Automatic Compilation of Data-Driven Circuits

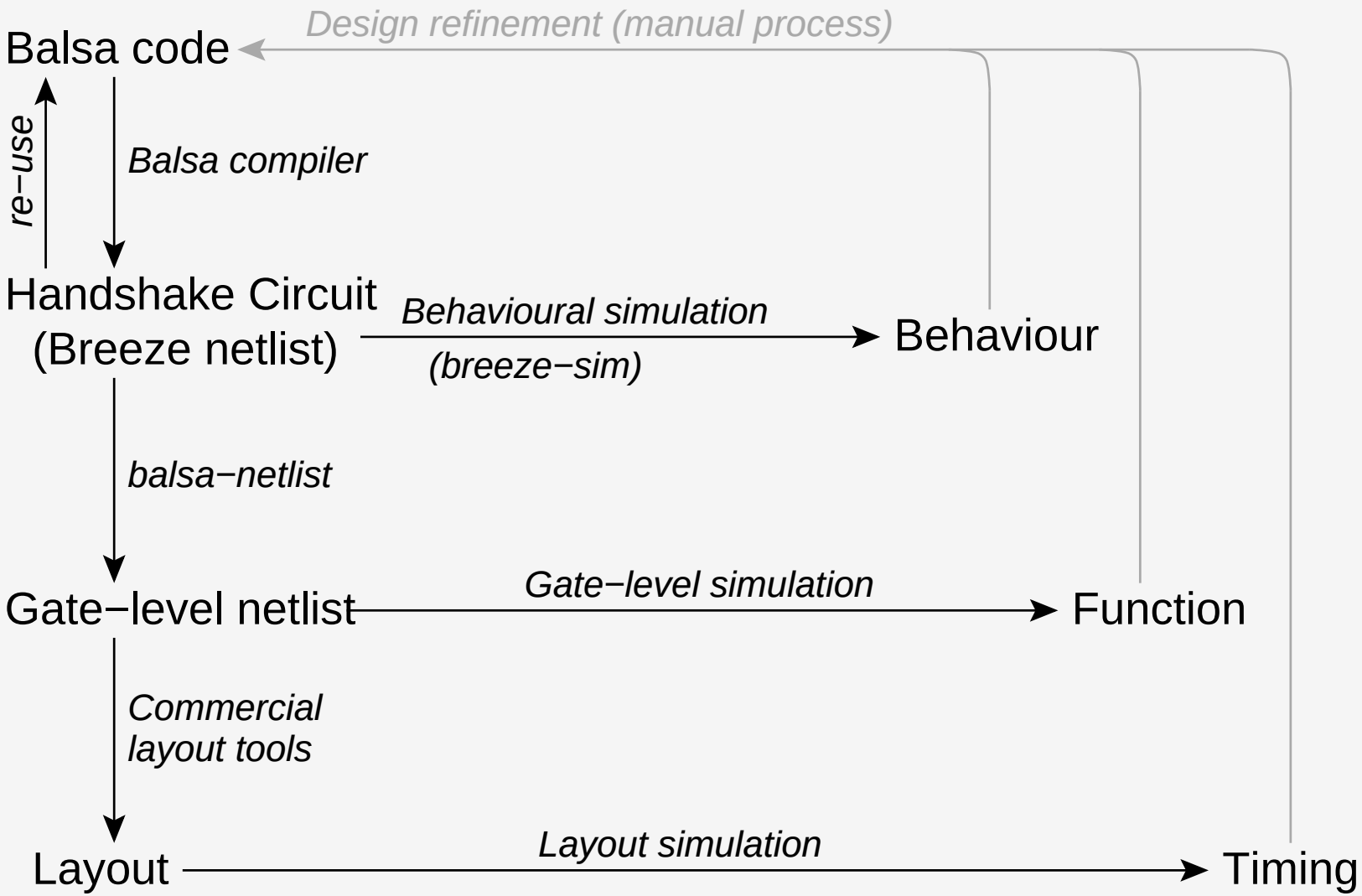
Sam Taylor, Doug Edwards, Luis Plana
University of Manchester
smtaylor|doug|lplana@cs.manchester.ac.uk

EPSRC Engineering and Physical Sciences Research Council

Summary

- Handshake Circuit paradigm is nice
- Control-driven style is flexible but slow
- Data-driven approaches provide better performance
- Combine data-driven approach with handshake circuit paradigm
- An alternative option for designers?

Balsa Design Flow

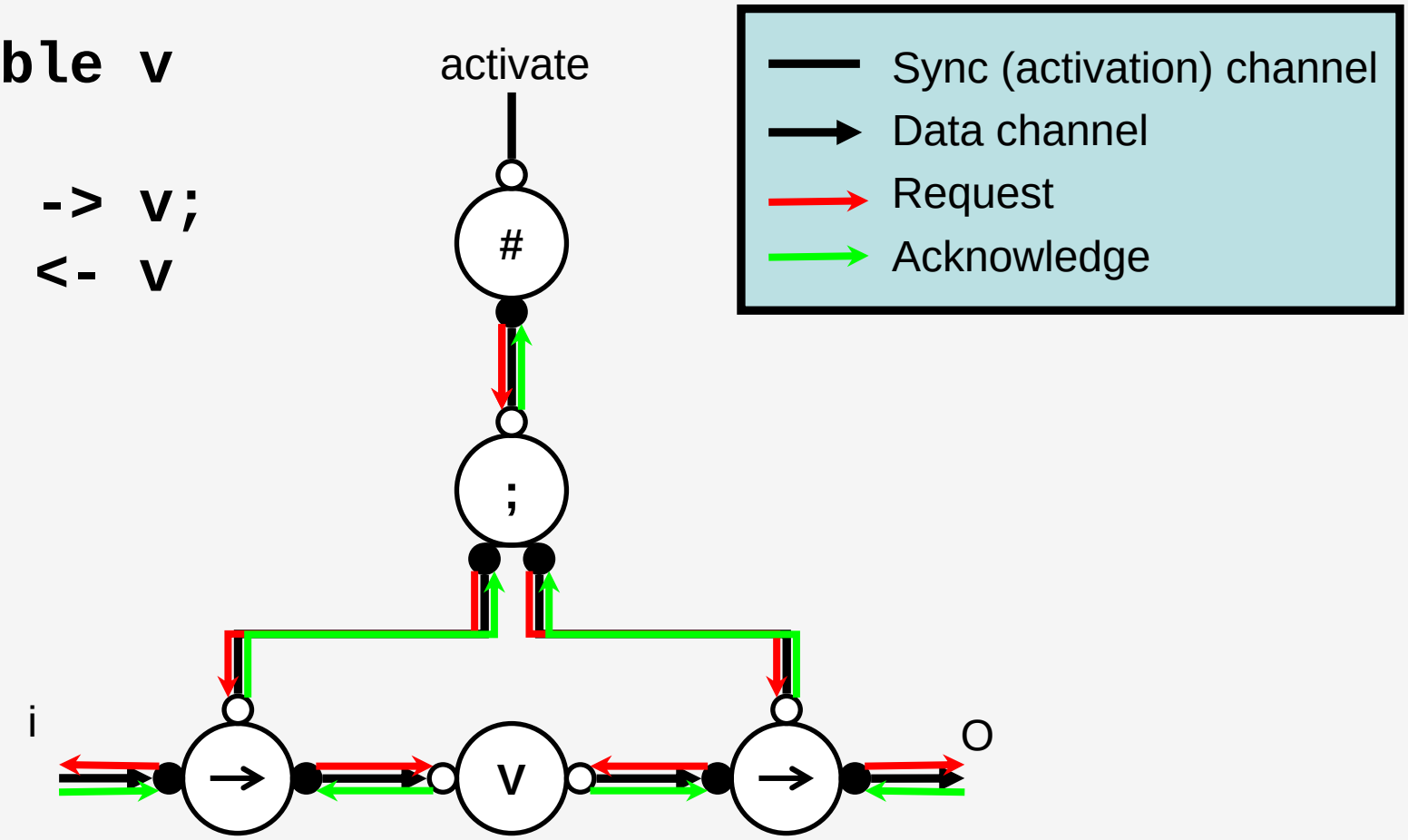


Handshake Circuits

- Intermediate representation independent of implementation styles
- Networks of small components communicating by handshakes
- Each component (relatively) straightforward to implement in isolation
- Successful method of implementing large circuits
- Syntax-directed translation

Balsa one-place buffer

```
variable v
loop
     i -> v;
     o <- v
end
```



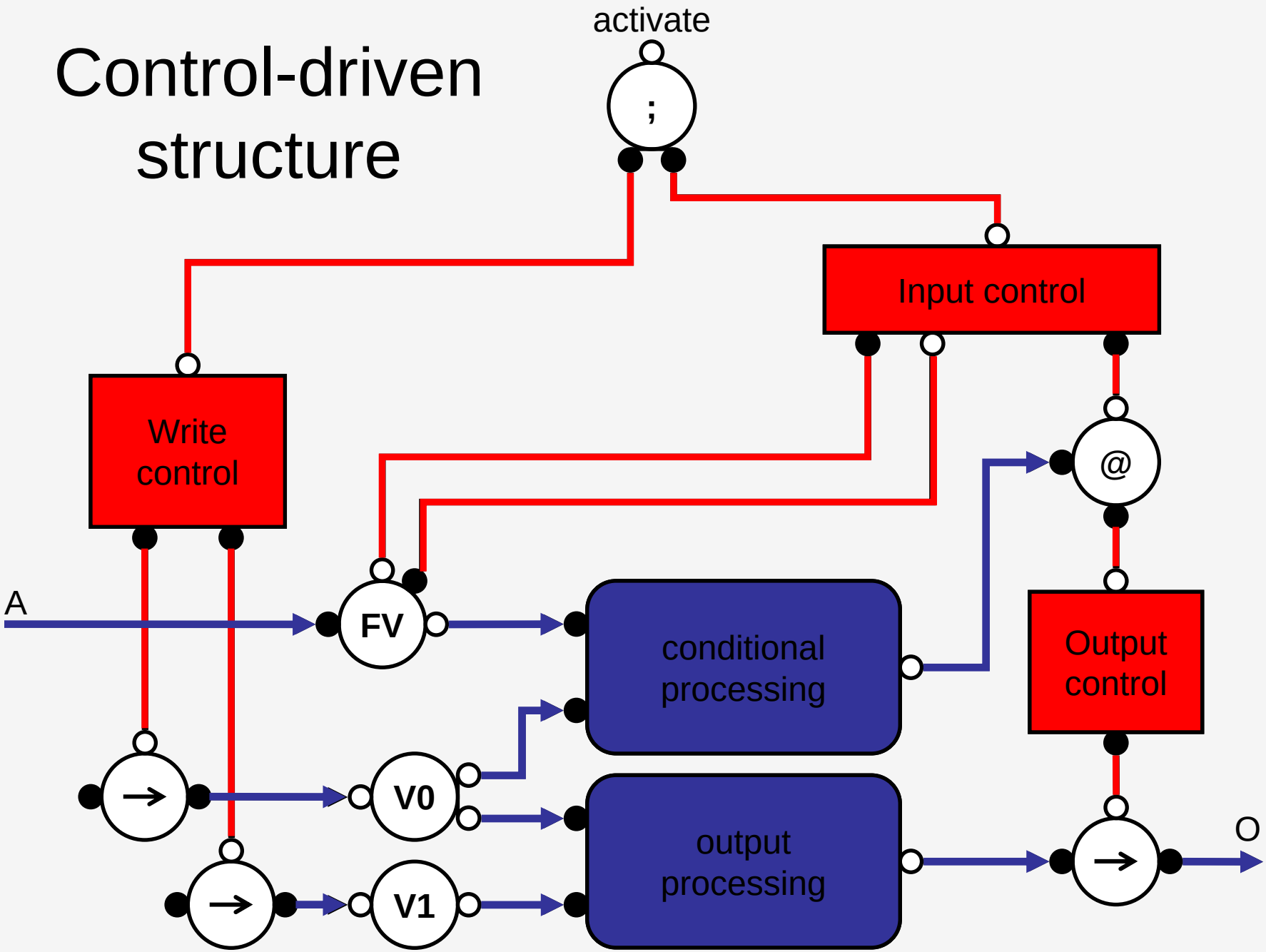
Advantages of control-driven structure

- Passive-ported variable is very flexible. Read and write in any order like a sequential programming language
- Familiar control structures - loops etc.
- Low power – nothing gets done that does not need doing.

Why does the structure of Balsa circuits make them slow?

- Control-driven compilation
- Monolithic control
- Lots of sequencers
- Frequent synchronisation between control and data
- Control Overhead. Data is always waiting for control.
- Data-driven style attempts to avoid all of these problems

Control-driven structure

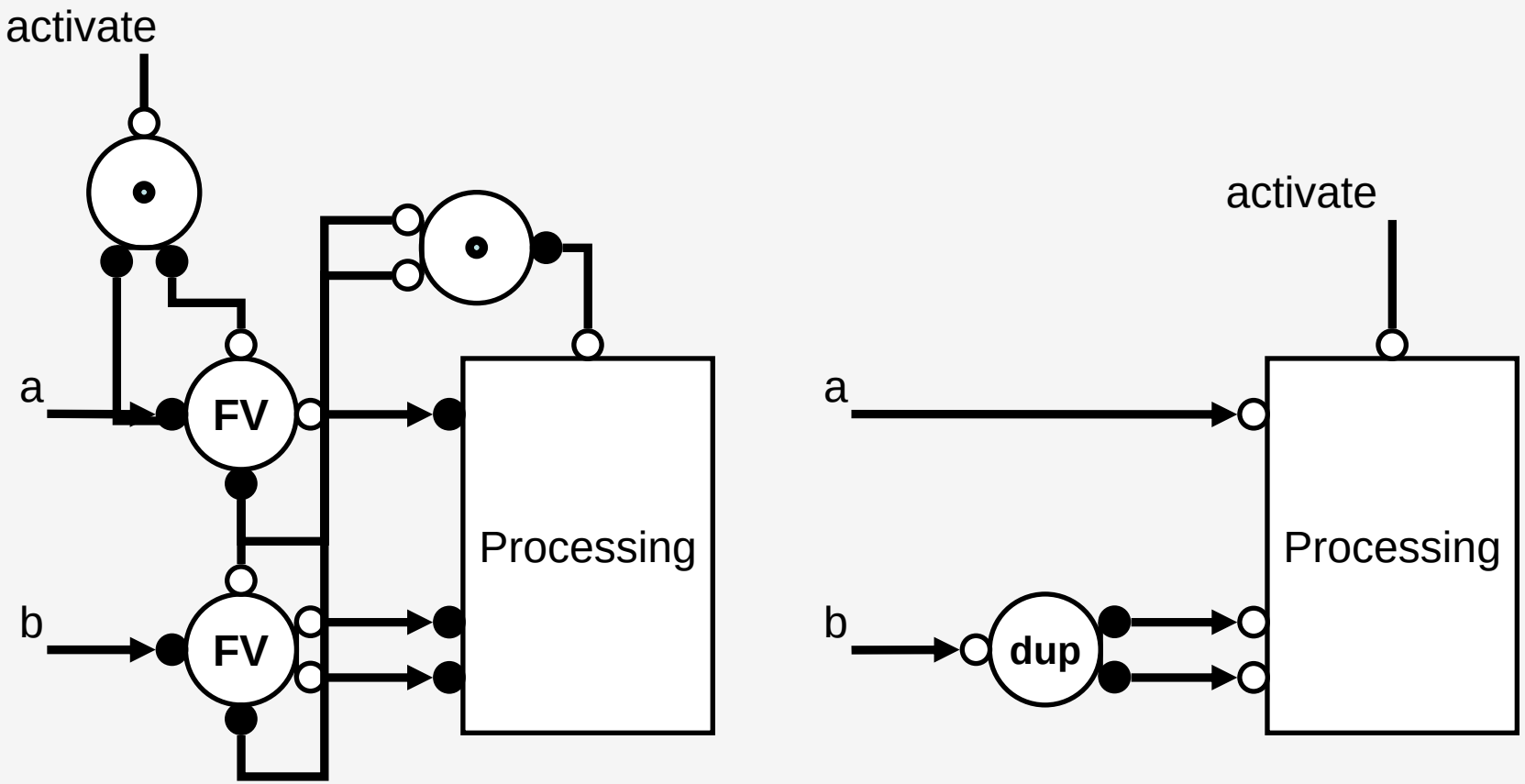


Three main issues

- All inputs are synchronised
- Sequential activation of ‘reads’ and ‘writes’
- Data processing operations occur sequentially after control instead of in parallel

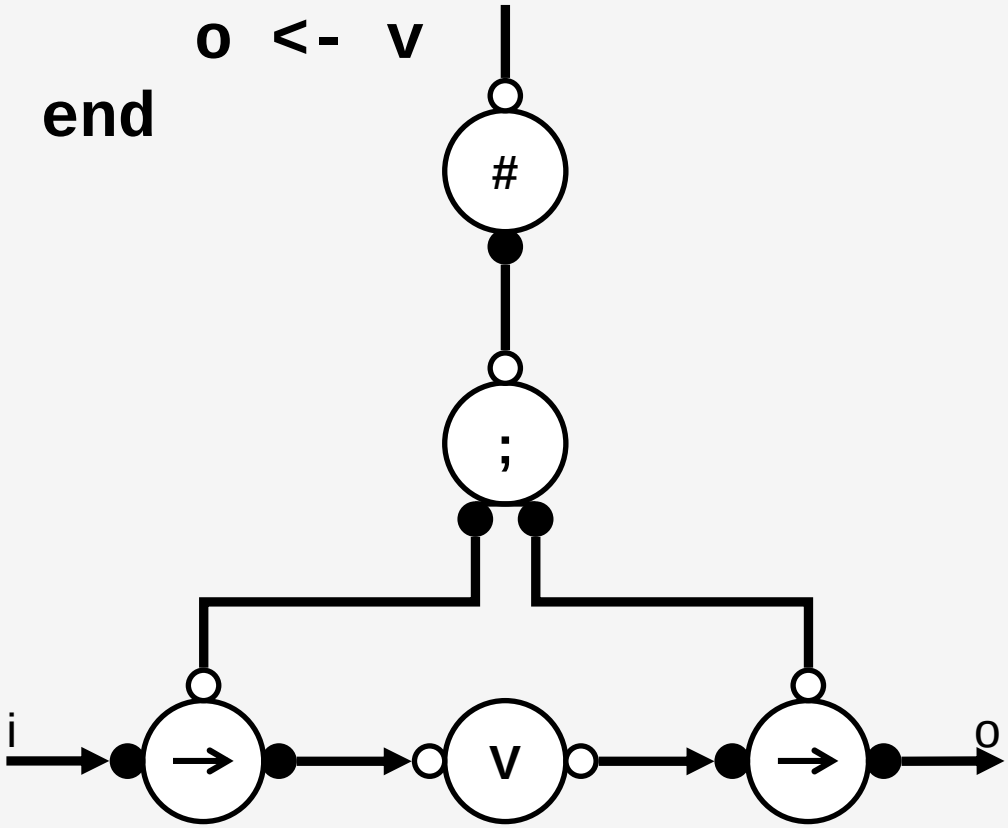
  So look at the main structures of Balsa handshake circuits and replace with data-driven alternatives

Input control



Localised sequencing

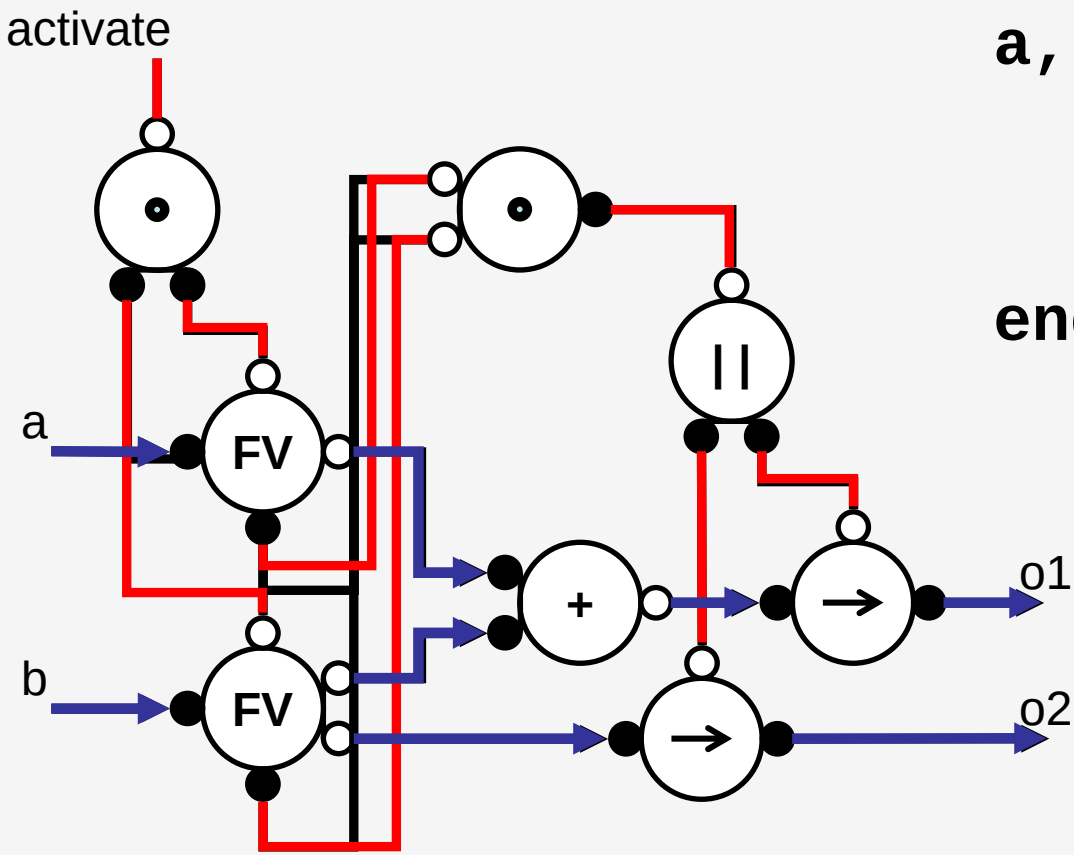
```
loop
     i -> v;
     o <- v
end
```




```
input  i
output v
during
     v <- i
end
```

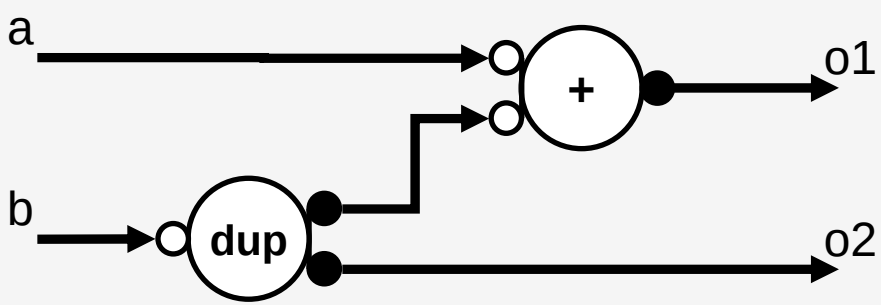
```
input  v
output o
during
     o <- v
end
```

Data processing

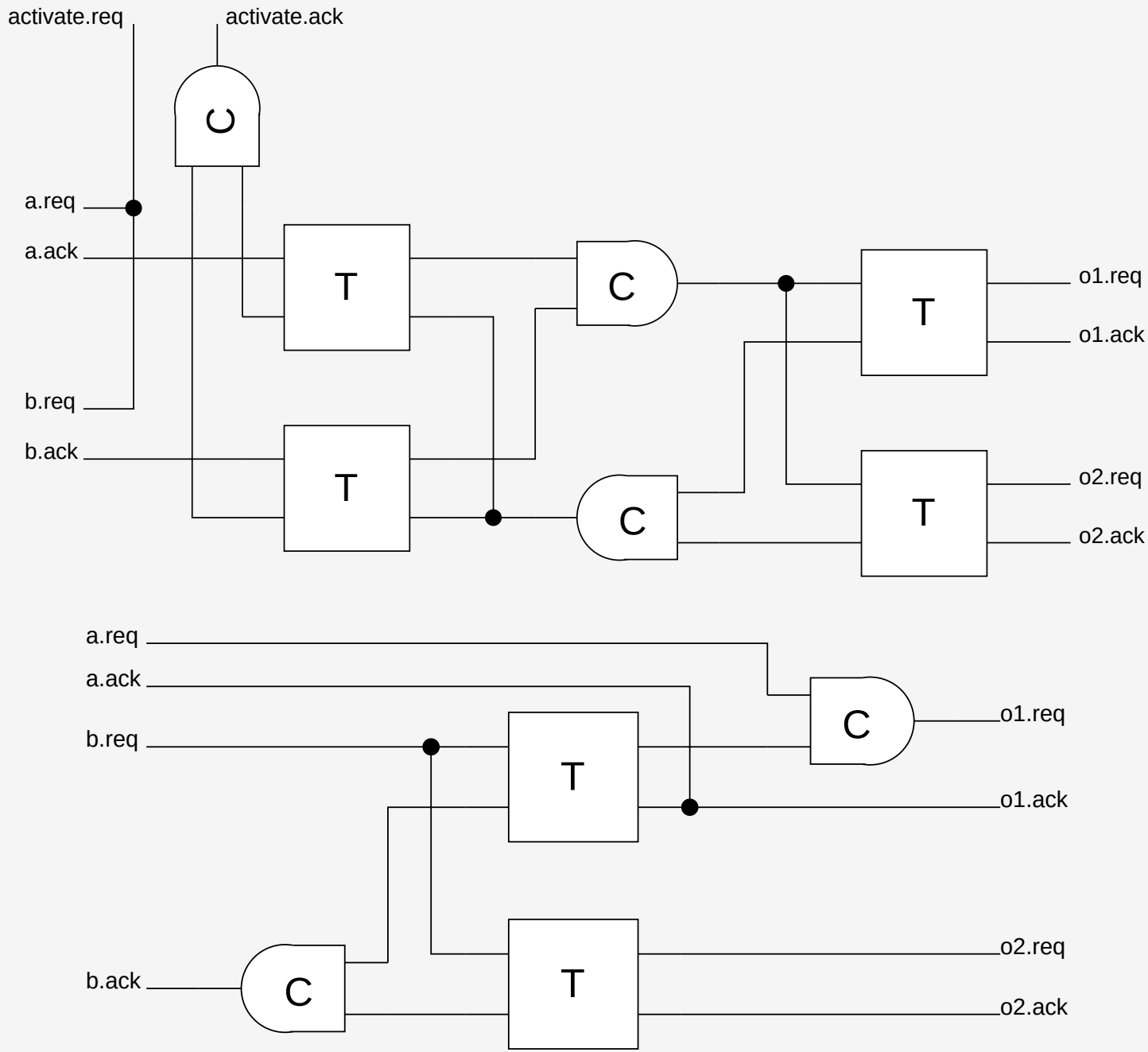



```
a, b -> then
     o1 <- a + b
     || o2 <- b
end
```

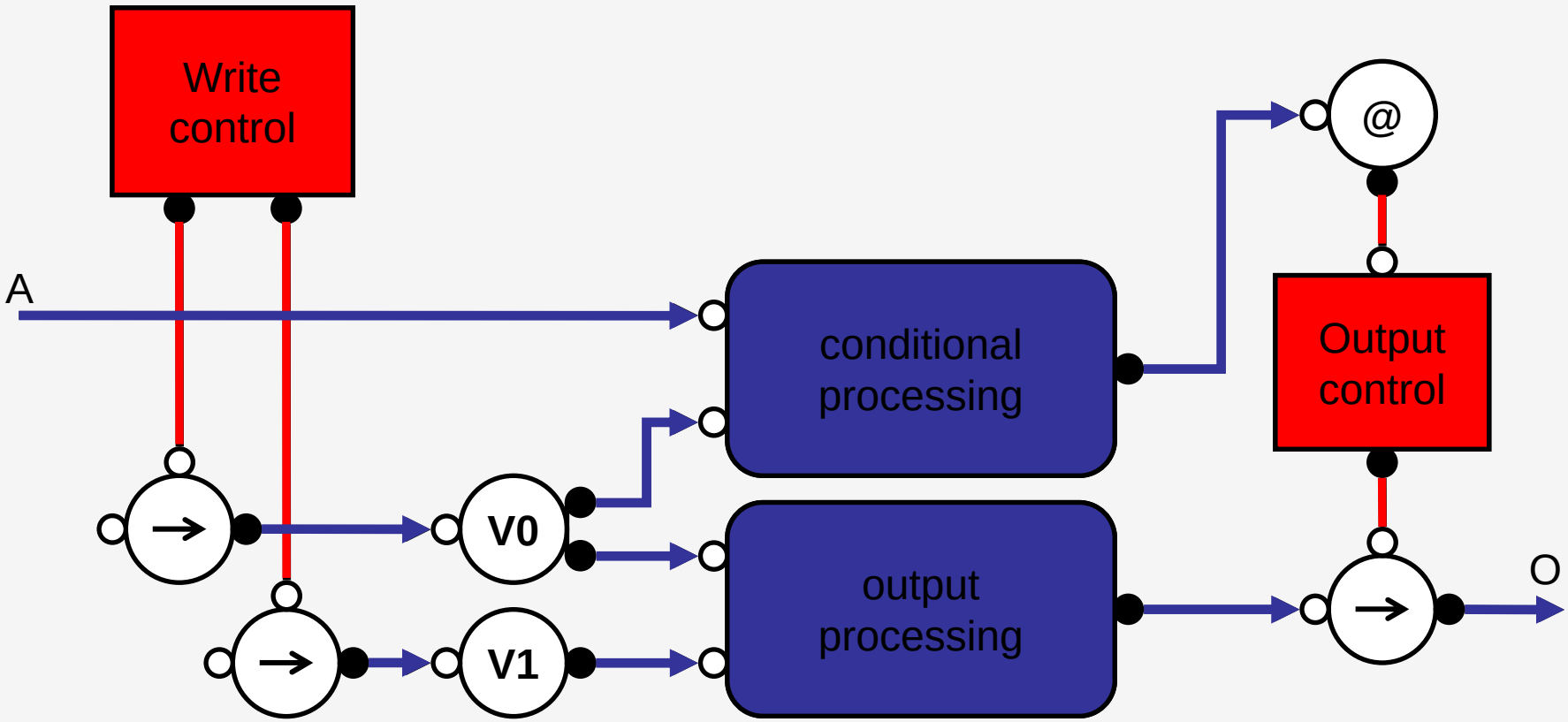
Data processing




```
input  a, b
output o1, o2
during
     o1 <- a + b
     o2 <- b
end
```


activate.req
activate.ack
C
a.req
a.ack
b.req
b.ack
T
T
C
C
T
T
o1.req
o1.ack
o2.req
o2.ack
a.req
a.ack
b.req
b.ack
C
T
C
T
C
o1.req
o1.ack
o2.req
o2.ack

Data-driven structure



Code

```
a, b -> then
    o1 <- a + b
    || o2 <- b
end
```

```
input  a, b
output o1, o2
during
    o1 <- a + b
    o2 <- b
end
```

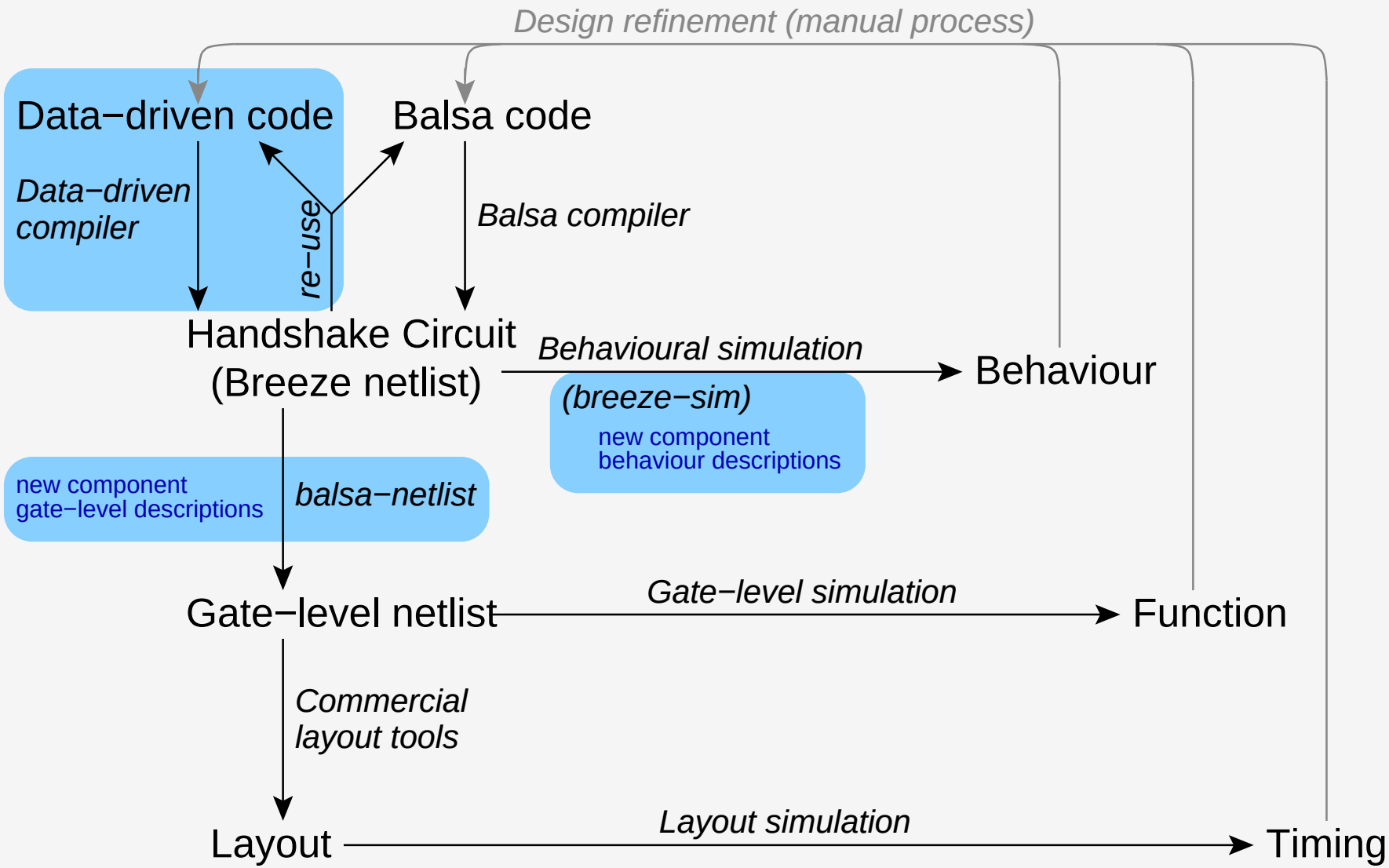
Each block in data-driven code is basically the description of a pipeline stage.

Balsa vs. data-driven philosophy

- Collect all inputs
- Decide what operation to do
- Do the operation
- Release the inputs

- List of operations
- Do all of these operations as soon as you can (speculate)
- Don't synchronise until you absolutely must
- Throw away the results of operations you don't need

Design Flow


Design refinement (manual process)
Data−driven code
Balsa code
Data−driven compiler
re−use
Balsa compiler
Handshake Circuit (Breeze netlist)
Behavioural simulation
(breeze−sim)
new component behaviour descriptions
Behaviour
new component gate−level descriptions
balsa−netlist
Gate−level netlist
Gate−level simulation
Function
Commercial layout tools
Layout
Layout simulation
Timing

nanoSpa

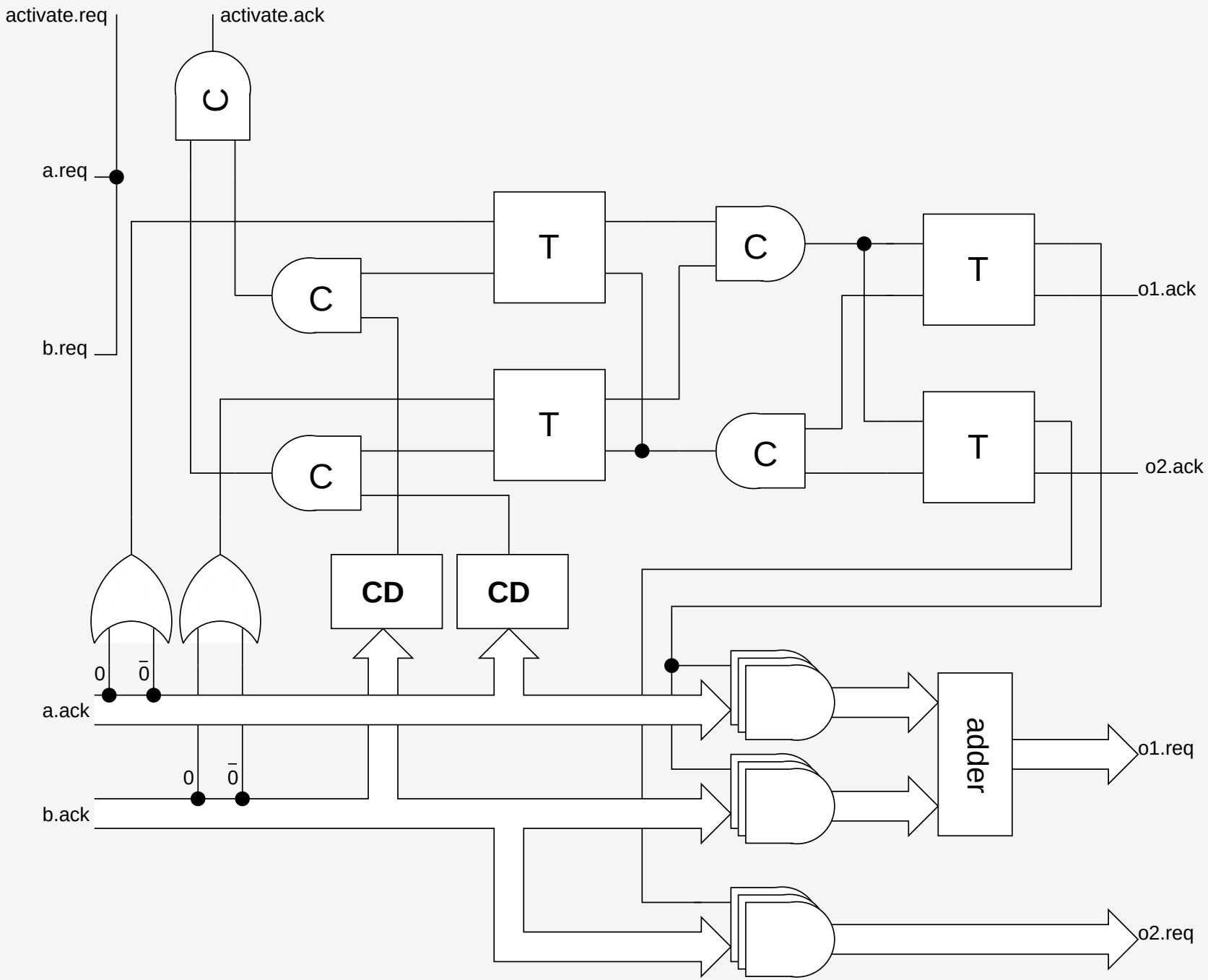
- Cut-down ARM processor
- Balsa design intended for maximum performance
- Data-driven equivalent with same architecture and handshake component implementation style (try to look just at improvement from structure)
- Data-driven bundled data and dual-rail implementations both about 1.5x improvement over Balsa version

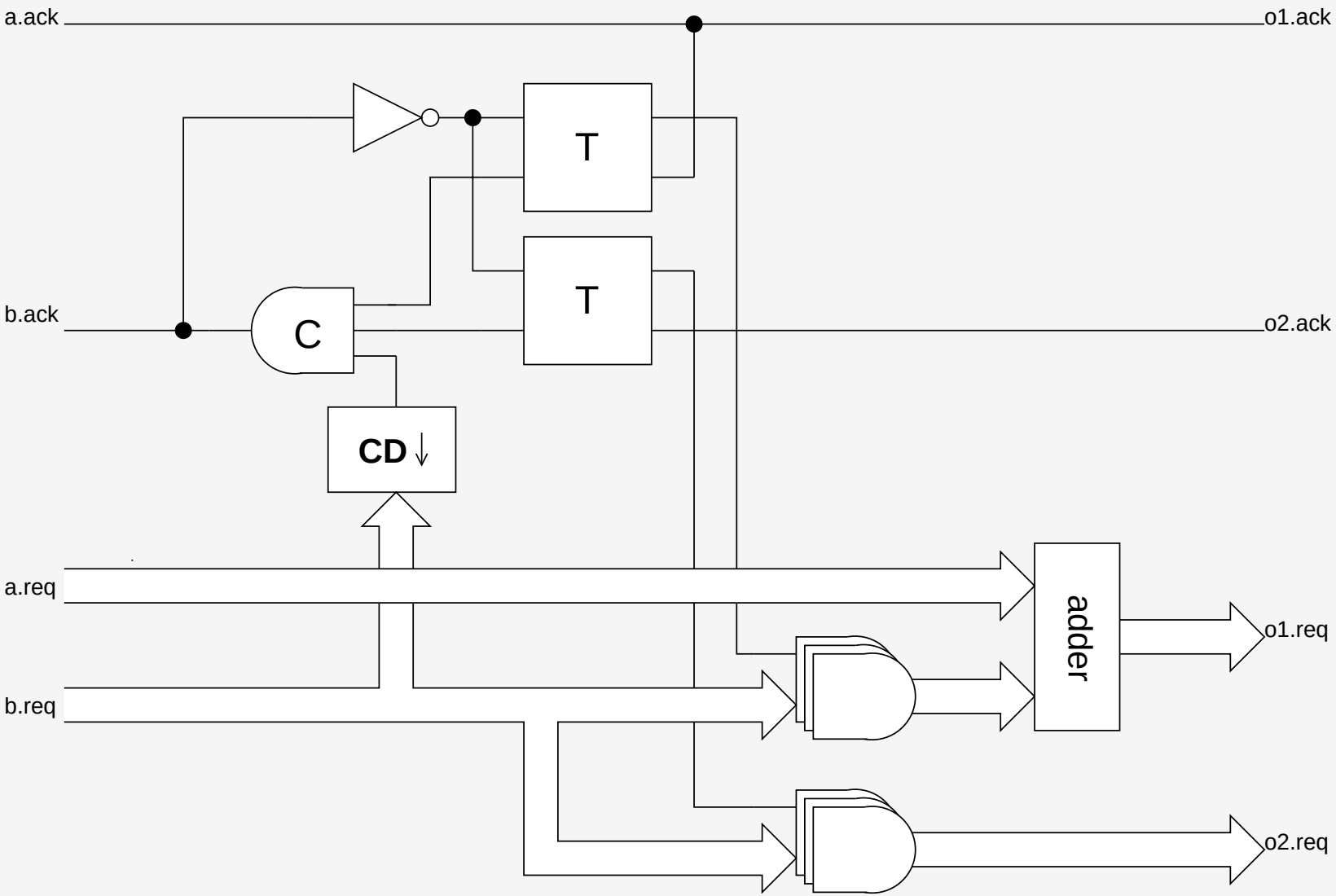
Syntax-directed translation?

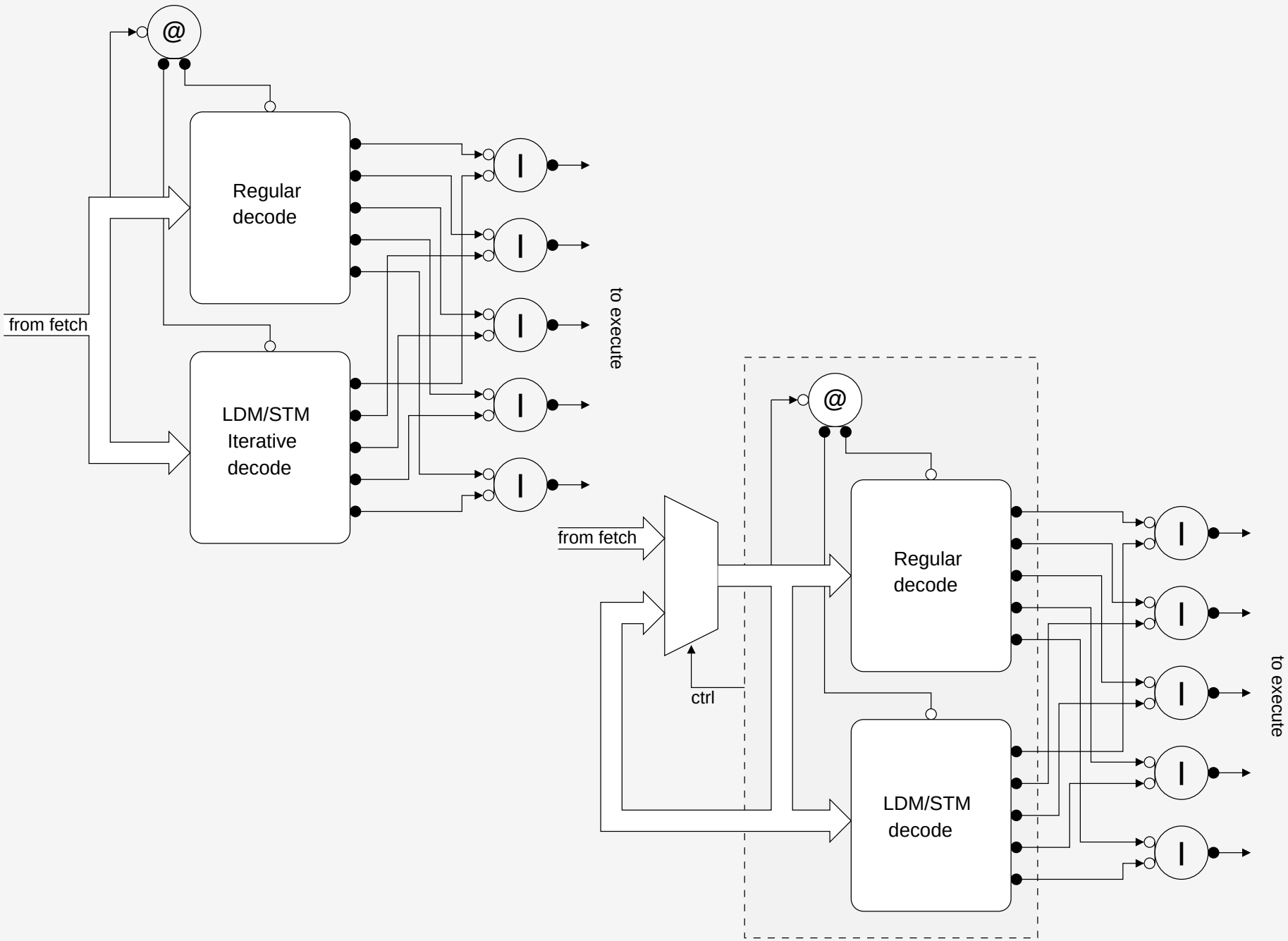
- To use syntax-directed translation I restricted the input language so that one could only write what I wanted to produce!
- This is probably fine for an experienced designer – it gives them what they want.
- Probably not fine for others – they don’t know how to think ‘asynchronous’.
- But the same thinking is needed to write fast Balsa.

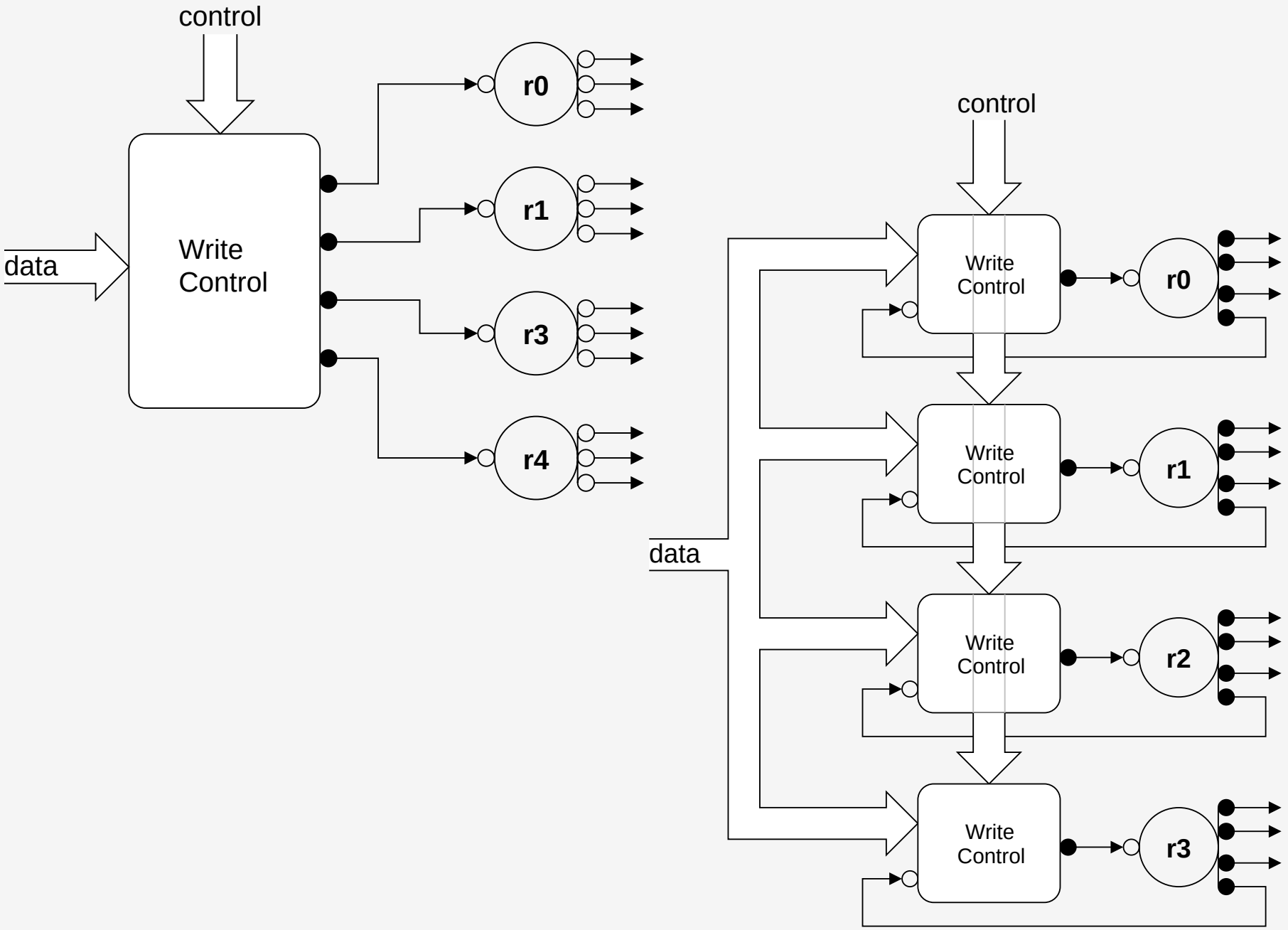
Conclusion

- The structure of control-driven handshake circuits is familiar and flexible but contributes to their poor performance
- Data-driven circuits perform better but are not as familiar and flexible
- Both styles can be combined in the same flow
- Future work could include automatic transformation from control to data-driven or at least more structures to assist data-driven design


activate.req
activate.ack
C
a.req
b.req
T
C
C
T
o1.ack
T
C
C
T
o2.ack
CD
CD
0
0̄
a.ack
0
0̄
b.ack
adder
o1.req
o2.req


a.ack
o1.ack
T
T
b.ack
C
o2.ack
CD ↓
a.req
adder
o1.req
b.req
o2.req


@
Regular
decode
from fetch
LDM/STM
Iterative
decode
to execute
@
from fetch
Regular
decode
ctrl
LDM/STM
decode
to execute


control
data
Write
Control
r0
r1
r3
r4
control
data
Write
Control
r0
Write
Control
r1
Write
Control
r2
Write
Control
r3