BCTCS12

Rainbow

A Hardware Description Framework for Micropipeline Designs

Donal Fellows

Department of Computer Science, University of Manchester

with

H. Barringer, G.D.Gough, B.W. Marsden, W. Visser, A.J. Williams

http://www.cs.man.ac.uk/fmethods/projects/ahv-project.html

April, 1996

Contents

Asynchronous Design

Some engineers (esp. S. Furber at Manchester with AMULET) turning to asynchronous CPU designs:

- Increase speed
- Reduce power consumption
- Reduce design problems (easier to compose)

There are problems with using the current toolset for this:

- Current tools are for synchronous design
- Difficult to analyse in native language

Sutherland's micropipelines gives one popular methodology, and this is followed in the design of AMULET.

The Components of Rainbow

- Red — specifications
- Yellow — behavioural descriptions, algorithmic style
- Green — strcutural descriptions, dataflow style
  - ◇ Includes a graphical layout editor
- Blue — signal level

- Action algebra — language of values and functions upon them

Features of Green

- Dataflow language model
- Bundled-data micropipeline communication model (*request*, *data* and *acknowledge*)
- Communication along point-to-point synchronised streams
- Hierarchical
- Denotational semantics by translation into APA[a]
- Polymorphic type system

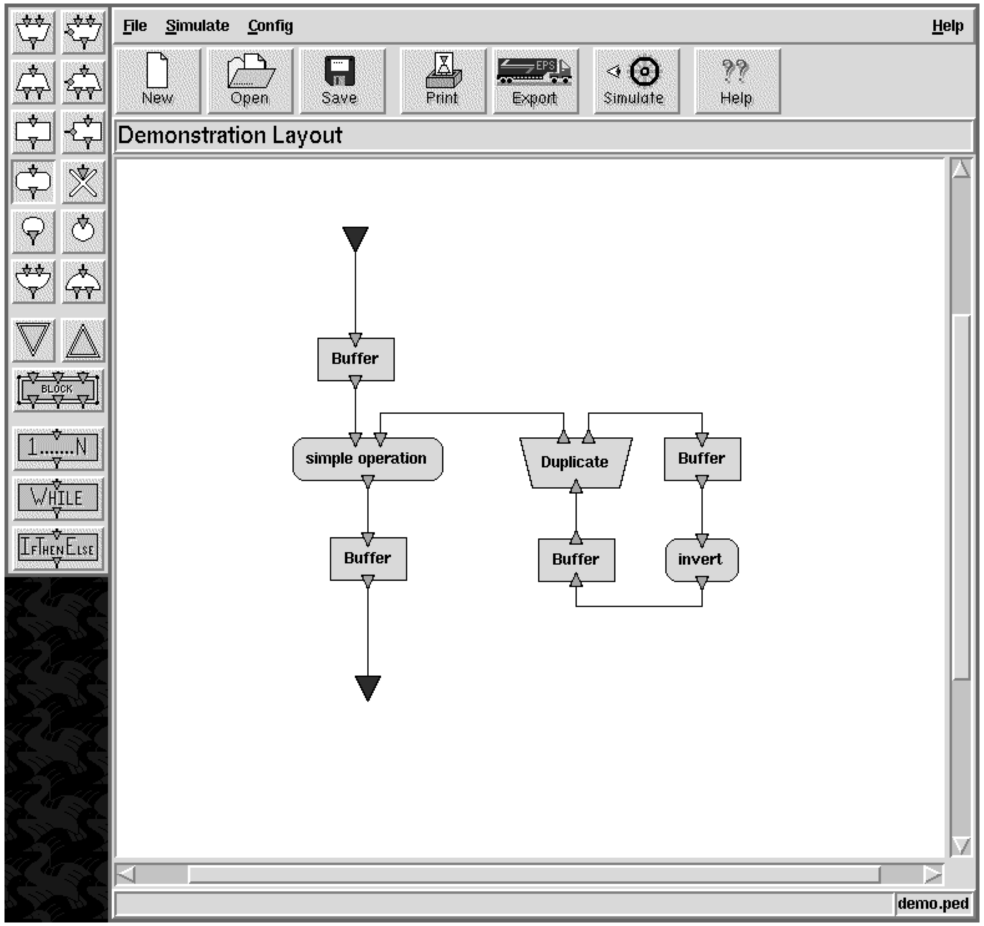
[a] Asynchronous Process Algebra — under development at Manchester

- General tables[a]
- Configuration parameters[b]
- Non-determinism easy to locate (only in arbitrators)
- State easy to locate (only in buffers)
- Both textual and graphical descriptions available.
- Fully integrable with other *Rainbow* components.

[a] for specifying stateless, deterministic stream functions

[b] to allow the creation of configurable hardware and generic libraries

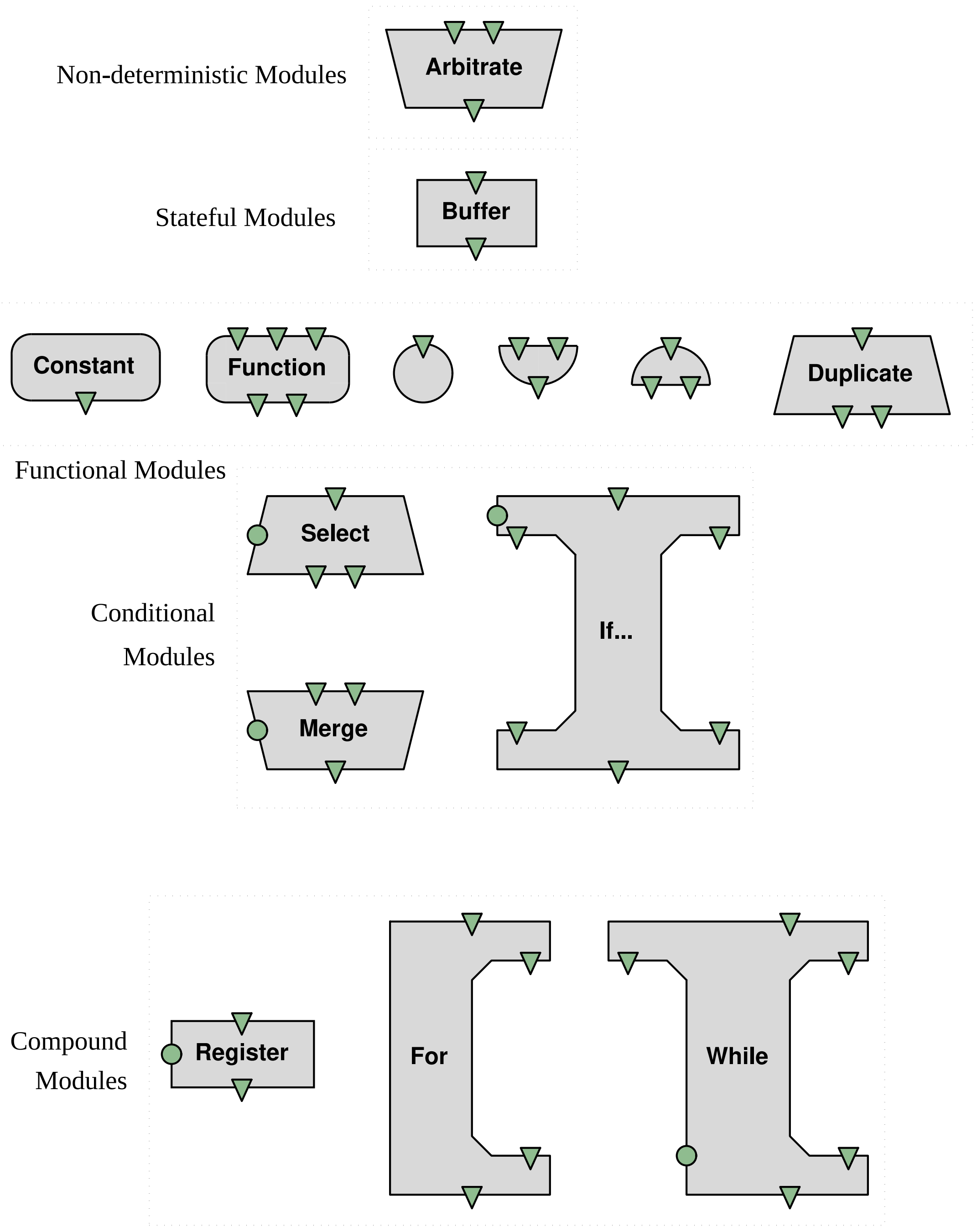
Green Graphical Editor



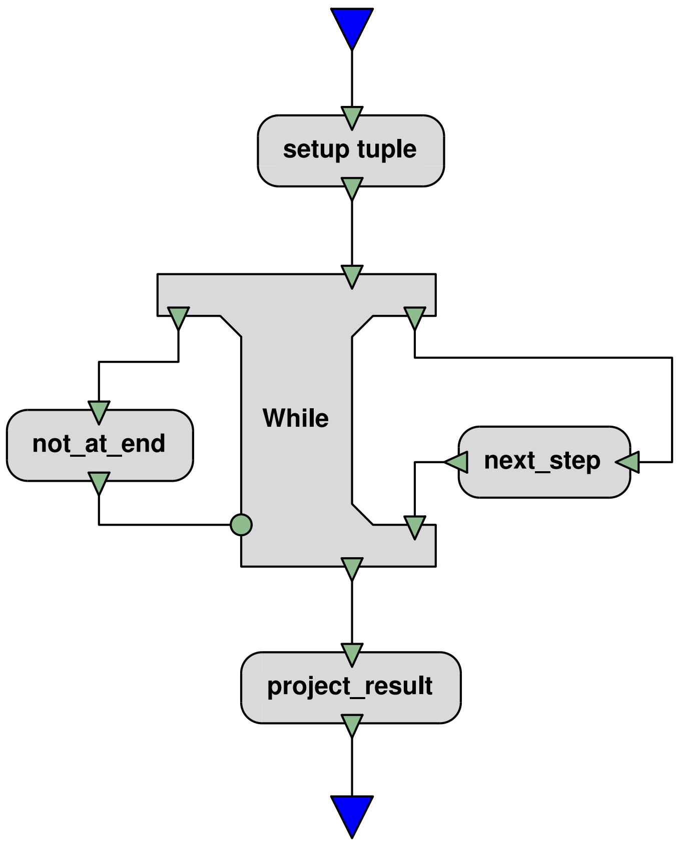
Features of the graphical editor include:

- Export of the current layout as PostScript
- Animation of schematic
- User-driven extensibility (written in Tcl/Tk)

Basic Green Modules



Examples of Green: Factorial



$$\text{setup\_tuple}(x) = (x, 2, 1)$$

$$\text{not\_at\_end}(x, i, a) = x \geq i$$

$$\text{next\_step}(x, i, a) = (x, i + 1, a \times i)$$

$$\text{project\_result}(x, i, a) = a$$

Examples of *Green*: Factorial (Continued)

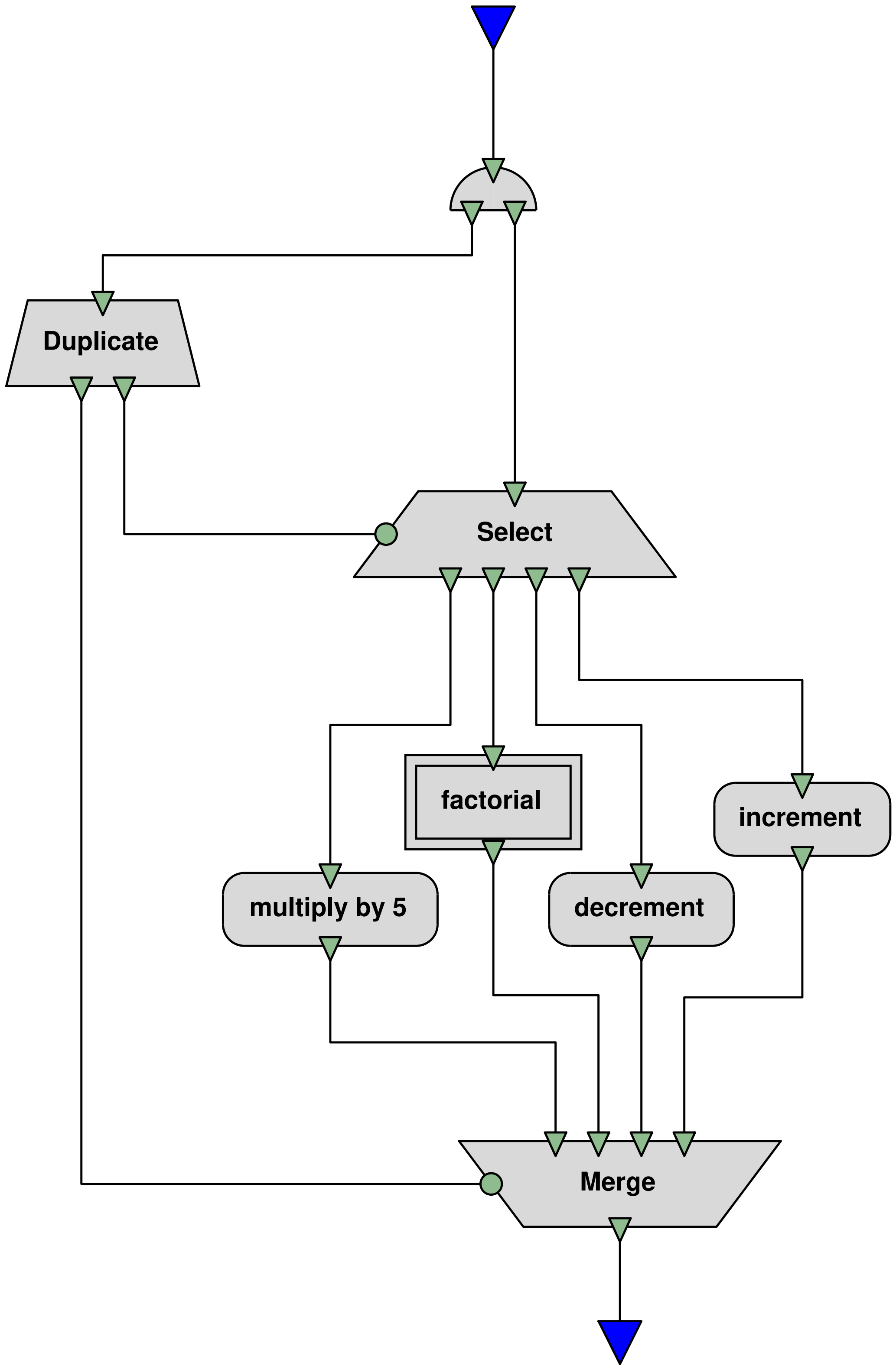
```
proc factorial(input in:'a; output out) =
    type factup =
        record {N:'a , i:'a , a:'a}.

    function not_at_end(value t) =
        t[N]>=t[i].
    function next_step(value t) =
        factup(t[N] , t[i]+1 , t[a]*t[i]).

    def finalval = (in,2,1) >>
        while not_at_end {
            next_step
        }.

    let out = finalval[a].
endproc factorial.
```

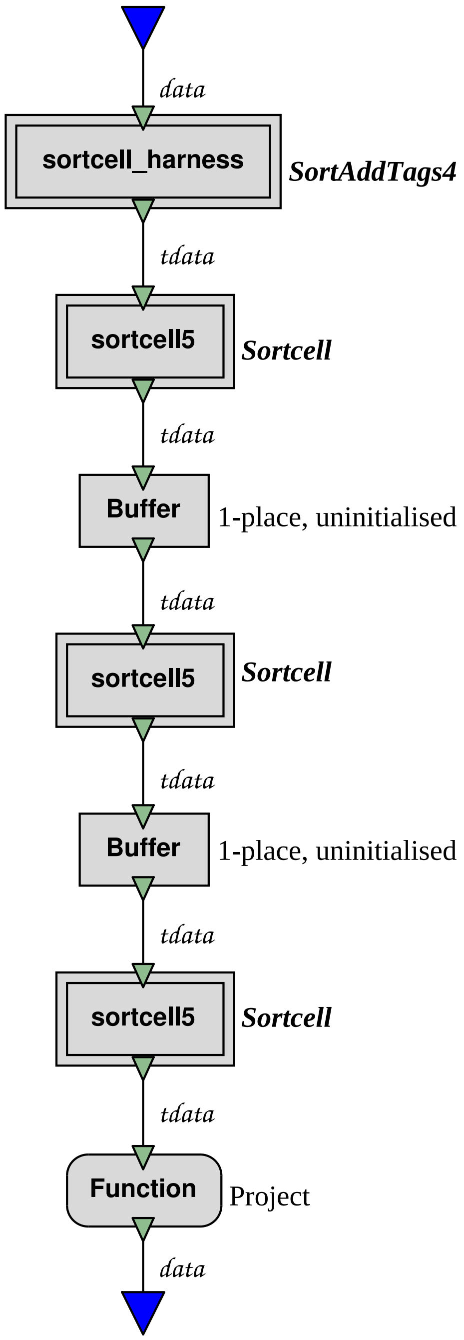
Examples of Green: ALU



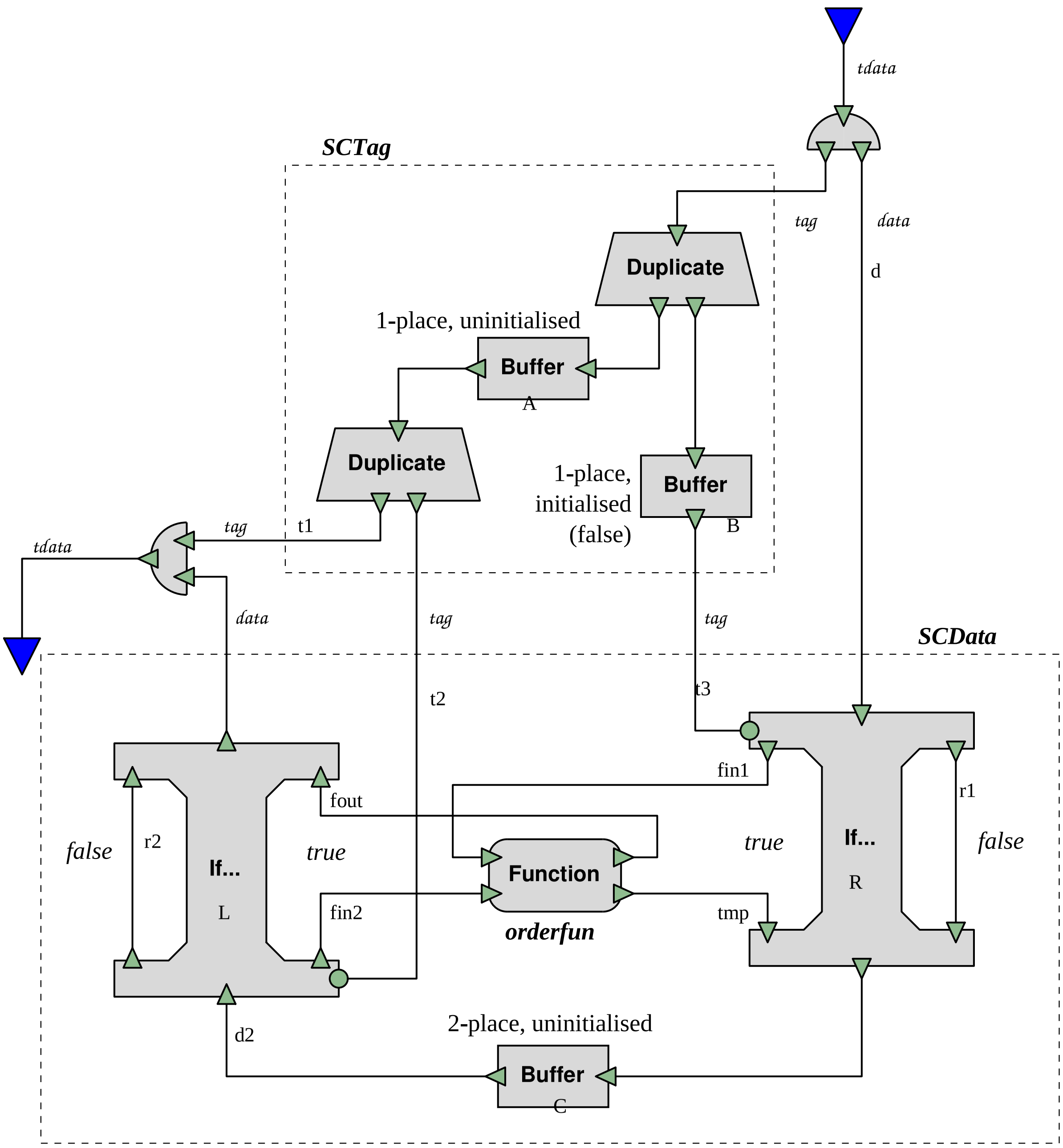
The layout builds a stream function that applies the function alu to every value input to create its output.

$$alu(c,x) = \begin{cases} x \times 5 & ,\text{if } c = 0 \\ x! & ,\text{if } c = 1 \\ x - 1 & ,\text{if } c = 2 \\ x + 1 & ,\text{if } c = 3 \end{cases}$$

Examples of Green: Block Sorter

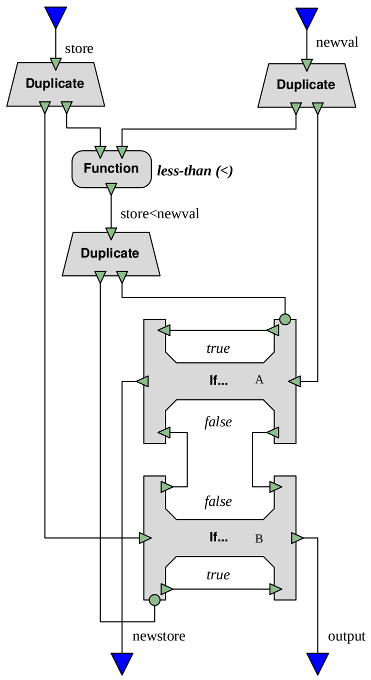


Sort Cell

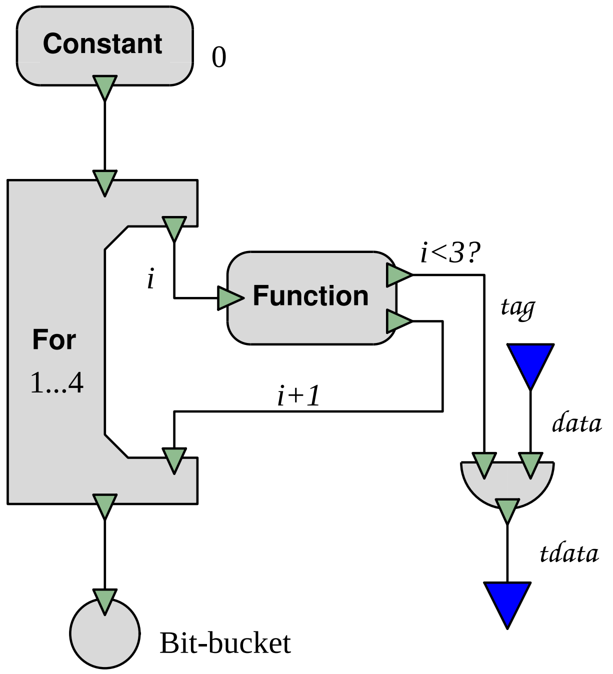


tdata
SCTag
tag
data
Duplicate
d
1-place, uninitialised
Buffer
A
Duplicate
1-place,
initialised
(false)
Buffer
B
tag
t1
tdata
data
tag
tag
SCData
t2
t3
fin1
fout
r1
false
r2
true
true
If...
false
If...
Function
R
L
fin2
tmp
orderfun
2-place, uninitialised
d2
Buffer
C

Sort Swap Function



Sorter Header Cell



Block Sort — Textual

```
proc Sort(function orderfun; input instream; output outstream) =
    //-----------------------------------------------------
    // Start of declarations

    proc Sortcell(input dstream; output ostr) =

        proc SCTag(input in; output o1, o2, o3) =
            let o1 = in >> buffer(1,-).
            let o2 = in >> buffer(1,false).
            let o3 = o1.
        endproc SCTag.

        proc SCData(input din, t1, t2; output dout) =
            label fin1,fin2,fout.

            let dout = din >>
                if (t1) then fin1:{
                    label tmp.
                    let (tmp,fout) = orderfun(fin1,fin2).

                    tmp
                } else {} >>
                buffer(2,-) >>
                if (t2) then fin2:{
                    fout
                } else {}.
        endproc SCData.

        def mid1,mid2,mid3 = SCTag(dstream#2).
        def dataout = (dstream[1] || mid2 || mid3) >> SCData.
        let ostr = (dataout,mid1).

    endproc Sortcell.
```

Block Sort — Textual (Continued)

```
    proc SortAddTags4(input in; output out) =
        function stepfun4(value x) = ( x=3 , x+1 ).
        label tag,fvar.

        let out = 0 >>
            for (4) fvar:{
                def tn = stepfun4(fvar).
                let tag = tn[0].

                tn[1]
            } >> sink >> (in,tag)
    endproc SortAddTags4.

    def final = {
        SortAddTags4(instream) >>
        Sortcell >>
        buffer(1,-) >>
        Sortcell >>
        buffer(1,-) >>
        Sortcell
    }.
    let outstream = final[0].
endproc Sort.
```

Work in Progress

- *Yellow*
- Completion of *Green* Semantics
- Simulators
- Integration of *Rainbow* components and tools
- Complexity management in the graphical environment

URL

`http://www.cs.man.ac.uk/fmethods/projects/ahv-project.html`