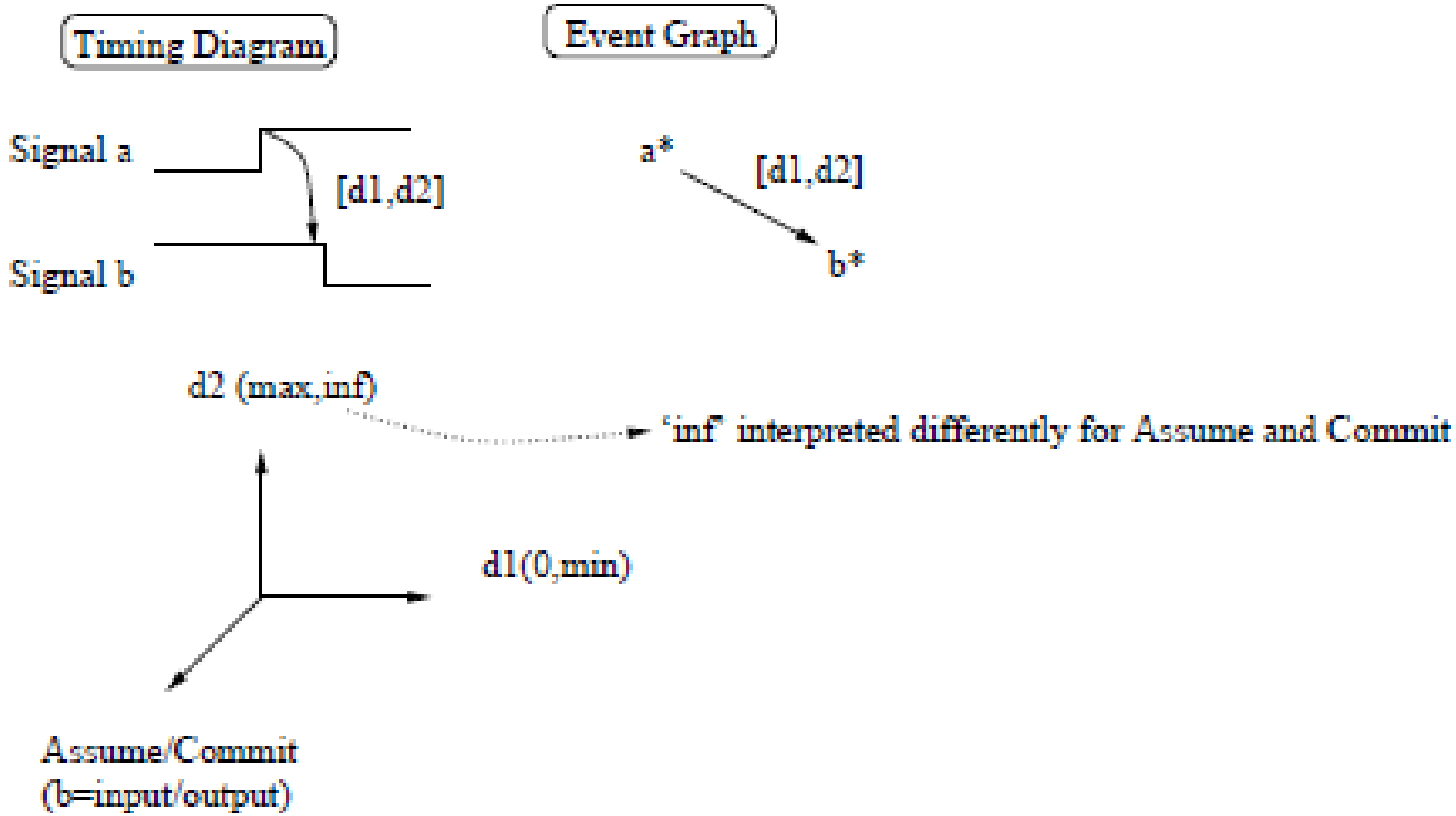
Designing asynchronous circuits with timing conditions

Vision statement for possible CAD development under Workcraft

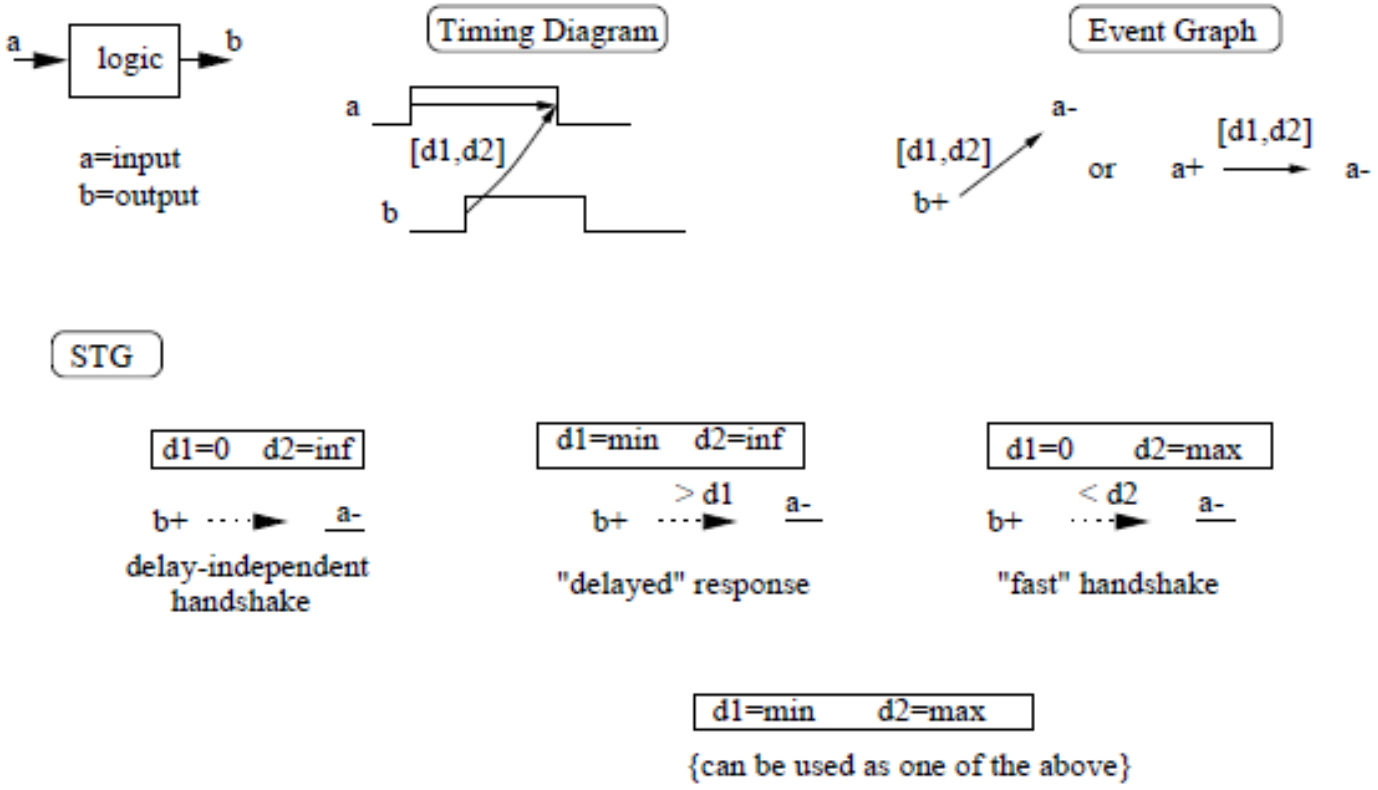
Original document:

A. Yakovlev, Synthesis from timing diagrams: rough notes and examples, Tech Memo, March 7, 1998; written on the basis of discussions with Ed Cerny and Luciano Lavagno

Basic classification of timing conditions



STG interpretation of Assume conditions



(1) Case $d1 = 0, d2 = \infty$

Corresponds to delay-independent handshake

(2) Case $d1 = min > 0, d2 = \infty$

Corresponds to delayed (by min value) response from the environment

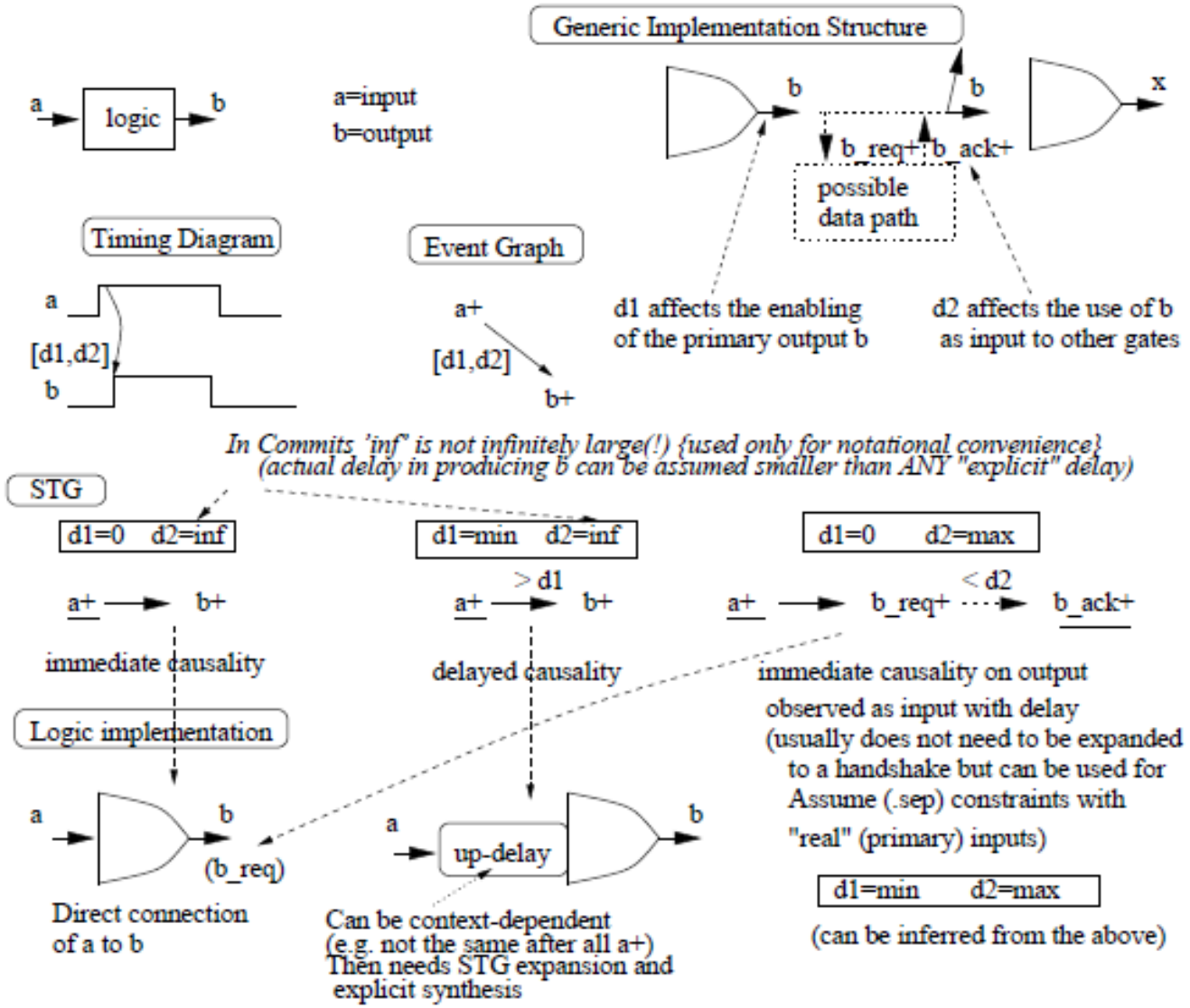
(3) Case $d1 = 0, d2 = max < \infty$

Corresponds to bounded response, e.g. from clock signal

(4) Case $d1 = min > 0, d2 = max < \infty$

Combination of (2) and (3), i.e. imaginary handshake with input clock

STG interpretation of commits



(1) Case $d1 = 0, d2 = \infty$

Corresponds to immediate causality between input a and the logic producing output b

(2) Case $d1 = min > 0, d2 = \infty$

Corresponds to delayed causality, e.g. for producing skew compensation (bundled data strobe or setup)

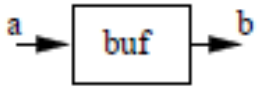
(3) Case $d1 = 0, d2 = max < \infty$

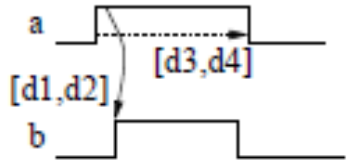
Corresponds to immediate causality but observing of the output b may be after some (up to max) delay

(4) Case $d1 = min > 0, d2 = max < \infty$

Combination of (2) and (3)

Simple buffer example





Initial Constraints:

Commit: [d1,d2] a+ → b+

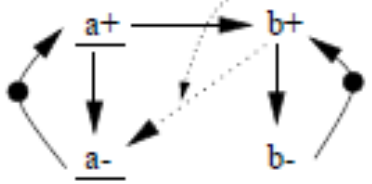
Assume: [d3,d4] a+ → a-

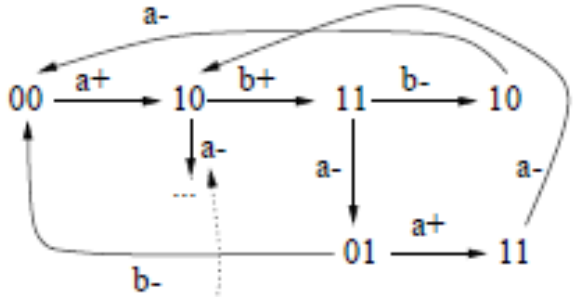
Let d2 < d3:

Hence (via Timing Analysis) we have a new Assume .sep b+ < a- (b+ before a-)

Since this .sep affects input (a-) we introduce an explicit arc

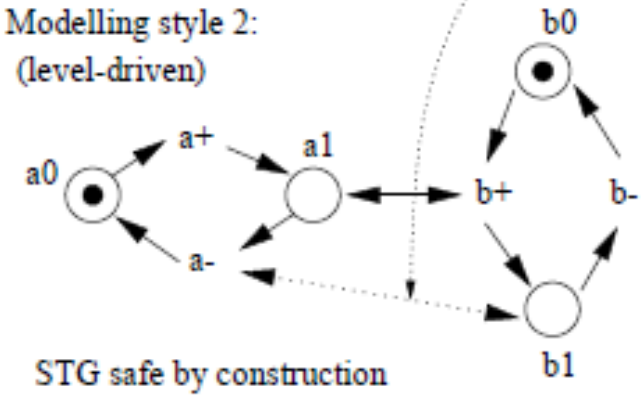
Modelling style 1:
(event-driven)





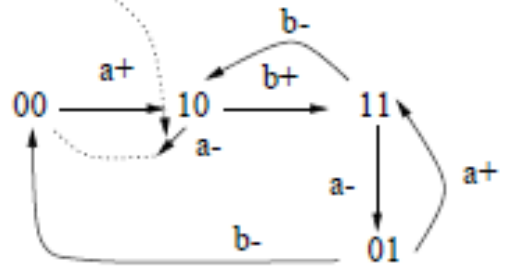
STG output-persistent by construction
but inbounded without "Assume"

Modelling style 2:
(level-driven)

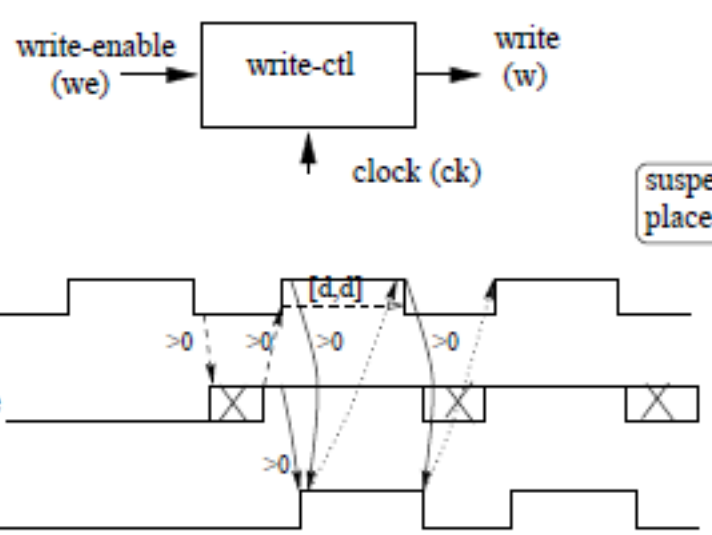


STG safe by construction
but not out-persistent without "Assume"

This transition is impossible due to "Assume"



Clocked write controller

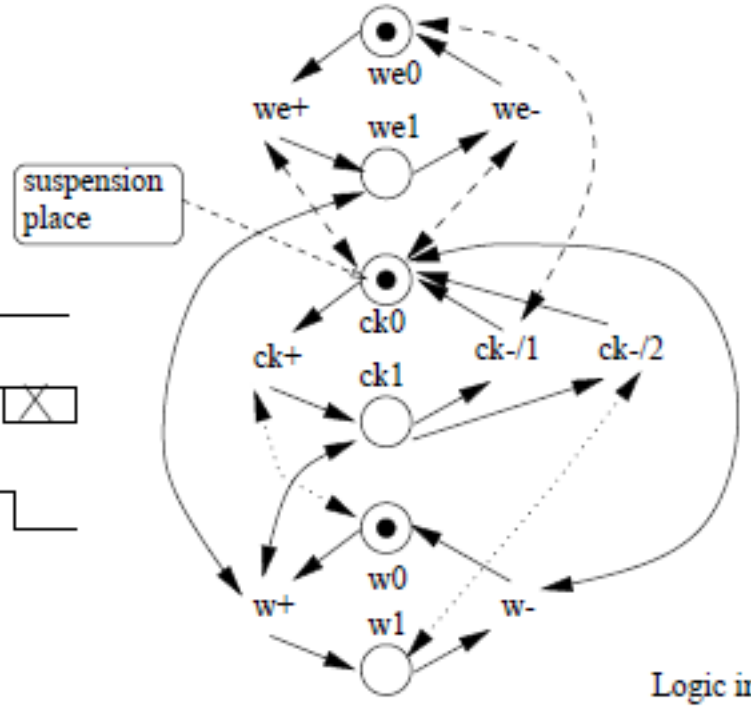


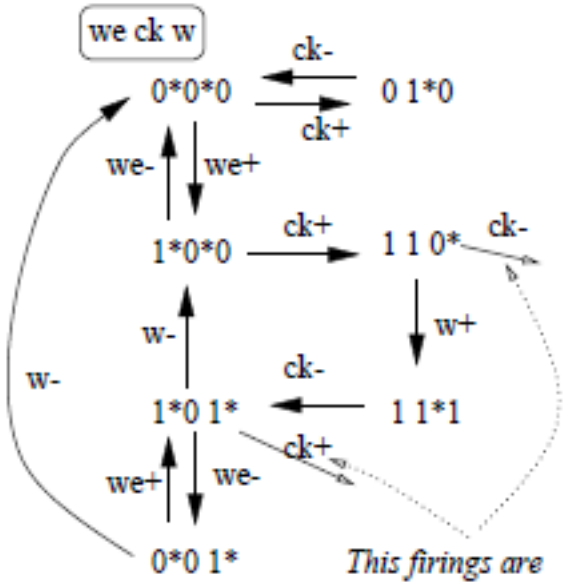
Constraints:

causal commit

order assume input-input

order assume output-input





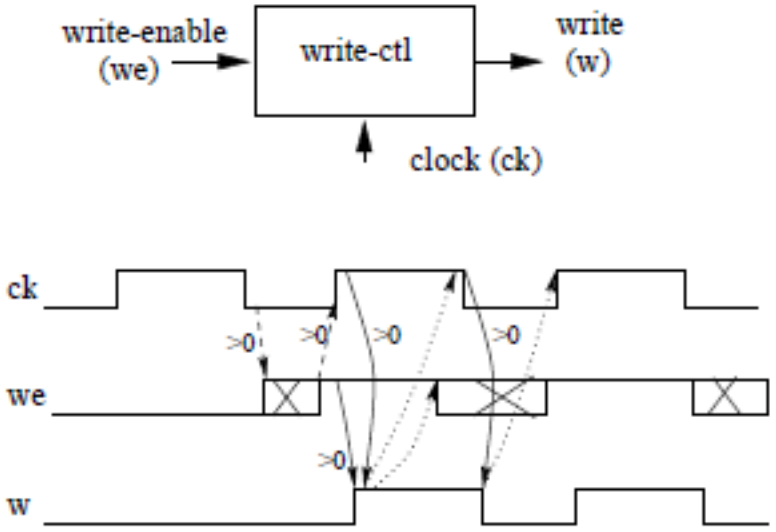
This firings are impossible due to Assumes on clock

Logic implementation:

w = ck we

Very simple clocked write controller

Here we allow ‘we’ to resume its change after ‘w’ goes high

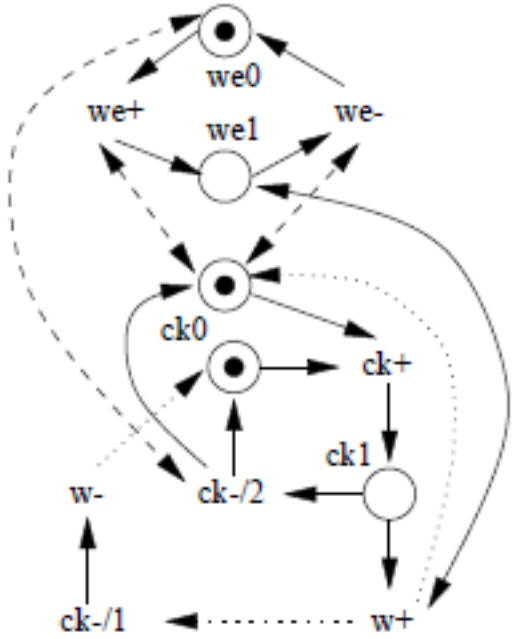


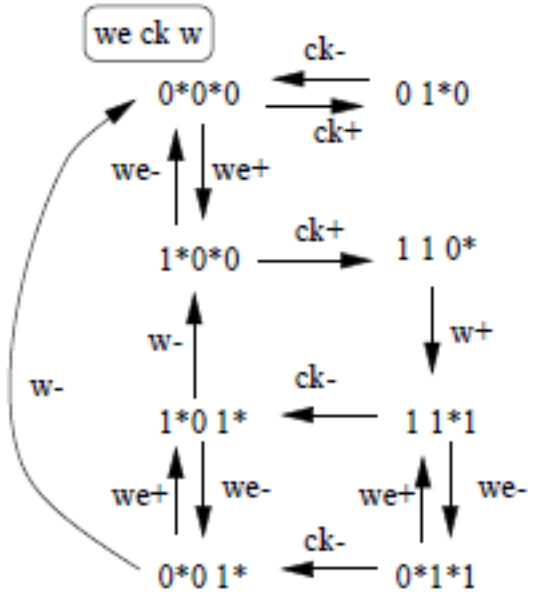
Constraints:

causal commit

order assume input-input

order assume output-input



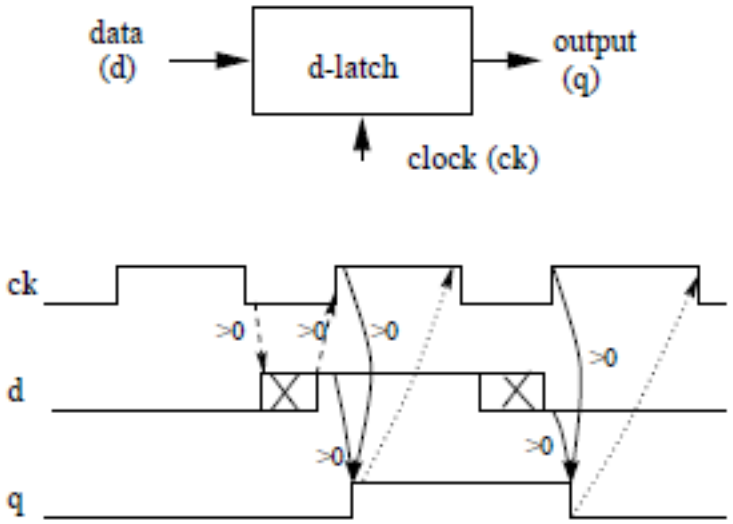


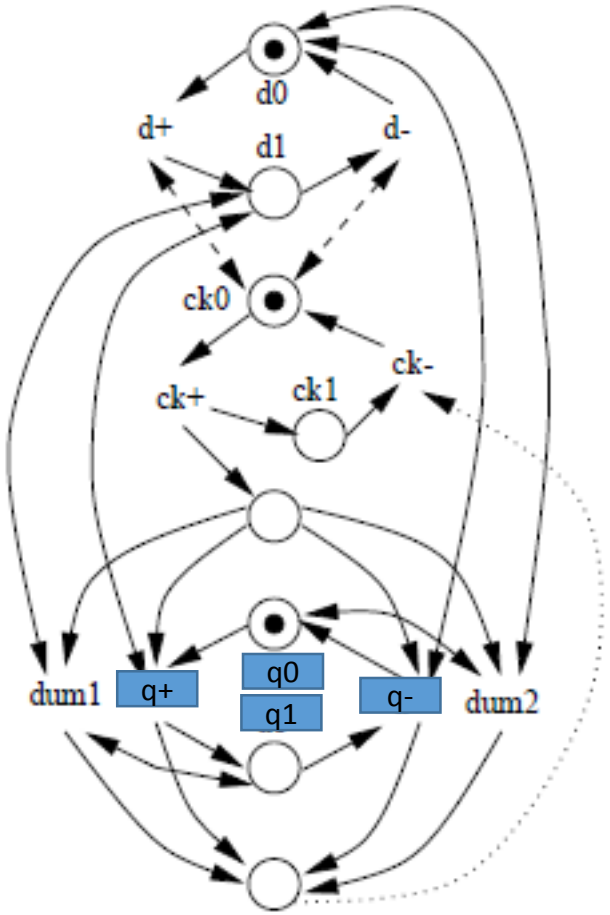
Logic implementation:

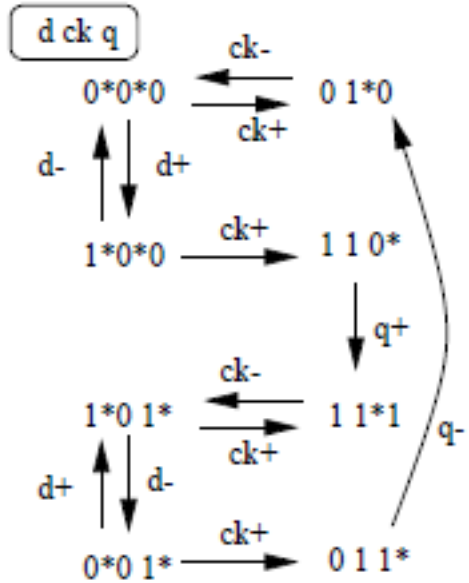
w = ck we + ck w

w = ck we if, additionally, .sep ck-/1 < we- is applied

Clocked D-latch



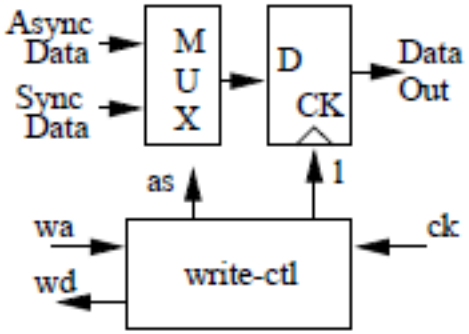




Logic implementation:

$q = ck\ d + ck'\ q$

Dual (Sync/Async) mode write ctl with external choice



Possible constraints in async part:
commit:

C1: 1+/1 -> 1-/1 (> min)

C2: wa- -> 1-/1

C3: 1-/1 -> wd-

to help csc or logic

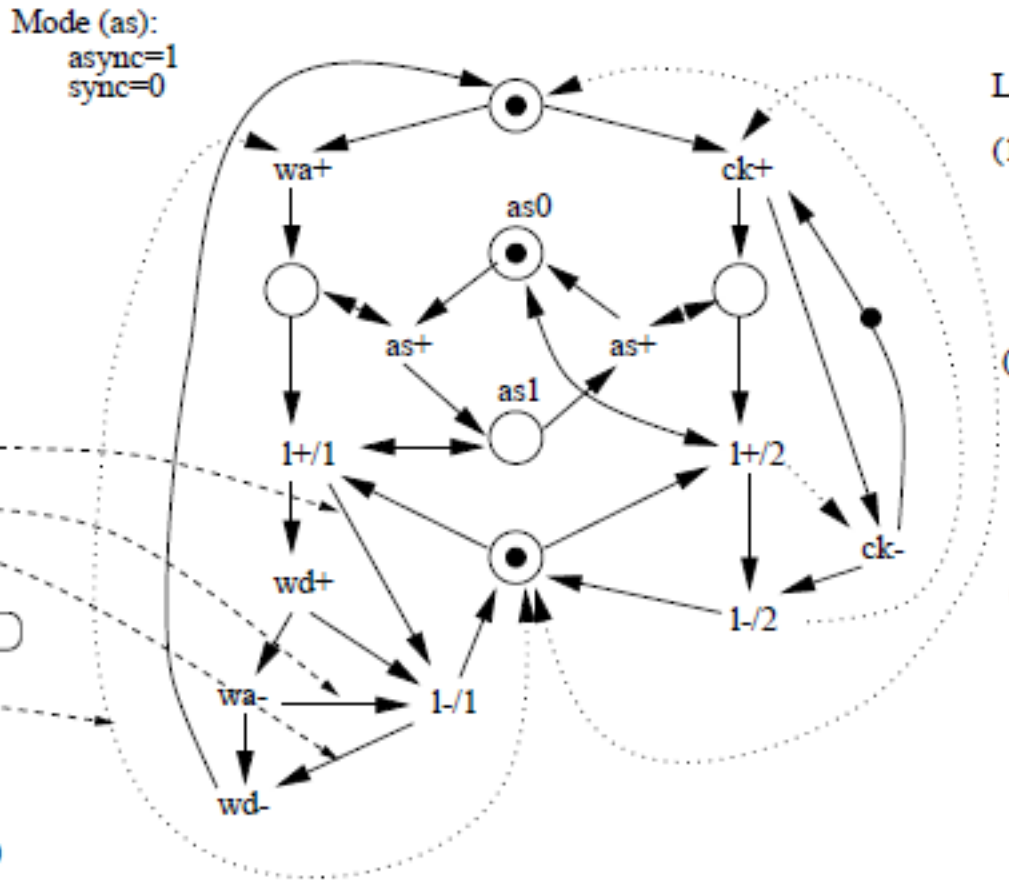
assume:

A1: 1-/1 -> wa+

(a bit milder than C3)

also: commit on set-up delay

C4: before 1+/1 and 1+/2 (???)



Logic implementations:

(1) no constraints in async part:

wd = csc0 (wa 1 + wd)
as = as ck' + wa
1 = csc0 (wd'(as +1) + ck)
csc0 = 1'(as'+wa) + csc0 (ck+wa)

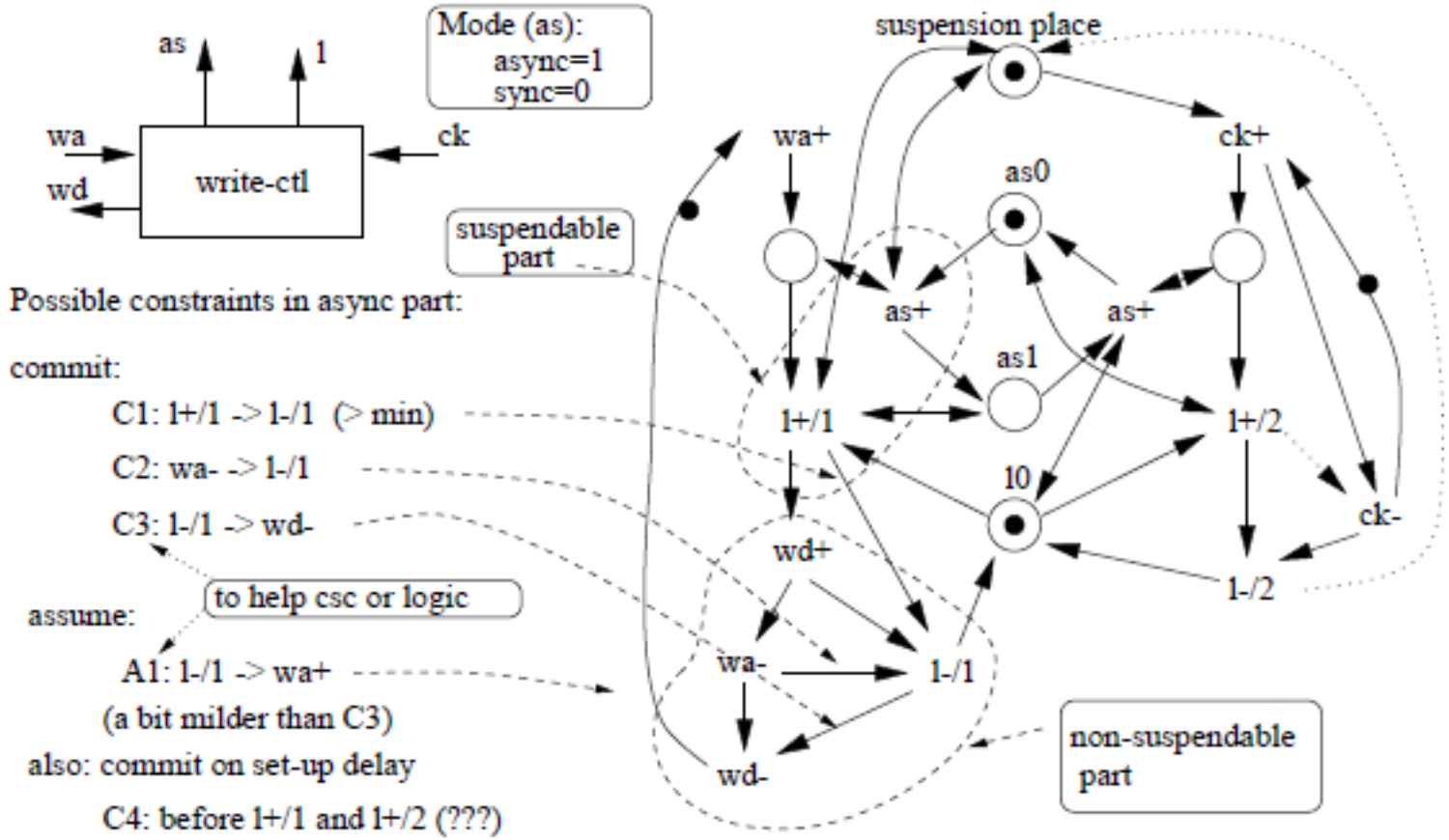
(2) with C2

wd = csc0 as (1 + wa')
as = as ck' + wa
1 = csc0 (wa as + as' 1 + ck)
csc0 = 1'(as'+wa) + csc0 (ck+wa)

(3) with C3 or with A1:

wd = as 1
as = wa + as ck'
1 = wa as + ck as'

Dual (Sync/Async) mode write ctl with pre-emption by Sync



Logic implementations:

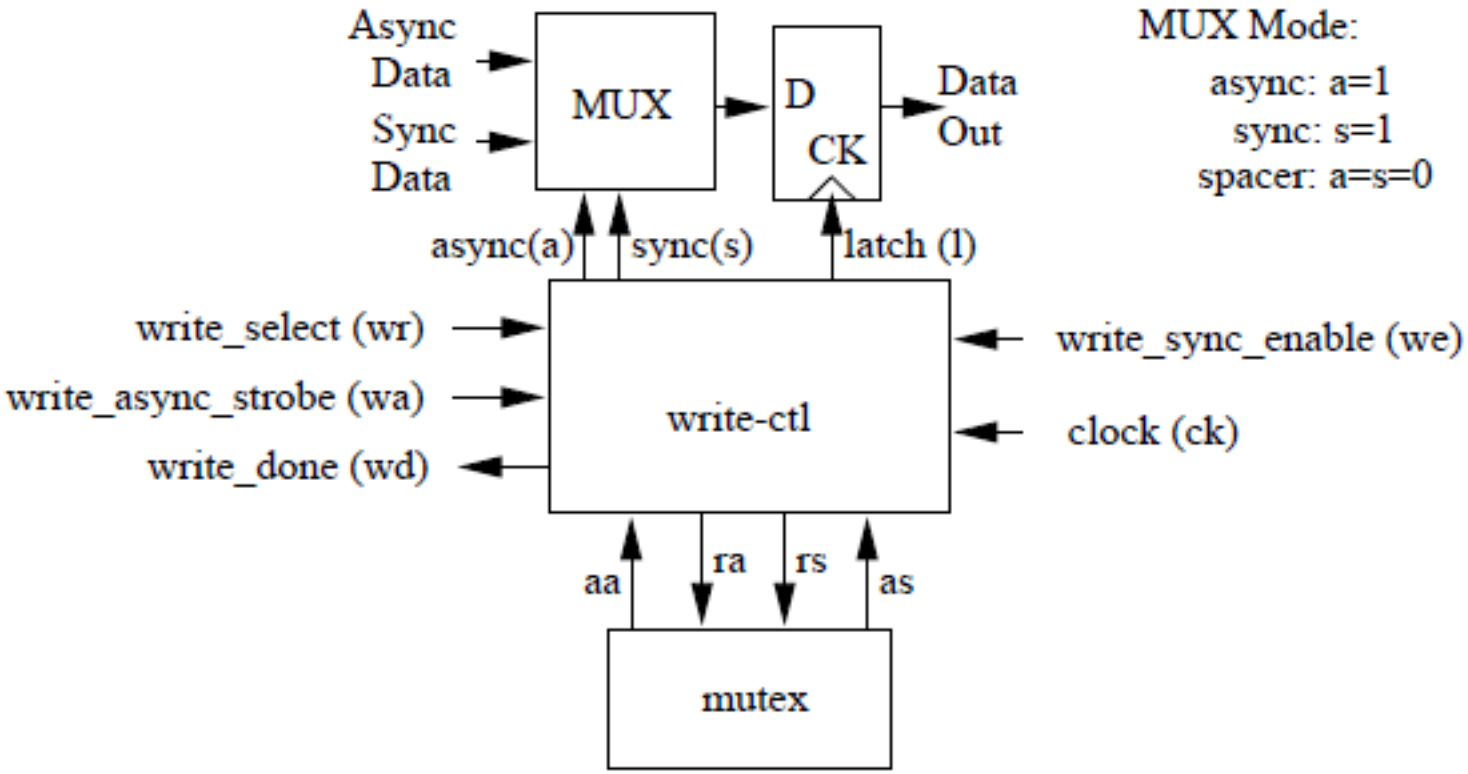
(1) no constraints in async part:

wd = csc0 (as 1 + wd)
as = wa ck' wd' 1' + as (ck' + 1)
1 = csc0 wd' as (ck' +1) + ck as'
csc0 = wa (1 + csc0 + as')

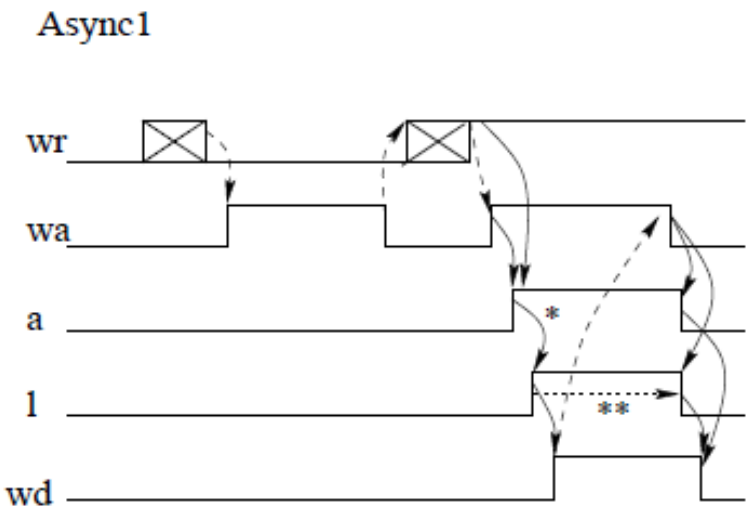
(3) with C3:

wd = as 1
as = wa ck' 1' + as (1 + ck')
1 = wa ck' as + ck (wa 1 + as')

Dual (Sync/Async) mode write ctl with Mutex



Dual mode write ctl: timing diagrams)



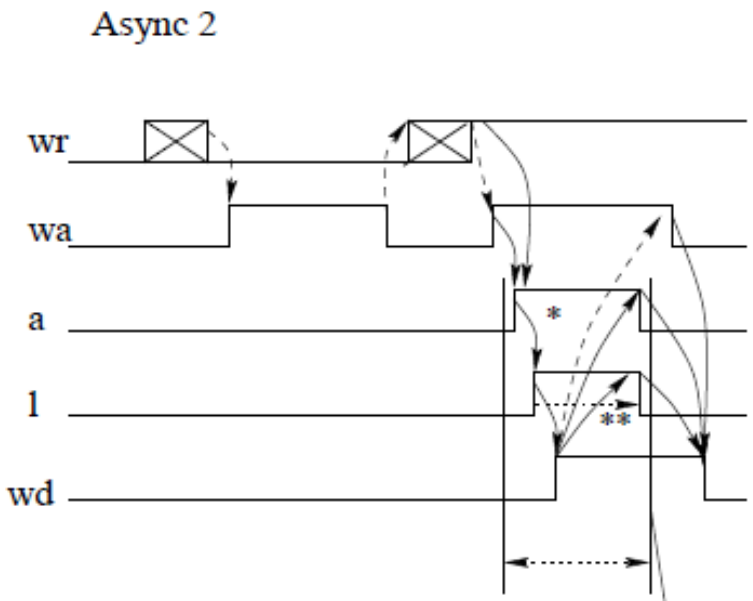
special commit constraints:

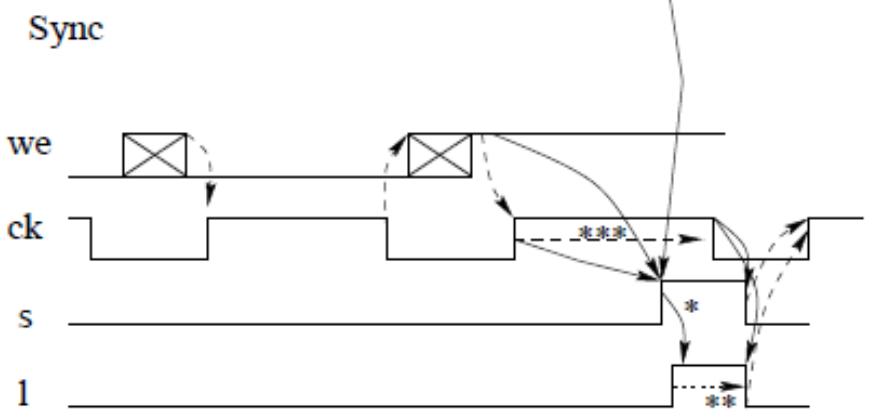
* maybe min set-up time

** maybe min hold time

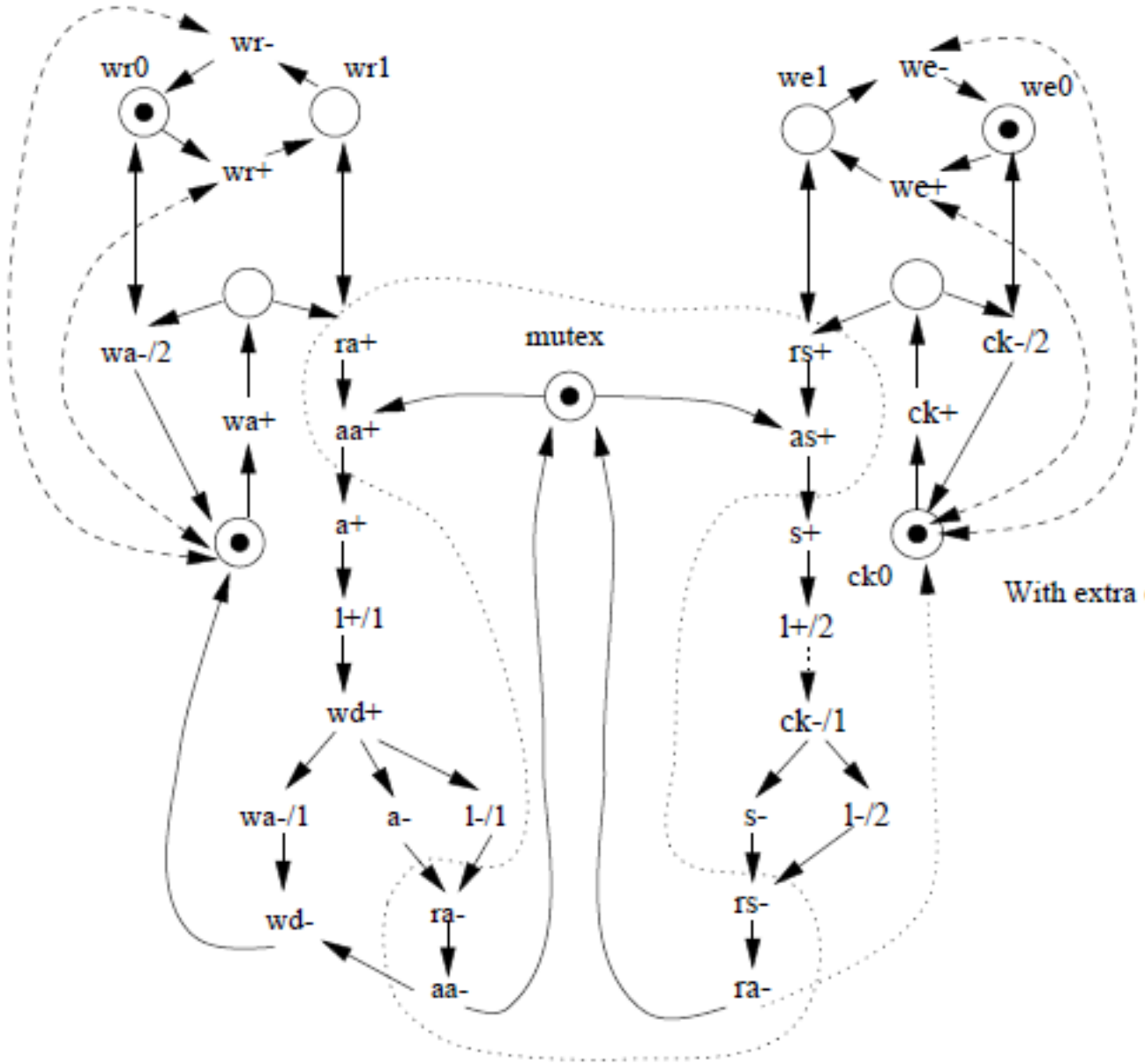
special assume constraint:

*** length of clock pulse must allow for at least one mutext tenure on async (to avoid missing sync cycles)





Dual mode write ctl with Mutex: STG and logic



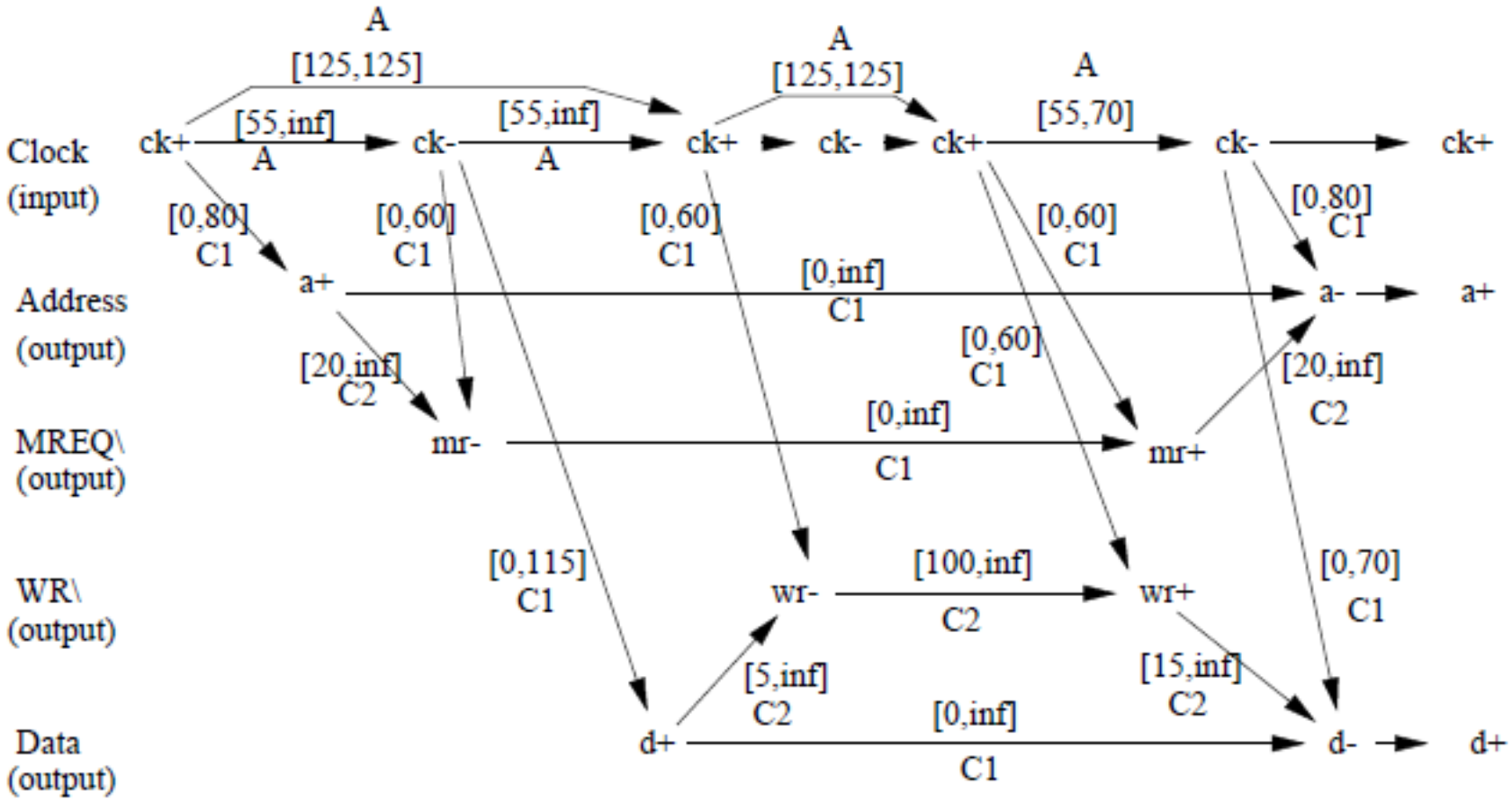
Logic implementation:

wd = aa l + wd (aa + wa)
a = wd' aa
s = ck as
l = wd' a + ck s
ra= wa wr wd' + l aa + a
rs = l as + we ck + s

With extra commits (wa-/1 -> {a-,l-/1}):

wd = l a + wd aa
a = wa aa
s = as ck
l = wa a + ck s
ra = l aa + wr wa + a
rs = l as + we ck + s

Example of Z84 write i/f: Initial Event Graph



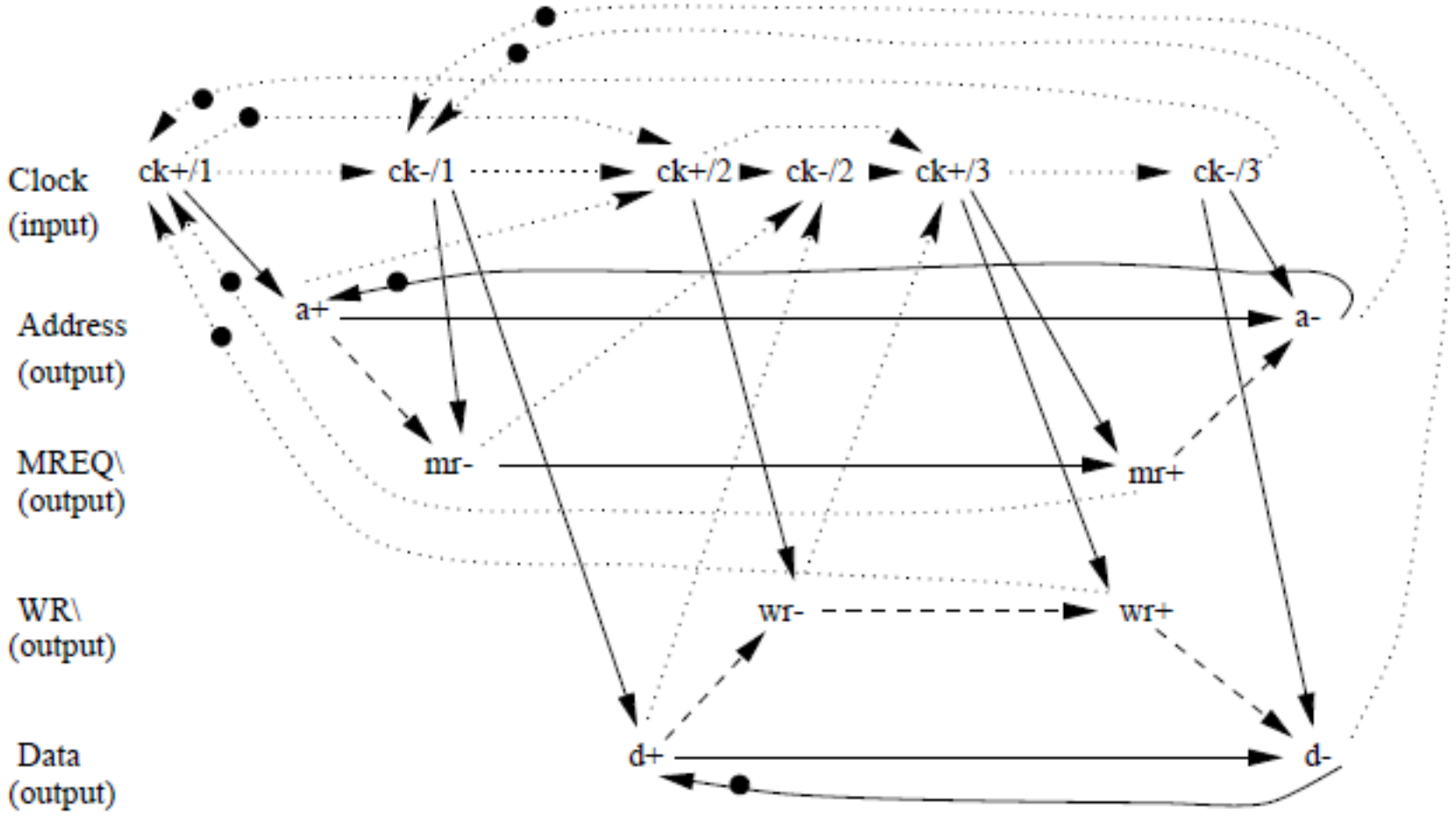
Constraint types: commit ({I,O} -> O) and assume ({I,O} -> I)

commit 1 (C1): [0,max] or [0,inf] => 'immediate causal' ASAP (for enabling); but maybe as long as max if with max, this can be used as assume (?)

commit 2 (C2): [min, inf] => 'delayed causal' ASAP (via min delay); must not be less than min

assume (A): [min,max] => 'timing assumptions' to avoid inconsistence, nonpersistence, state coding conflicts and to optimize logic (wrt area and/or speed)

Example with Z84 write i/f: deriving STG from initial constraints



Constraint types: commit ({I,O} -> O) and assume ({I,O} -> I)

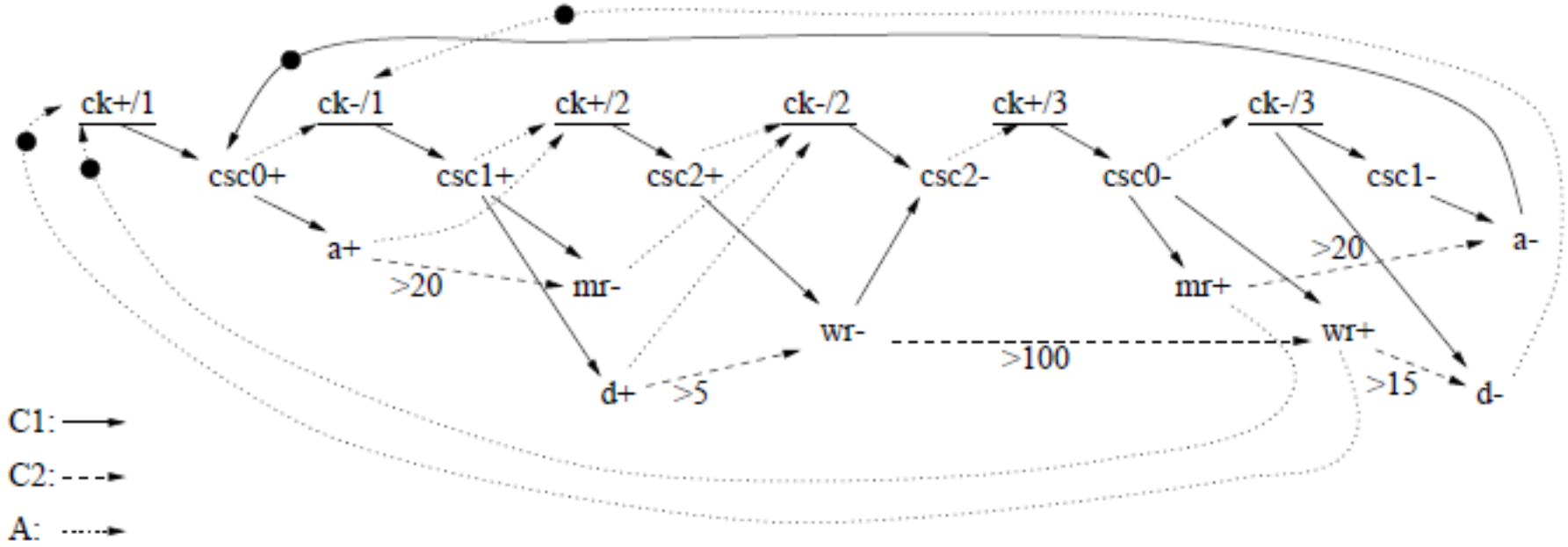
commit 1 (C1): [0,max] or [0,inf] => 'immediate causal' ASAP (for enabling); but maybe as long as max if with max, this can be used as assume (?)

commit 2 (C2): [min, inf] => 'delayed causal' ASAP (via min delay); must not be less than min

assume (A): [min,max] => 'timing assumptions' to avoid inconsistence, nonpersistence, state coding conflicts and to optimize logic (wrt area and/or speed)

Additional assume constraints inferred from C1 (with max) and A to help state coding and logic

Example with Z84 write i/f: STG-based synthesis



Logic implementation:

a= csc0 + csc1 + mr'

mr = (csc0 csc1 a)'

wr = csc0' + d' + csc2' wr

d = csc1(csc0 + ck) wr'

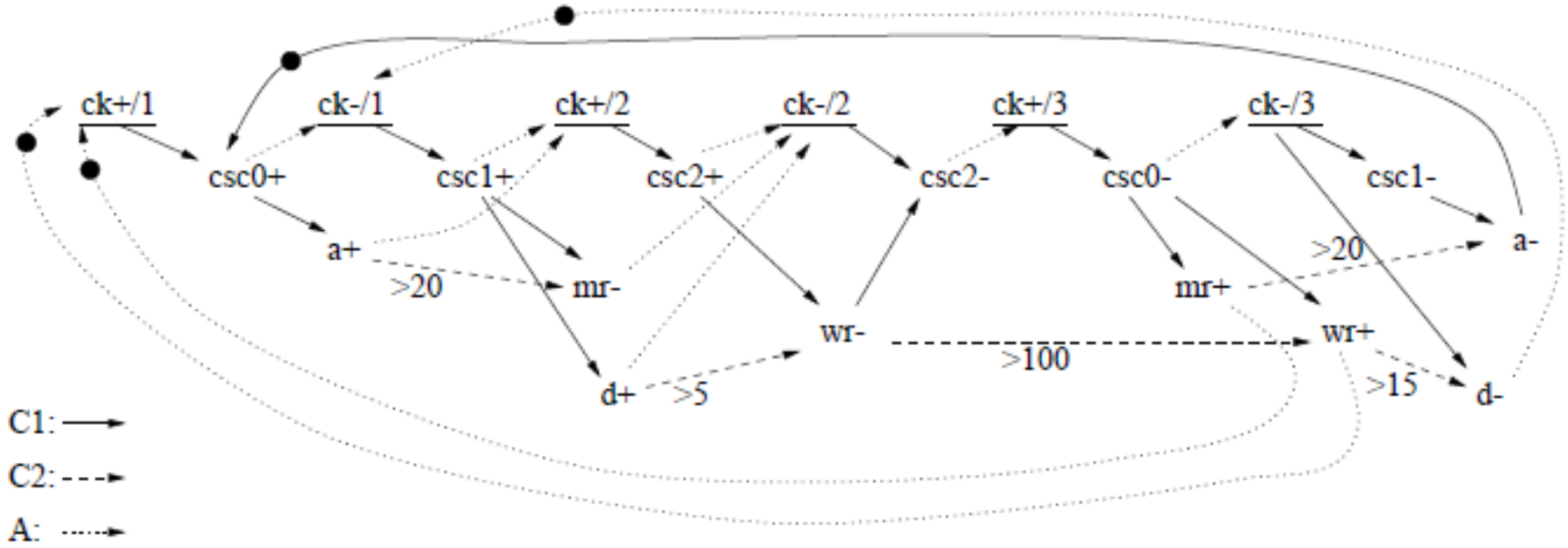
csc0 = csc2 + ck a' + csc0 (ck' + wr)

csc1 = csc0 ck' + csc1 ck

csc2 = ck wr csc0 csc1 + csc2 (ck + wr)

commit C2 constraints have to be satisfied by inserting delays into trigger inputs

Example with Z84 write i/f: using stricter (burst) constraints



Logic implementation:

a= csc0 + csc1 + mr'

mr = (csc0 csc1 a)'

wr = csc0' + d' + csc2' wr

d = csc1(csc0 + ck) wr'

csc0 = csc2 + ck a' + csc0 (ck' + wr)
csc1 = csc0 ck' + csc1 ck
csc2 = ck wr csc0 csc1 + csc2 (ck + wr)

commit C2 constraints have to be satisfied by inserting delays into trigger inputs

STG synthesis problems

Problem 5.1 *Given a Timing Diagram specification (or better say, an Event Graph, whose arcs are annotated with timing constraints), synthesize an 'equivalent' STG with three types of arcs (or places and arcs if choice is involved): "immediate causality" arcs, "delayed causality" arcs and "input order" arcs.*

Problem 5.2 *Find an optimal (speed/area of future logic)* asynchronous schedule *in terms of "precedence" (b always occurs after a) and "delayed precedence" (b always occurs after a with a delay of δ units [4]) arcs that satisfies the original Timing Diagram (Event Graph).*

Related work

A. El-Aboudi, E. -. Aboulhamid and E. Cerny, "Synthesis of interface controllers from timing diagram specifications," *Proceedings of the IEEE 1998 Custom Integrated Circuits Conference (Cat. No.98CH36143)*, Santa Clara, CA, USA, 1998, pp. 89-92.
doi: 10.1109/CICC.1998.694913

P. Vanbekbergen, G. Goossens and H. De Man, "Specification and analysis of timing constraints in signal transition graphs," *[1992] Proceedings The European Conference on Design Automation*, Brussels, Belgium, 1992, pp. 302-306.
doi: 10.1109/EDAC.1992.205943

K. -. Chung, R. K. Gupta and C. L. Liu, "An algorithm for synthesis of system-level interface circuits," *Proceedings of International Conference on Computer Aided Design*, San Jose, CA, USA, 1996, pp. 442-447.
doi: 10.1109/ICCAD.1996.569835