Asynchronous Design: Quo Vadis?

Tyne Bridge, 1928
Newcastle upon Tyne

Annibale Carracci, 1602
National Gallery, London

Alex Yakovlev

Microelectronics System Design Group
School of Electrical, Electronic and Computer Engineering,
Newcastle University, UK

Outline

- Asynchronous Design Principles
- Praises and curses
- (Some of the) Models, Techniques and Tools for Asynchronous Design
- Synchronization and Arbitration
- "Asynchronous History" (in brief)
- "Who is who" in Async design
- Where do we go?

Asynchronous Behaviour

- Synchronous vs Asynchronous behaviour in general terms, examples:
  - Orchestra playing with vs without a conductor
  - Party of people having a set menu vs a la carte
- Synchronous means all parts of the system acting globally in tact, even if some or all part 'do nothing'
- Asynchronous means parts of the system act on demand rather than on global clock tick
- Acting in computation and communication is, generally, changing the system state
- Synchrony and Asynchrony can be in found in CPUs, Memory, Communications, SoCs, NoCs etc.

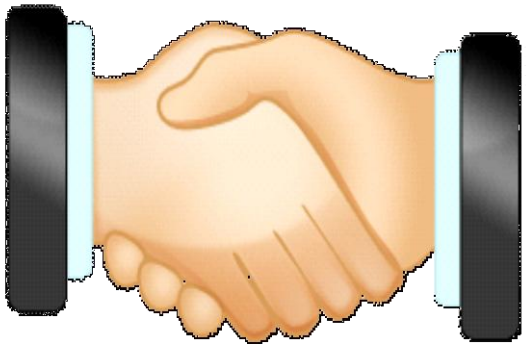
Why think about Timing and Synchrony

Gateshead Millennium Bridge, Newcastle, 2000

Key Principles of Asynchronous Design

- Asynchronous handshaking
- Delay-insensitive encoding
- Completion detection
- Causal acknowledgment (aka indication or indicatability)
- Strong and weak causality (full indication and early evaluation)
- "Time comparison" (synchronisation, arbitration)

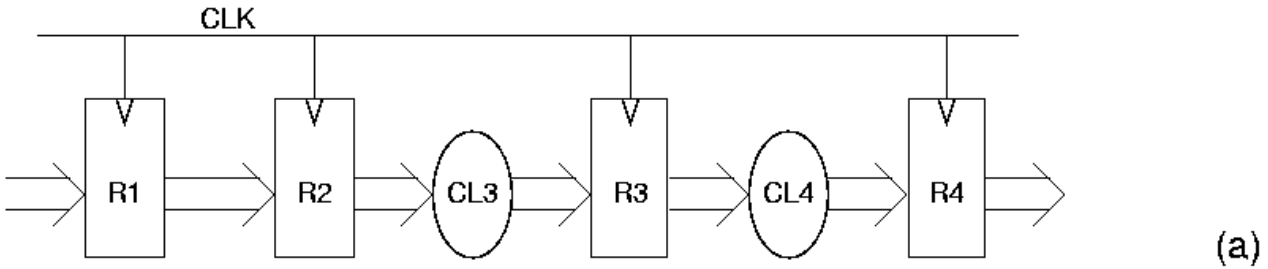
Why and what is handshaking?



Mutual Synchronisation is via Handshake

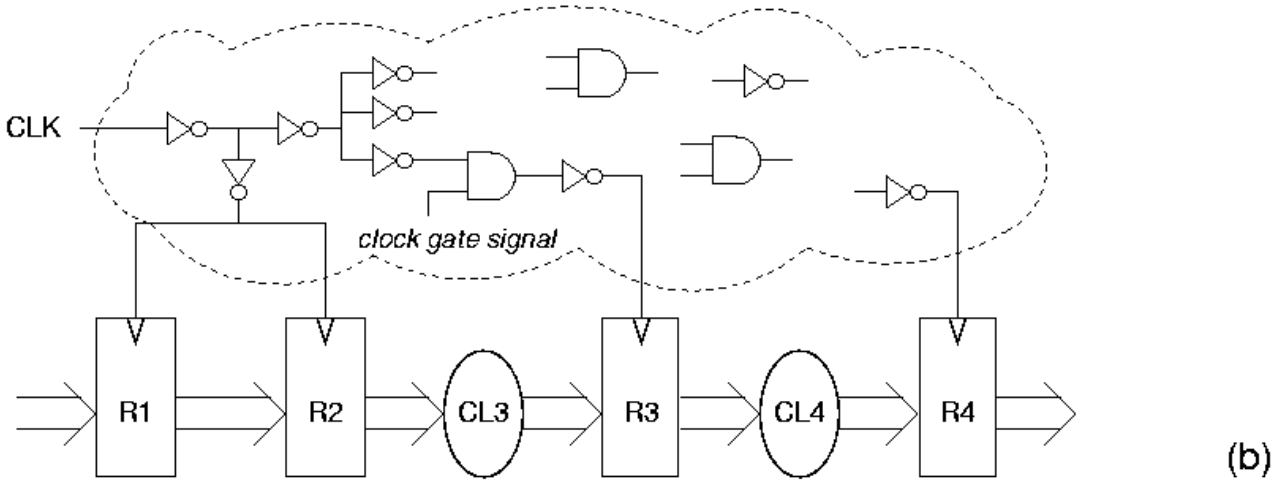
Synchronous clocking

How we think



(a)

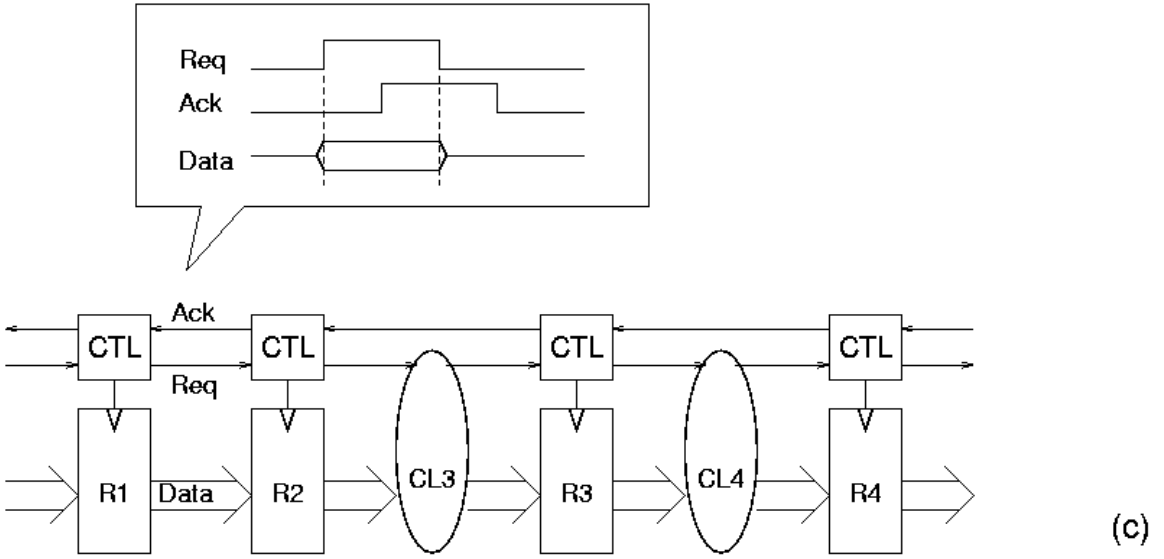
What we design



(b)

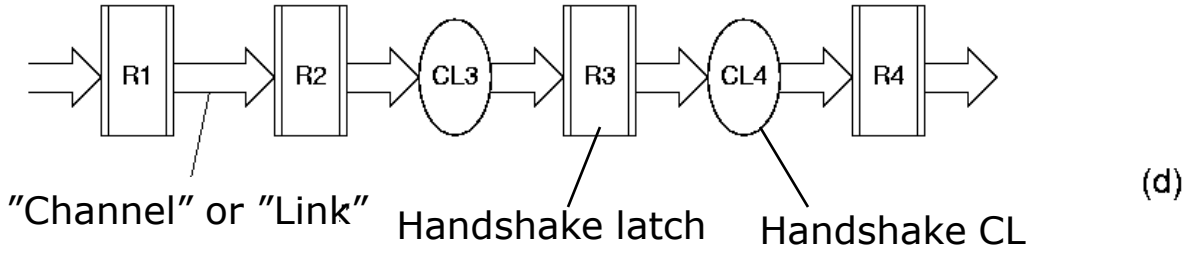
Asynchronous handshaking

What we design



(c)

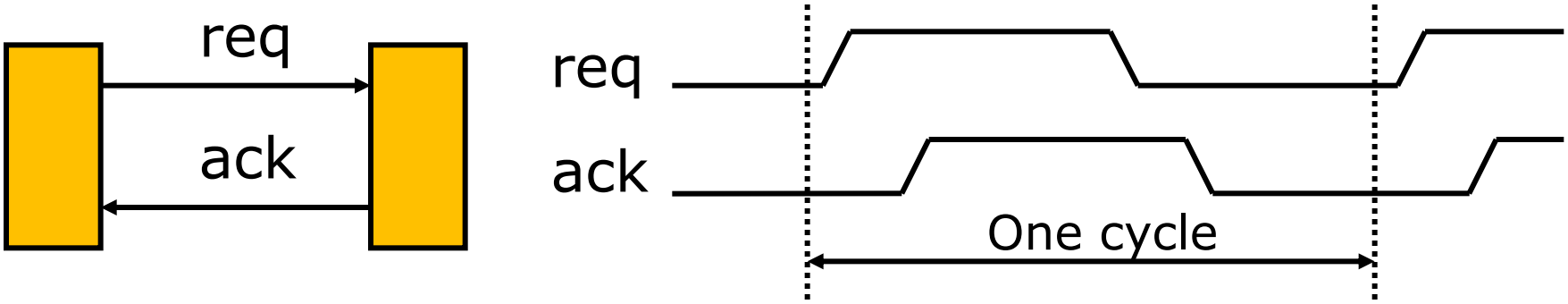
How we think



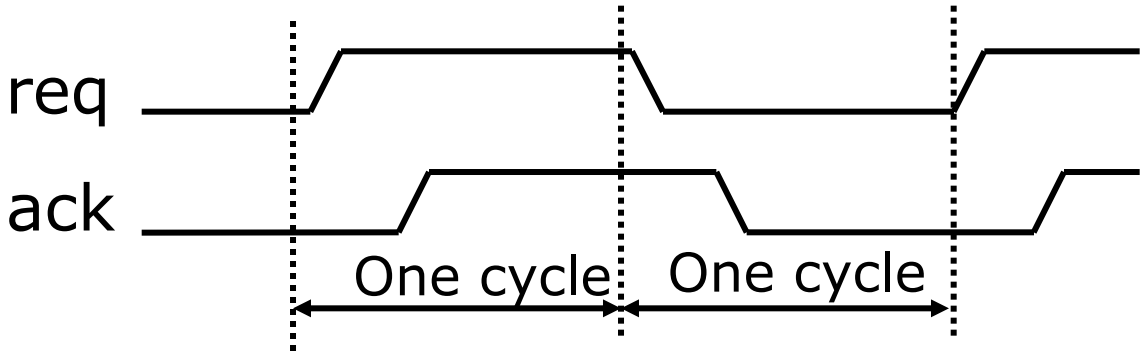
(d)

Handshake Signalling Protocols

Level Signalling (RTZ or 4-phase)

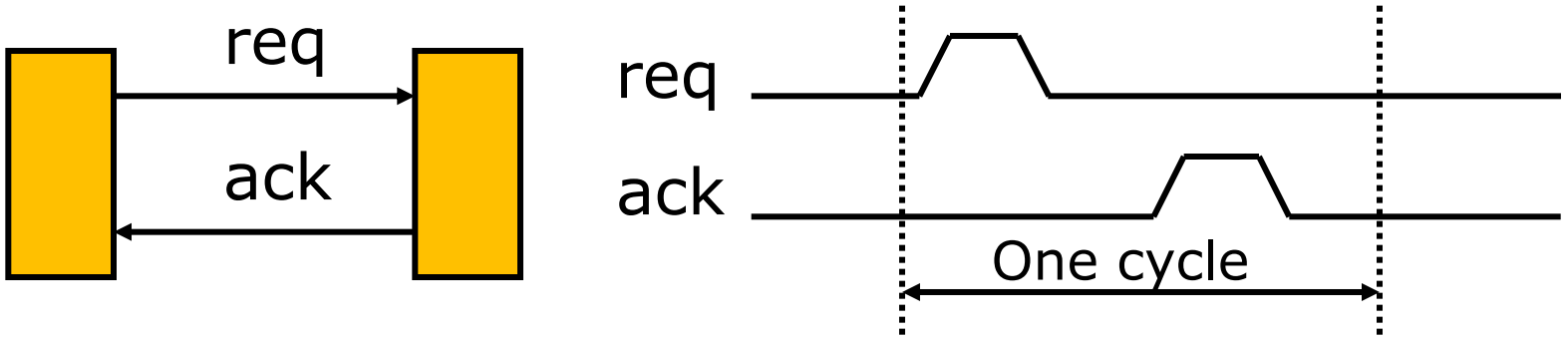


Transition Signalling (NRZ or 2-phase)

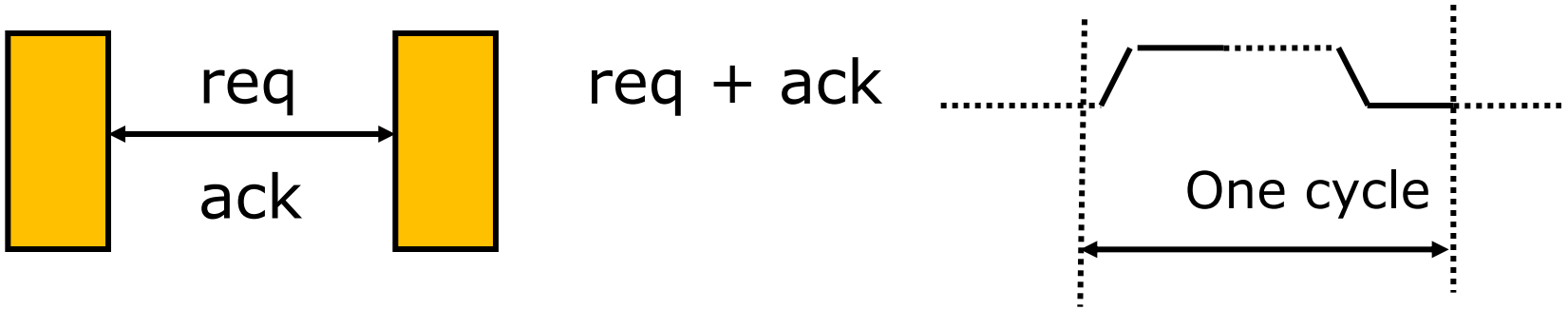


Handshake Signalling Protocols

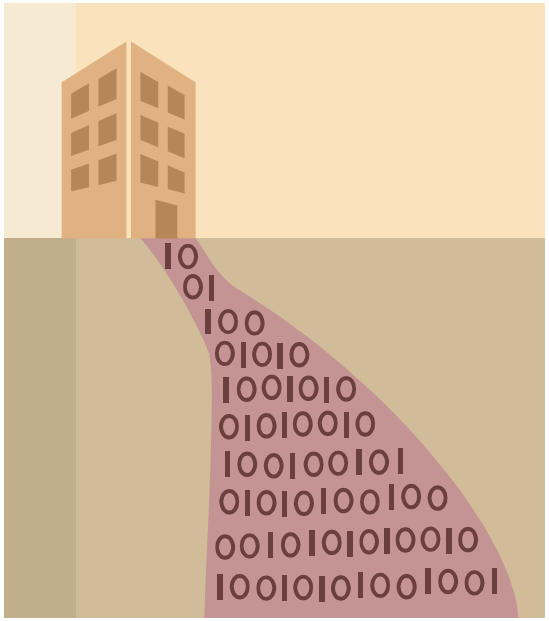
Pulse Signalling



Single-track Signalling (GasP)

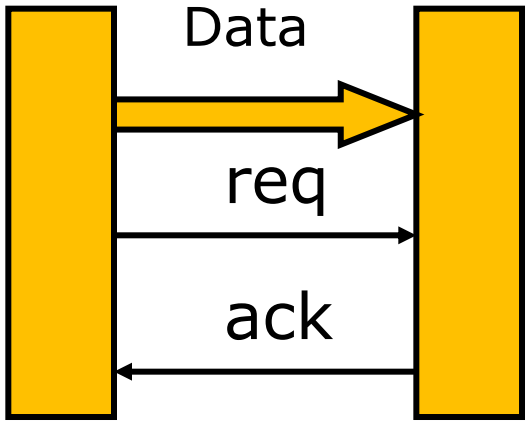


Why and what is delay-insensitive coding?

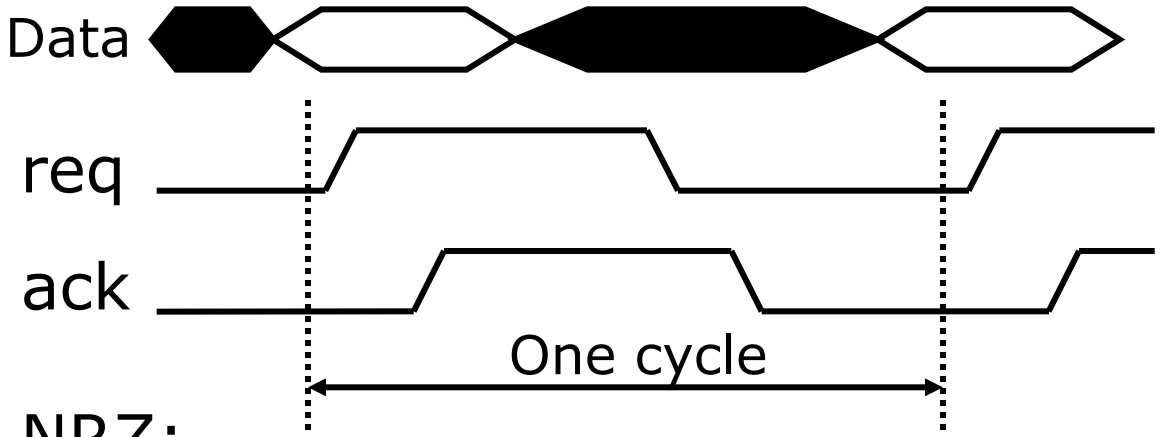


Data Token = (Data Value, Validity Flag)

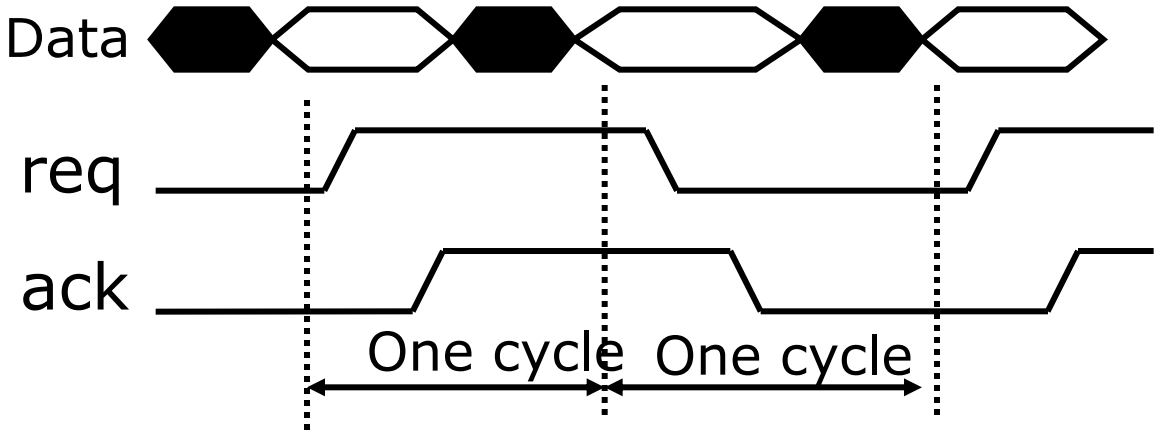
Bundled Data



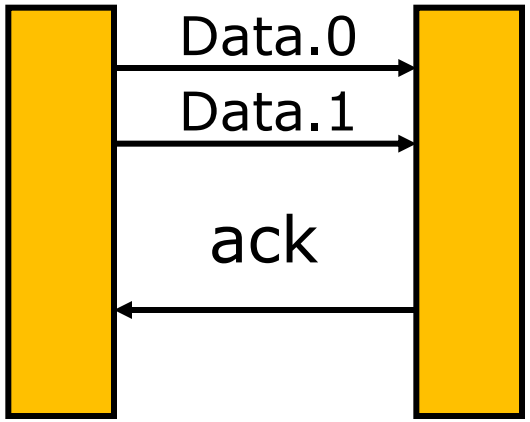
RTZ:

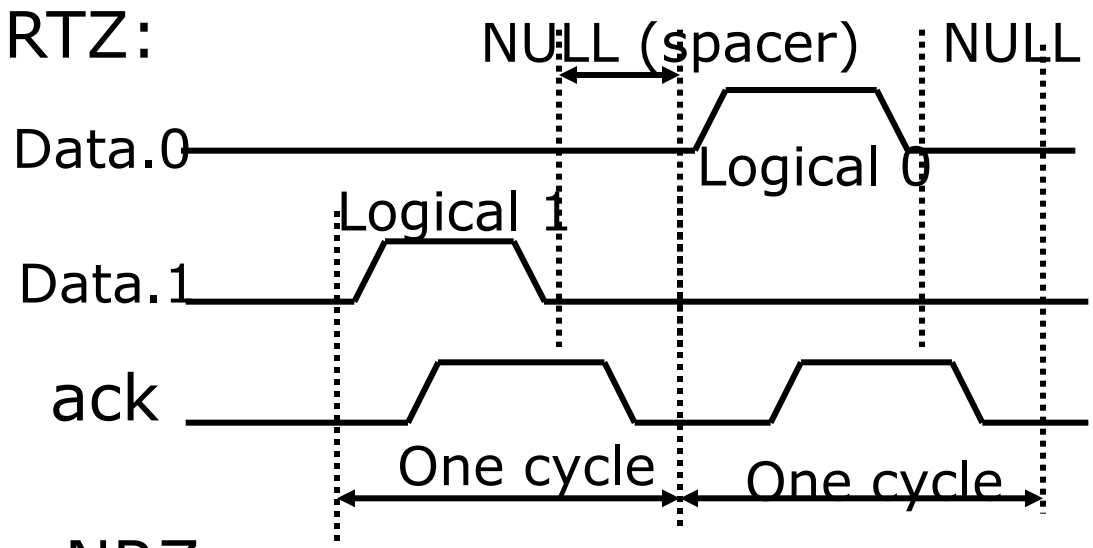


NRZ:

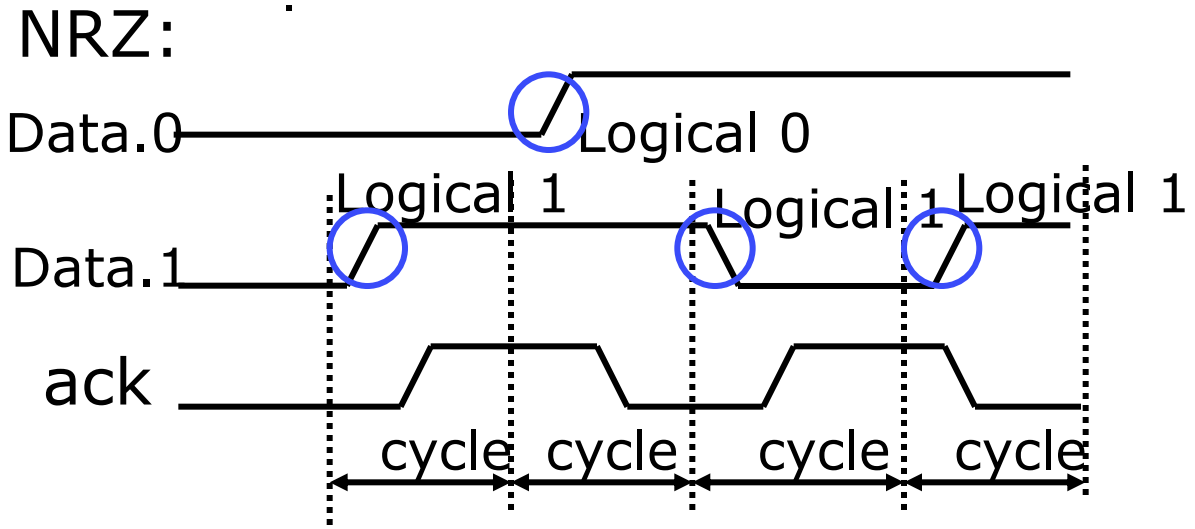


DI encoded data (Dual-Rail)





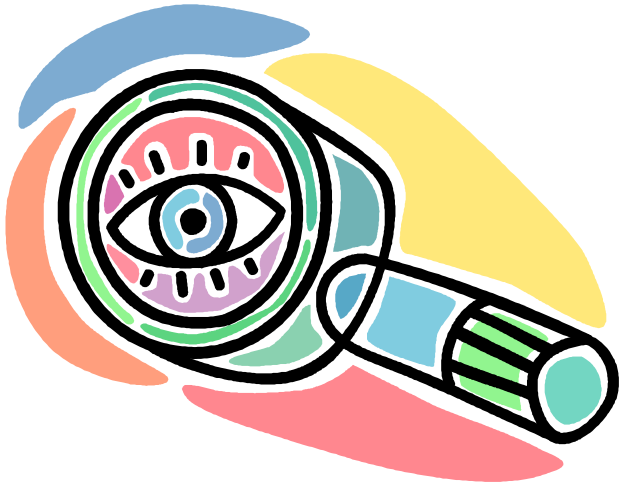
NRZ coding leads to complex logic implementation; special ways to track odd and even phases and logic values are needed, such as LEDR



DI codes (1-of-n and m-of-n)

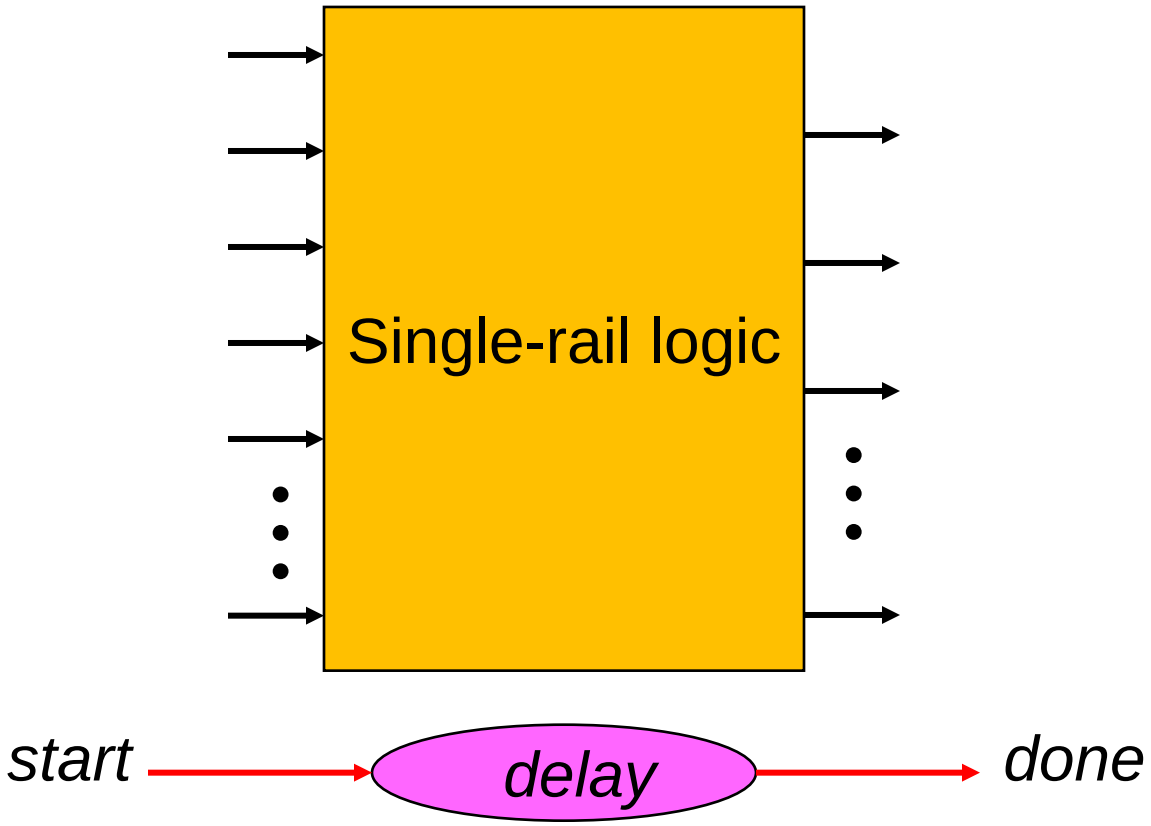
- 1-of-4:
  - 0001=> 00, 0010=>01, 0100=>10, 1000=>11
- 2-of-4:
  - 1100, 1010, 1001, 0110, 0101, 0011 – total 6 combinations (cf. 2-bit dual-rail – 4 comb.)
- 3-of-6:
  - 111000, 110100, …, 000111 – total 20 combinations (can encode 4 bits + 4 control tokens)
- 2-of-7:
  - 1100000, 1010000, …, 0000011 – total 21 combinations (4 bits + 5 control tokens)

Why and what is completion detection?



Signalling that the Transients are over

Bundled-data logic blocks

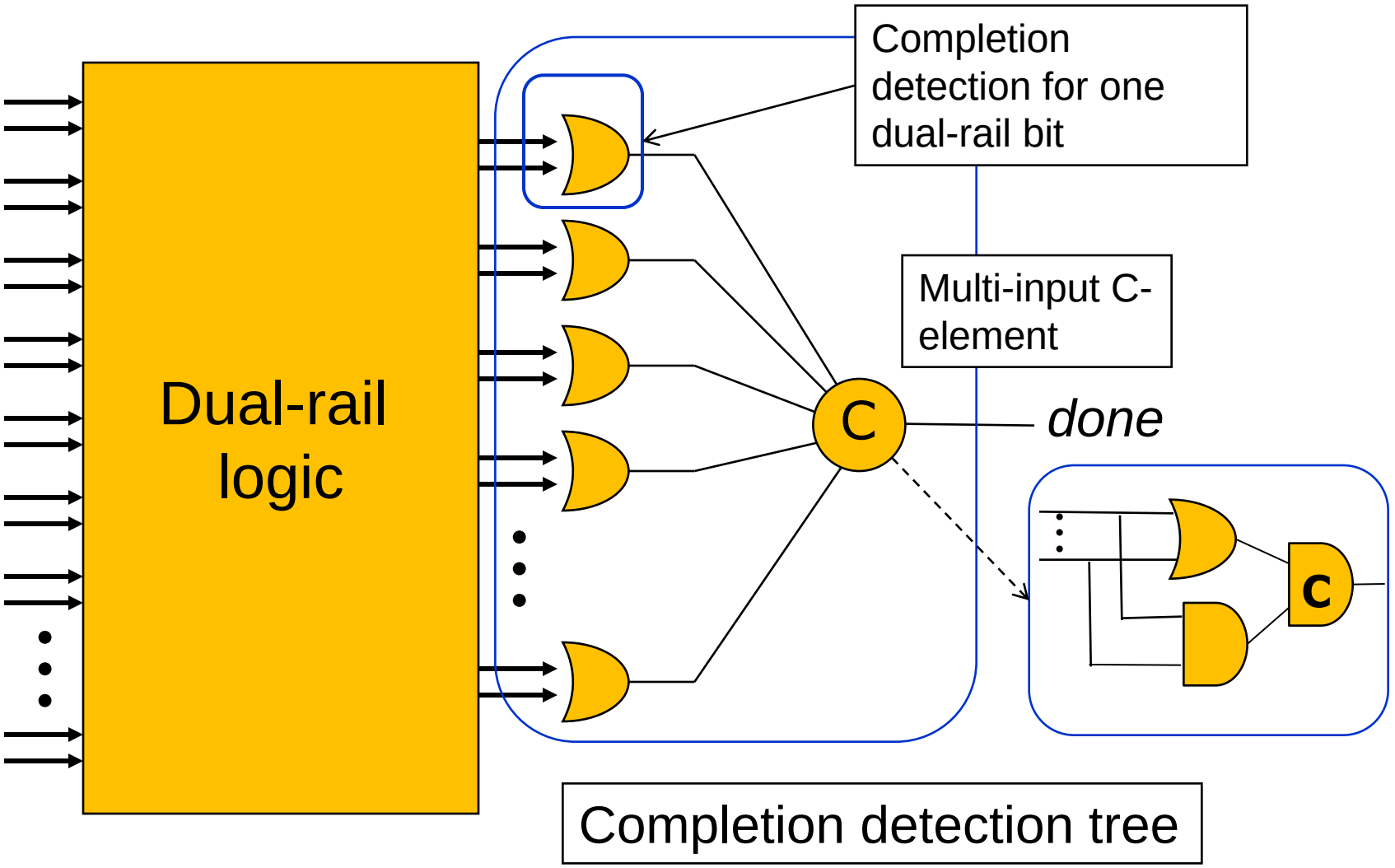


Conventional logic + matched delay

Completion is implicit: by done signal

The delay must scale with the worst case delay path, So … not really self-timed

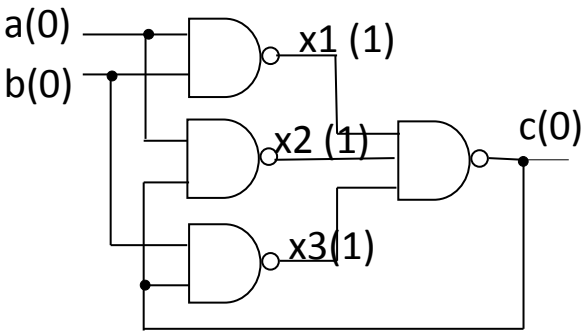
True completion detection

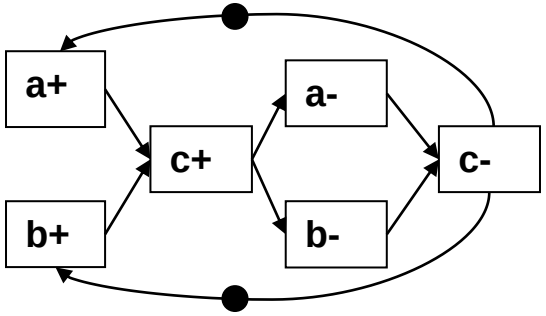


Why and what is causal acknowledgment?

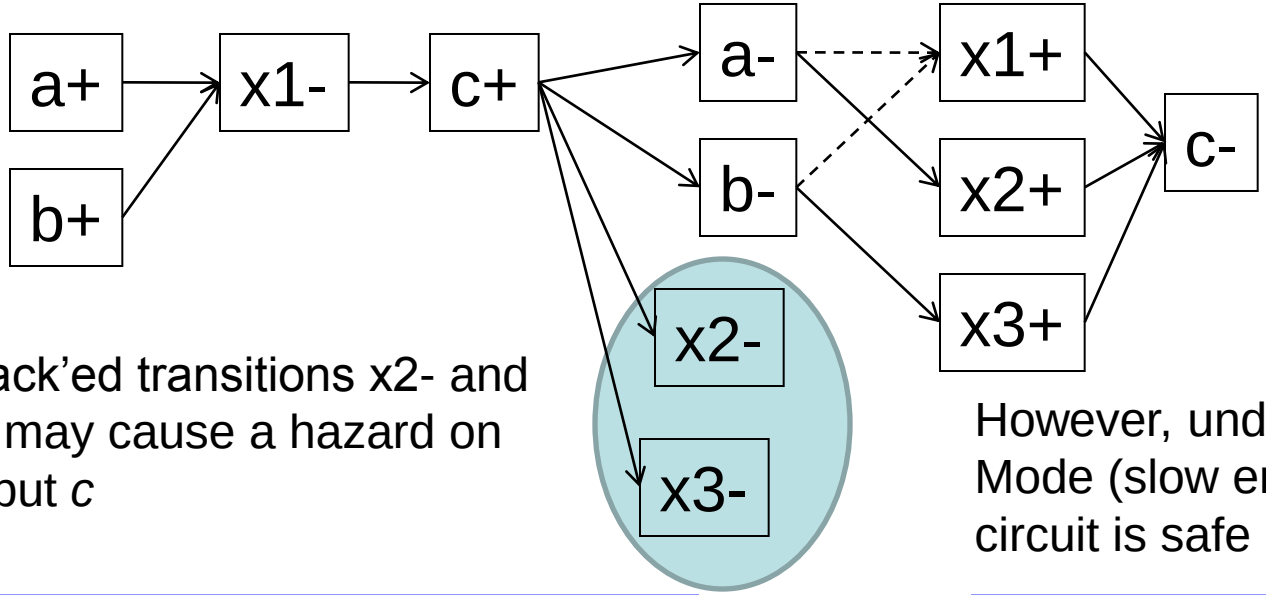
Every signal event must be acknowledged by another event

Causal acknowledgment





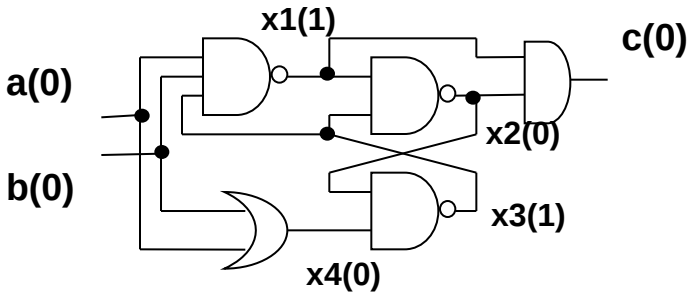
C-element implementation using simple gates

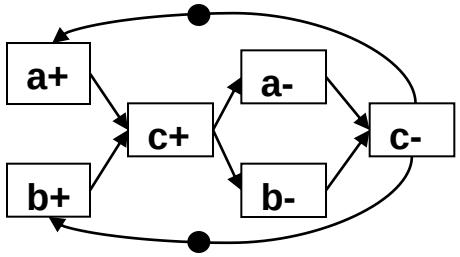


Unack'ed transitions x2- and x3- may cause a hazard on output *c*

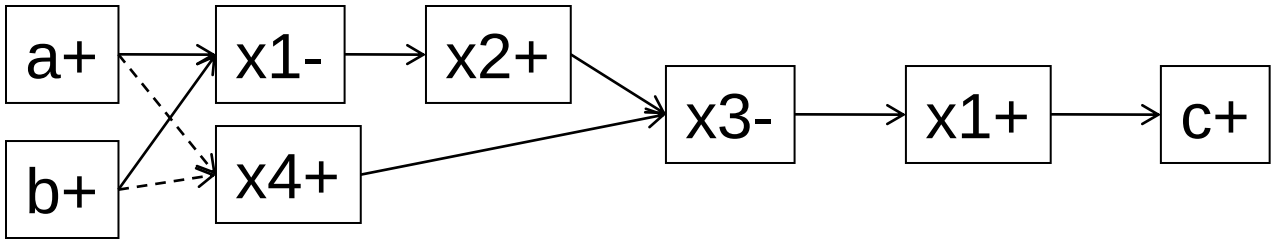
However, under Fundamental Mode (slow environment) the circuit is safe

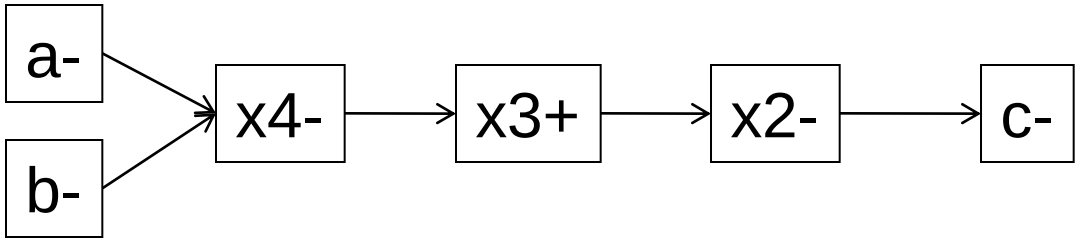
Principle of causal acknowledgement





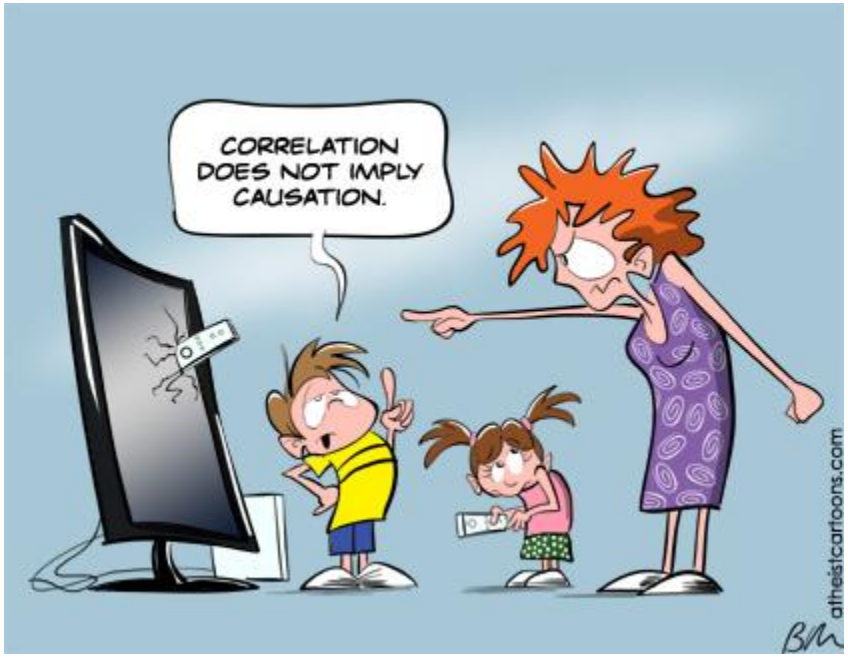
C-element implementation using simple gates





Each transition is causally ack'ed, hence no hazards can appear

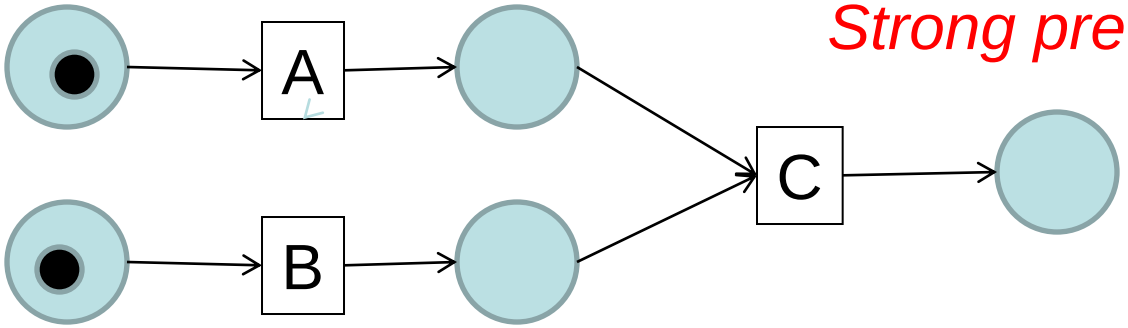
Why and what are strong and weak causality ?



Degree of necessity of precedence of some events for other events

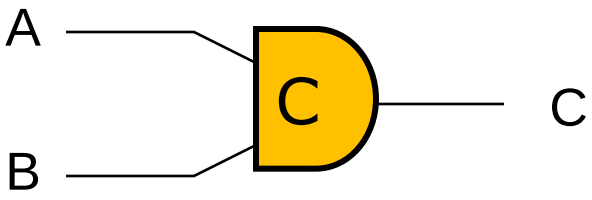
Strong Causality

- Petri net transitions synchronising as rendez-vous



Strong precedence

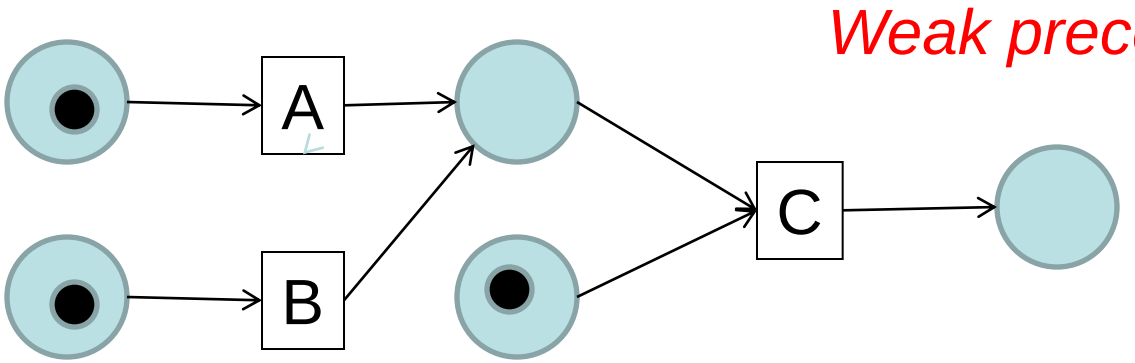
- Logic circuits: Muller C-element (in 0-1 and 1-0 transitions), AND gate (in 0-1 transitions), OR gate (in 1-0 tranisitions)



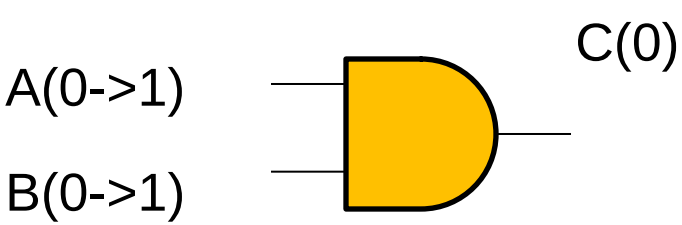
A	B	C^+
0	0	0
0	1	C
1	0	C
1	1	1

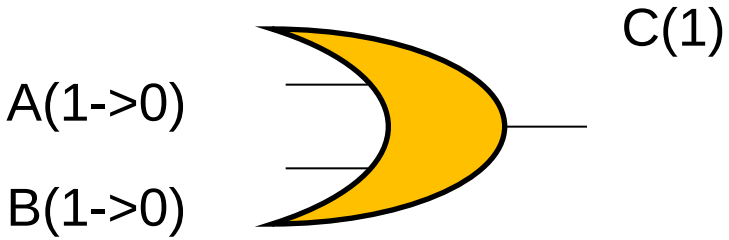
Weak Causality

- Petri net transitions communicating via places

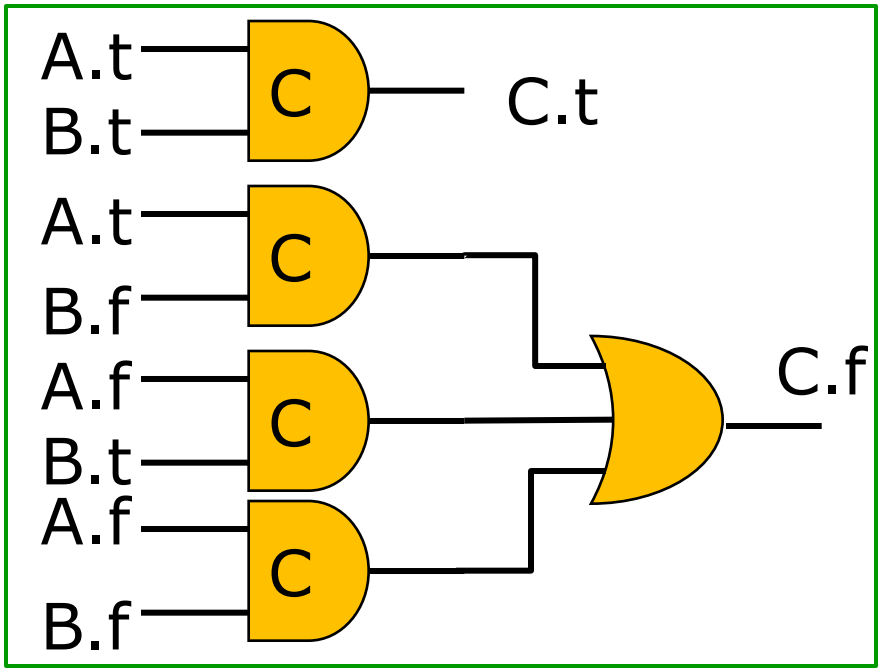


- Logic circuits: AND gate (in 1-0 transitions), OR gate (in 0-1 transitions)

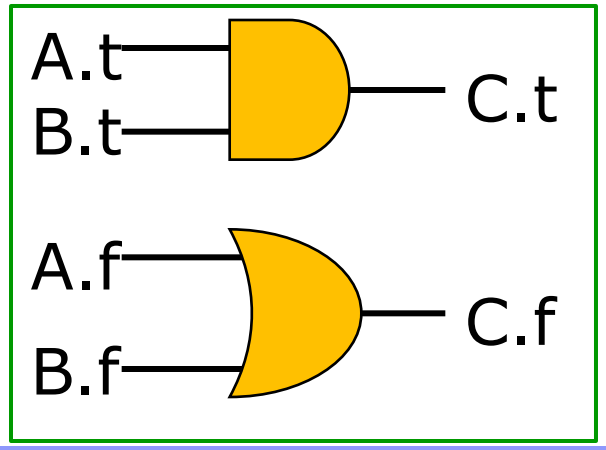




Full indication versus Early Evaluation



Dual-rail AND gate with full input acknowledgement



Dual-rail AND gate with “early propagation”

Allows outputs to be produced from NULL to Codeword only when some (required) inputs have transitioned from NULL to Codeword (similar for Codeword to NULL)

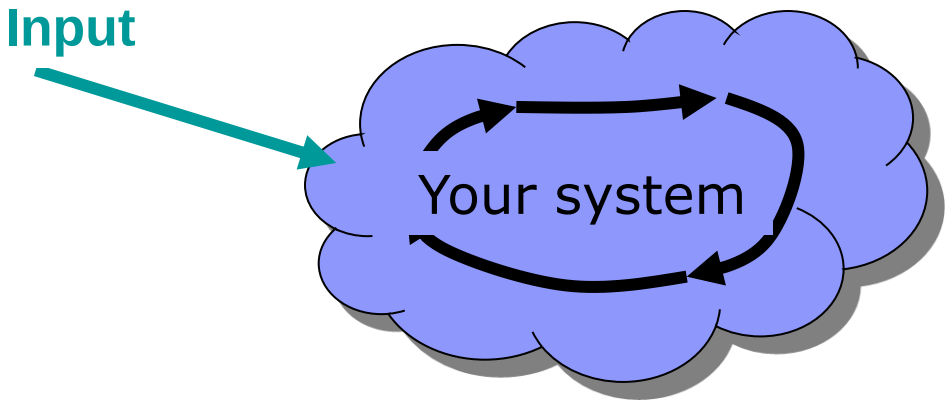
Why and what is timing comparison?

Telling if some event happened before another event

Synchronizers and arbiters

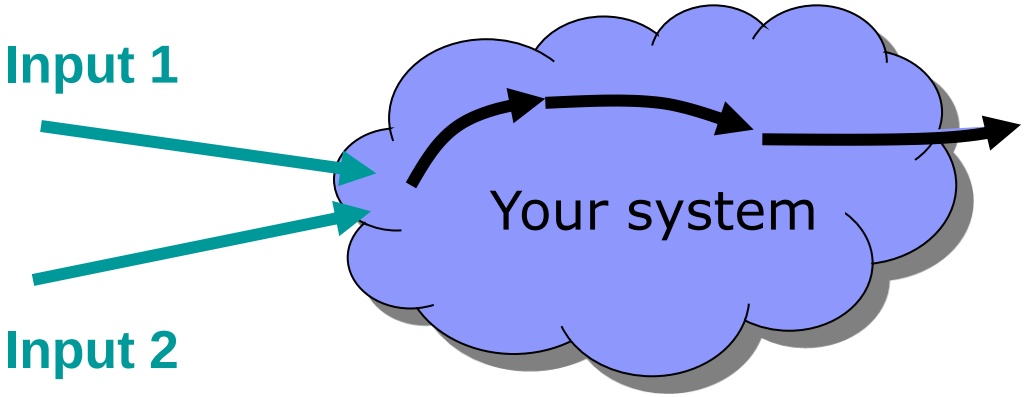
- Synchronizer

Decides which clock cycle to use for the input data

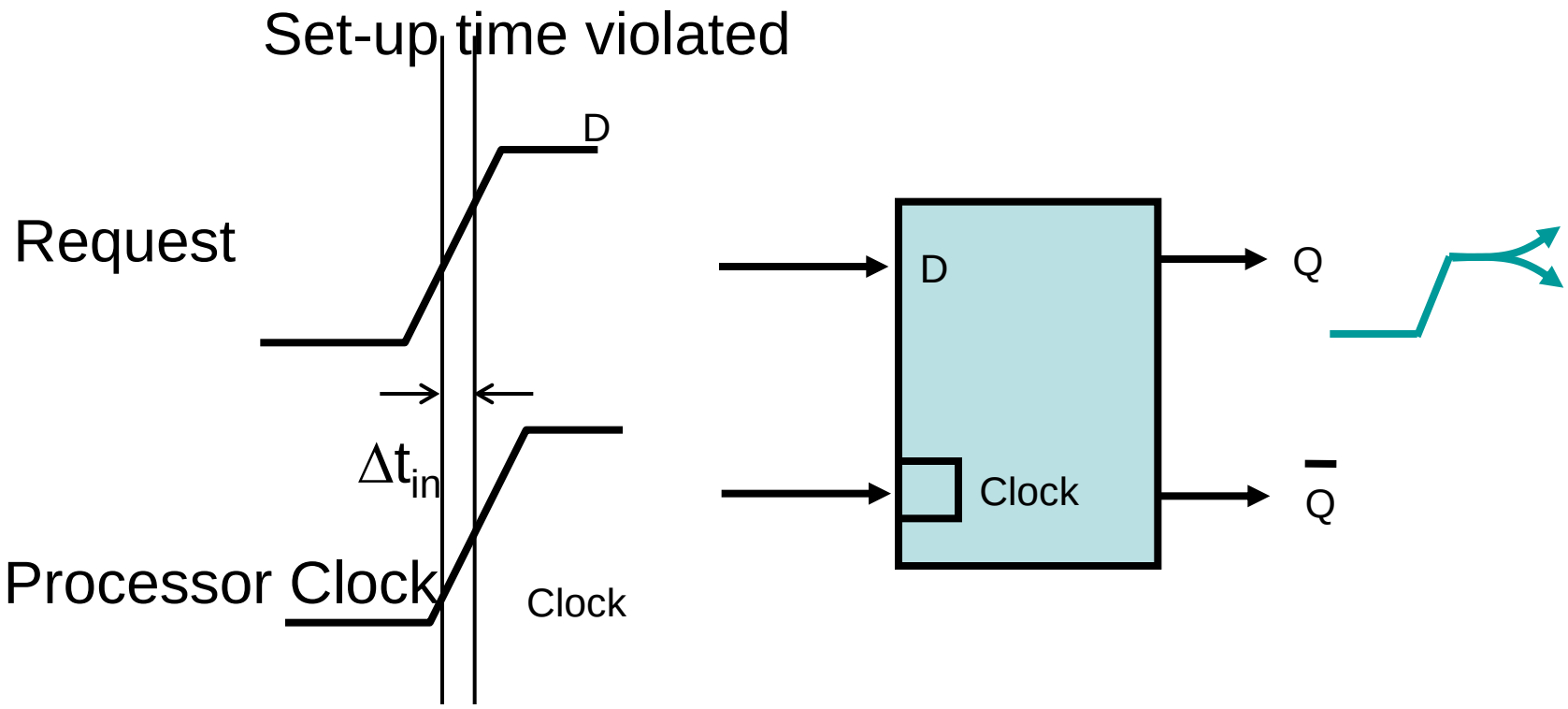


- Asynchronous arbiter

Decides the order of inputs

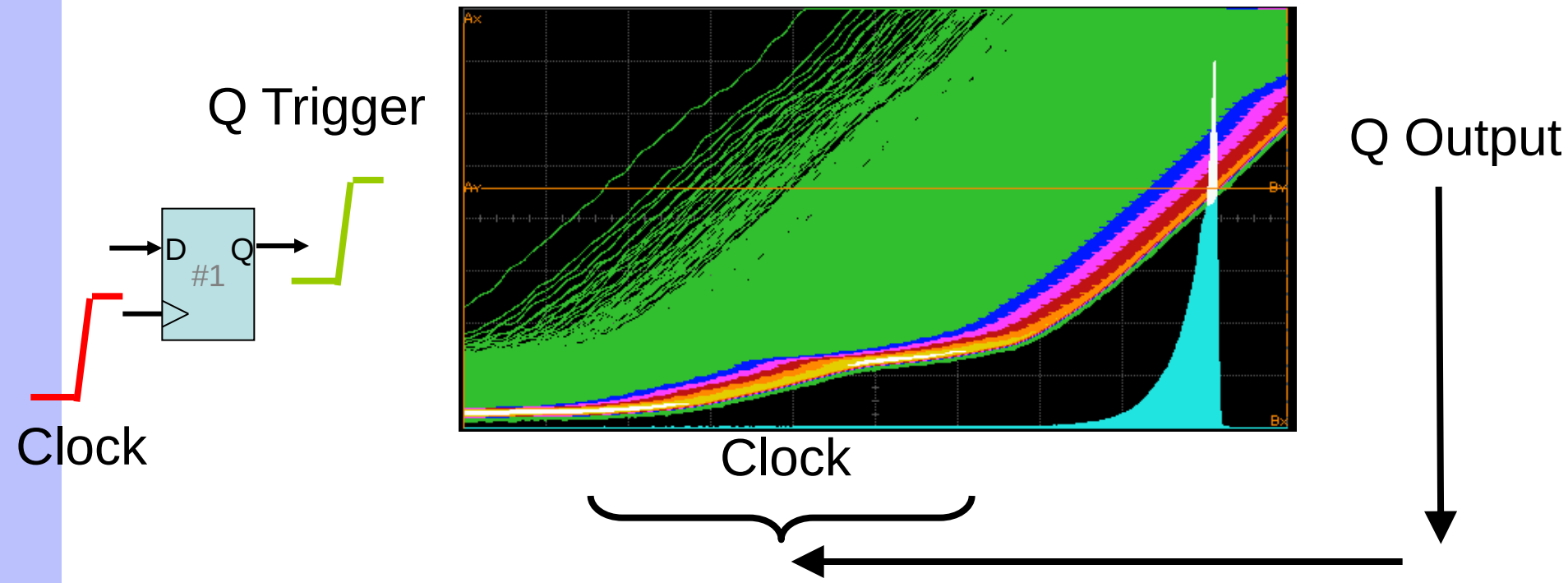


Metastability is....



Δt_{in} -> 0 Not being able to decide…

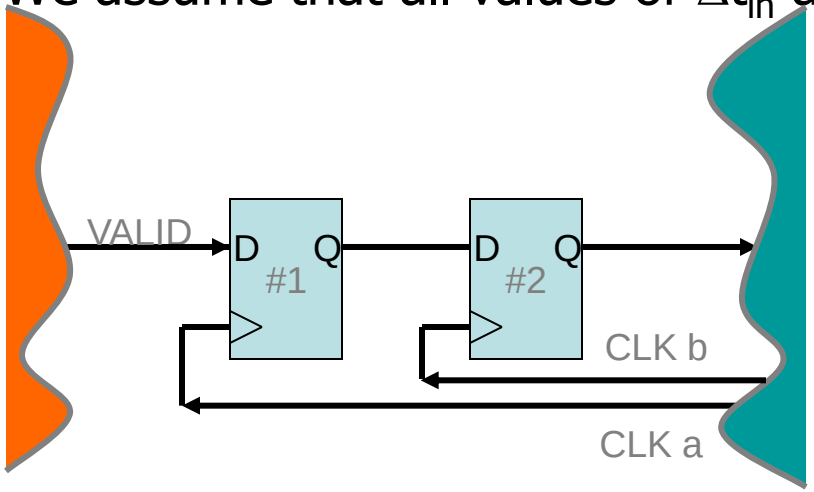
Typical responses



- We assume all starting points are equally probable
- Most are a long way from the "balance point"
- A few are very close and take a long time to resolve

Synchronizer

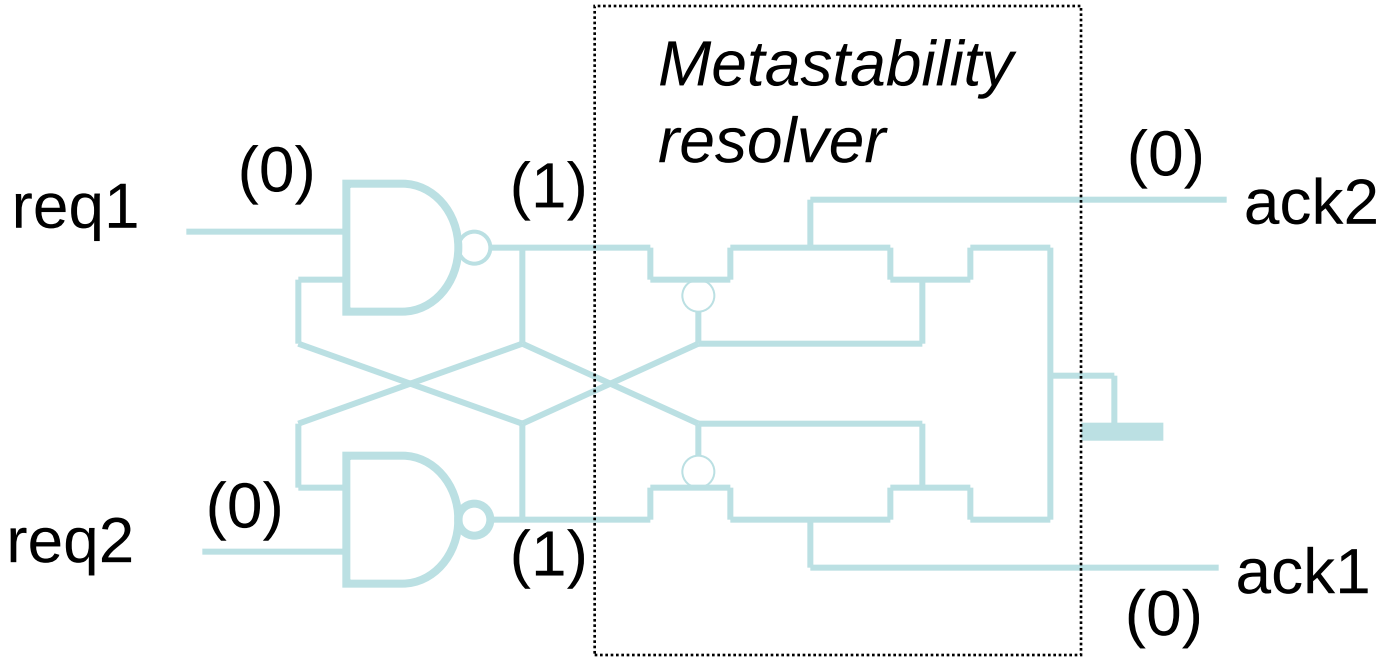
- t is time allowed for the Q to change between CLK a and CLK b
- τ is the recovery time constant, usually the gain-bandwidth of the circuit
- T_w is the "metastability window" (aperture around clock edge in which the capture of data edge causes a delay that is greater than normal propagation delay of the FF)
- τ and T_w depend on the circuit
- We assume that all values of Δt_{in} are equally probable



$$MTBF = \frac{e^{t/\tau}}{T_w \cdot f_c \cdot f_d}$$

Two-way arbiter (Mutual exclusion element)

Basic arbitration element: Mutex (due to Seitz, 1979)



An asynchronous data latch with metastability resolver can be built similarly

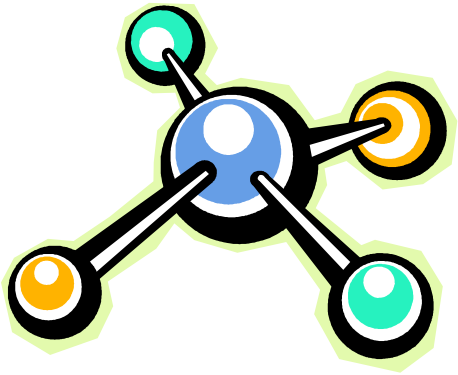
Praises…

- People have always been excited by asynchronous design, and motivated by:
  - Higher performance (work on average not worst case delays)
  - Lower power consumption (automatic fine-grain "clock" gating; automatic instantaneous stand-by at arbitrary granularity in time and function; distributed localized control; more architectural options/freedom; more freedom to scale the supply voltage)
  - Modularity (Timing is at interfaces)
  - Lower EMI and smoother Idd (the local "clocks" tend to tick at random points in time)
  - Low sensitivity to PVT variations (timing based on matched delays or even *delay insensitive*)
  - Secure chips (white noise current spectrum)
  - Plus, … a lot of scope and fun for research (there are many unexplored paths in this forest!)

… Curses

- So why have async designers been often "crucified" in the past?
  - Overhead (area, speed, power)
    - Control and handshaking
    - Dual-rail and completion detection costs
  - Hard to design
    - yes and no, … It's different – there are very many styles and variants to go and one can easily get confused which is better
  - Very few **practical** CAD tools (but many academic tools)
    - Tools are quite specific to particular design styles and design niches; hence don' t cover the whole spectrum
    - Complexity of timing and performance models
    - Difficulty with sign-off (for particular frequency requirements)
    - … But the situation is improving
  - Hard to Test
    - Possible, but not as mature as sync

Models and techniques for design

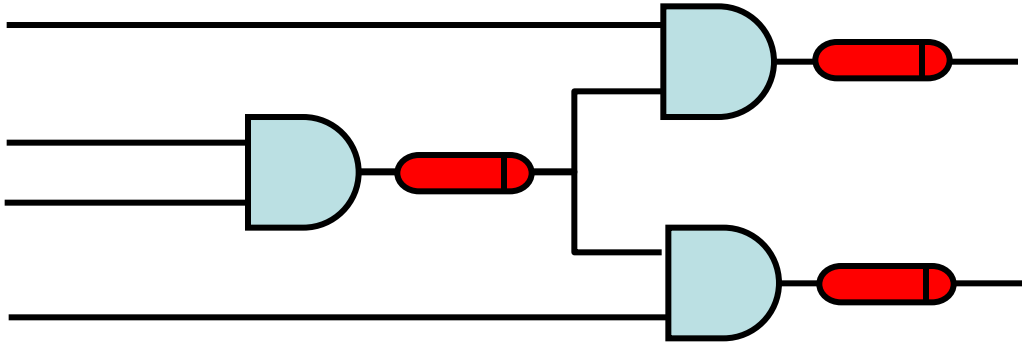


Models and techniques for asynchronous design

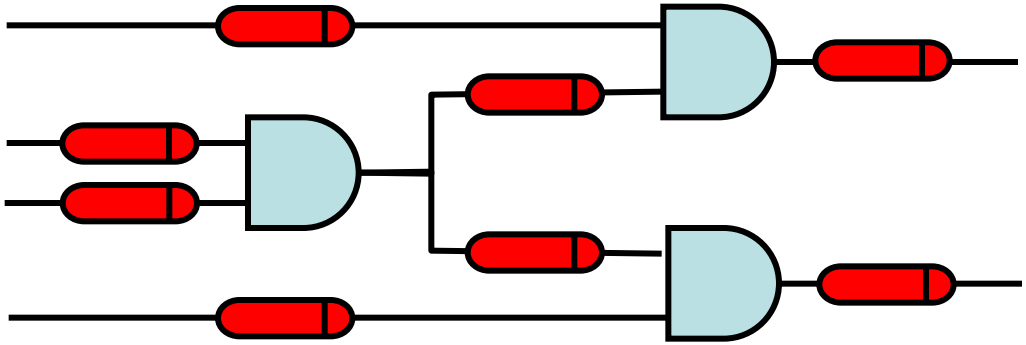
- Models:
  - Delay model (inertial, pure, gate delay, wire delay, bounded and unbounded delays)
  - Models of environment (fundamental mode, input-output)
  - Models of switching behaviour (state-based, event-based, hybrid)
- RTL level:
  - Data and control paths separate (data flow graphs, FSMs, STGs, Synchronised Transitions)
  - Pipeline based (Combinational logic plus registers and latch controllers, e.g. micropipelines, gate-level pipelining)
  - Process-based (CSP-like, Balsa, Haste, Communicating Hardware Processes)
- High-level models
  - Flow graphs (Marked graphs, extended MGs), Petri nets, Markov Chains
  - Behavioural HDLs (C, SystemC)

Gate vs wire delay models

- Gate delay model: delays in gates, no delays in wires

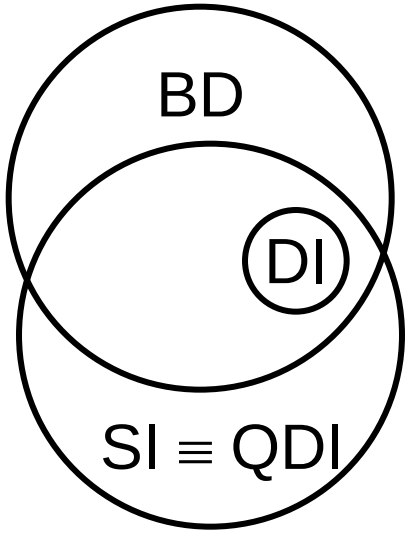


- Wire delay model: delays in gates and wires



Classes of asynchronous circuits

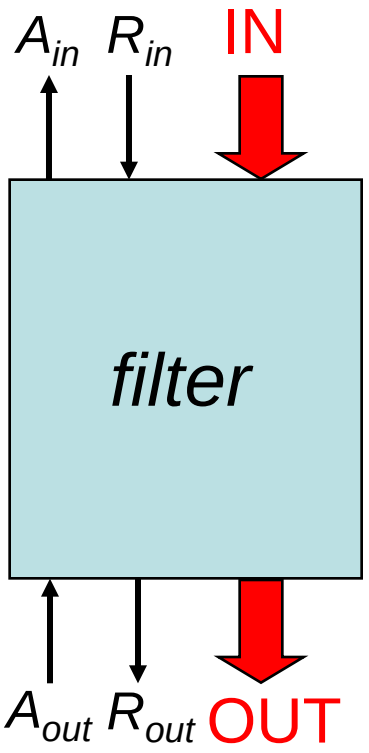
- Bounded delays (BD): realistic for gates and wires.
  - Technology mapping is easy, verification is difficult
- Speed independent (SI): Unbounded (pessimistic) delays for gates and "negligible" (optimistic) delays for wires.
  - Technology mapping is more difficult, verification is easy
- Delay insensitive (DI): Unbounded (pessimistic) delays for gates and wires.
  - DI class (built out of basic gates) is almost empty
- Quasi-delay insensitive (QDI): Delay insensitive except for critical wire forks (*isochronic forks*).
  - In practice it is the same as speed independent



Designing in the small: Control Logic Synthesis

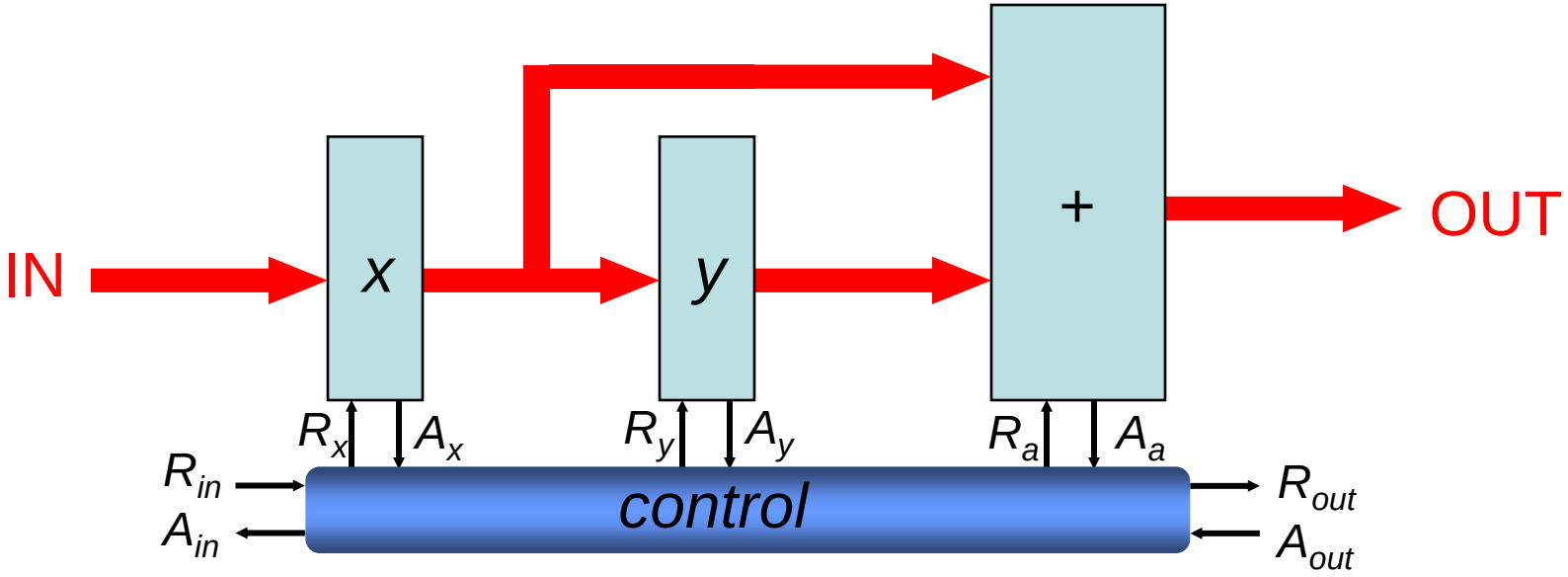
A simple filter example:

```
y := 0;
loop
    x := READ (IN);
    WRITE (OUT, (x+y)/2);
    y := x;
end loop
```



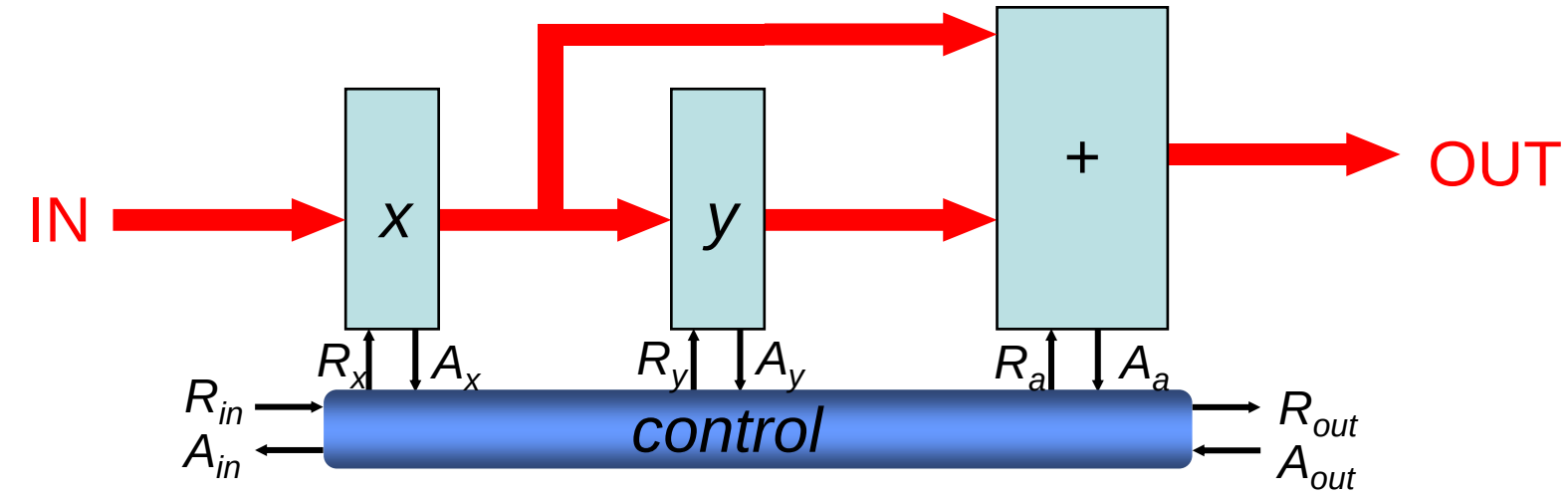
We build the filter by separating the data path and control path. Data path can be designed using dual-rail of bundled data. Control can be synthesized using the Signal Transition Graph model (Rosenblum, Yakovlev, Chu, 1985)

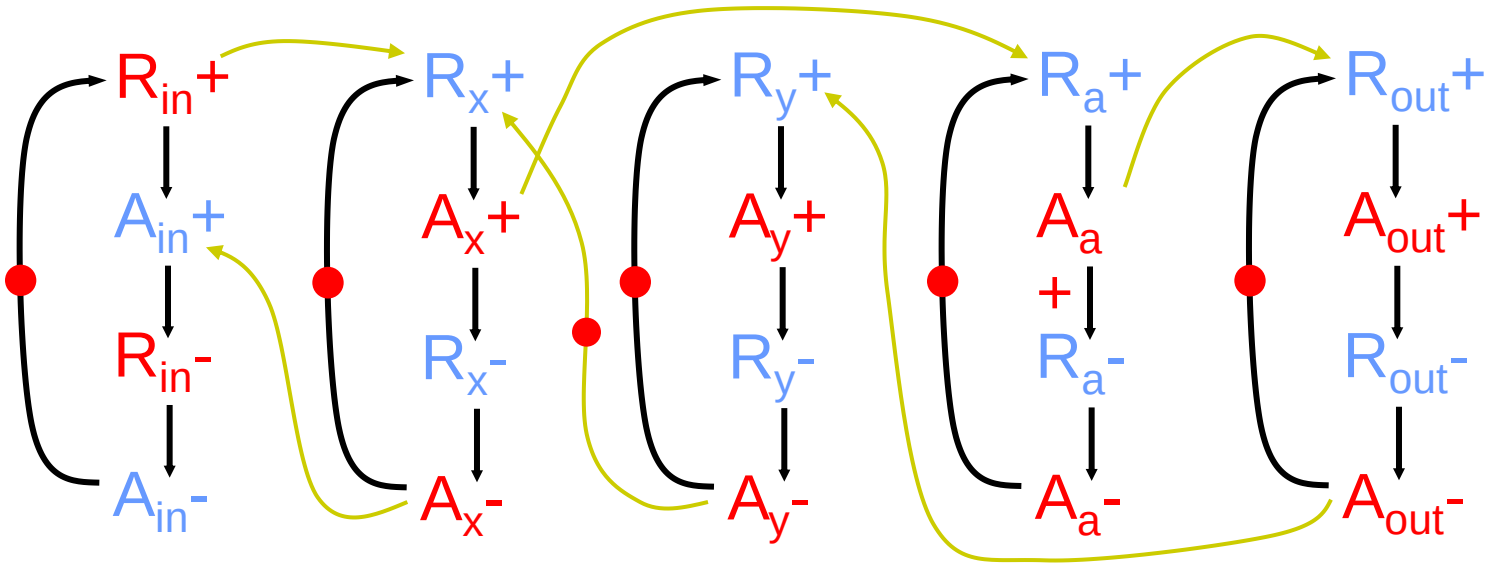
Data path description



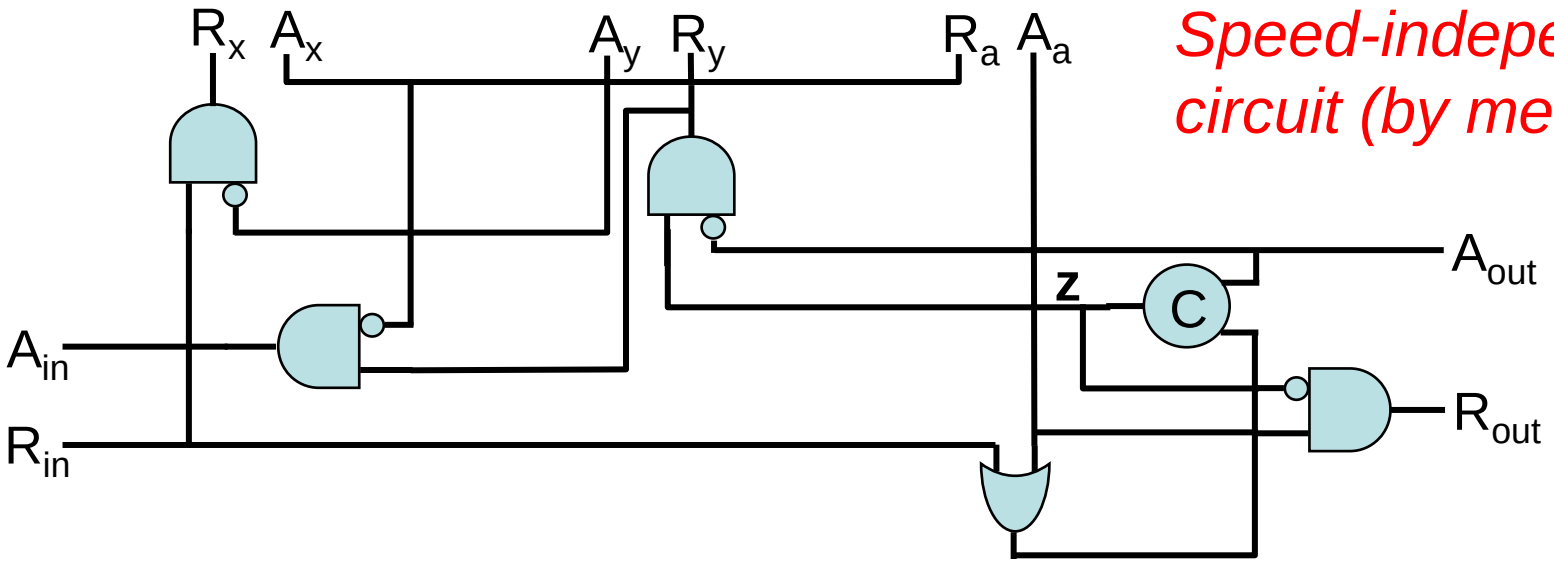
- *x* and *y* are level-sensitive latches (transparent when R=1)
- + is a bundled-data adder (matched delay between R_a and A_a)
- R_{in} indicates the validity of IN
- After A_{in}+ the environment is allowed to change IN
- (R_{out},A_{out}) control a level-sensitive latch at the output

Control specification using Signal Transition Graphs

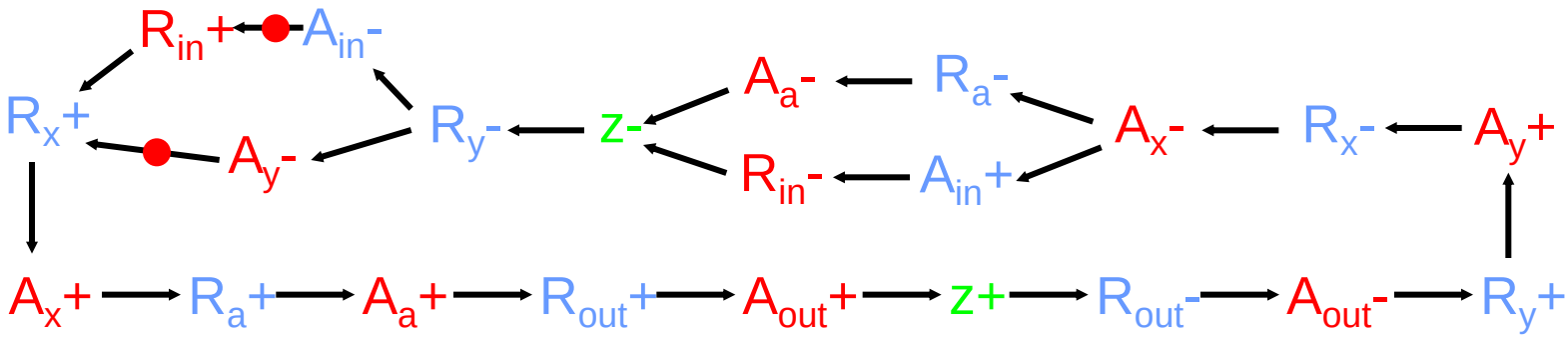




Control implementation

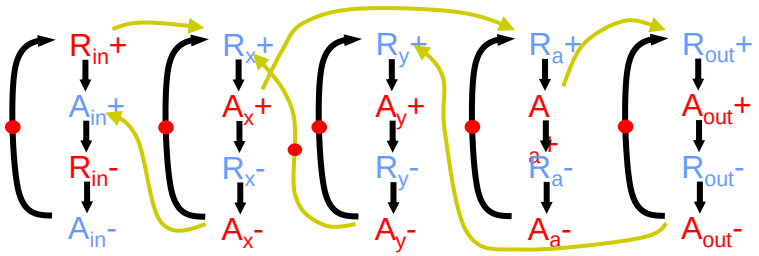


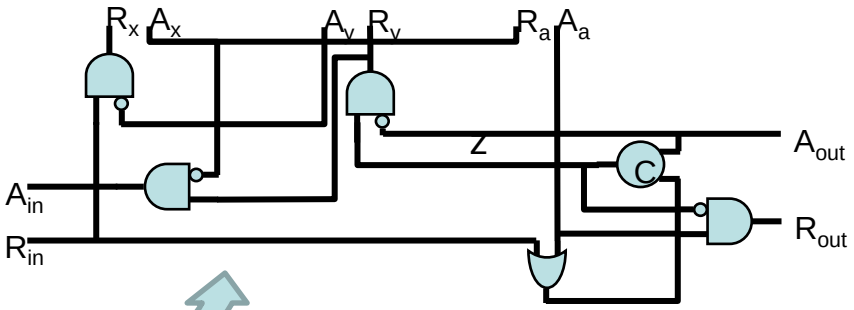
Speed-independent circuit (by method)

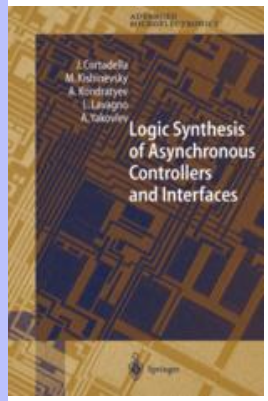


z is internal state signal added by the synthesis method

From STG specification to logic implementation







Synthesis method and Petrify tool

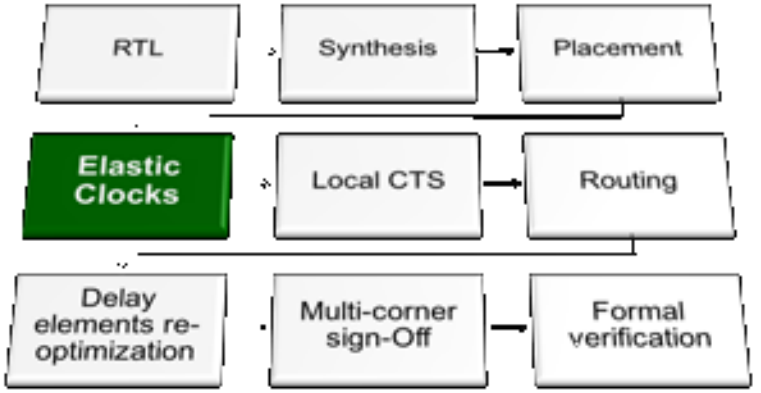
J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002. (Petrify software can be downloaded from: http://www.lsi.upc.edu/~jordicf/petrify/)

It is possible to synthesize circuits with up to 50 signals – limited by state explosion (state space is generated inside the tool)

Recent developments at Newcastle, Barcelona, Augsburg and Potsdam have led to methods based on direct mapping of Petri nets, structural decomposition and use of Petri net unfoldings.

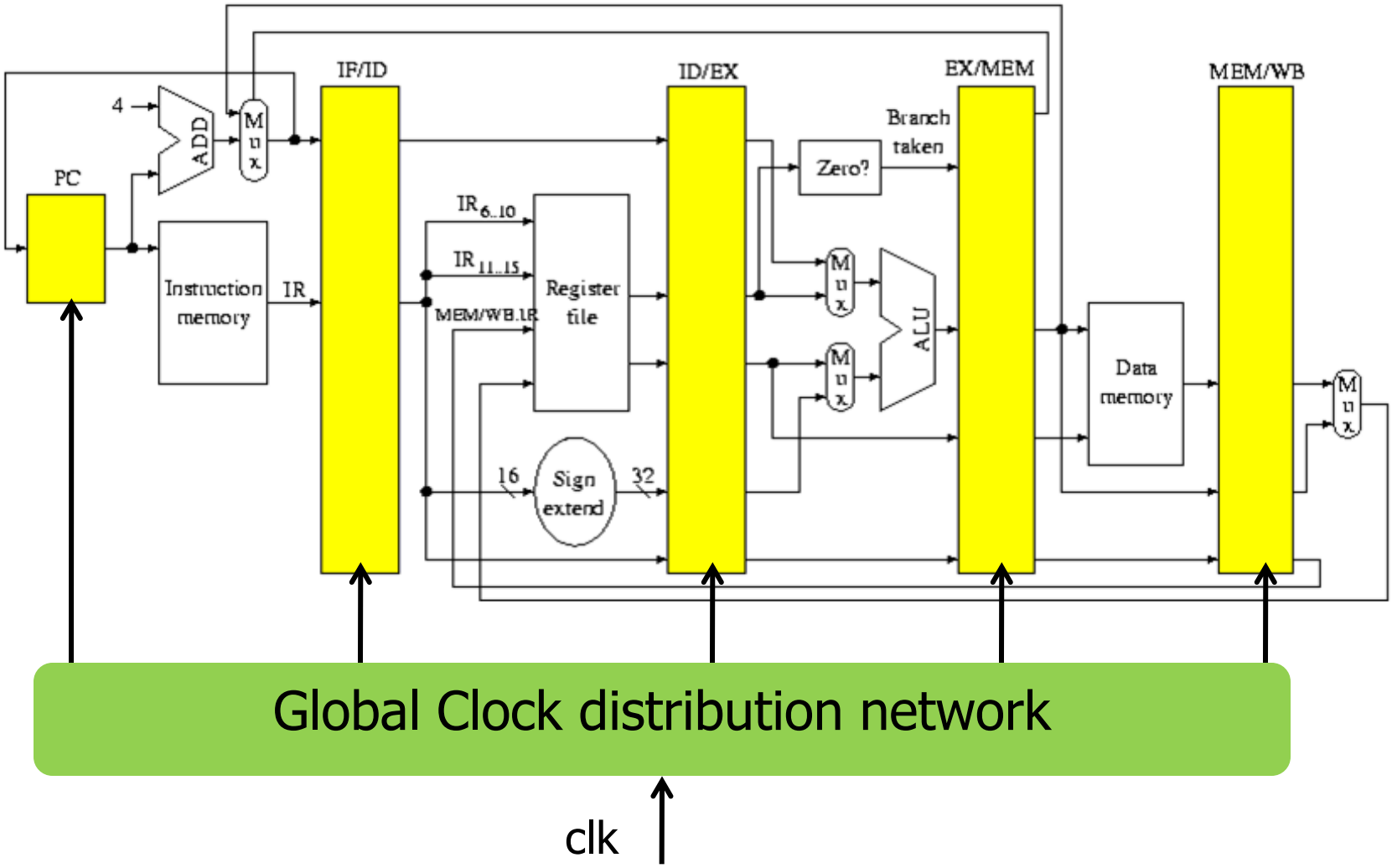
Designing async in the large: "desynchronisation"

- Think synchronous
- Design synchronous:
  - One clock
  - Edge-triggered flip-flops
- De-synchronize (automatically)
  - Remove clock
  - 1 edge-triggered flip-flop = 2 latches
  - Add latch controllers (any mix of "valid" controllers is allowed)
  - Add Joins and Forks
  - Add delay elements
- Run it asynchronously (possible dynamic voltage scaling)

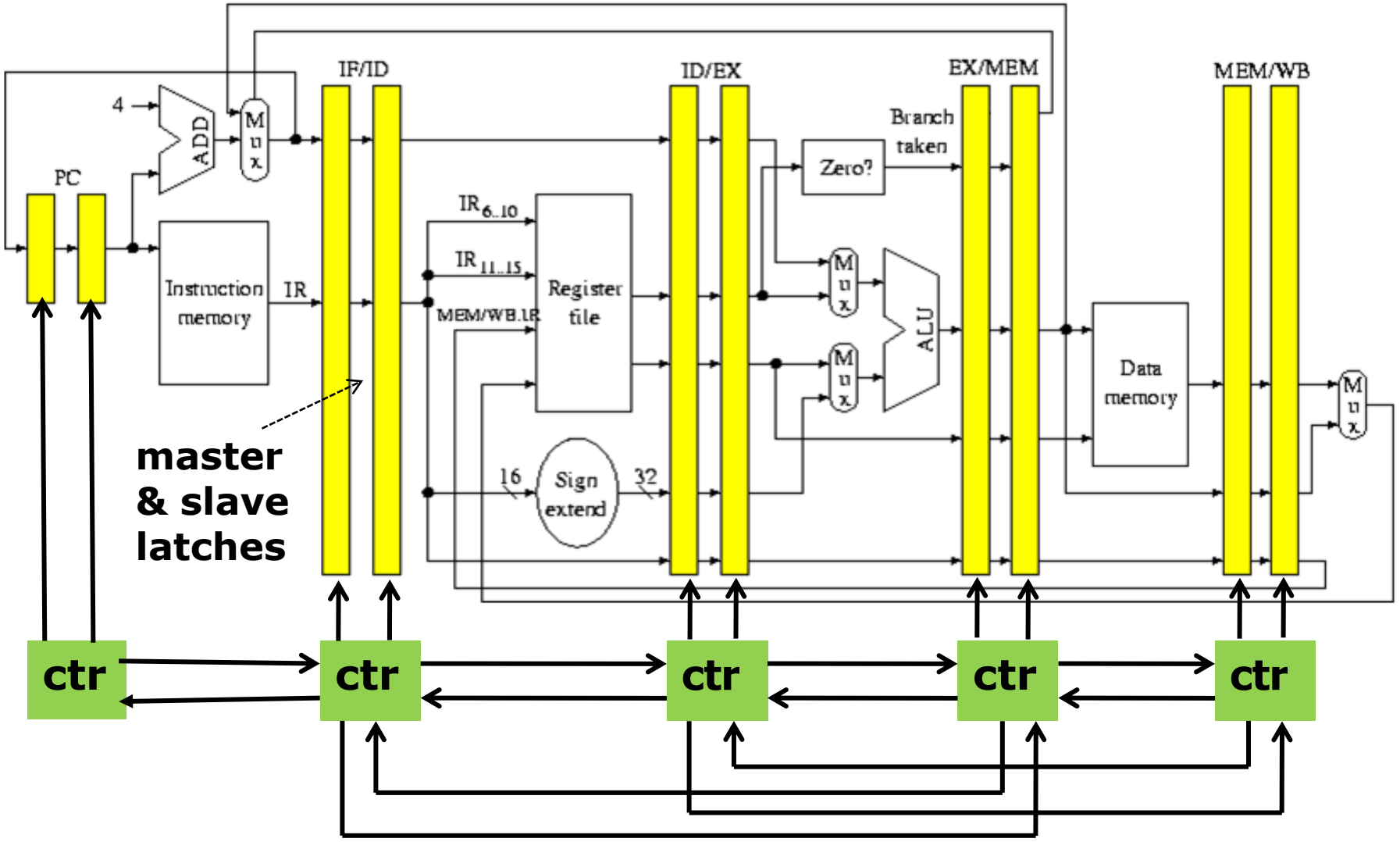


Source: J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou, "Desynchronization: Synthesis of asynchronous circuits from synchronous specifications ", IEEE Transactions on Computer-Aided Design, pp. 1904–1921, October 2006.

Example: Synchronous DLX



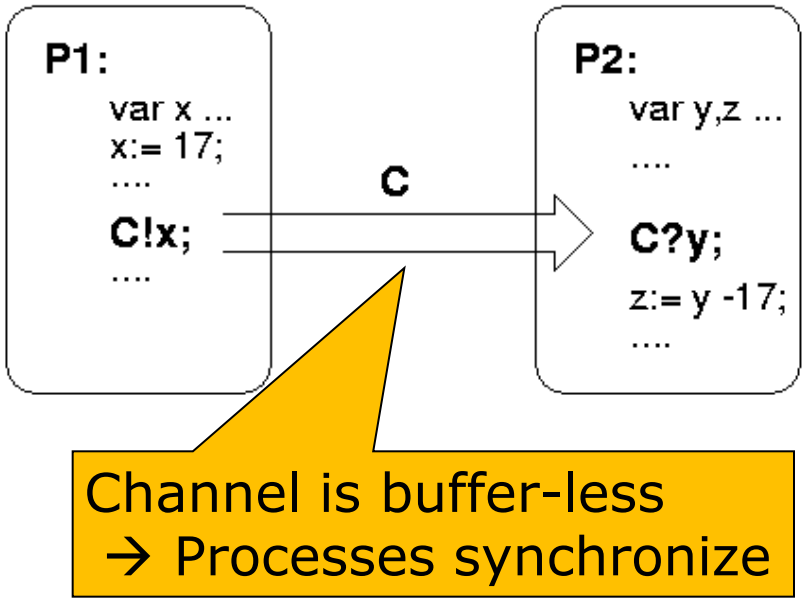
"Desynchronized" (Elastic) DLX



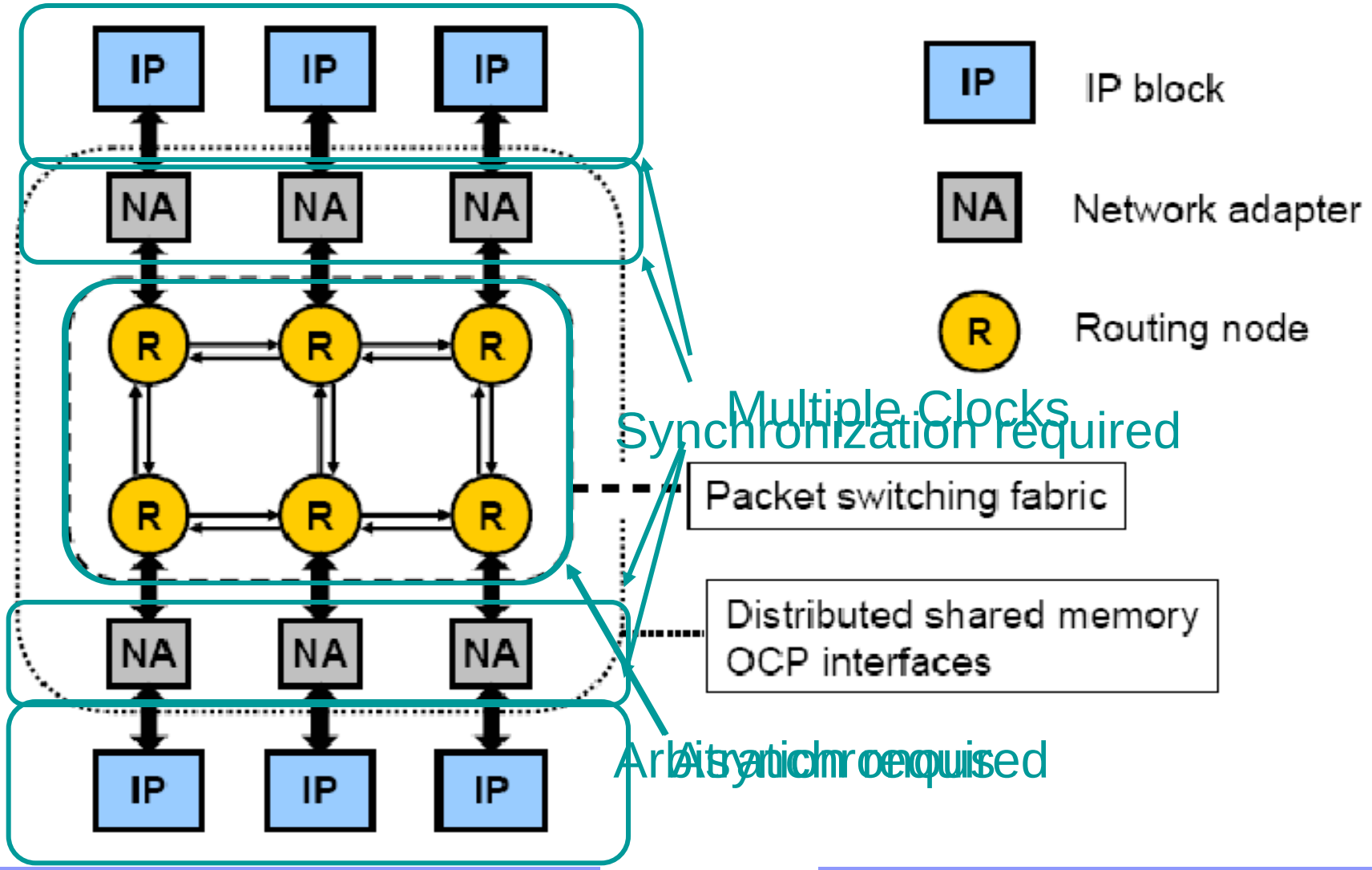
Designing async in the large: use of HDLs

- VHDL or Verilog:
  - Event driven + parallel processes. Fine, but …
  - … "programming" of req-ack handshake is tedious.
- Inspiration from parallel programming languages:

  CSP, OCCAM, …
  - Message passing across communiction channels (Send, Receive, Probe)
- Asynchronous HDL's
  - Haste (Tangram) Handshake Solutions
  - Balsa U. of Manchester
  - CHP Caltech
  - …



Asynchronous Networks (Sparsø, ASYNC 2005)



Back to synchronizers and arbiters

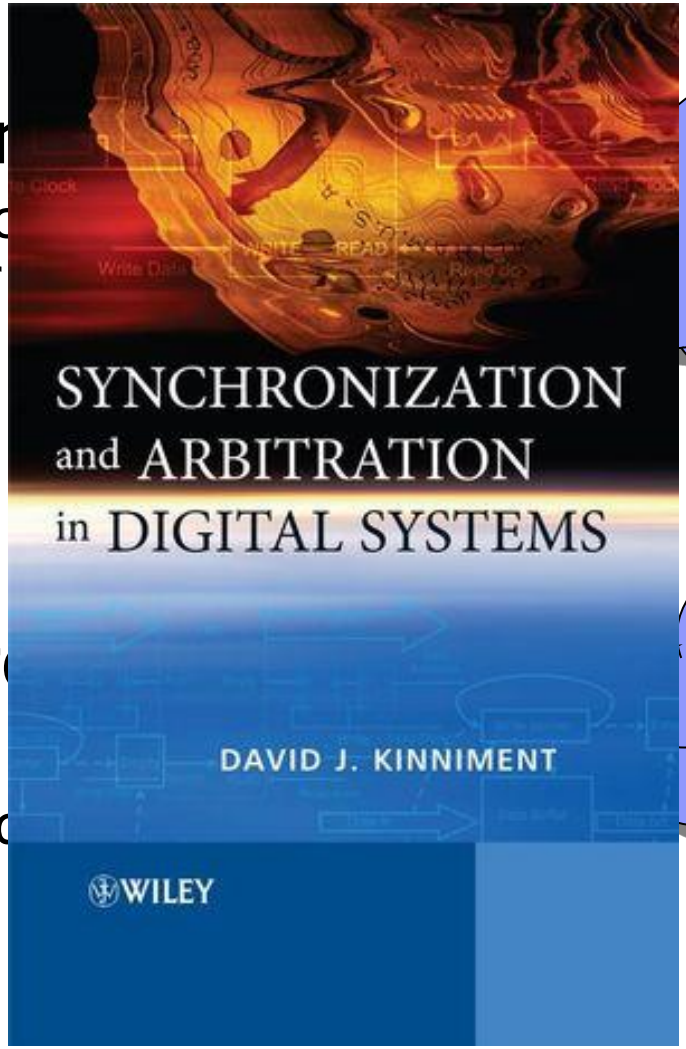
- Synchron

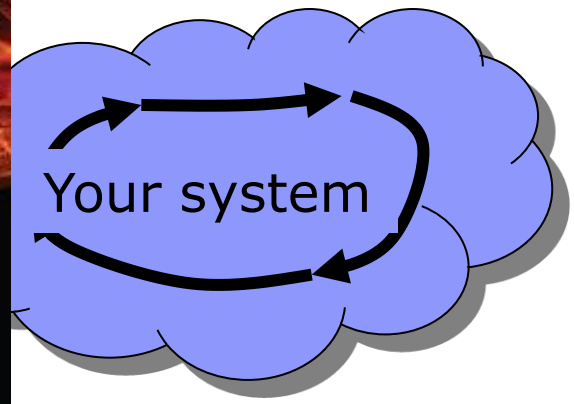
Decides which c
cycle to use for
input data

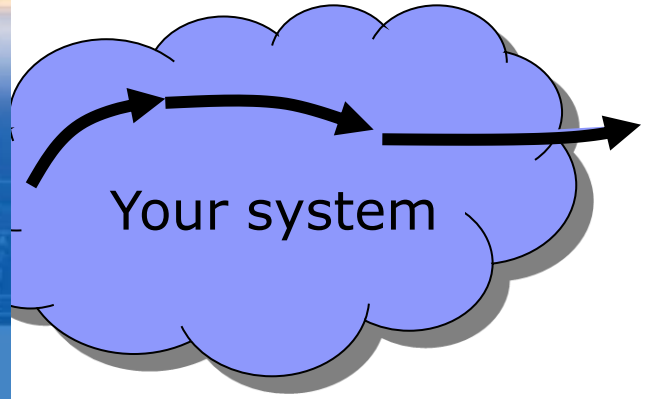
- Asynchr

arbiter

Decides the orc
inputs

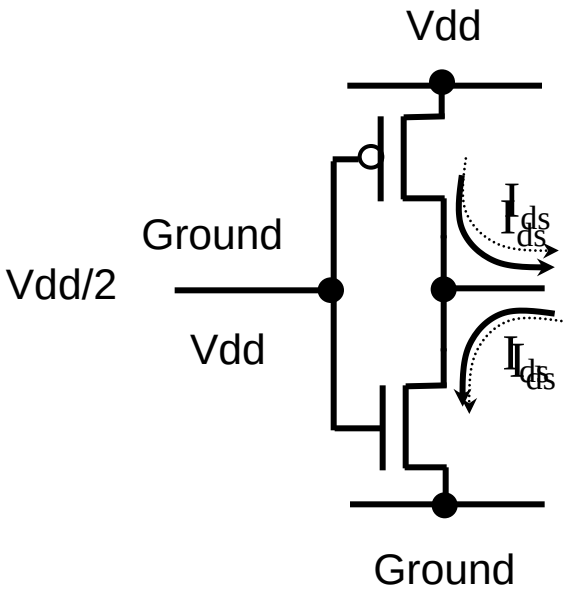






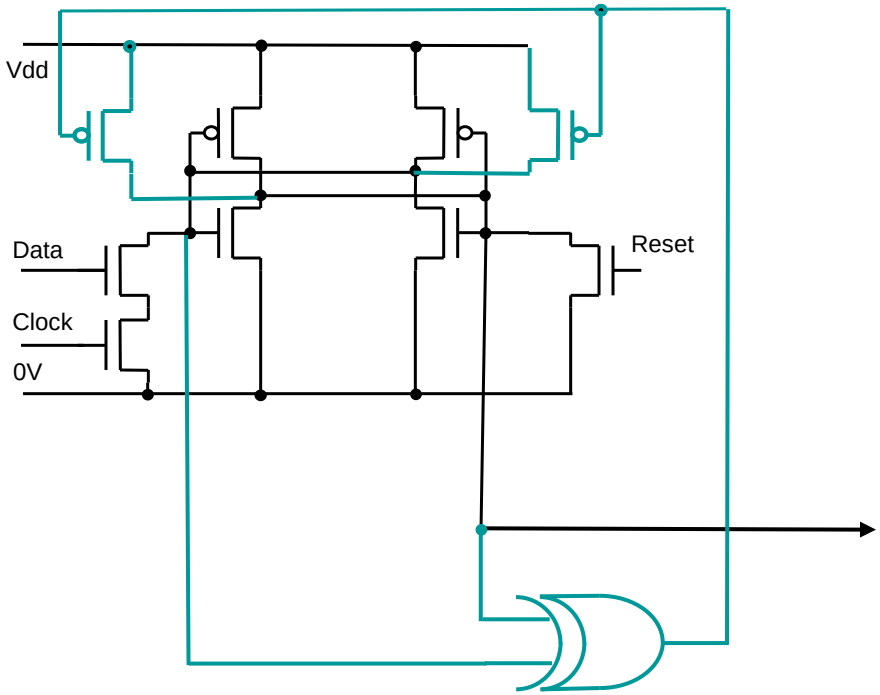
Synchronisers and future processes

- Synchronizers and arbiters don't work well in nanometre technologies, especially at low Vdd
- Worse than gates! Why?
- A gate input is either HIGH
  - Output pulled down
- Or LOW
  - Output pulled up
- A metastable gate is neither
  - Both transistors can be off
- Vdd/Vth decreases with process shrink,
  - Transconductance g_m very low
- Synchronization time constant $\tau = C/g_m$



Robust synchronizer (Zhou et al, 2006)

- Jamb latch synchronizer slow for low V_{dd}, low temp
- In metastability, both outputs are the same
- Extra p-types are switched fully on, so g_m increases, and τ improves
- Because extra p-types are only on during metastability, set/reset transistors can now be small and do not load latch



History – "classics"

- 1950s and early 60s:
  - Theory of Languages, Automata, Switching Circuits (Kleene, Mealy, Moore)
  - Asynchronous Automata and Switching Circuits (Huffman)
  - Speed-Independent Circuits (Muller, Bartky) – analysis, early synthesis, compositionality
  - Building real async computers using SI circuits (!) from first transistors (Illiac II)

  Leitmotiv: theory of asynchronous switching behaviour
- Late 60s, early 70s:
  - Building real async computers in TTL and ECL (DEC PDP-16, MU-5, Atlas)
  - Metastability and synchronisation (Catt, Molnar, Chaney, Kinniment)

  Leitmotiv: observing asynchronous phenomena in vivo

History – "middle ages"

- 70s:
  - Macromodule project at Washington University St. Louis (Clarke, Molnar);
  - Design methods based on data/control-flow graphs, Petri nets, project MAC at MIT (Dennis, Holt, Patil)
  - Aperiodic Automata, USSR Academy of Science (Varshavsky et al)
  - Leitmotiv: modularity to cope with complexity and concurrency
- 80s:
  - VLSI (clocked design dominates in industry and leads to mature CAD);
  - Self-timed circuits in massively parallel architectures to get speed (Seitz); Postoffice design (Davis et al); Micropipelines (Sutherland)
  - Early foundations for Async CAD (analysis and synthesis),interfaces (Varshavsky et al); Signal Transition Graphs (Rosenblum et al, Chu)
  - Leitmotiv: async is pushed on the fringe: first experiments in VLSI and understanding that CAD is crucial

History – modern times

- 90s:
  - Theory of delay-insensitive circuits (Josephs, Udding)
  - CAD developments: verification (Ebergen), synthesis and Petrify tool (Cortadella et al, Nowick), syntax-direct compilation and Tangram at Philips (van Berkel)
  - Processor Designs (Martin, Furber); design experiments at HP, Sun and Intel (RAPPID, 1999)

  Leitmotiv: The main thrust is that async is good for Low power
- 2000s:
  - High-speed designs for internet switches (Fulcrum)
  - GALS, Asynchronous Interconnects, NoCs (Vivet et al, Bainbridge)
  - Synchronizers for SoCs (Kinniment, Ginosar, Greenstreet)
  - "More practical" CAD: desynchronisation (Cortadella et al), Haste (Peeters)

  Leitmotiv: Robustness against variability; mitigating timing closure

History - summary

- Early stages: relatively small circuits that could be designed by hand from specification, either by synthesis, or by direct mapping, or by assemble-verify techniques, and clock was not needed – good times for ASYNC!
- Clock was introduced into the design practice later to avoid the complexities of dealing with causal relations
- Clock fitted better in the FSM approach, and allowed to avoid complex procedures with hazard-elimination or avoidance
- But, with concurrent processes and Petri nets, clock is becoming a trouble, even though we can keep measuring time and performance
- With NoCs, variability, timing closure and power problems , global clocking is a problem – good times for ASYNC!
- PLUS: many people design asynchronous circuits but do NOT admit that – name "asynchronous" implies something negative …

Who is Who in Async : Industry

- Europe
  - Elastix, Spain – USA/CA
  - Handshake Solutions, Netherlands
  - Tiempo, France
  - Silistix, UK
- USA
  - Achronix,
  - Fulcrum,
  - IBM,
  - Intel,
  - Timeless,
  - Theseus *(if still exists?)*
  - Sun - Oracle

⇒ Mostly startup companies on CAD tools & some circuit niches

⇒ A few R&D labs within major companies (IBM, Intel, Sun)

Who is Who in Async: Academia

- In the USA
  - Caltech
  - Columbia Univ.
  - Cornell Univ.
  - Portland State University (ARC lab)
  - Univ. North Carolina at Chapel Hill
  - Univ. of British Columbia (Canada)
  - Univ. of South California
  - Univ. of Utah
- In Europe
  - CEA-LETI, France
  - IHP, Germany
  - Cambridge Univ., UK
  - Newcastle Univ., UK
  - Manchester Univ., UK
  - Politechnico de Torino, Italy
  - Technical University of Denmark, Denmark
  - Technion, Israel,
  - TIMA, France
  - TU Vienna, Austria
  - UPC, Spain
- In Japan
  - Himeji Institute of Technology
  - University of Tokyo

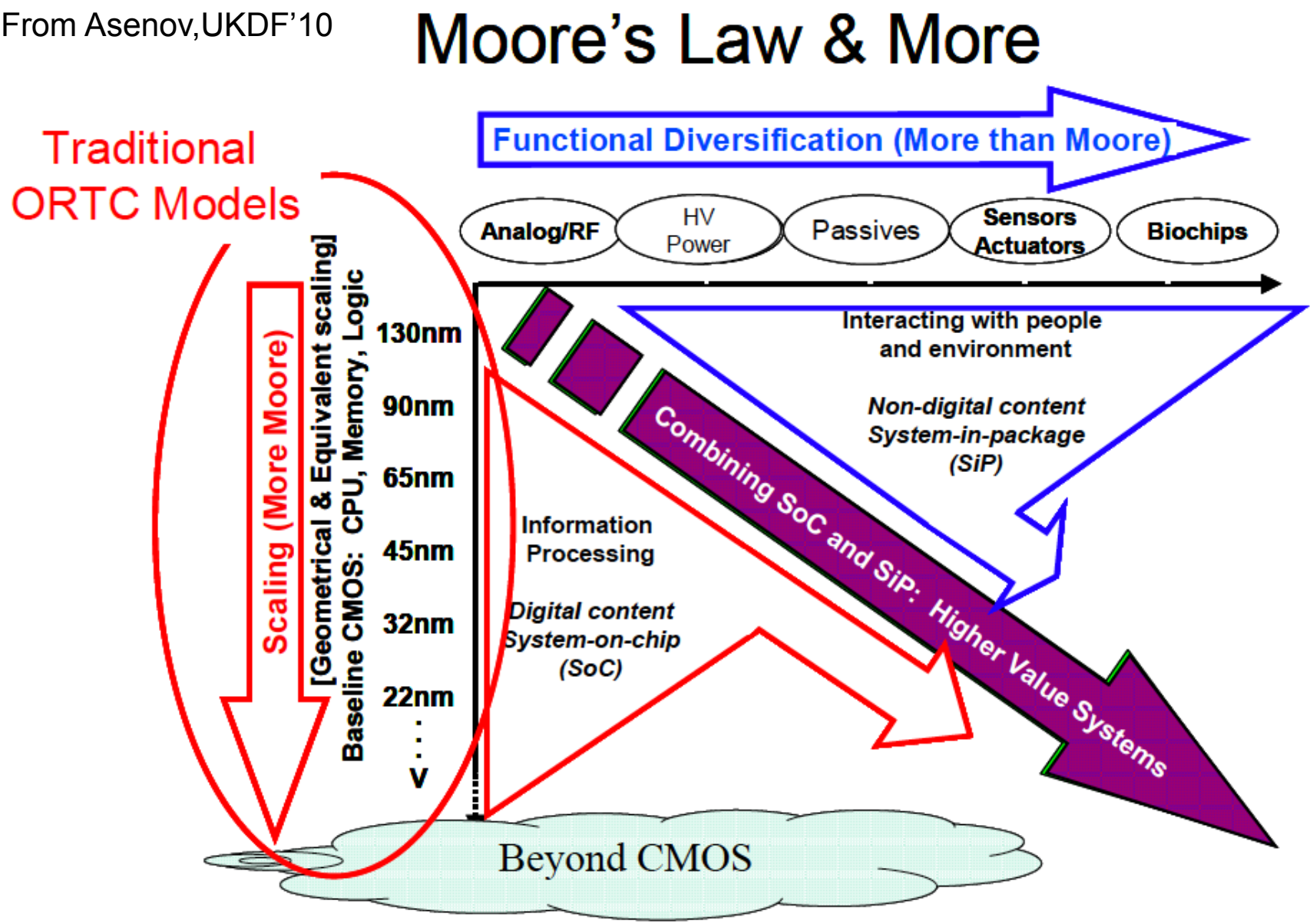
⇒ About a few hundred people around the world…

⇒ ASYNC IEEE symposia running annually since 1994

Where do we go next?

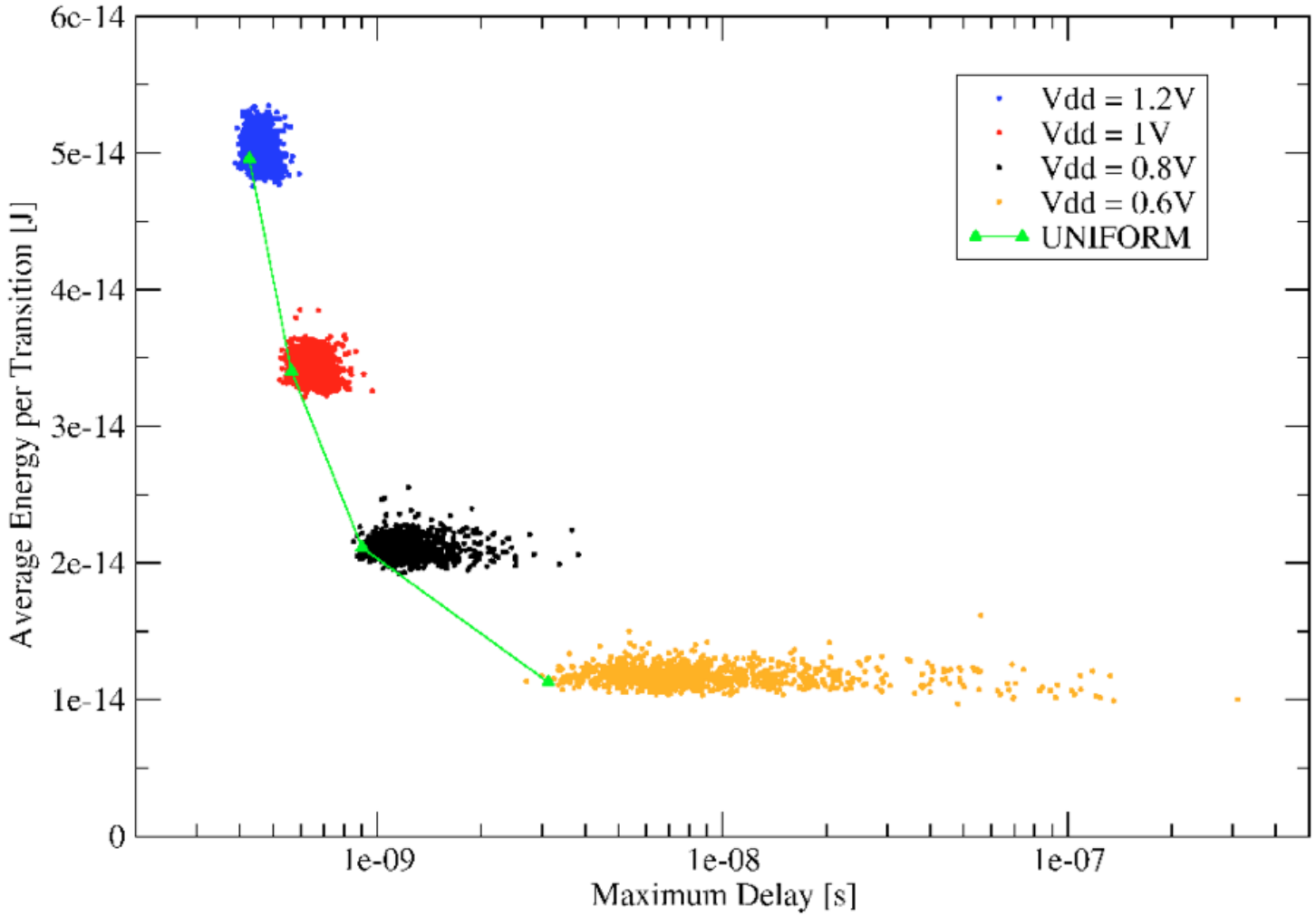
Technology Trends

From Asenov,UKDF'10



Performance/power/yield trade off

From Asenov,UKDF'10

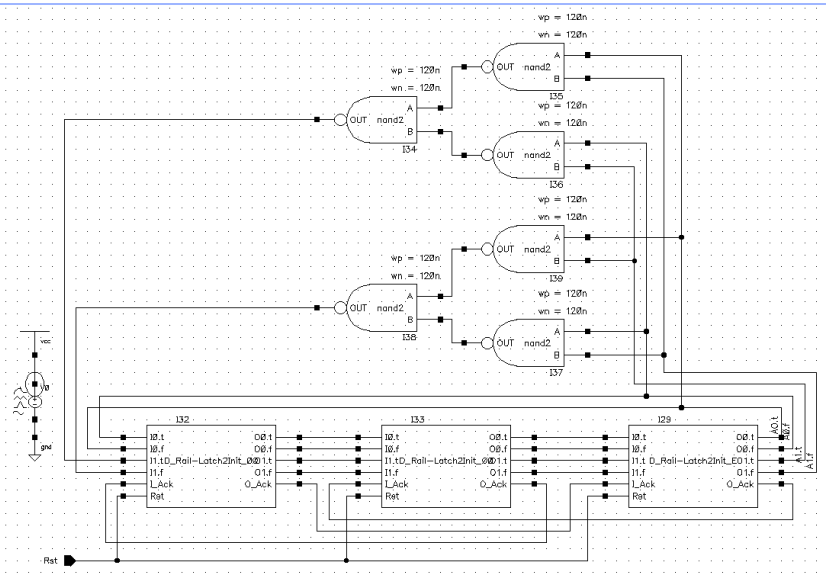


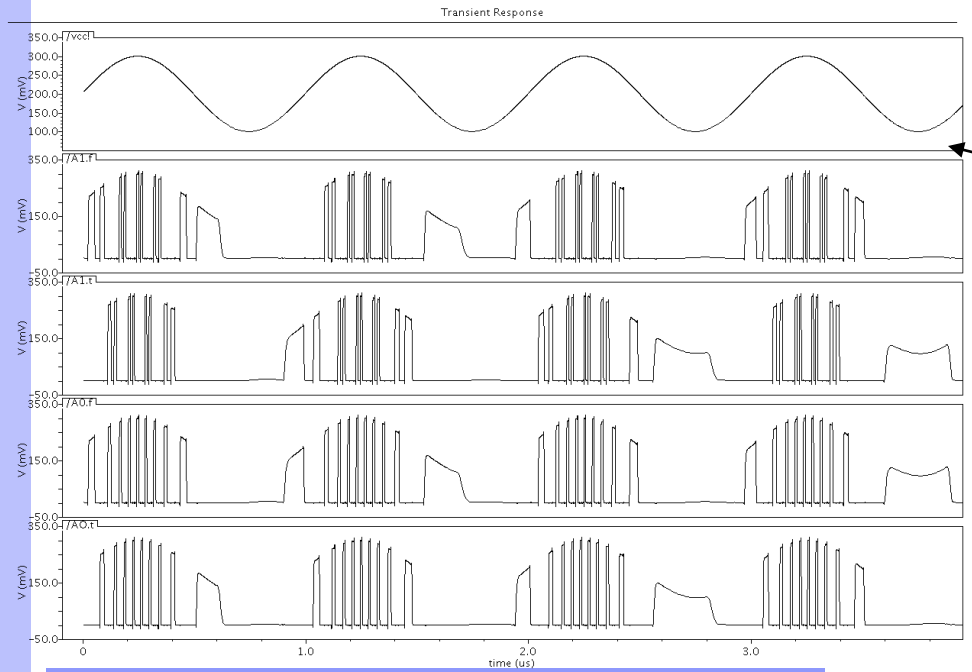
Outlook for asynchronous ...

- New models
  - Combining choice, concurrency and causality better (e.g. Conditional Partial Order Graphs – CPOGs)
  - Models for elastic behaviour (with or w/o clocking)
  - Hybrid, discrete-continuous-stochastic (to enable design for variability and uncertainty – e.g. design to a certain MTBF level)
- Time and power-elastic interfaces
  - Bio to silicon and silicon to bio (e.g. synthesis of comm channels between brain and prosthetic limbs or other actors)
- Energy-proportional, power-adaptive designs
  - Energy-harvesting based systems (design from the premise that power is constrained and performance optimised; traditional: performance constrained, power is optimised)
- Designs resilient to transient defects (SEUs) and adaptive to parametric instability and degradation
- And ... let's remember ASYNC design works well for "small circuits" !!!

Example: Asynchronous Logic for AC supply

2-bit Sequential Dual-rail Asynchronous Counter





A1.f

A1.t

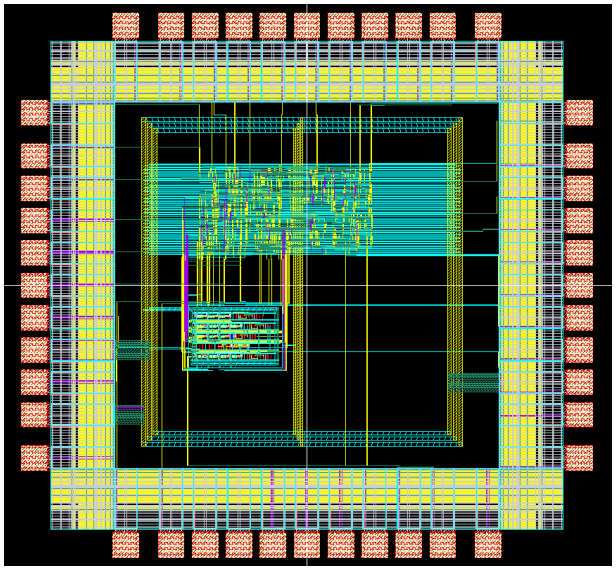
A0.f

A0.t

Supply: AC
200mV±100mV
Frequency: 1Mhz

MSD Group at Newcastle

- Academic Members:
  - *Alex Bystrov, Graeme Chester, Nick Coleman, David Kinniment, Albert Koelmans, Gordon Russell, AlexYakovlev*
- Research Associates:
  - *Frank Burns, Fei Xia, Danil Sokolov, Delong Shang, Julian Murphy, Basel Halak , Andrey Mokhov, Ivan Poliakov*
- 25+ postgraduate research students
- Close Links to CS school Theory of Concurrent and Async Systems Group – *Maciej and Marta Koutny and Victor Khomenko*



Metastability test chip (Newcastle 2006)

Group's Current Project Map

Levels of abstraction
(classes of objects studied)