



Statically-allocated languages for hardware stream processing

Simon Frankau Alan Mycroft Simon Moore

Motivation



- ◆ We want to investigate *high-level* hardware description languages
- ◆ Abstract away the details of timing, signalling and wires
- ◆ *Software-like* descriptions:
 - ◇ Minimise explicit parallelism
 - ◇ Let synthesis tools deal with pipelining, scheduling etc.
- ◆ Uses:
 - ◇ Rapid development
 - ◇ Development by non-specialists
 - ◇ Reconfigurable hardware

Existing languages



- ◆ **SAFL** is a *statically-allocated* pure functional language

- ◇ In the vein of functional languages used for software
- ◇ Poor support for I/O—a call/return mechanism with no state held
- ◇ Example:

```
fun mult(x, y, acc) =  
  if (x=0 or y=0) then acc  
  else mult(x<<1, y>>1, if y[0:0] then acc+x else acc)
```

- ◆ **LUSTRE** is a synchronous dataflow language

- ◇ Reactive systems processing streams of data items—useful for I/O
- ◇ Streams have clocks—compile-time consistency check
- ◇ Very different approach to conventional programming
- ◇ Example:

```
node WatchDog (set, reset, deadline: bool) returns (alarm: bool);  
var is_set: bool;  
let  
  alarm = deadline and is_set;  
  is_set = set -> if set then true  
               else if reset then false else pre(is_set);  
tel.
```



- ◆ SASL is a statically-allocated eager functional language (similar to SAFL)
- ◆ Lazily-evaluated linear lists, using a CONS operator (:::)
- ◇ CONS becomes a write, CONS-matching becomes a read
- ◆ A mixture of functional-style calling conventions and multiple I/O streams
- ◆ *Implicit timing*—no explicit clocks, demand-driven generation of stream data
- ◇ Simplifies e.g. stream merging
- ◆ Example:

```
fun add_blocks (head ::: rest, acc) =  
  case head of Item i => add_blocks (rest, acc+i)  
            | Marker => acc ::: add_blocks(rest, 0)
```

Language constraints



Need static allocation restrictions:

1. Type system restricts language (e.g. removes streams of streams)
2. Use SAFL's constraints (e.g. tail recursion only)
3. Add *linearity* and *stability* constraints

Otherwise, problems occur:

```
fun desynchronise (stream) =  
  zip(stream, filter(stream))
```

```
fun build_up (stream, item) =  
  build_up(item::stream, item)
```

Further work



- ◆ Synthesis techniques
- ◆ Implementation
- ◆ Further language extensions (e.g. nondeterminism)