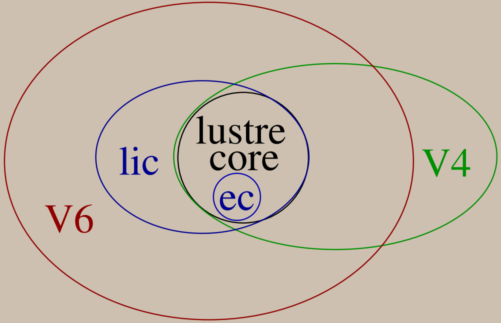
Lustre potatoes



Lustre potatoes

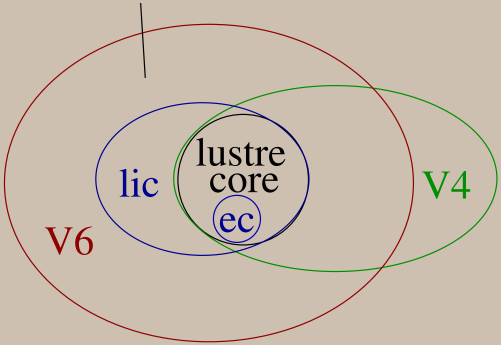


Lustre potatoes

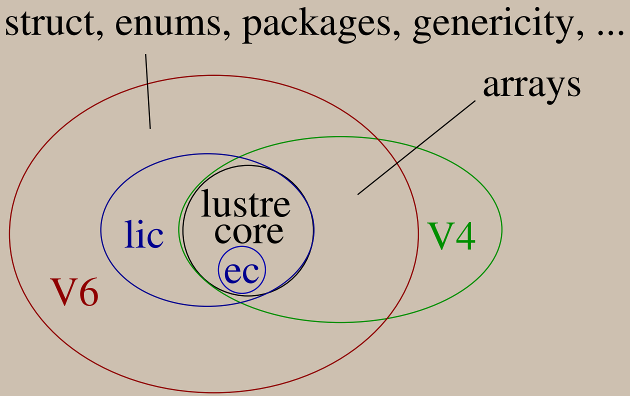


Lustre potatoes

struct, enums, packages, genericity, ...



Lustre potatoes



struct, enums, packages, genericity, ...

arrays

lustre core

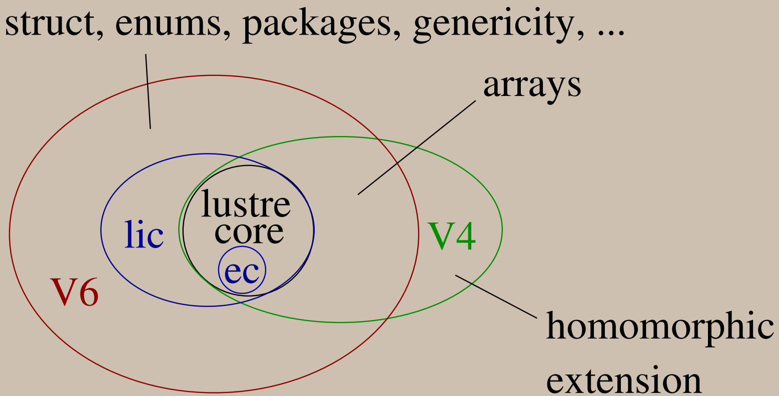
lic

ec

V4

V6

Lustre potatoes



struct, enums, packages, genericity, ...

arrays

lic

lustre core

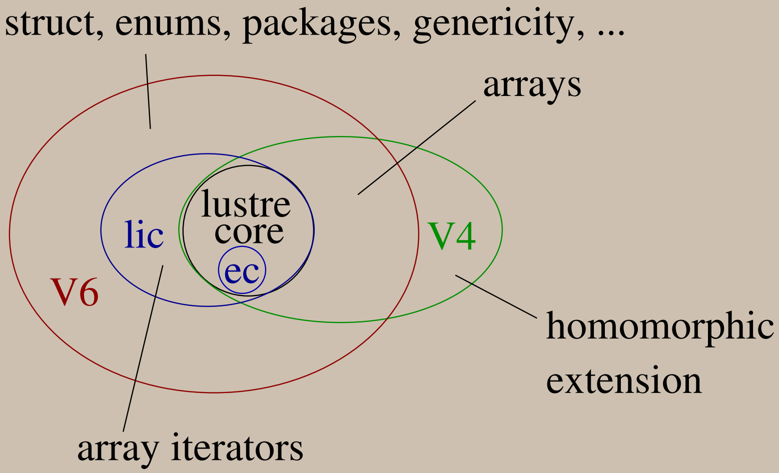
ec

V4

V6

homomorphic extension

Lustre potatoes



struct, enums, packages, genericity, ...

arrays

lustre core

lic

ec

V4

V6

homomorphic extension

array iterators

The Lustre V6 compiler

The back-end

The role of the backend is to generate sequential code

We defined (yet) another intermediary format to represent sequential code: SOC (Synchronous Object Code)

The idea is that translating this format into any sequential language is easy, and done at the very end

The back-end

maps each node to a Synchronous Object Component (SOC)

- A SOC is made of:
  - a set of memories
  - a set of methods: typically, an init and a step method

The back-end

maps each node to a Synchronous Object Component (SOC)

- **A SOC is made of:**
  - **a set of memories**
  - **a set of methods: typically, an init and a step method**
- **each method is made of a sequence of guarded atomic operations**

The back-end

maps each node to a Synchronous Object Component (SOC)

- A SOC is made of:
  - a set of memories
  - a set of methods: typically, an init and a step method
- each method is made of a sequence of guarded atomic operations
- atomic operation (named actions) can be
  - another SOC method call
  - an assignment (a wire)

The back-end

From node to SOC

For each node, we:

- Identify memories
- Explicitely separate the control (clocks) from the computations

  ⟶ set of guarded equations
- Split equations into more finer-grained steps: actions

  ⟶ a set of guarded actions (a wire or a call)
- Find a correct ordering for actions (sheduling)

  ⟶ a sequence of guarded actions

The back-back-end

From SOC to C

- pretty-print the SOC into, let's say, C

- provide a C implementation of every predefined (non-temporal) operators

Lustre V6 compiler

An alpha release is available

`http://www-verimag.imag.fr/~synchron/lustre-v6/`

- **The front-end `lus2lic` seems ok**

Lustre V6 compiler

An alpha release is available

`http://www-verimag.imag.fr/~synchron/lustre-v6/`

- **The front-end `lus2lic` seems ok**
- `lus2lic --lustre-v4`**: added last friday; seems to work**

Lustre V6 compiler

An alpha release is available

`http://www-verimag.imag.fr/∼synchron/lustre-v6/`

- **The front-end** `lus2lic` **seems ok**
- `lus2lic --lustre-v4`**: added last friday; seems to work**
- **The back-back: generates C code... But its not finished.**

Thanks for your attention