1995 Israel Workshop on Asynchronous VLSI
Design Styles

Alex Yakovlev

DESIGNING ARBITERS USING PETRI NETS

DESIGNING ARBITERS USING PETRI NETS

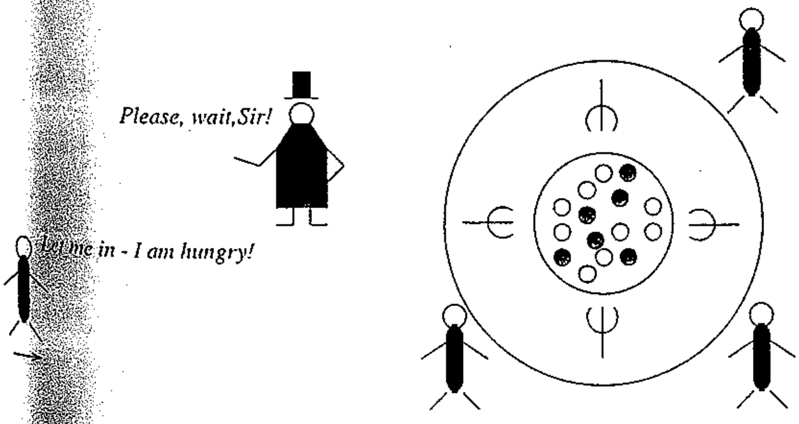
Alex Yakovlev
University of Newcastle upon Tyne, U.K.

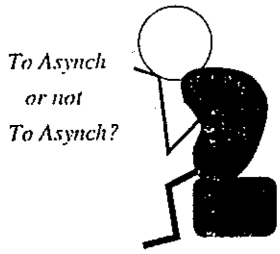
with Contributions from
Luciano Lavagno, Jordi Cortadella and Alex Semenov

Israel Workshop on Asynch VLSI
Nof Genosar - March - 1995

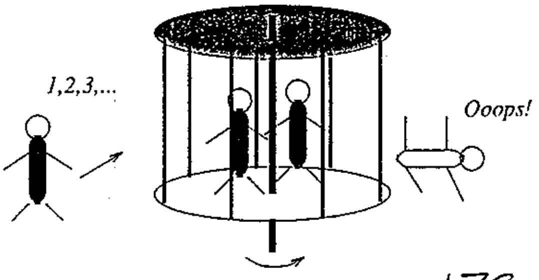
WHY "DESIGNING ARBITERS" ?

* Resource Allocation





* Synchronisation



WHY "DESIGNING ASYNCHRONOUS ARBITERS" ?

E.g.:

* Routing Chips

* Pipeline synchronisation in Amulet and Sproull's CFP

* Post Office bus arbitration

* Philips DCC Error Corrector Chip

* Hazard-free Transparent Latching

...

Attempts to construct a clocked arbiter face with problems:

Who will clock it ? Who will synchronise clock with the rest of the arbiter ?
Who will clock this synchroniser? ...

If the rate of asynchronous requests is high, the metastability rate
(and hence failure rate) is high, too

arb1.2

WHY "USING PETRI NETS" ?

Petri nets

* discrete and asynchronous by nature

* abstract from data path

* conflicts, causality and concurrency are represented in natural form

* verifiable (at the discrete-event/finite state level)

* circuit synthesis tools (SIS, Forcage, ASSASSIN, ...)

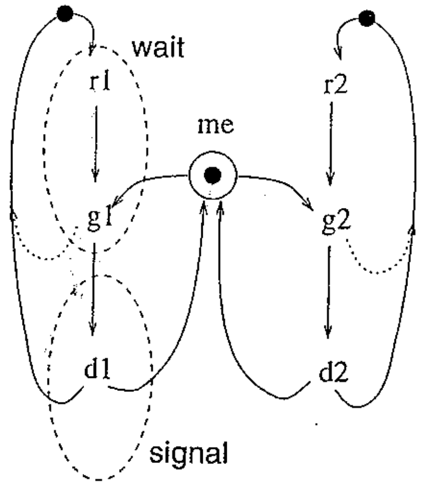
* Petri net and STG level transformations (based on semaphores)

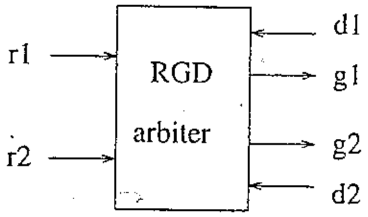
* anything else than Petri nets ???

... human design skills? - no, that's cheating!

arb1.3

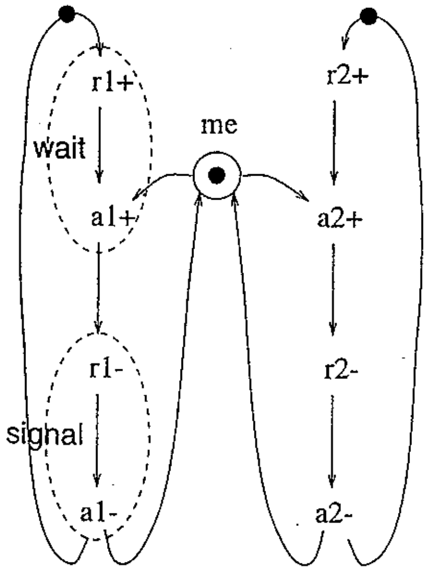
Semaphore Implementation
(2-phase)

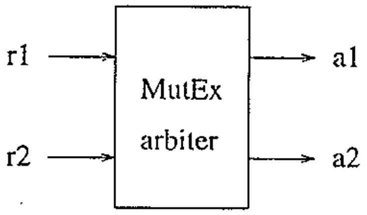




arb2.1

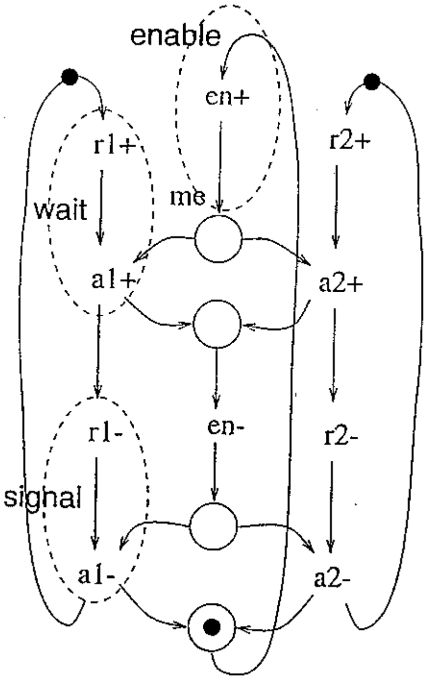
Semaphore Implementation
(4-phase)



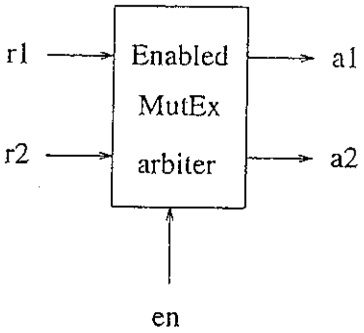


arb2.2

Semaphore Implementation
(4-phase, with Enabling)



arb2.3

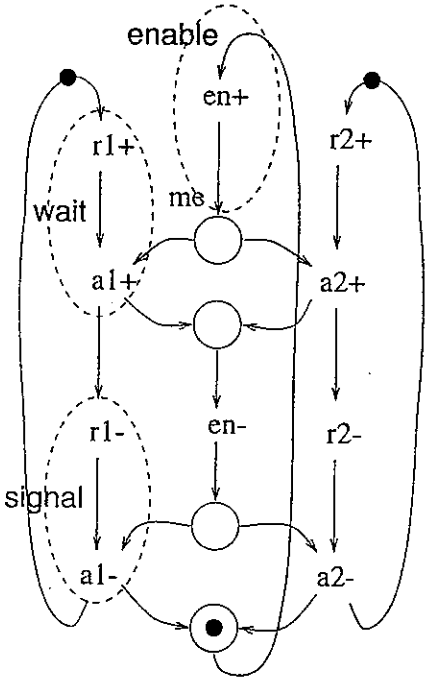


Important Semantical Constraint:

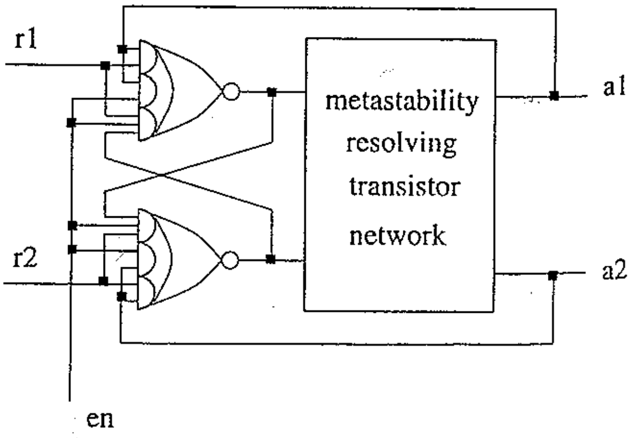
Semaphore can ONLY be enabled when signal EN is high

This is to prevent arbitration "in advance"

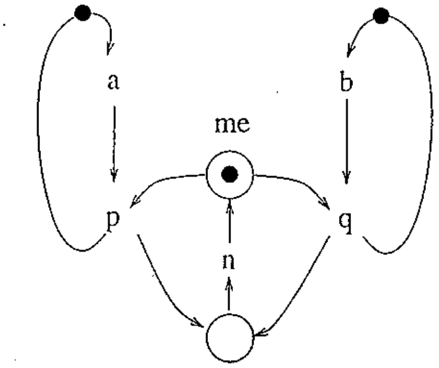
Semaphore Implementation
(4-phase, with Enabling)

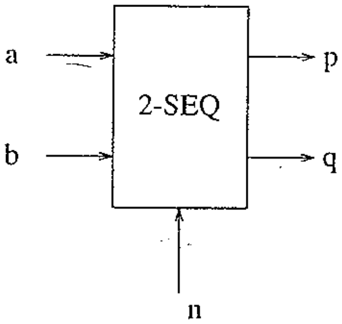


arb2.4



Semaphore Implementation
(2-input Sequencer with "lazy" arbitration)



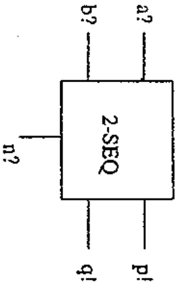


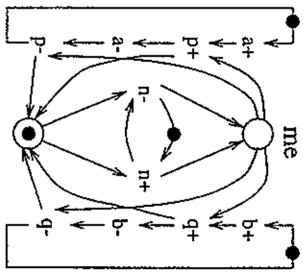
Important Semantic Constraint:

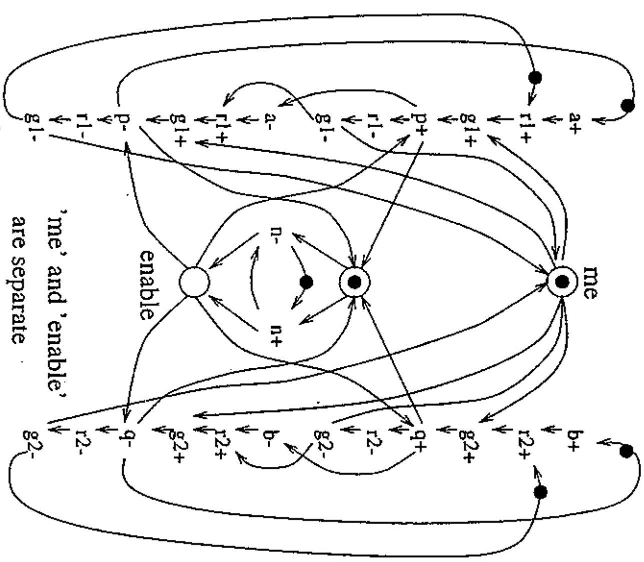
Arbitration can ONLY occur after an event on wire 'n'

arb2.5

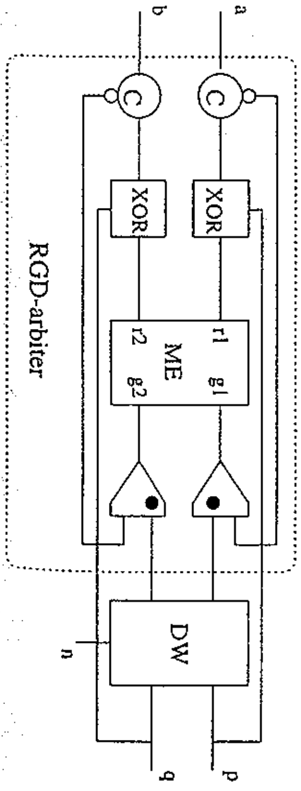
2-input Sequencer with "Eager" Arbitration







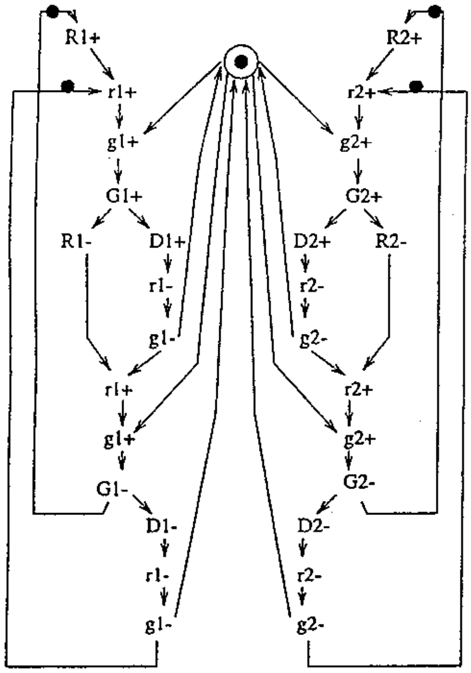
'me' and 'enable' are separate



arb2.6

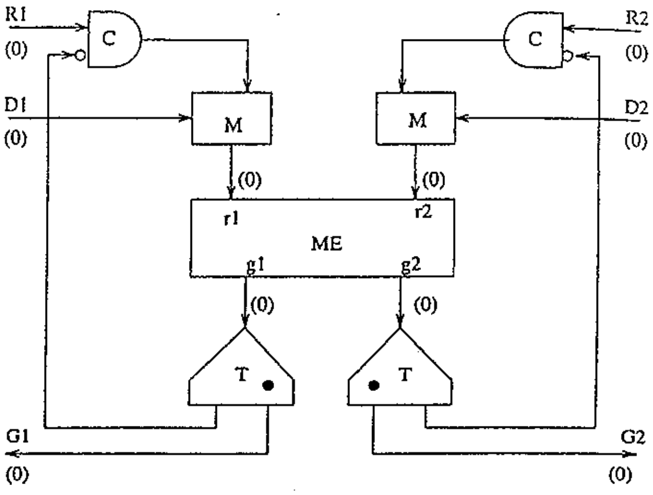
LOGIC SYNTHESIS OF RGD ARBITER

Signal Transition Graph:



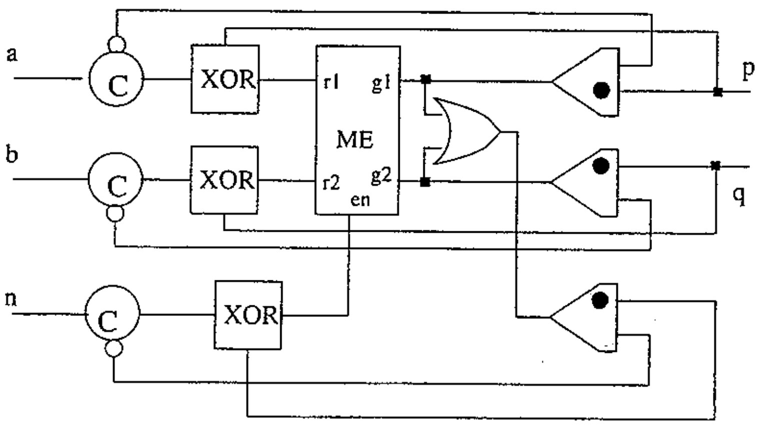
Implemenetation using two-phase signalling components C = C-element
M = Merge (mod 2 sum)
T = Toggle

ME operates in four-phase signalling scheme



It is possible to obtain an alternative implementation, using STG-based synthesis
This implementation would be on logic gates and ME element

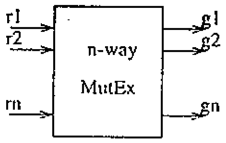
2-input Sequencer (with "lazy" arbitration)



g1+ and g2+ are only produced
when an event on wire n has occurred

MULTI-WAY ARBITERS
(or multi-way binary semaphores)

4-phase:



2-phase:

n-way RGD arbiter

Multi-way Arbiters are usually decomposed so as to use a 2-way arbiter as a building block. Decomposition can use different architectures:

* a (usually, TREE) cascaded ("fixed" token source) arbiter

based on a Tree-Arbiter (TA) cell (e.g. Plummer)

* a RING - distributed ("migrating" token source) arbiter

based on a Token-Ring (TR) cell

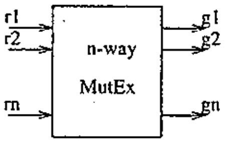
TR with "busy" token (e.g. Brunvand's Token Ring arbiter)

TR with "lazy" (demand-driven) token (e.g. Martin's DME)

- "advance" polling (e.g. Martin's DME)
- polling "with pre-emption" (e.g. , ???)

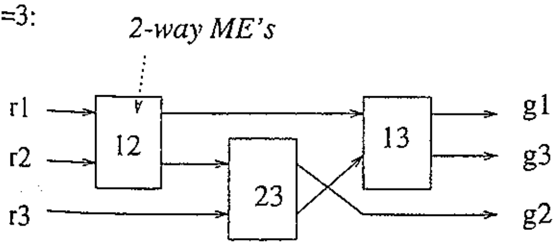
MULTI-WAY ARBITERS

4-phase:

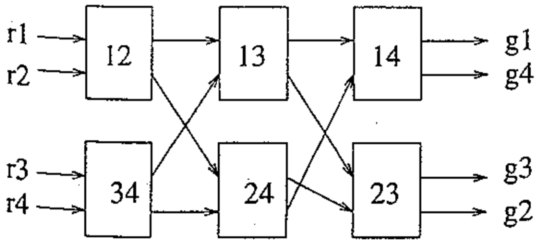


Pairwise mesh interconnection or 2-out-of-n based cascading:

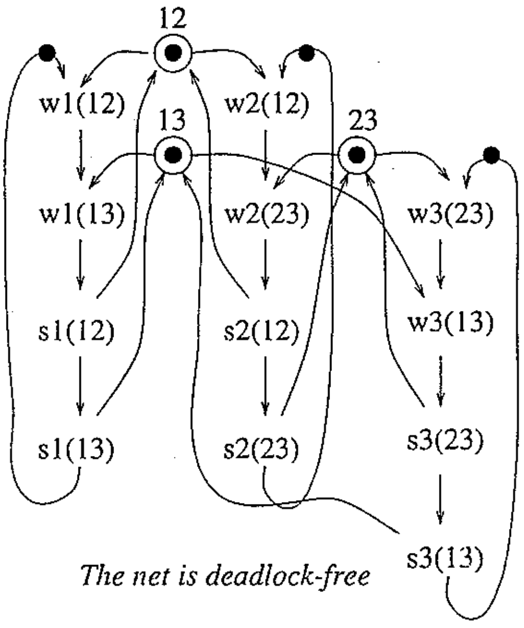
n=3:



n=4:



Petri Net model for n=3

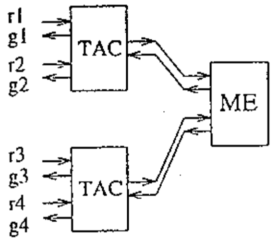


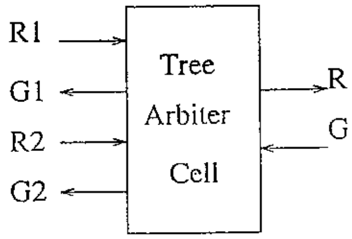
The net is deadlock-free

CASCADED ARBITER

2-phase, TREE-based

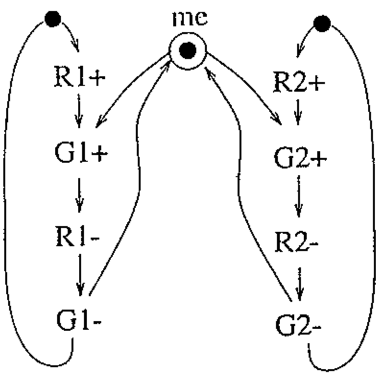
n=4:





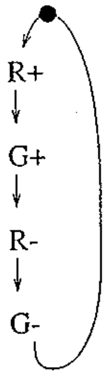
Basic protocols: (1) 4-phase handshake on each (R,G) pair

(2) Mutual Exclusion between G1 and G2



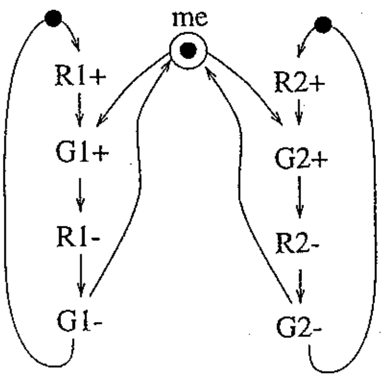
(3)

How to synchronise them?



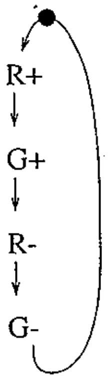
arb3.3

CASCADED ARBITER



(3)

How to synchronise them?



Extra Causality Constraints:

(3.1) R1+ or R2+ precede R+

(3.2) G+ precedes G1+ or G2+

(3.3) R1- or R2- precede R-

ALL MUST BE in the specification

How can we proceed within these (1), (2) and (3.1 -3.3) ?

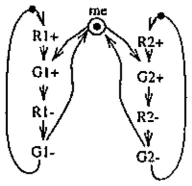
arb3.4

CASCADED ARBITER

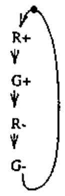
(3.1) R1+ or R2+ precede R+ (3.2) G+ precedes G1+ or G2+

(3.3) R1- or R2- precede R-

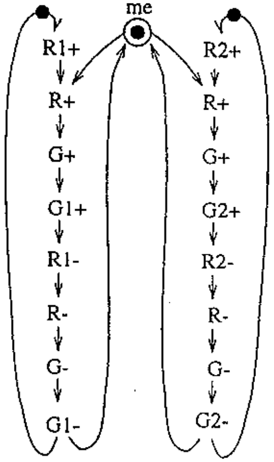
In Seitz' solution:



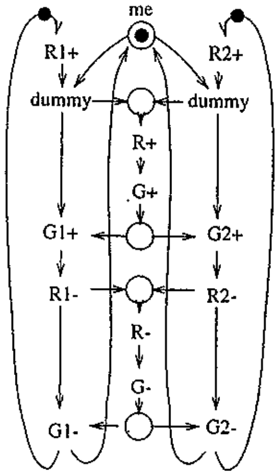
and



are synchronised as follows:



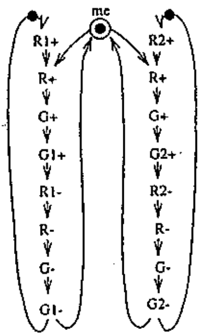
that is



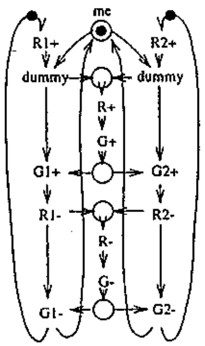
arb3.5

insert q_1, q_2 as inputs!

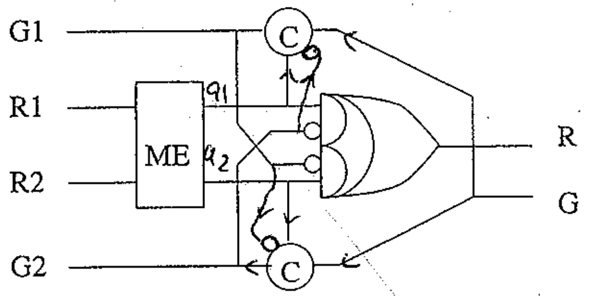
CASCADED ARBITER



that is



which gives the following (intuitively clear) circuit:



existing in different variations of gates

arb3.6

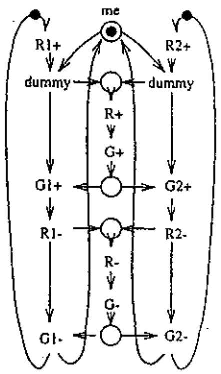
$G1 = q_1 \overline{G2}\, G + G1 \cdot G$

$G2 = q_2 \overline{G1}\, G + G2 \cdot G$

CASCADED ARBITER

(3.1) R1+ or R2+ precede R+ (3.2) G+ precedes G1+ or G2+

(3.3) R1- or R2- precede R-



Looking for better performance:

(1) Why does the MutEx-ing need to be before R+ ?

(2) Why do G1- and G2- need to be after G- ?

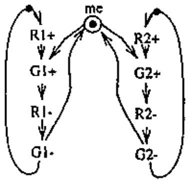
arb3.7

CASCADED ARBITER

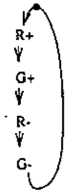
(3.1) R1+ or R2+ precede R+ (3.2) G+ precedes G1+ or G2+

(3.3) R1- or R2- precede R-

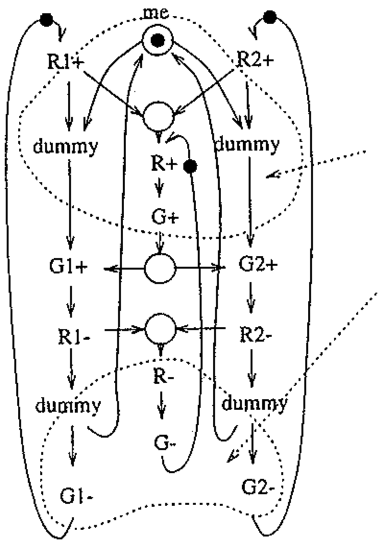
So,



and



may be synchronised as follows:



Where:

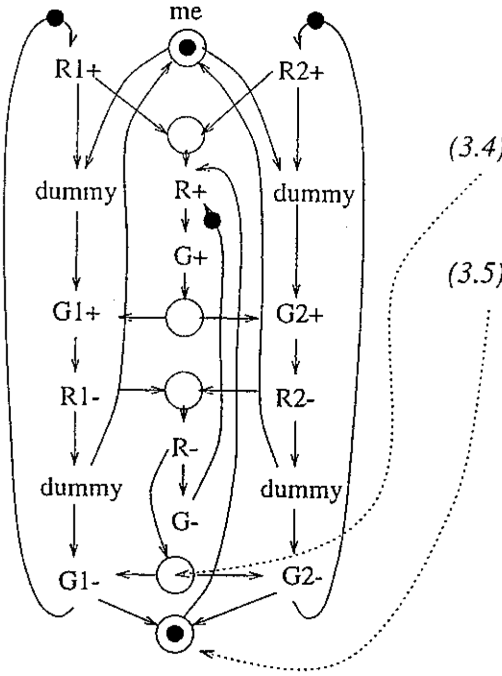
(1) MutEx resolution is in parallel with request propagation

(2) Release of forward and backward handshakes is also concurrent

arb3.8

CASCADED ARBITER

To avoid State Coding problem complexity, we trade-off performance for simplicity - by "slightly" constraining concurrency :



(3.4) G1- or G2- is preceded by R-

(3.5) The next occurrence of R+ can only be when G1 and G2 are released.

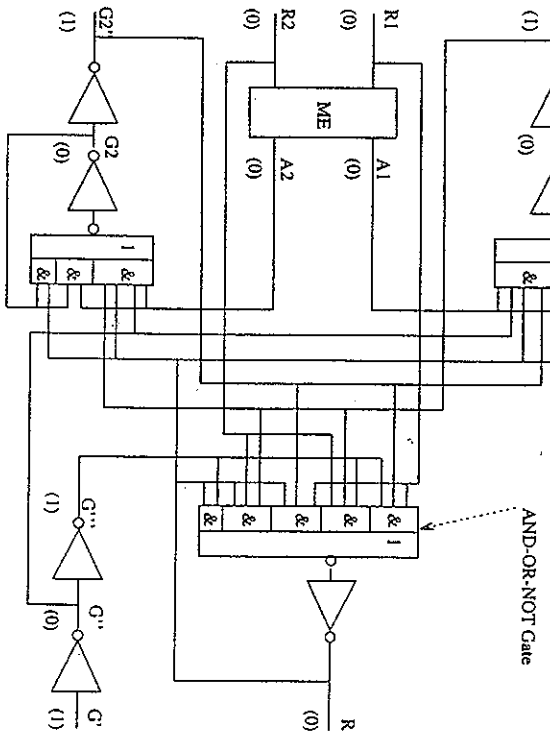
Now dummies can be replaced by semaphore actions implemented by a 2-way ME

arb3.9

arb3.10

CASCADED "LOW-LATENCY" ARBITER

Obtained with SIS synthesis tools:



Alternative solution (M. Josephs and J. Yantchev):

Separate two "speeding" concerns:

"Fast" request propagation -> Tree-Arbiter Element

"Eager" acknowledgement -> Quick-Return-Linkage

CASCADED ARBITER

Are there any more "opportunities" left in its original specification ?

FAIRNESS ? ...

Do we always need to release the Forward Handshake linkage ?

A story about "unfair" card playing ...

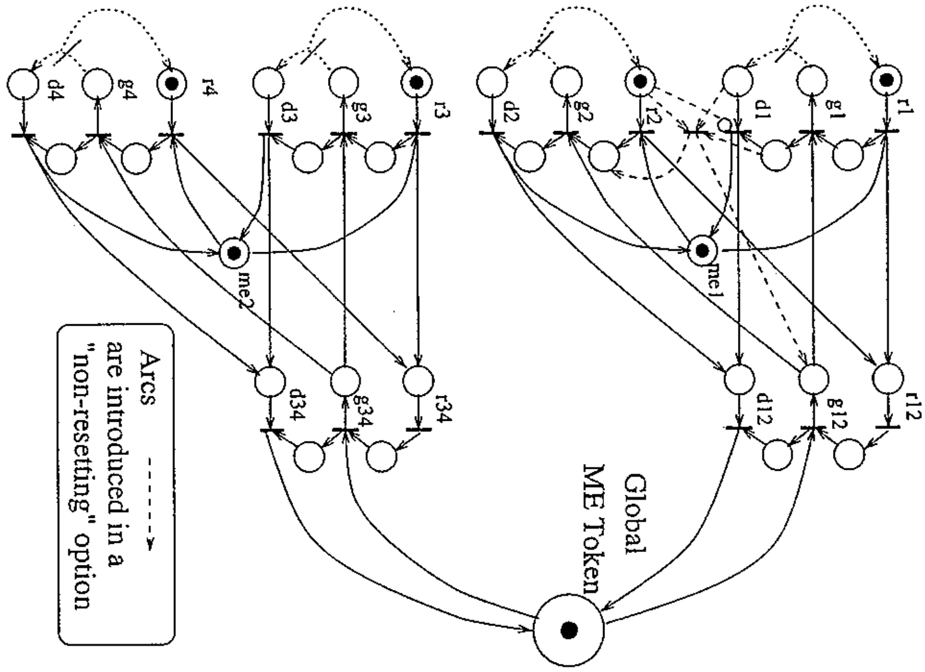
or how to help your "friend" by "short-cutting" the Request-Grant path

arb3.11

arb3.12

CASCADED ARBITER

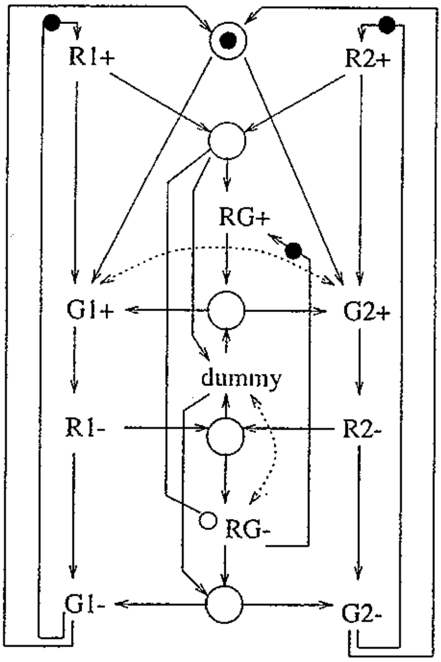
(Token Propagation Model)



CASCADED ARBITER

NON-RESETING ARBITER

Inhibitor net Model



RG stands for the (R,G)
(R+ -> G+) = RG+
(R- -> G-) = RG-

If R2+ arrives before R-handshake RG is not released

Arbitration points:

(1) between R1+ and R2+
(2) between R1+ and R2-
(3) between R2+ and R1-

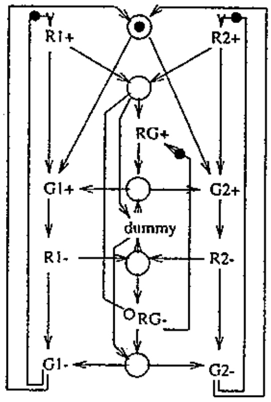
Conflicts are denoted by

arb3.13

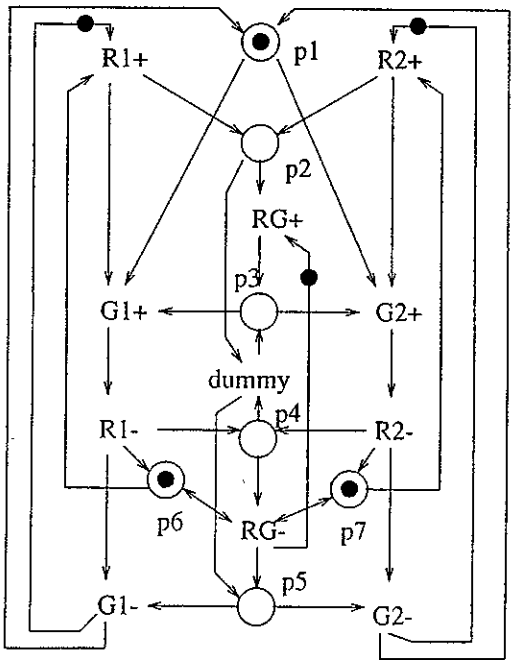
CASCADED ARBITER

NON-RESETING ARBITER

Inhibitor net Model



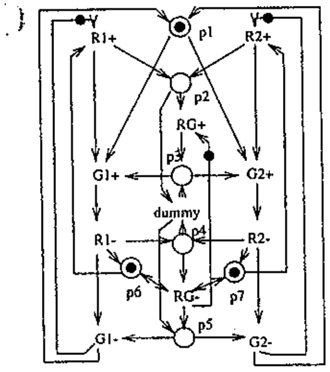
Equivalent ordinary Petri net model



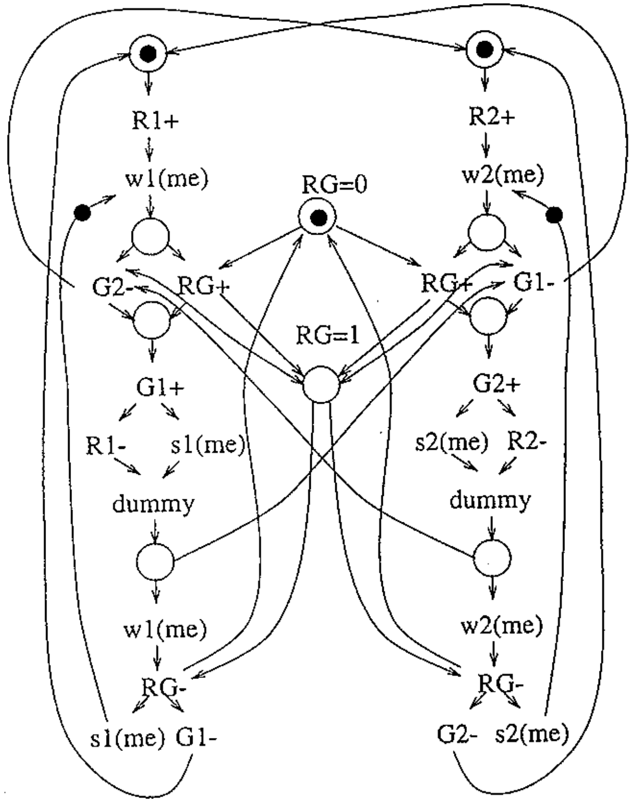
arb3.14

CASCADED ARBITER
NON-RESETING ARBITER

Initial Spec:



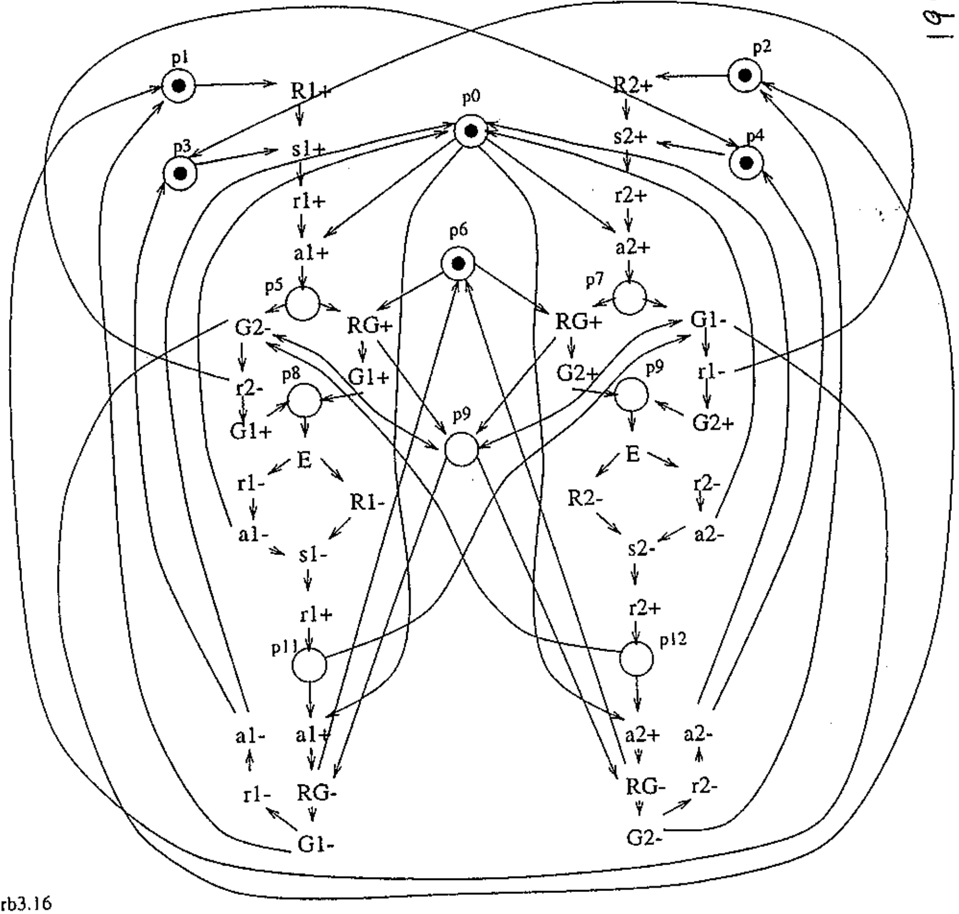
Refinement with single ME (sempahore)
(non-low-latency option)



arb3.15

CASCADED ARBITER
NON-RESETING ARBITER

STG version for synthesis:

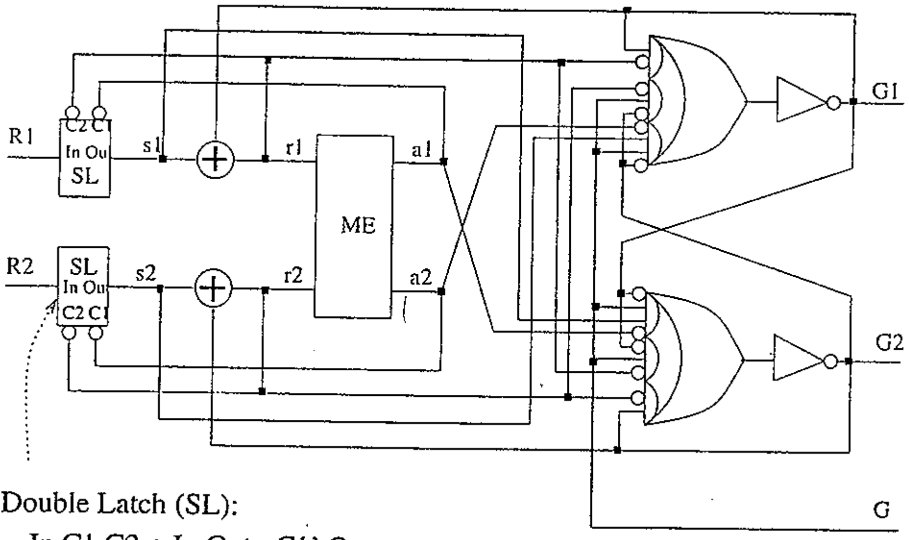


arb3.16

192

CASCADED ARBITER

NON-RESETING ARBITER



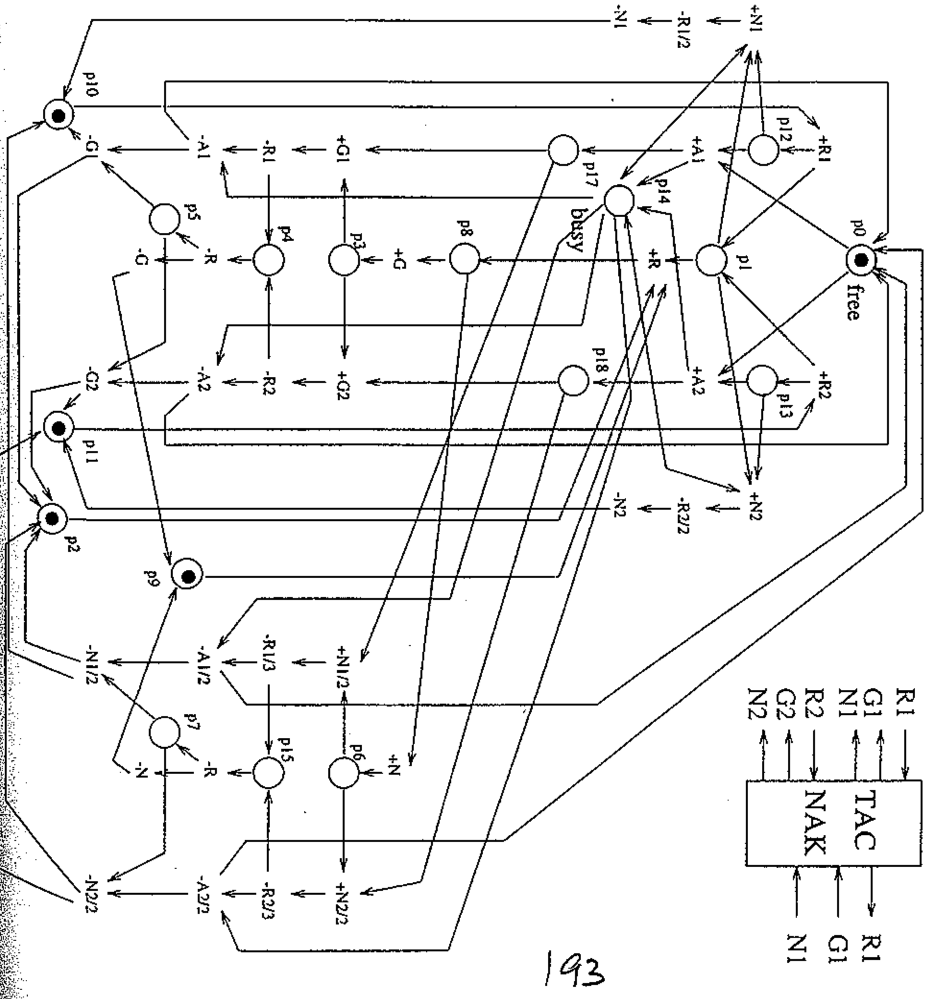
Set-Double Latch (SL):
Out = In C1 C2 + In Out +C1' Out

Implementation for R:
R= a1 s1 + a2 s2 + R(a1' a2' + a1' s2 + a2' s1)

arb3.17

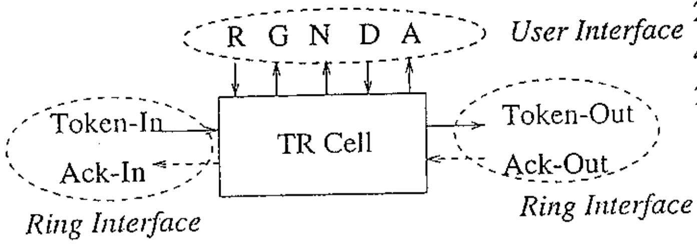
CASCADED ARBITER

Low Latency NAK-ing Arbiter STG



RING-BASED ARBITERS

Token Ring with "busy" token:



4-phase (RG)
2-phase (RGD)
4-phase nak-ing (RGN)
2-phase nak-ing (RGDNA)

Types:	User Interface	Ring Interface	
RG-4/P	4-phase (RG)	4-phase/Propagate	*Kishinevsky, Varshavsky, 86*
RGD-2/HS	2-phase (RGD)	2-phase/Handshake	
RGDNA-2/P	2-phase (RGD)	2-phase/Propagate	*Ebergen et al.,93*
RG-2/P	4-phase (RG)	2-phase/Propagate	*??? Brunvand, 87* *Gopalakrishnan,94* *(used 'stoppable'token)*
...	...	...	
RG-4/HS	4-phase (RG)	4-phase/Handshake	

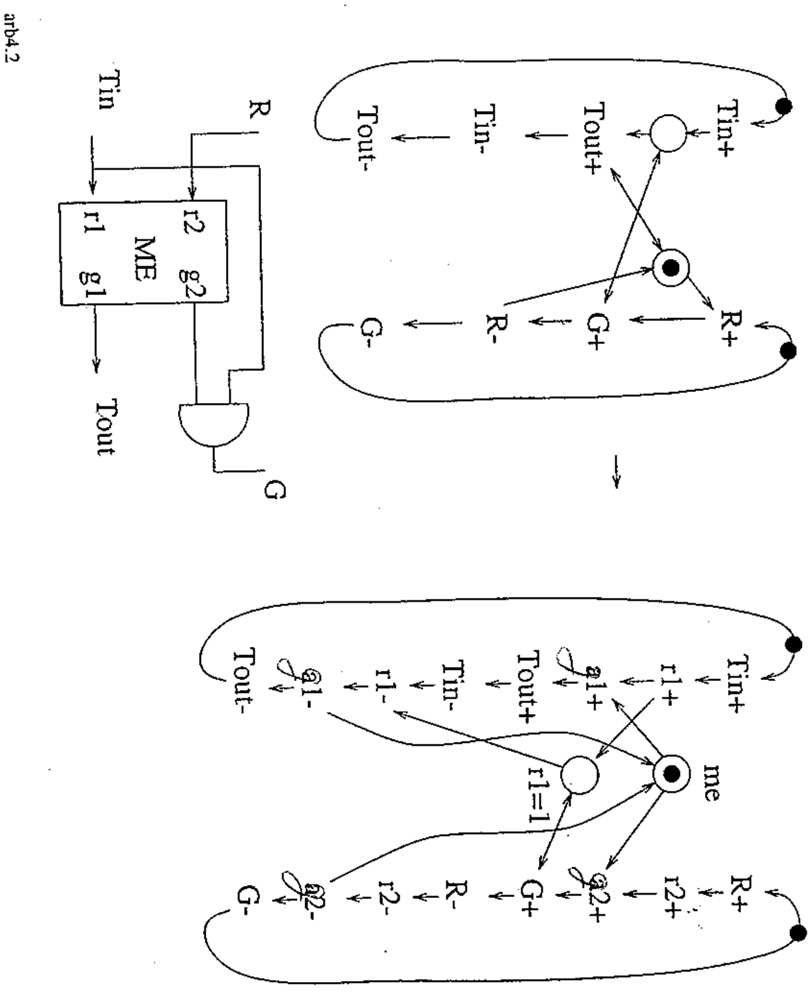
arb4.1

RING-BASED ARBITERS

E.g.: RG-4/P 4-phase (RG) *User Interface*
4-phase/Propagate *Ring Interface*

STG specification:

STG for synthesis:

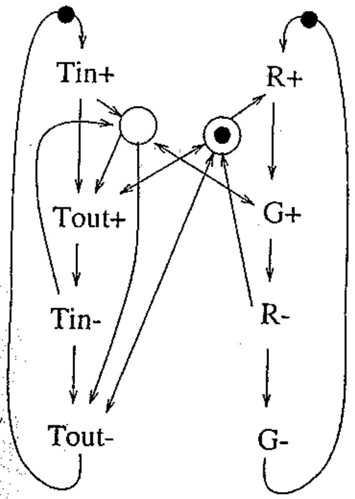


RING-BASED ARBITERS

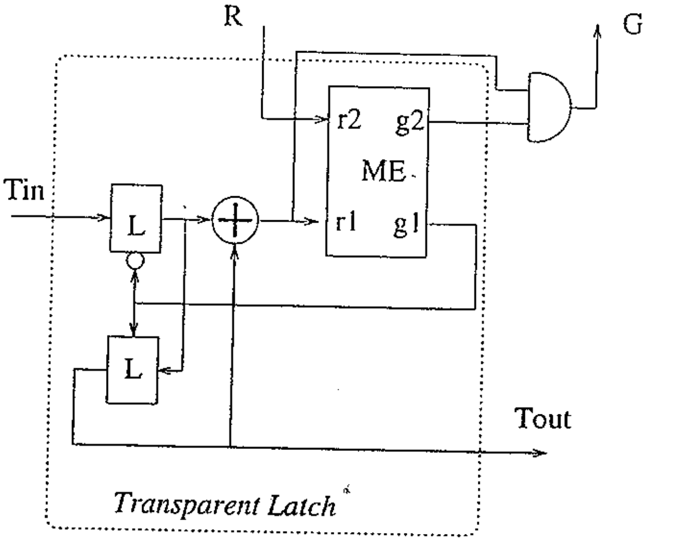
E.g.: RG-2/P — 4-phase (RG) — *User Interface*

2-phase/Propagate — *Ring Interface*

STG specification:



Nothing Surprising ...



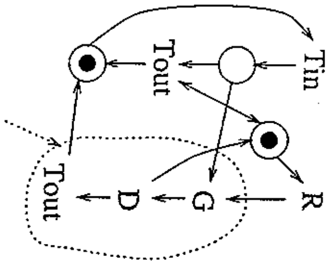
arb4.3

RING-BASED ARBITERS

E.g.: RGD-2/P — *User Interface:* 2-phase (RGD)

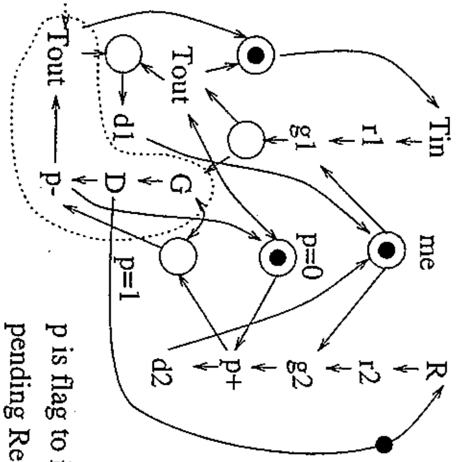
Ring Interface: 2-phase/Propagate

PN specification:

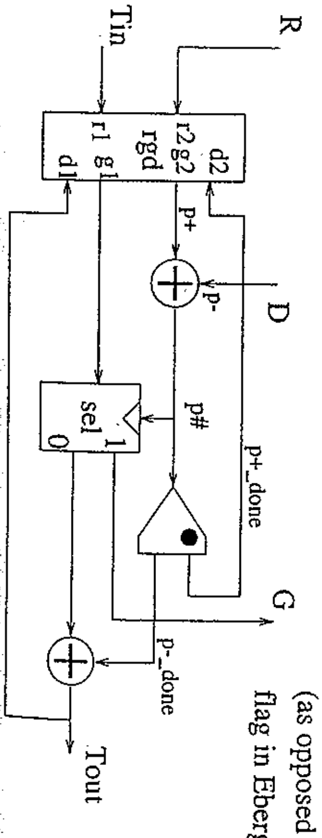


Fairness option:
Token is always pushed out!

PN for synthesis



p is flag to indicate
pending Request
(as opposed to token
flag in Ebergen et al)

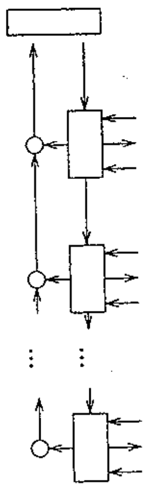


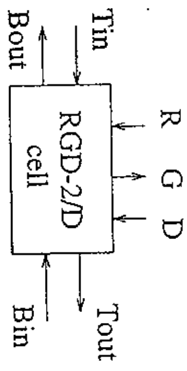
arb4.4

RING-BASED ARBITERS

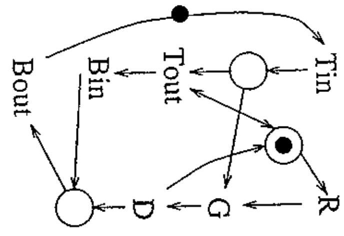
E.g.: RGD-2/D   *User Interface:* 2-phase (RGD)
Ring Interface: 2-phase/Daisy-chain

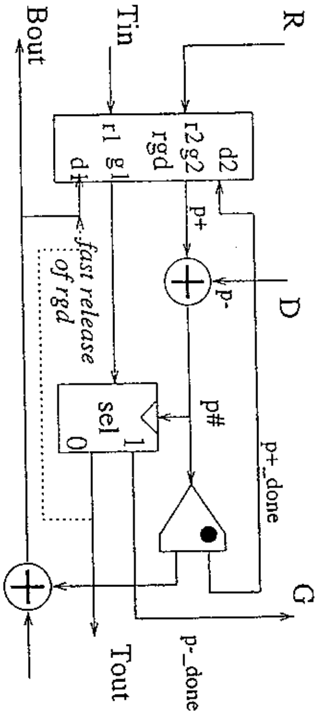
Daisy-chain structure





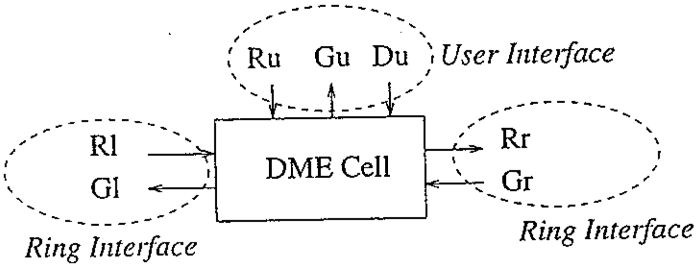
PN specification:





RING-BASED ARBITERS

Token Ring with "lazy" token (or DME)

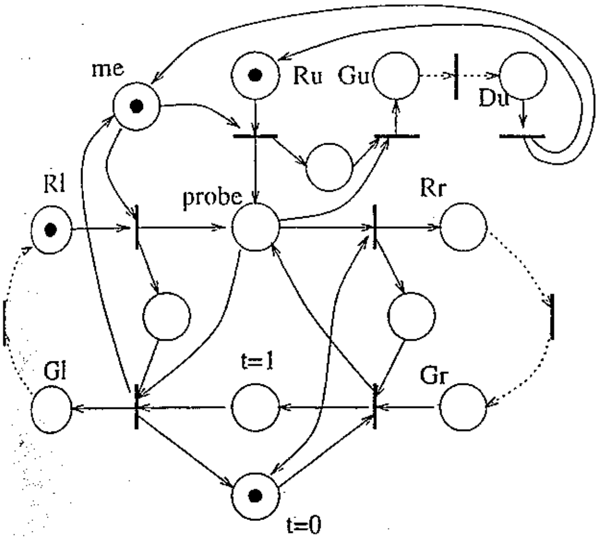


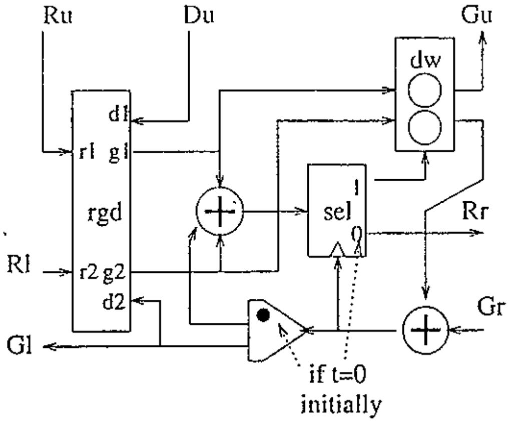
2-phase (RGD)
4-phase (RG)

Types:	User Interface	Ring Interface	
RG-4/HS	4-phase (RG)	4-phase/Handshake	*Martin,85*
RGD-2/HS	2-phase (RGD)	2-phase/Handshake	
...	...	...	

RING-BASED ARBITERS

E.G. DME RGD-2/HS — User Interface: 2-phase (RGD)
Ring Interface: 2-phase Handshake



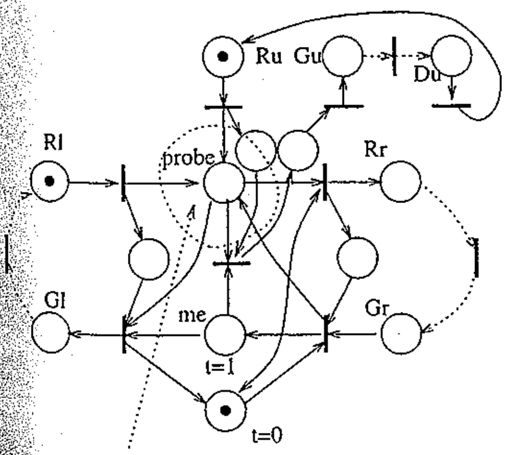


arb4.6

RING-BASED ARBITERS

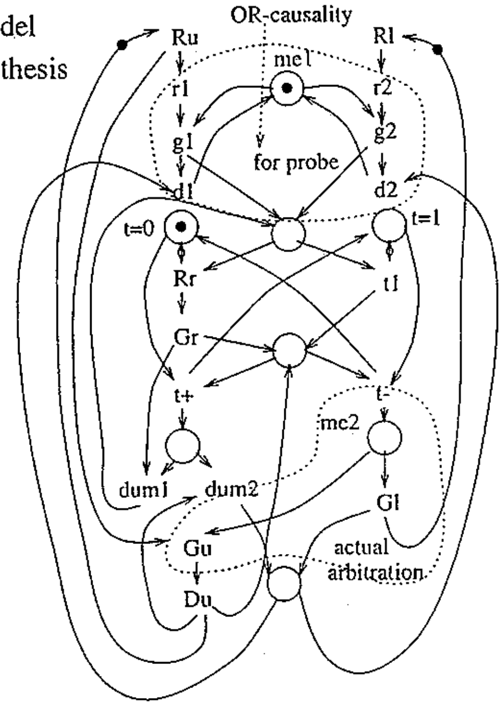
E.G. DME RGD-2/HS *with postponed decision* — User Interface: 2-phase (RGD)
Ring Interface: 2-phase Handshake

Initial PN model



OR-causal probe
(2-safe place!)

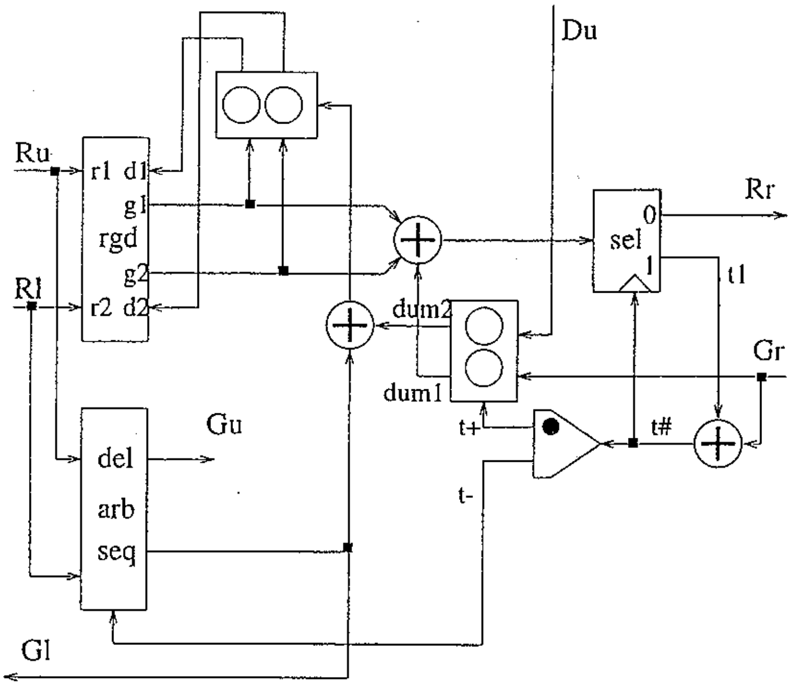
PN model
for synthesis



arb4.8

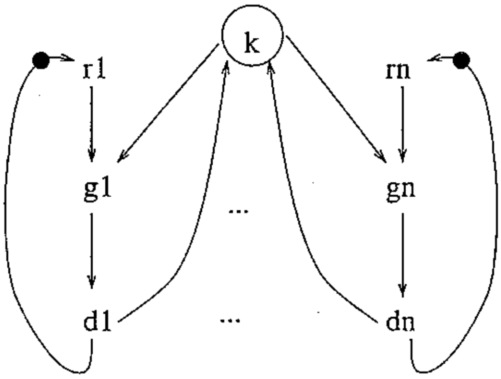
RING-BASED ARBITERS

E.G. DME RGD-2/HS — User Interface: 2-phase (RGD)
with postponed decision — Ring Interface: 2-phase Handshake



arb4.9

MULTI-TOKEN ARBITERS



Can a multi-resource multi-way arbiter be built using 2-way MutEx'es ?

Two potential approaches:

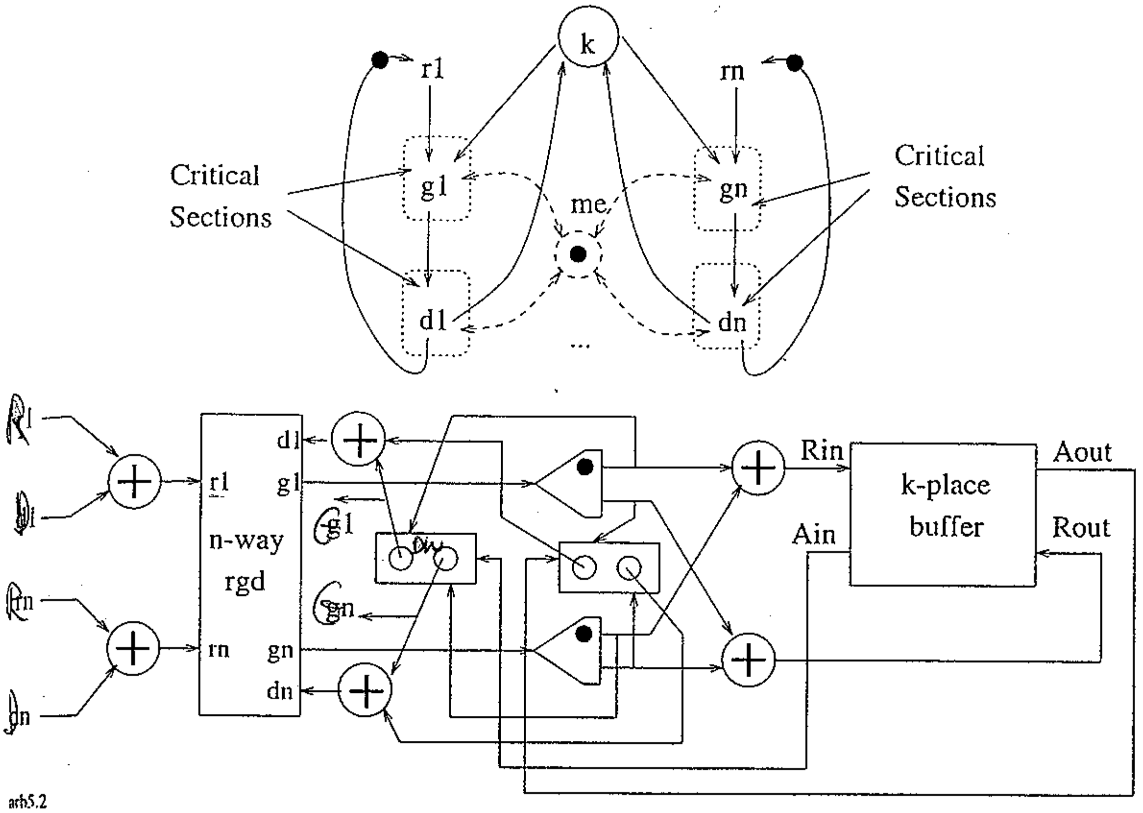
* The "Fixed" Token Source Case

* The "Migrating" Token Source Case

arb5.1

MULTI-TOKEN ARBITERS

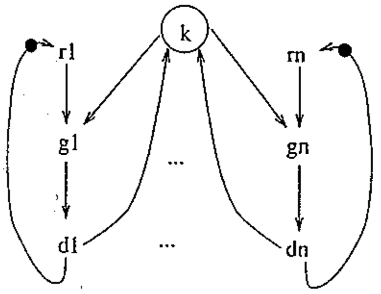
"Fixed" Token Source



arb5.2

MULTI-TOKEN ARBITERS

"Migrating" Token Source



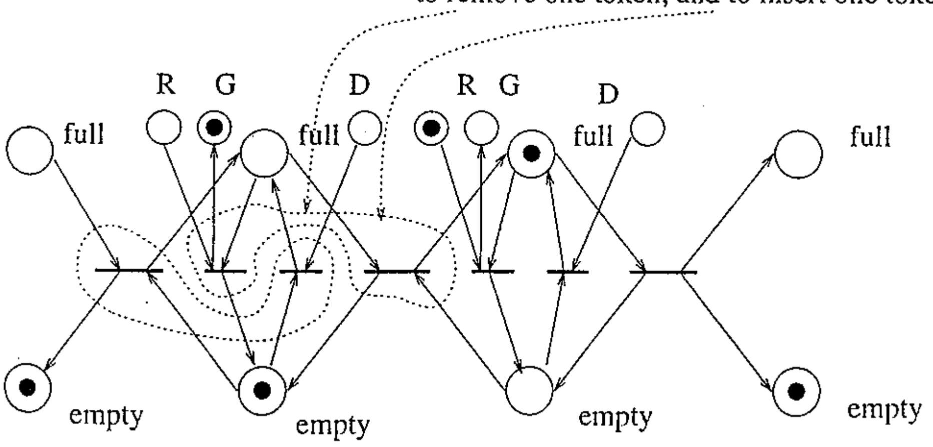
Tokens circulate in a Ring Pipeline

At any time a token can be removed by any requestor

(1) Blocking case - all subsequent tokens wait until the requestor has finished

(2) Non-blocking case - all other tokens continue to pass through

Requires two arbitration points in each cell: to remove one token; and to insert one token

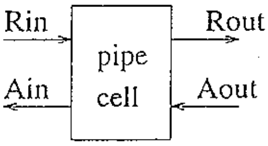


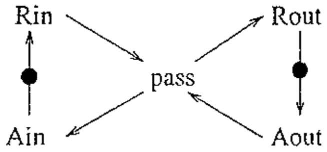
arb5.3

MULTI-TOKEN ARBITERS

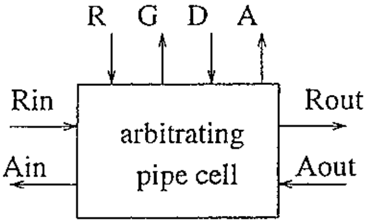
"Migrating" Token Source

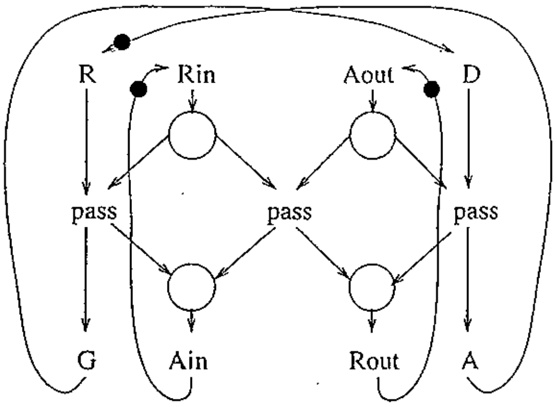
Ordinary Pipeline Cell:





Arbitrating Pipeline Cell:



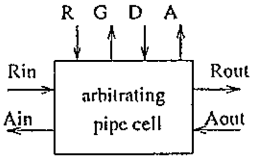


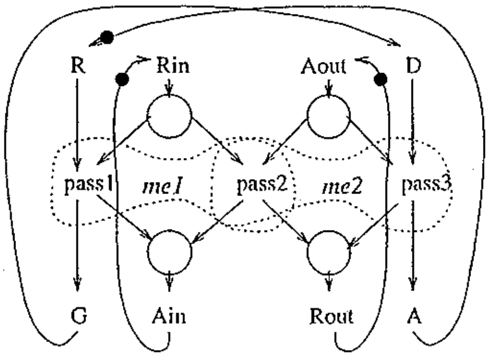
arb5.4

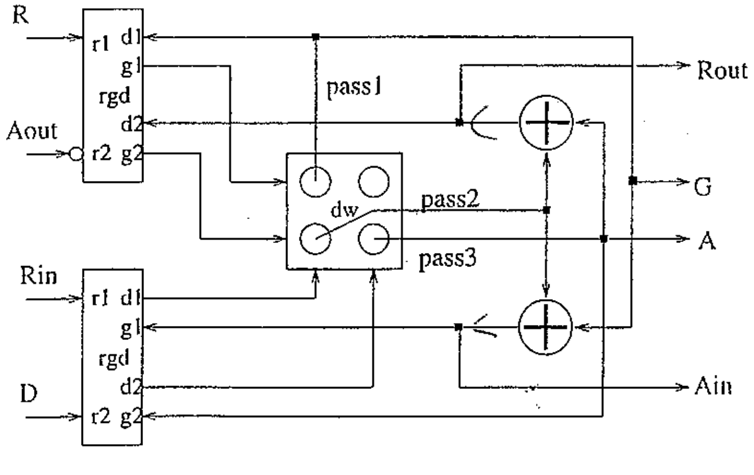
MULTI-TOKEN ARBITERS

"Migrating" Token Source

Arbitrating Pipeline Cell:





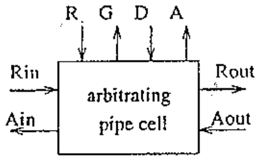


arb5.5

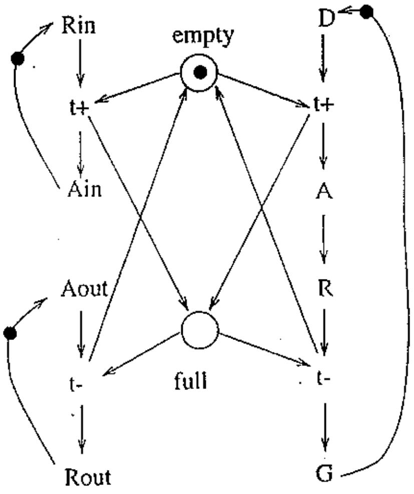
MULTI-TOKEN ARBITERS

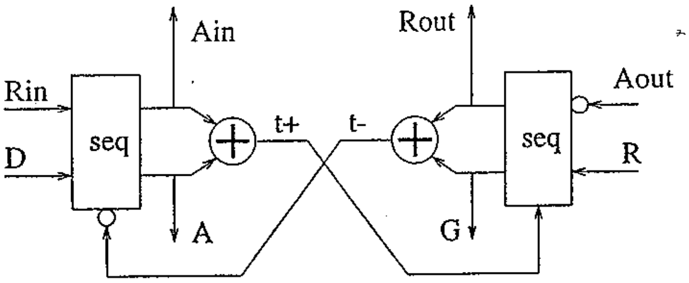
"Migrating" Token Source

Arbitrating Pipeline Cell:



Alternative version with internal memory (Empty/Full)



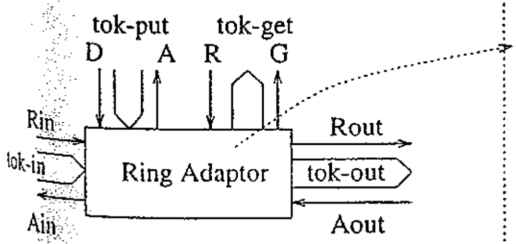


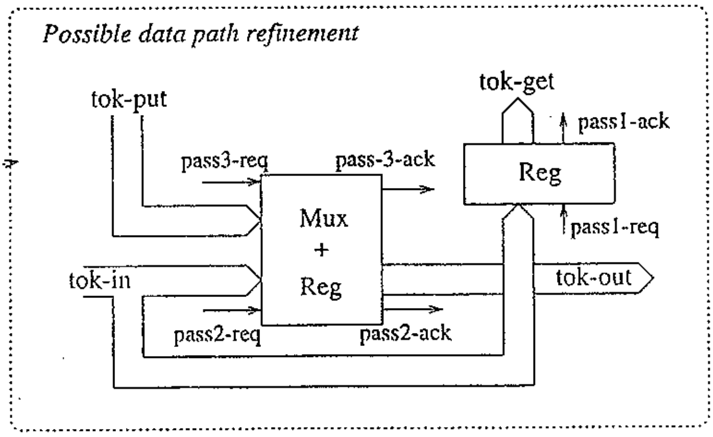
arb5.6

MULTI-TOKEN ARBITERS

Possible Applications:

* Multi-Token Ring Data Channel





* Multi-Token Ring Data Flow Processor

... rough ideas ???

arb5.7