Beyond classical chip design
lecture 4

Circuit model (continued)

Further Reading

Alain J. Martin: *Synthesis of Asynchronous VLSI Circuits.* Tech report California Institute of Technology, 1991.

Alain J. Martin and Mika Nyström: *Asynchronous techniques for system-on-chip design.* Proceedings of the IEEE Volume 94, Issue 6:1089 - 1120, June 2006.

What we had...

- Production rules
- Gate, circuit
- A implements B
- B can simulate A

Remember...

Stable transition functions:

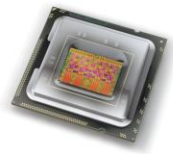
i can make a transition to c at time t &

i **cannot** make a transition to c at time $t+1$

->

i made a transition at time $t+1$

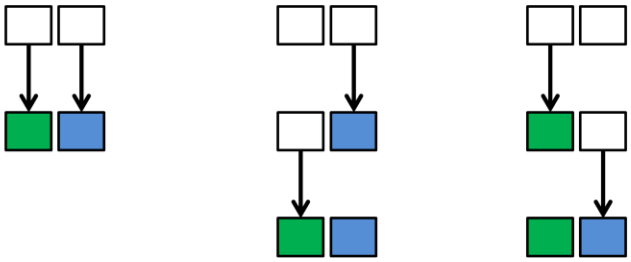
(and thus is in c at time $t+1$)



Remember...

"distributed schedule" $s(t) \subseteq [n]$

stable + distributed schedule ->
"linearizable to" schedule [later]



Now...

Stable production rules:

Rule (i) can make a transition to c at time t &

Rule (i) **cannot** make a transition to c at time $t+1$

->

Rule (i) made a transition at time $t+1$

Stability

Stable production rules:

Rule (i) is enabled at time t &

Rule (i) is **disabled** at time $t+1$

->

Rule (i) made a transition at time $t+1$

can be even simplified more from the last slide, since a rule (i) can make only a non-trivial transition to the same next state x-up, e.g.

enabled :<-> guard is true and the transition is non-trivial
disabled :<-> not enabled

Linearizable

Linearizable:
Given initial state x and
distributed schedule prefix $s(0)$
let state y = x with enabled rules in $s(0)$
make a step;

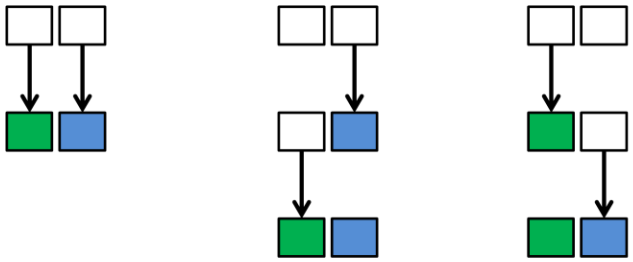
then exists schedule prefix $s'(0), s'(1), \ldots, s'(k)$
of rules in $s(0)$ such that
y = x with enabled nodes in $s'(0), s'(1), \ldots, s'(k)$
make a step.

linearizable: intuitively: possible to transform into a purely sequential schedule which has the same final result.
what we want: all prefixes of distributed schedules linearizable.
this follows from the given definition. we simply apply the definition for each set of the distributed schedule and generate a purely sequential schedule
with intermediate states.

Linearizable

"distributed schedule" $s(t) \subseteq [n]$

stable -> linearizable [hw]



Top-level Specifications

Communicating hardware processes

Communicating hardware processes

a la CSP [Hoare, 78]

local variables x, y, ...

ports A, B, ...

- assignment y := x

Communicating hardware processes

Composition:

- parallel c1 || c2
- serial c1; c2
- loop *[c1]

Communicating hardware processes

- blocking communication primitives

  A!x A?y

Communicating hardware processes

- selection

  [P1 -> c1 || P2 -> c2 || ...]

  [P1 -> skip] = [P1]

selection […] blocks only until one of P1, P2, etc is true and then executes the respective actions c1, c2, etc.

Communicating hardware processes

- selection

  [P1 -> c1 || P2 -> c2 || ...]

  [P1 -> skip] = [P1]

- arbitrated selection = choice

  [P1 -> c1 | P2 -> c2 | ...]

arbitrated selection: exactly one of the c1, c2, … becomes executed.

Execution

global state

enabled rule

execution

constraints: fairness, partial order constraints, timed constraints

defined analogously to PRs

Example: the C-Element

C1: CHP from production rules.

$$*[[(a \wedge b) \rightarrow z\uparrow \,||\, (\neg a \wedge \neg b) \rightarrow z\downarrow]]$$

C2: sequential CHP. [hw]

$$*[[a \wedge b]; z\uparrow; [\neg a \wedge \neg b]; z\downarrow]$$

Sequencing

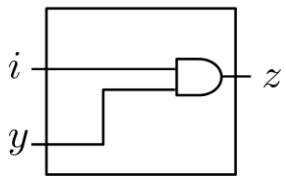
Problem. Implement a;b.

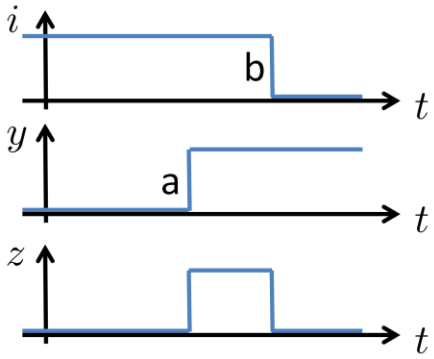
one of the most fundamental problems

Sequencing

Problem. Implement a;b.

… almost all circuits

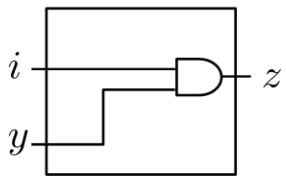


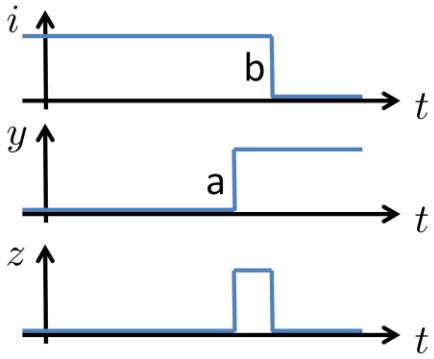


Sequencing

Problem. Implement a;b.

… almost all circuits

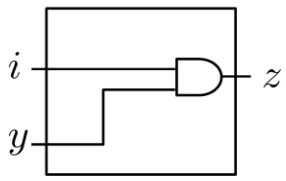


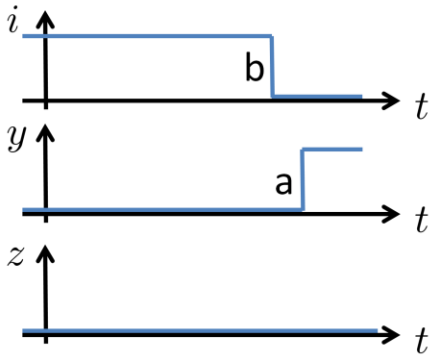


Sequencing

Problem. Implement a;b.

… almost all circuits





Sequencing

Problem. Implement a;b.

… almost all circuits

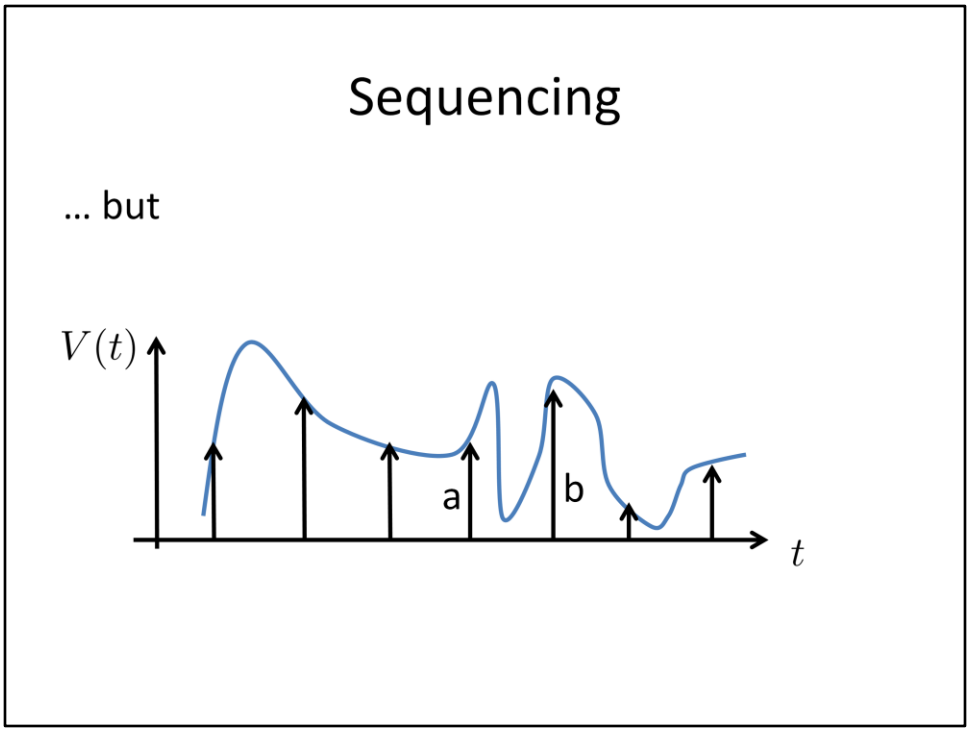
-> executions are just distorted in time

Sequencing

Problem. Implement a;b.

… almost all circuits

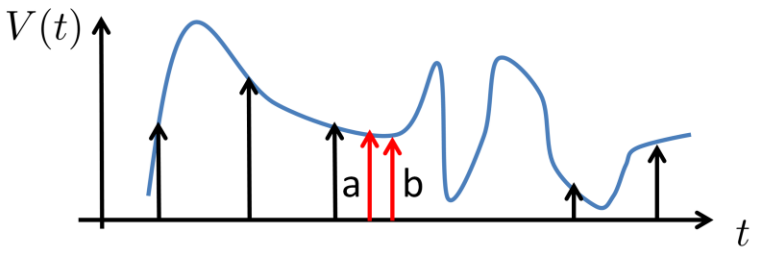
order + worst-case upper bounds.



here exact timing plays a crucial rule, not just ordering: FFT of a signal

Sequencing

… but



Sequencing

Two fundamental solutions.

1. Externally triggered.

2. Trigger themselves.

Sequencing

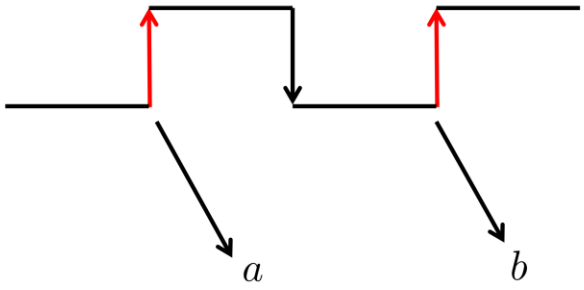
Two fundamental solutions.

1. Externally triggered **= synchronous model**

2. Trigger themselves **= clockless model**

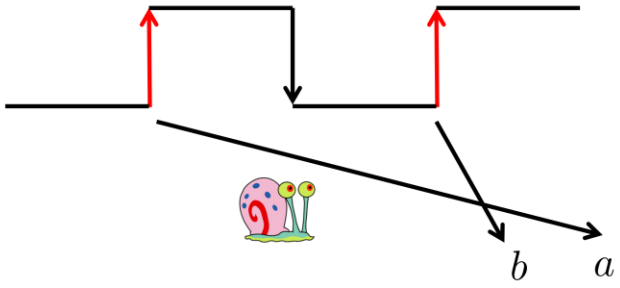
Sequencing

Synchronous model.



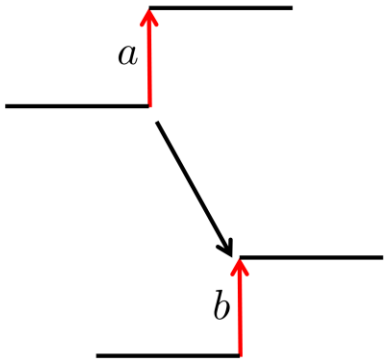
Sequencing

Synchronous model.



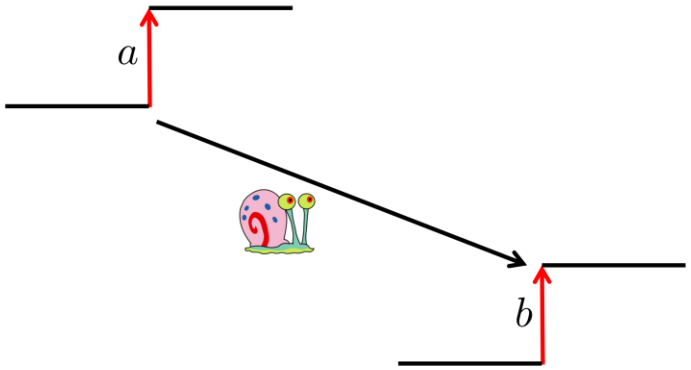
Sequencing

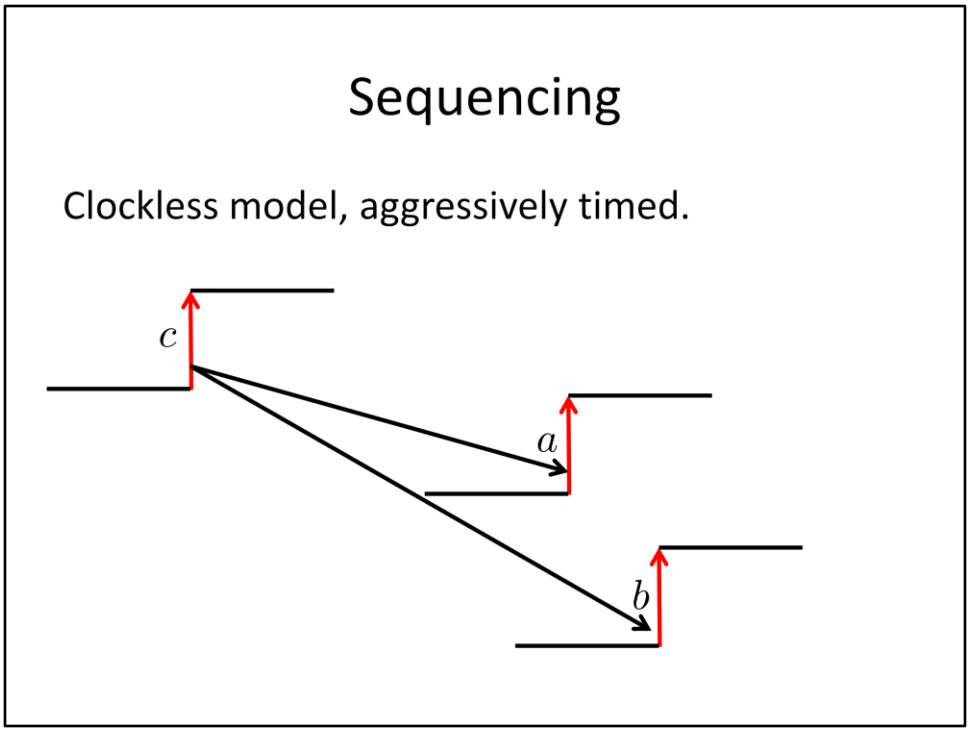
Clockless model.



Sequencing

Clockless model.





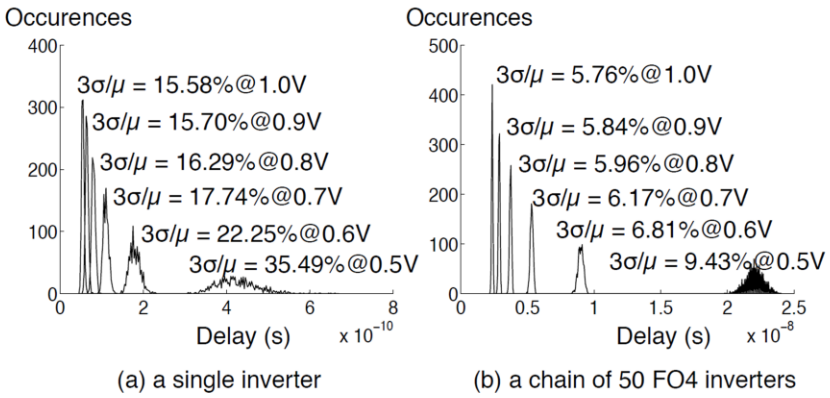
clockless can be heavily time dependent, not at all robust to delay variations.

Which one to choose...

… depends, e.g., on

- extent of delay variations
- allowed power consumption

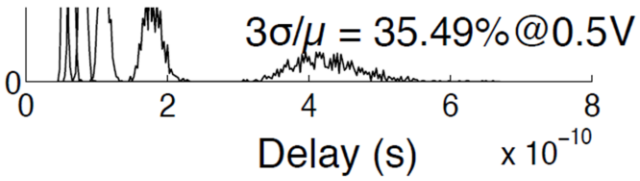
Delay Variations



(a) a single inverter (b) a chain of 50 FO4 inverters

90nm technology variations, Seo et al., DAC'12

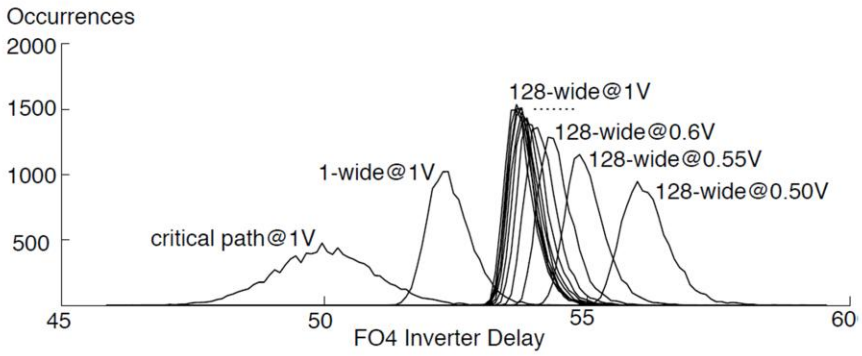
Remember…



assuming normal distribution

$P(\text{Delay} \in [\mu - 3\sigma, \mu + 3\sigma]) \approx 0.9973$

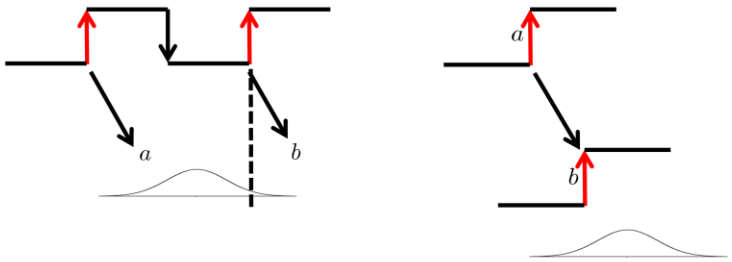
Delay Variations



critical path 90nm (near th), Seo et al., DAC'12

Delay Variations

P(fail) <-> throughput [hw]



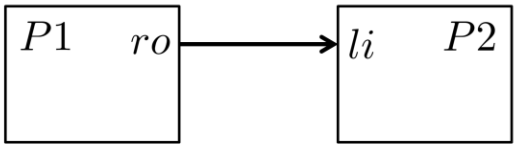
The synchronous (left) data send fails if the delay of a is longer than the clock period.

Sequencing

Require: a;b.

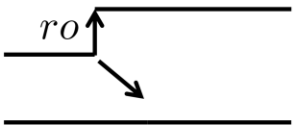
$P1 : \ldots a \ldots$

$P2 : \ldots b \ldots$



$P1 : a; ro \uparrow$

$P2 : [li]; b$



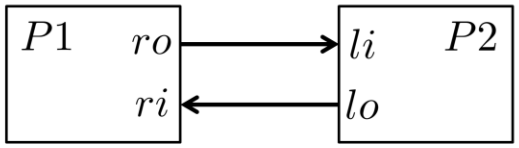
doing this once. single line sufficient. how do it several time? -> multishot problem

Sequencing - multishot

Require: a;b.

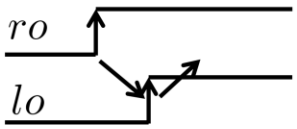
$P1 : *[\ldots a \ldots]$

$P2 : *[\ldots b \ldots]$



$P1 : a; ro\uparrow; [ri]$

$P2 : [li]; b; lo\uparrow$



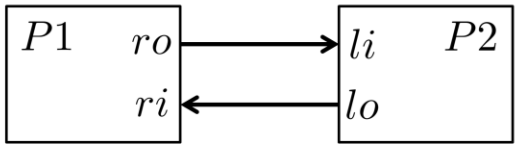
multishot = solve several times, not just once -> add a link back from receiver to sender to signal when ready for the next shot.

Sequencing - multishot

Require: a;b.

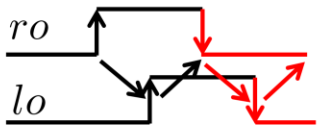
$P1 : *[\ldots a \ldots]$

$P2 : *[\ldots b \ldots]$



$P1 : a; ro\uparrow; [ri]; a; ro\downarrow; [\neg ri]$

$P2 : [li]; b; lo\uparrow; [\neg li]; b; lo\downarrow$



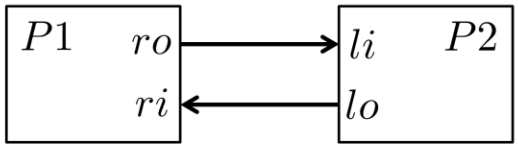
different program parts for even and odd sequencing of a;b

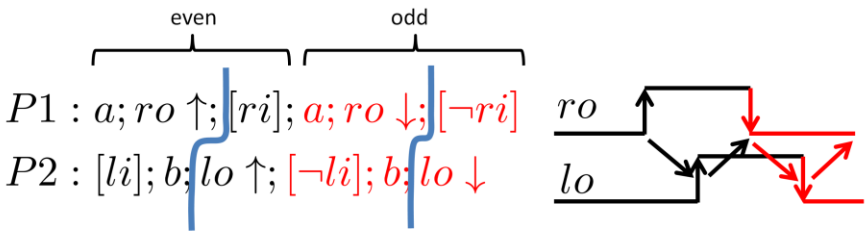
Sequencing - multishot

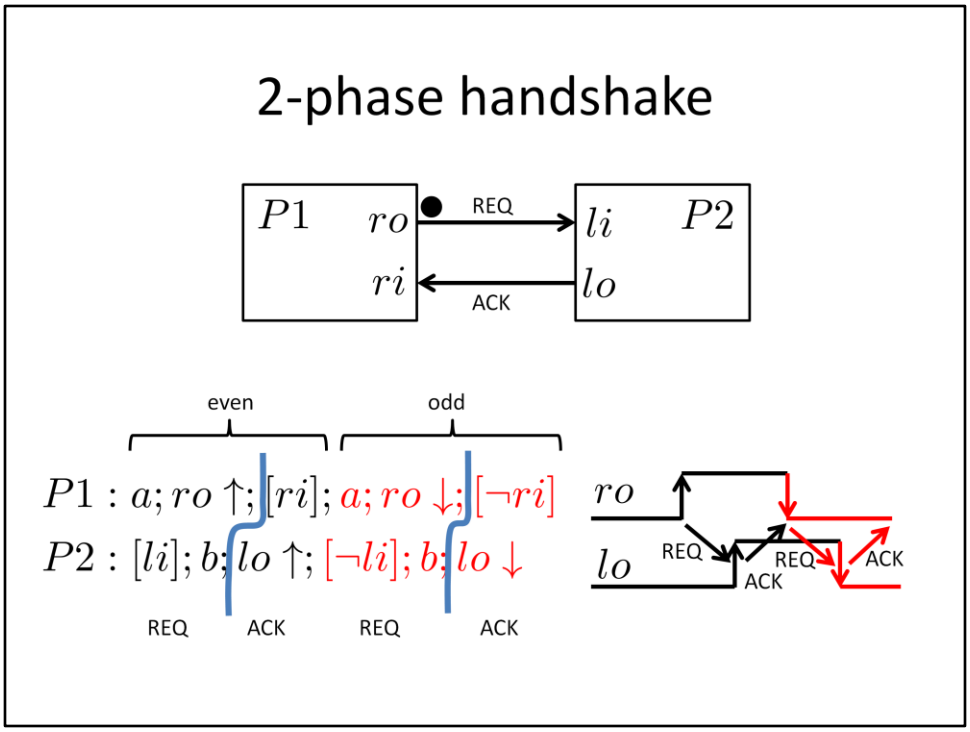
Require: a;b.

$P1 : *[\ldots a \ldots]$

$P2 : *[\ldots b \ldots]$

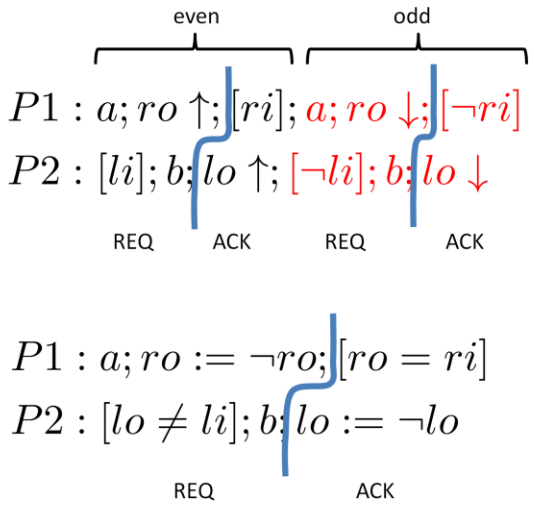




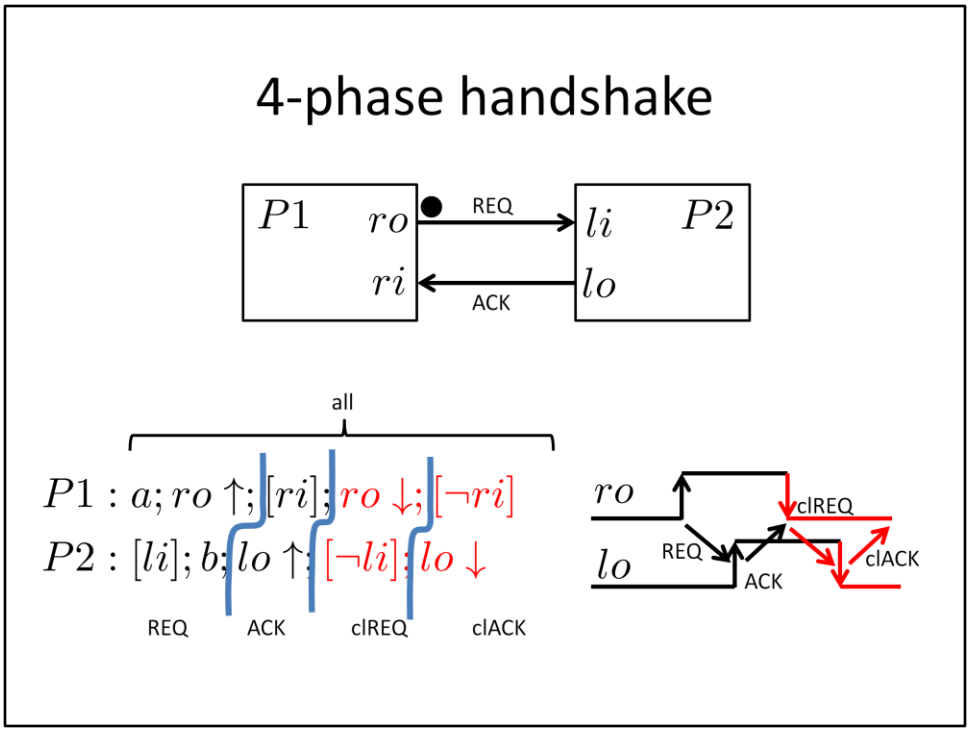


black dot = active node.
active = starts with send
passive = starts with wait
always match an active interface with a passive one.
per sequencing -> 2 phases (req and ack phase: „2-phase handshake“)

2-phase handshake



even and odd instances can be expressed by the same code.



4 phases per sequencing.

multishot-sequencing

Techniques:

- clocked: 1pair/D
- 2-phase: 1pair/2D
- 4-phase: 1pair/4D

D = communication delay from sender module to recevier module or vice versa.
Note that D differs in fact for all the above techniques -> depends which one is best.