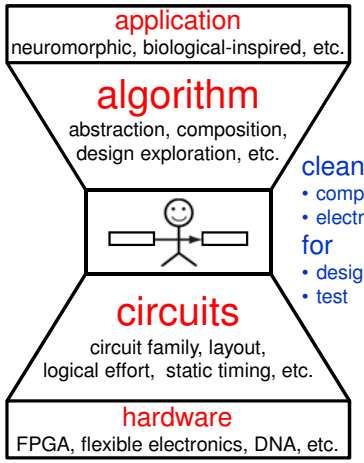
Asynchronous Computing

4. Initialization, Test, and Debug: Action versus State

Marly Roncken and Ivan Sutherland
Asynchronous Research Center (ARC)
Maseeh College of Engineering and Computer Science
Portland State University
July 2018

Vision

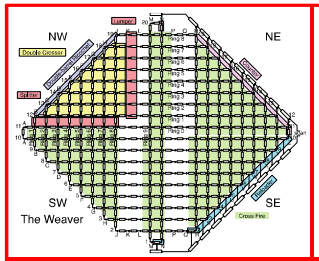


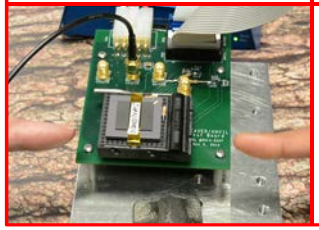
clean interface between
- computer scientists
- electrical engineers

for
- design
- test

After: Kees van Berkel. Handshake Circuits, Fig. 1.1. Cambridge University Press, 1993.

so MANY signals – so LITTLE access





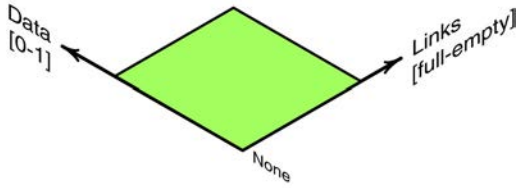
- Like software
  - so many lines – so few exports
- use
  - interactive code debug
  - to set states
  - and breakpoints for single-step code, etc.

combine the best of both worlds

- Like hardware
  - so many wires – so few pins
- use
  - scan to share pins to read or write states
  - MrGO to control actions

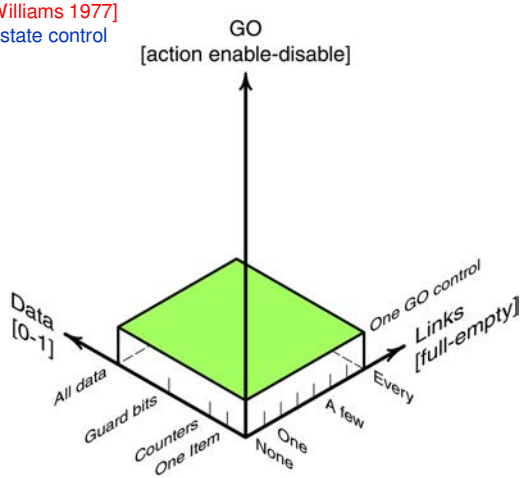
Test control (1/3): none

- external signals only



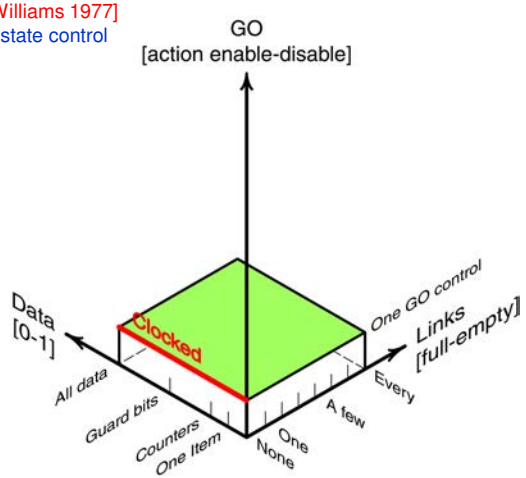
Test control (2/3): scan

- [Eichelberger-Williams 1977]
- global action + state control



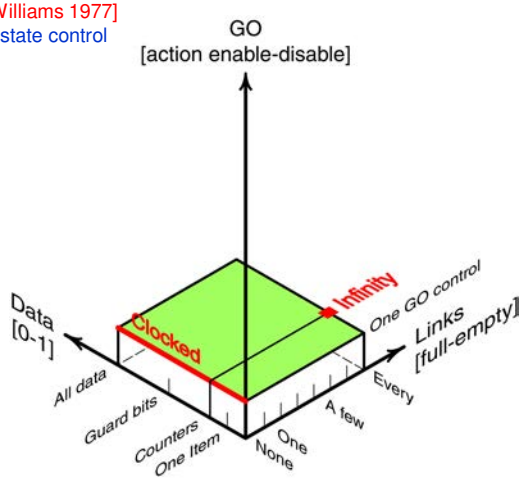
Test control (2/3): scan

- [Eichelberger-Williams 1977]
- global action + state control



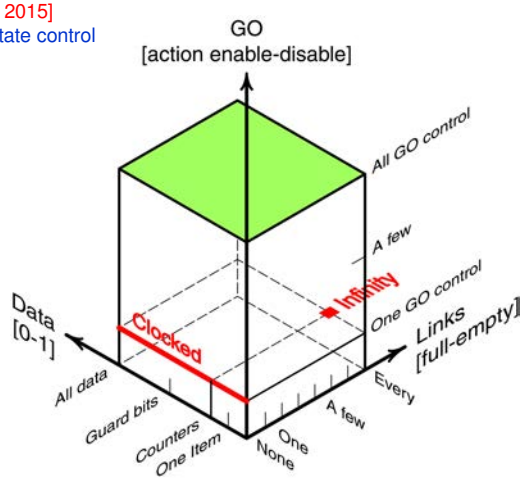
Test control (2/3): scan

- [Eichelberger-Williams 1977]
- global action + state control



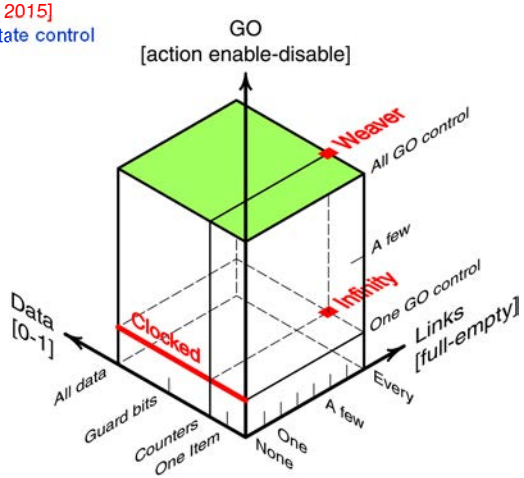
Test control (3/3): scan + GO-per-action

- [Roncken et al. 2015]
- local action + state control



Test control (3/3): scan + GO-per-action

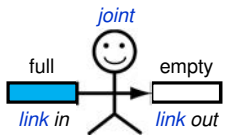
- [Roncken et al. 2015]
- local action + state control



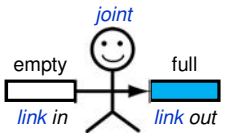
GO: (individual) local action control

Building blocks: action reminder

WHEN to act:
in is full
and
out is empty

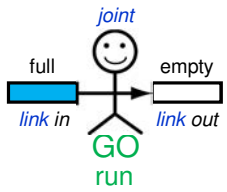


WHAT to do:
- copy data
- drain *in*
- fill *out*

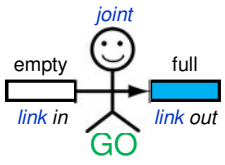


Building blocks: action with GO control

WHEN to act:
in is full
and
out is empty
and
GO



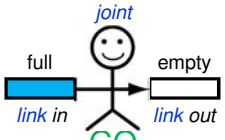
WHAT to do:
- copy data
- drain *in*
- fill *out*

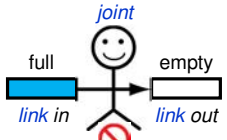


Building blocks: action with GO control

WHEN to act:

in is full
and
out is empty
and
GO

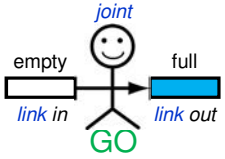




stop + freeze

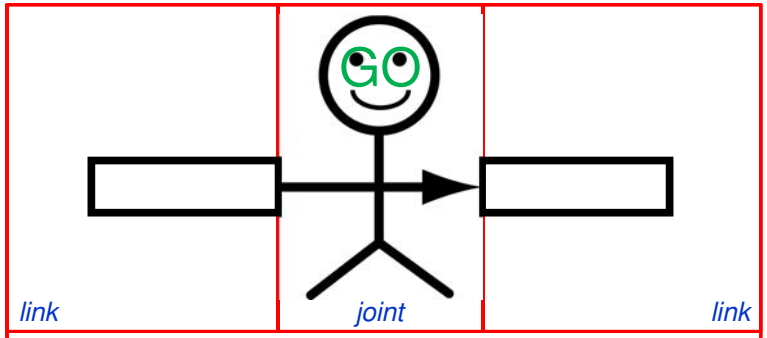
WHAT to do:

- copy data
- drain *in*
- fill *out*



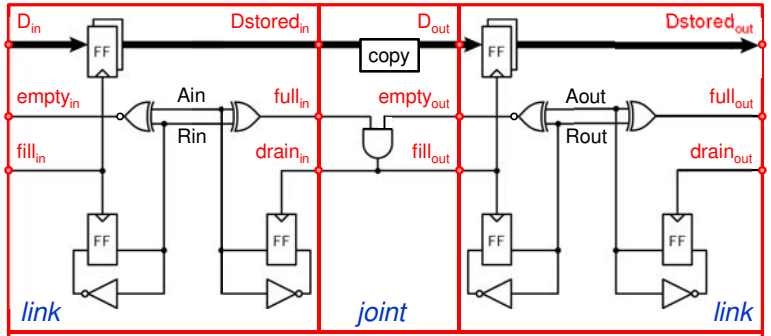
no action

Building blocks: design with GO control



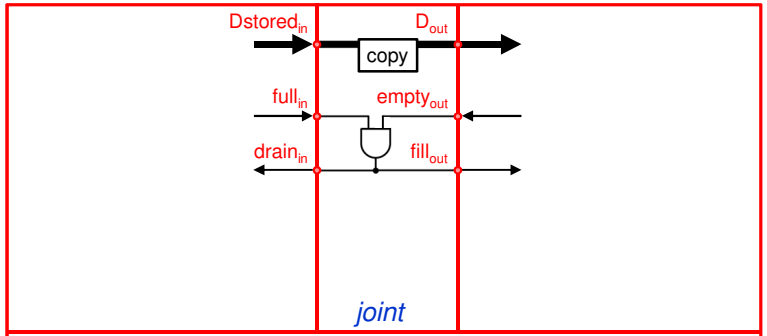
design reminder

Building blocks: design with GO control



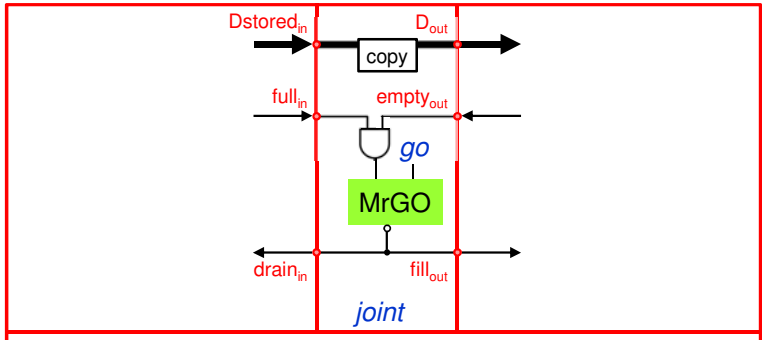
design reminder

Building blocks: design with GO control



design reminder

Building blocks: design with GO control

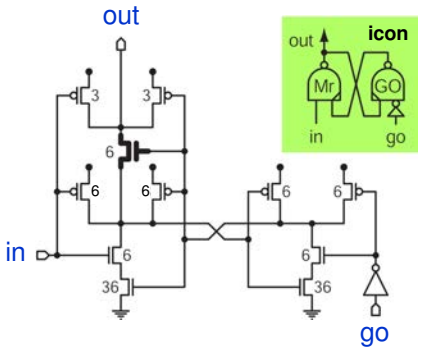


Solution MrGO: pronounced "Mister GO"

- *go* is high (GO) : run
- *go* is low (⃠) : stop and freeze
- arbiter for safe stop : "proper stopper"
- scan chain delivers *go* signals

MrGO: dedicated action control

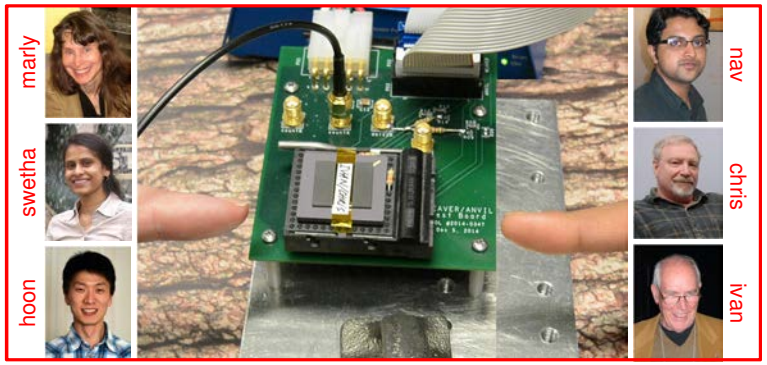
- go is high (GO) – grant in high to make out low
- go is low (⃠) – keep out high
- arbiter for safe stop – "proper stopper"
- scan chain delivers go signals



Get real!

MrGO approved

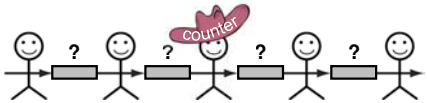
- Two working silicon experiments: Weaver and Anvil
- use link-joint building blocks with full-empty interfaces
- and MrGO + JTAG-scan for test, debug, characterization



At-speed test and debug using MrGO + scan

Testing a counter at speed

INITIALIZE

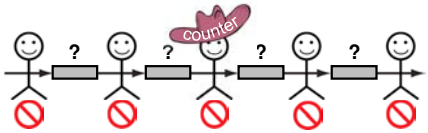


RUN

EVALUATE

Testing a counter at speed

INITIALIZE
1. freeze all joints

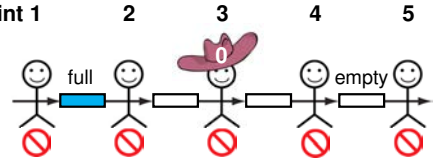


RUN

EVALUATE

Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   - full-empty links
   - counter data

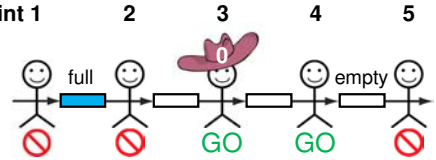


RUN

EVALUATE

Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   - full-empty links
   - counter data
3. unfreeze "runway" (3,4)



RUN

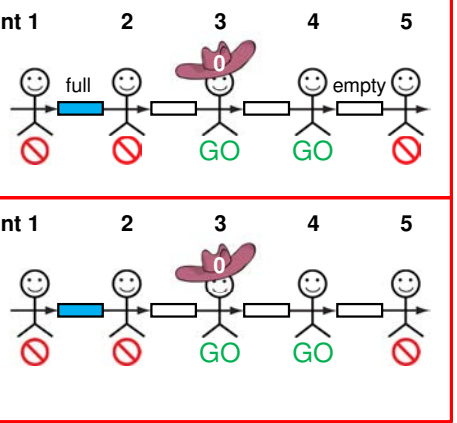
EVALUATE

Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   • full-empty links
   • counter data
3. unfreeze "runway" (3,4)

RUN

EVALUATE

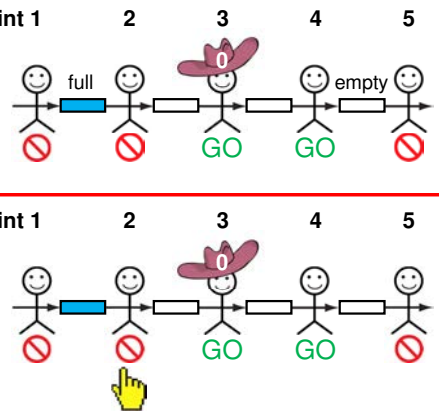


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   • full-empty links
   • counter data
3. unfreeze "runway" (3,4)

RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

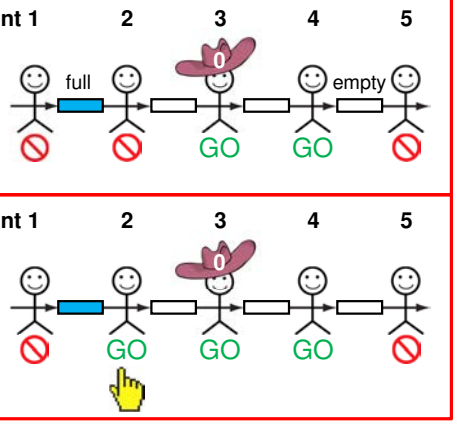


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   • full-empty links
   • counter data
3. unfreeze "runway" (3,4)

RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

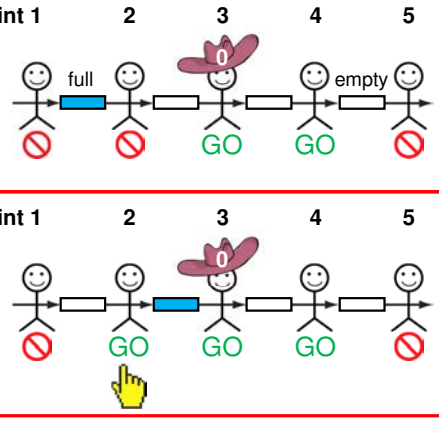


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   • full-empty links
   • counter data
3. unfreeze "runway" (3,4)

RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

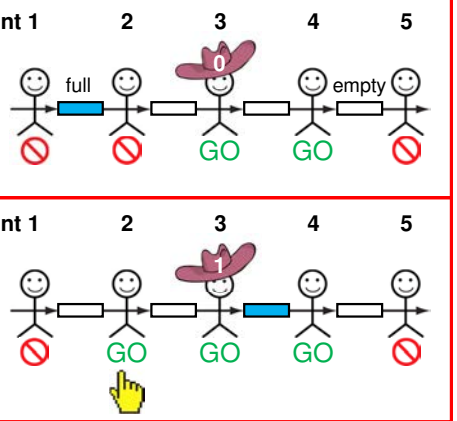


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   - full-empty links
   - counter data
3. unfreeze "runway" (3,4)

RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

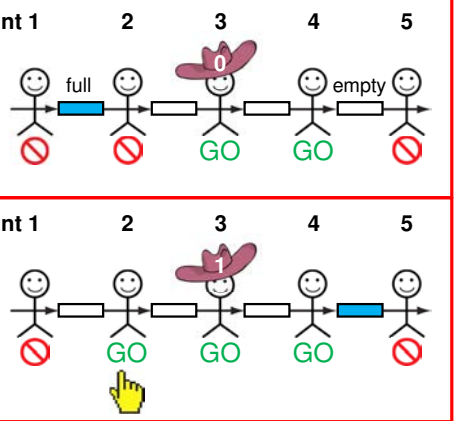


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   - full-empty links
   - counter data
3. unfreeze "runway" (3,4)

RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE

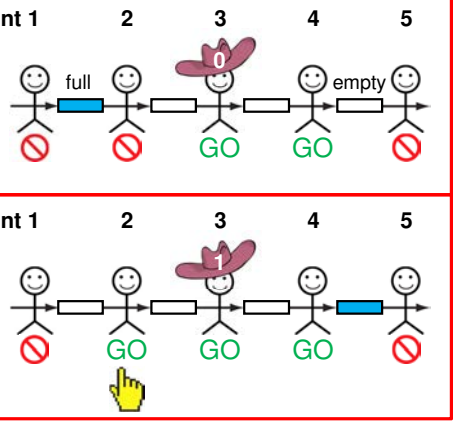


Testing a counter at speed

INITIALIZE
1. freeze all joints
2. set state
   - full-empty links
   - counter data
3. unfreeze "runway" (3,4)

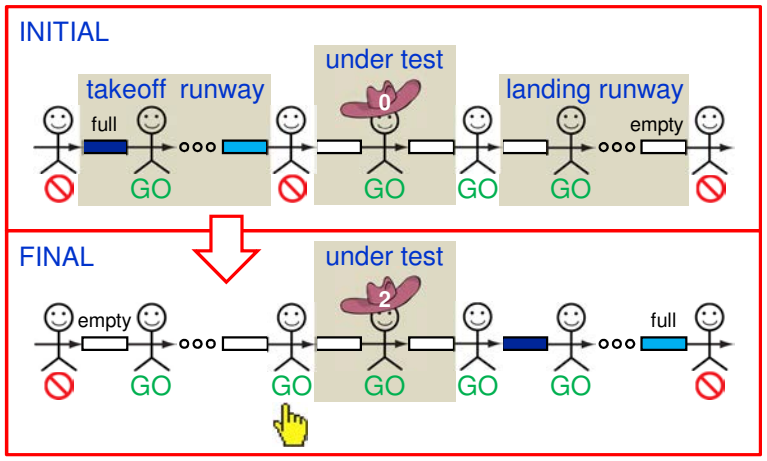
RUN
1. unfreeze entry (2)
2. wait for action to finish

EVALUATE
- read counter data



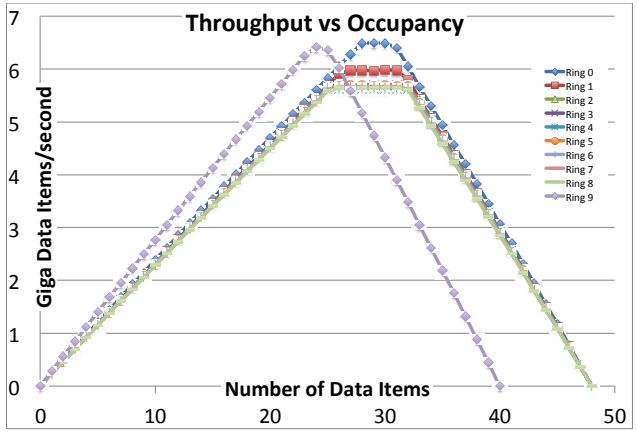
Testing a burst of data using MrGO + scan

Testing a burst of data at speed



Characterization of throughput using MrGO + scan

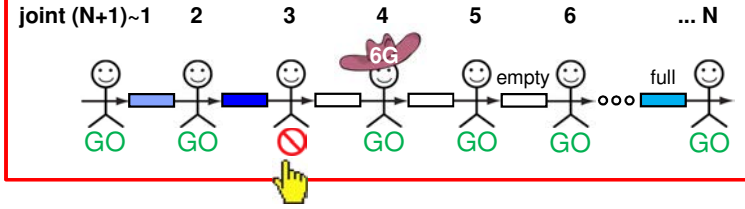
Performance characterization (Weaver)



Performance characterization

DO (ALL > *i* > 0 links)
  counter=0
  run 1 second with *i* full links
  arbitrated stop
  read counter
OD

FINAL for *i* ~ 60% links



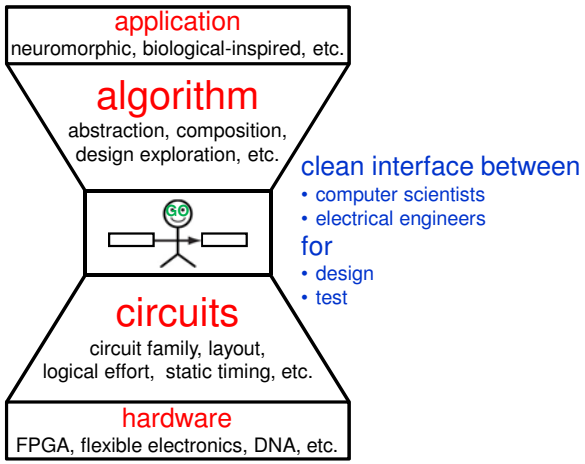
MrGO-scan: operation rules

- Informal:
  - don't scan the system when it's running, except to stop it
- More formal:
  - separate initialization-or-inspection from computation
  - stop actions before scanning state that's used by those actions
  - when changing go signals first change those getting disabled
  - use three separate scan chains for go, full-empty, and data

Summary

- Actions and states are equal partners
  - from the bottom up
  - MrGO
    - controls each action with a clean start and stop
    - separates initialization from computation
  - scan
    - reads and writes the states of data, full-empty, go signals
- Interfaces matter
  - design them for collaboration and re-use
  - MrGO-scan interface
    - works for computer scientists and electrical engineers
    - unifies interactive debug for code and silicon

Vision (picture)



After: Kees van Berkel. Handshake Circuits, Fig. 1.1. Cambridge University Press, 1993.

Vision (text)

We want to use this clean and simple interface to enable computer scientists and electrical engineers to collaborate and to design and test— jointly—the systems of the future whose computations—we believe—will be distributed over space and time and will be of a self-timed nature.