How to design little digital, yet highly concurrent, electronics?

Alex Yakovlev

Newcastle University

Newcastle upon Tyne, U.K.

Outline

- **“Little” Digital electronics: Why going asynchronous?**
- **Six Asynchronous Design Principles**
- **(Some of the) Models, Techniques and Tools for Asynchronous Design**
- **Asynchronous control logic synthesis using STGs**
- **Case study: Async control for power converter**
- **Messages to take away**

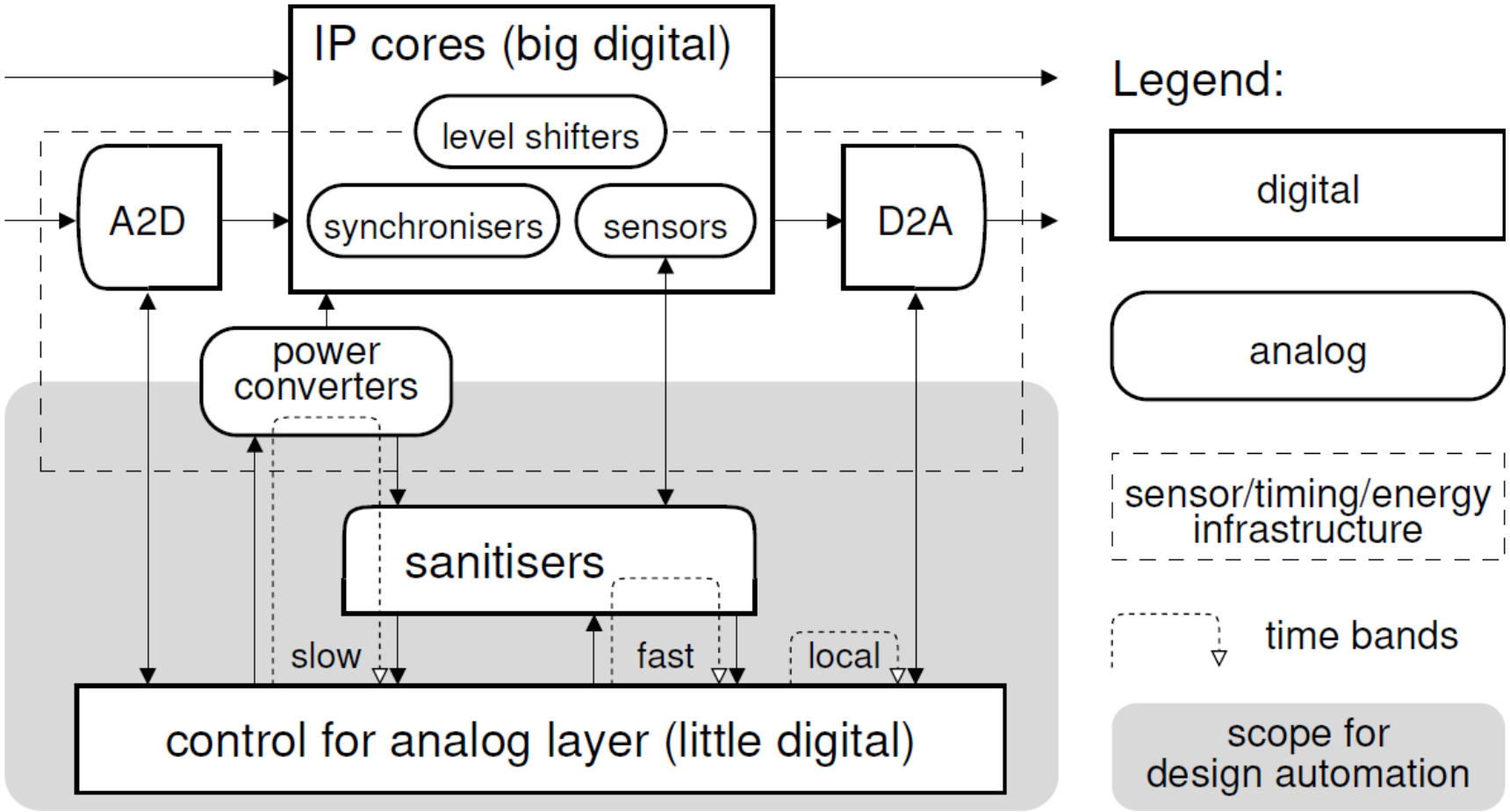
Asynchronous Behaviour

- **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.**

Parallel vs Concurrent

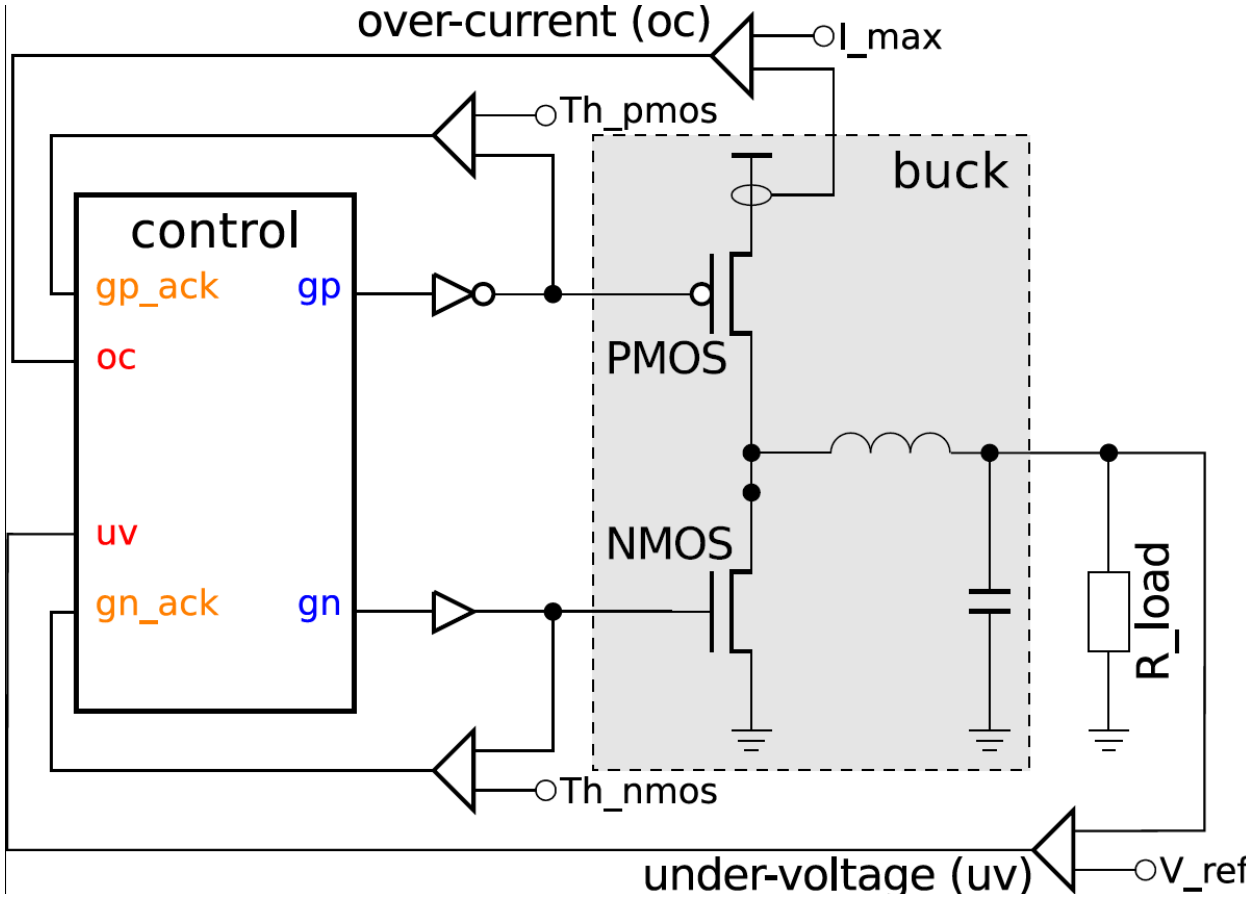
- **Synchronous circuits can be VERY parallel (executing many things at the same time!) but NOT concurrent (independently firing events), because clock is sequential, and actions are done in total SYNC – hence total, instead of partial, order**
- **Asynchronous circuits are fundamentally concurrent, they are self-timed – i.e. many clocking threads which synchronize by themselves in many different ways – hence partial order**

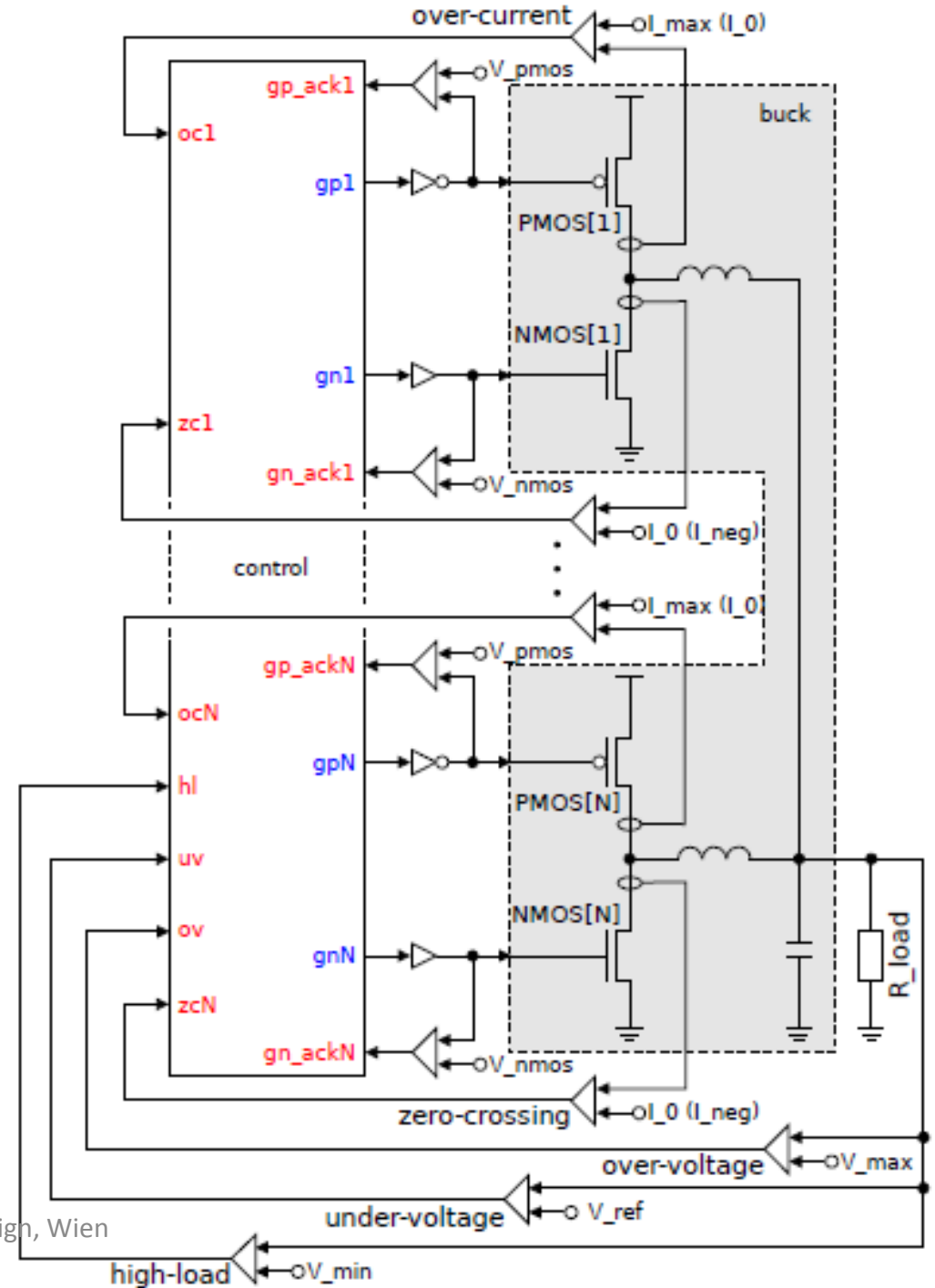
Emergence of little digital electronics



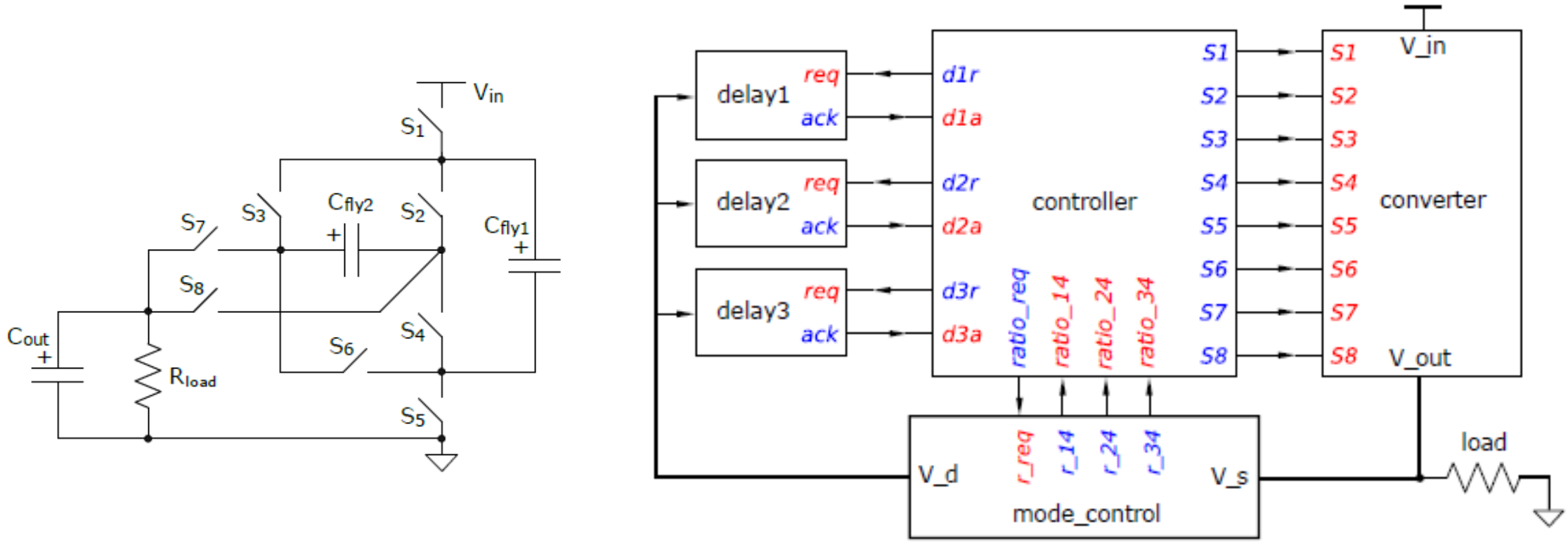
- **Analog and digital electronics are becoming more intertwined**
- **Analog domain becomes complex and needs digital control**

Example: Buck (DC-DC) converter control





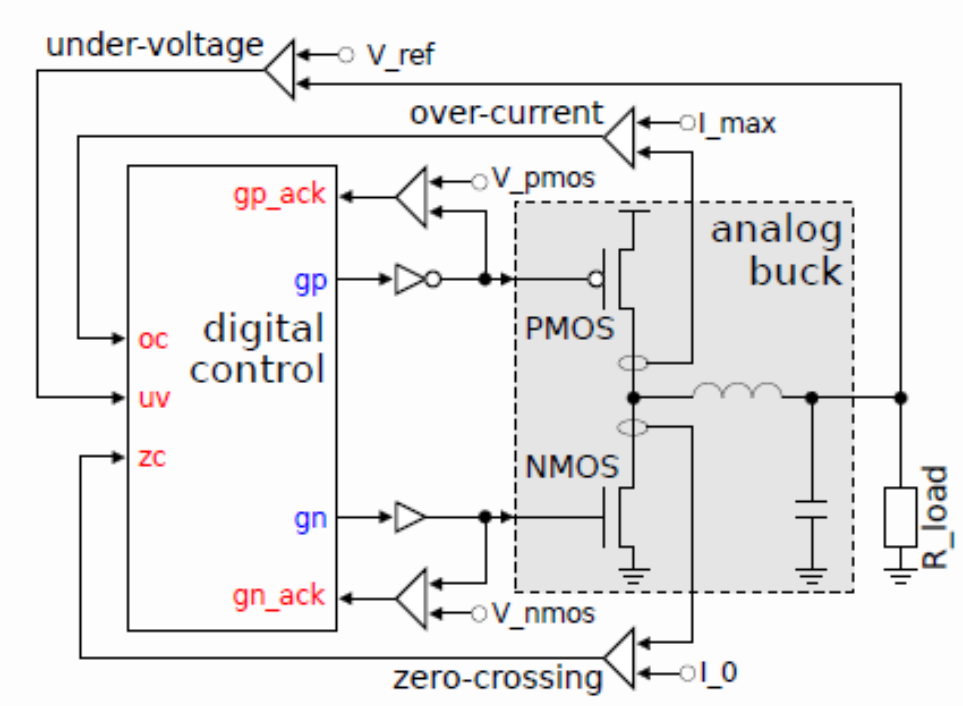
Example: Switched Capacitor (DC-DC) Converter control



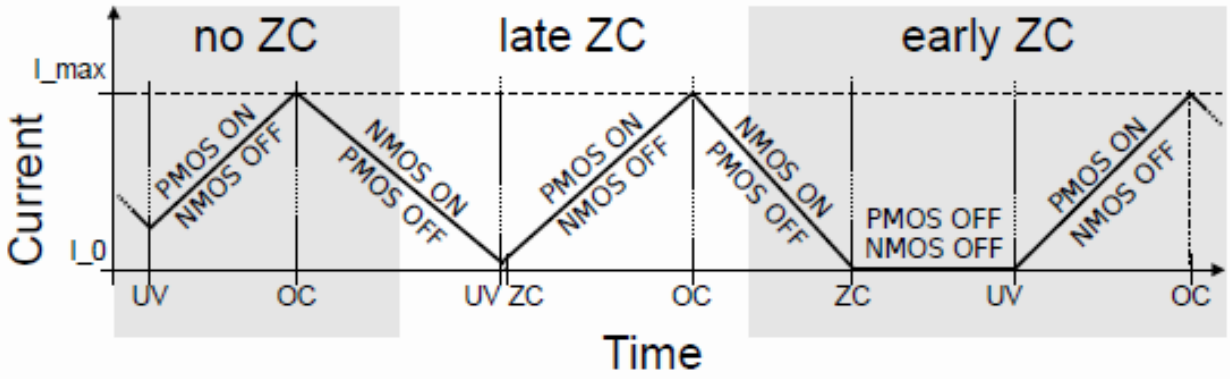
Asynchronous vs Synchronous for little digital

- Synchronous control
  - ☺ Conventional RTL design flow
  - ☹ Slow response (defined by the clock period)
  - ☹ Power consumed even when idle
  - ☹ Non-negligible probability of a synchronisation failure
- Asynchronous control
  - ☺ Prompt response (delay of few gates)
  - ☺ No dynamic power consumption when inactive
  - ☹ Non-conventional methodology and tool support

Example: Buck converter



Phase diagram specification:



Building asynchronous circuits in AMS context requires extending traditional assumptions about speed-independence …

Buck conditions:

- under-voltage (UV)
- over-current (OC)
- zero-crossing (ZC)

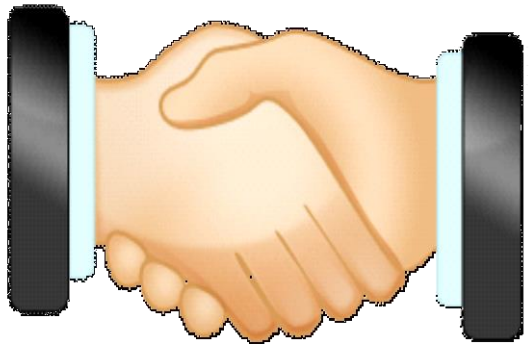
Operating modes:

- no zero-crossing
- late zero-crossing
- early zero-crossing

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)

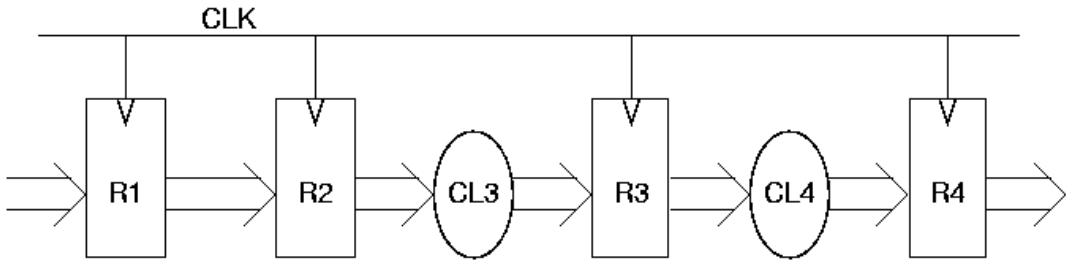
Why and what is handshaking?



Mutual Synchronisation is via Handshake

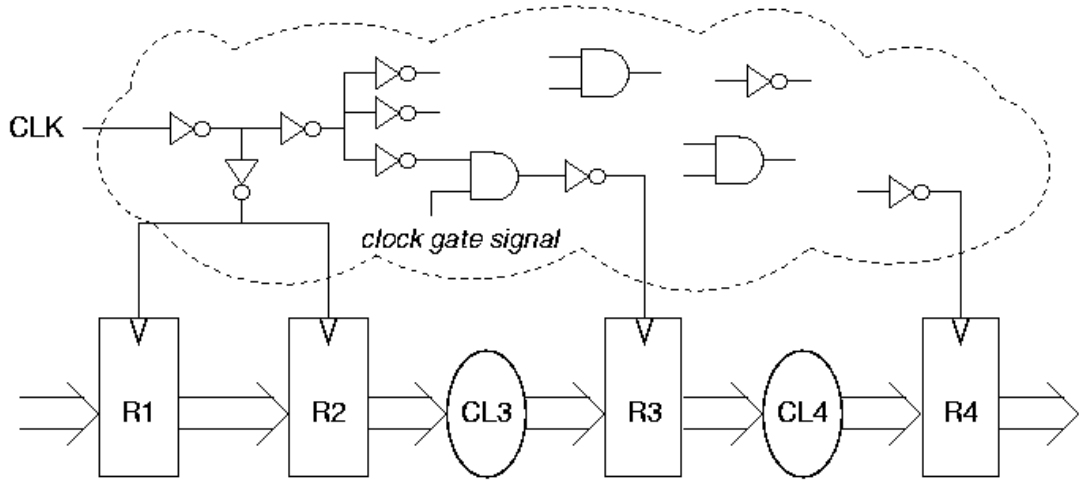
Synchronous clocking

How we think



(a)

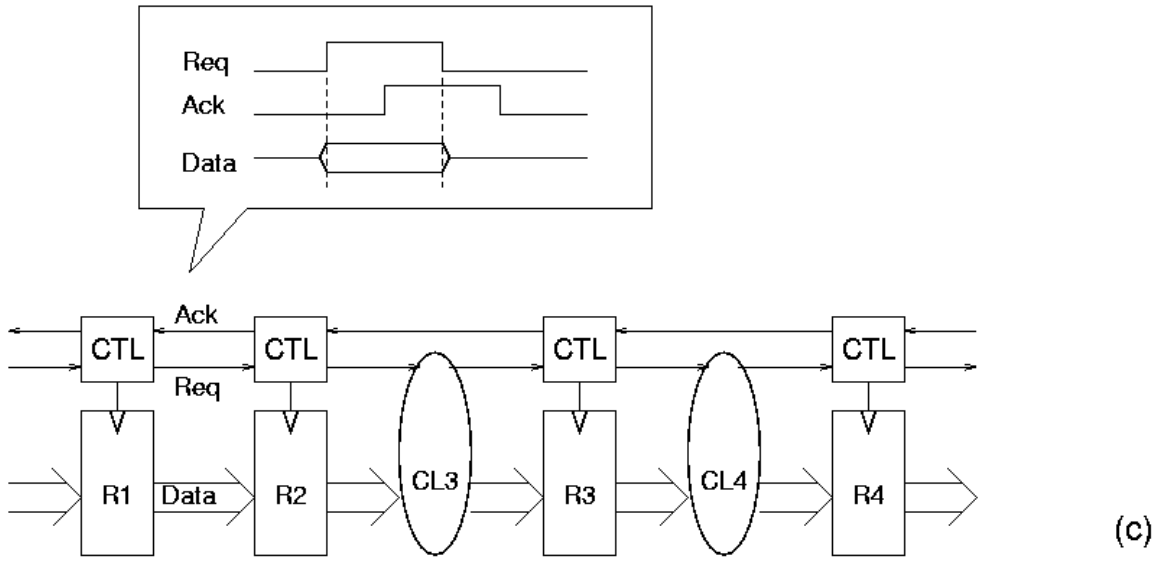
What we design



(b)

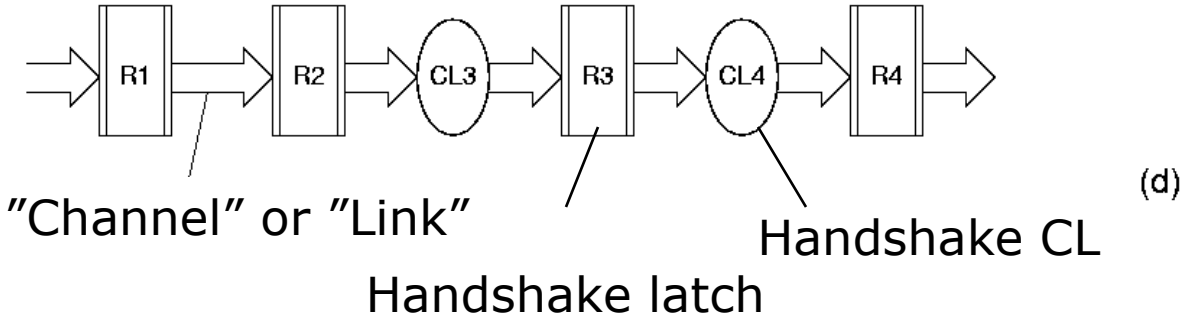
Asynchronous handshaking

What we design



(c)

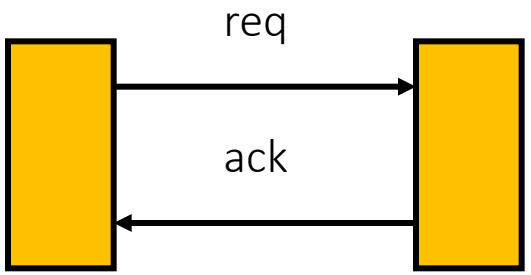
How we think



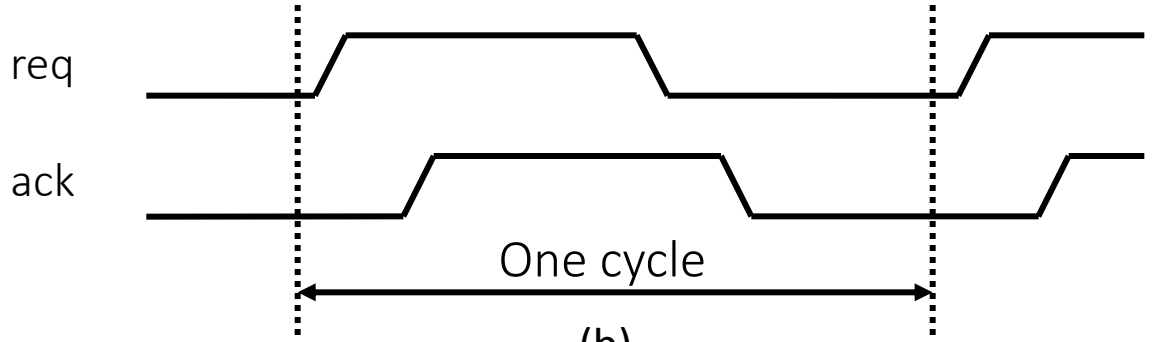
(d)

Handshake Signalling Protocols

Level Signalling (RTZ or 4-phase)

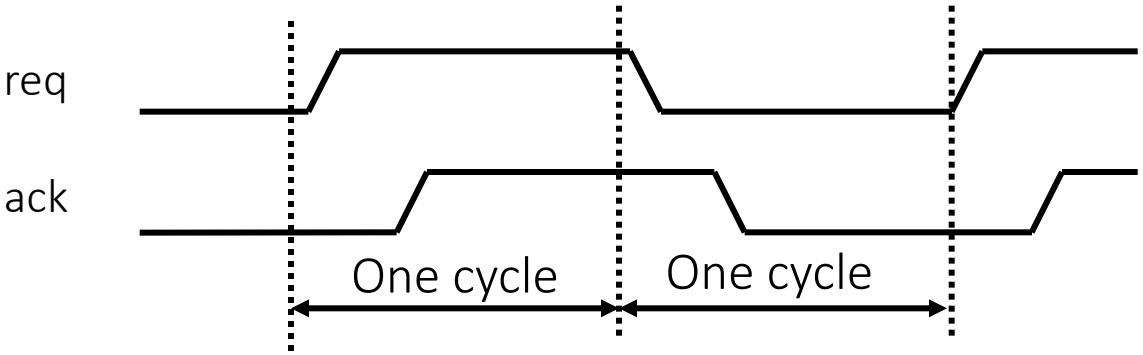


(a)

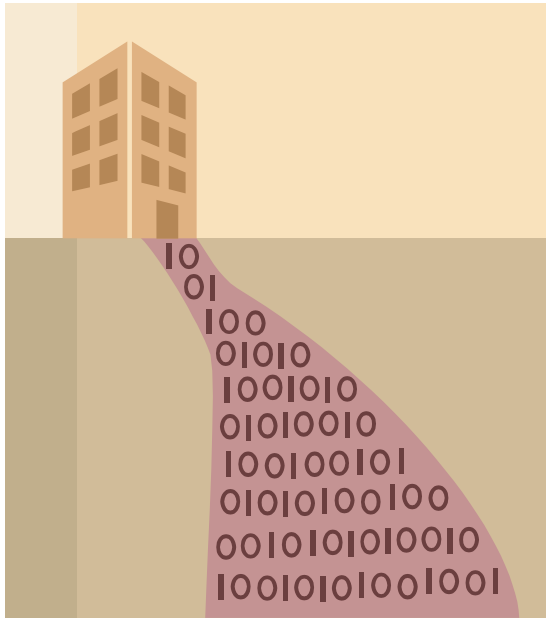


(b)

Transition **Signalling** (NRZ or 2-phase)

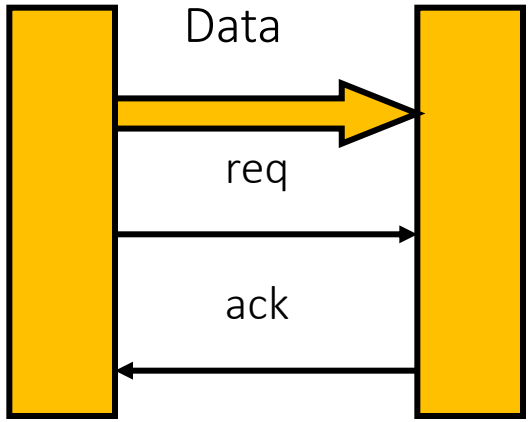


Why and what is delay-insensitive coding?

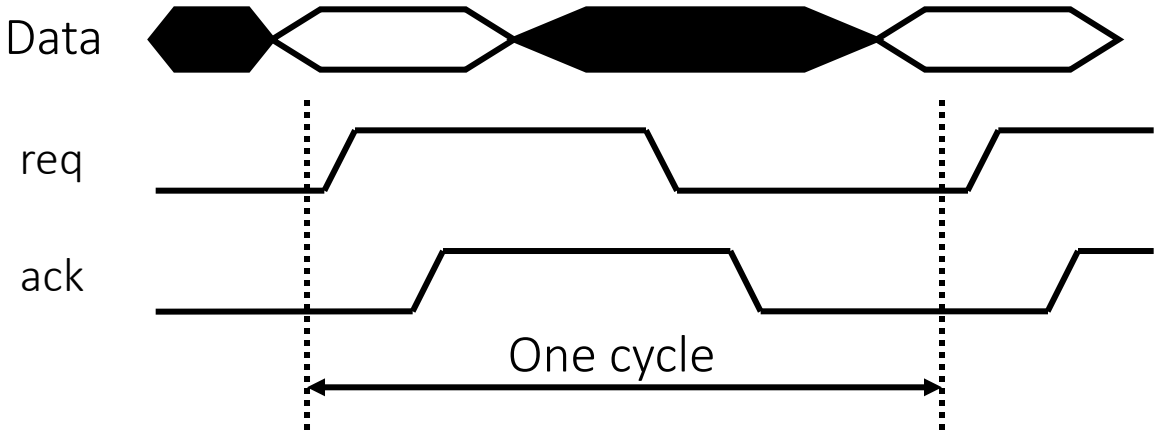


Data Token = (Data Value, Validity Flag)

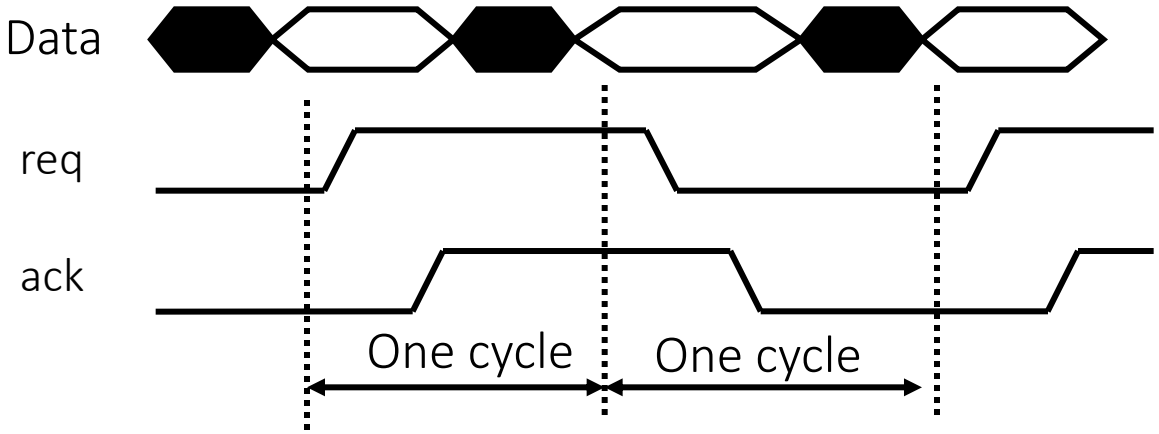
Bundled Data



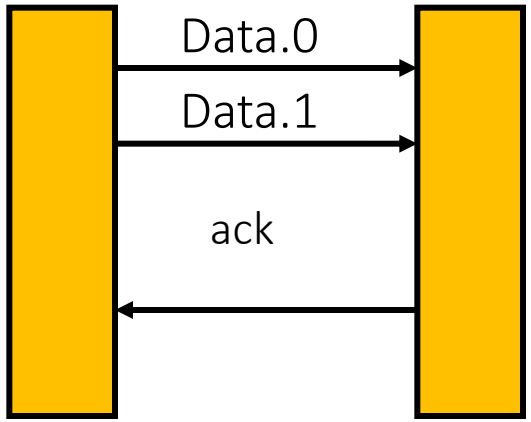
Return to Zero:



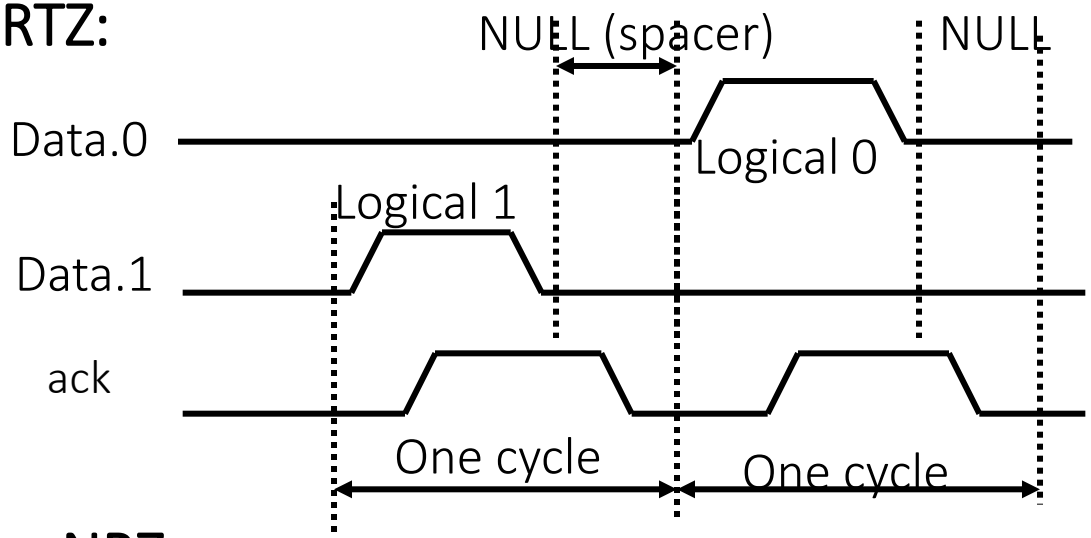
Non-Return-to-Zero



DI encoded data (Dual-Rail)

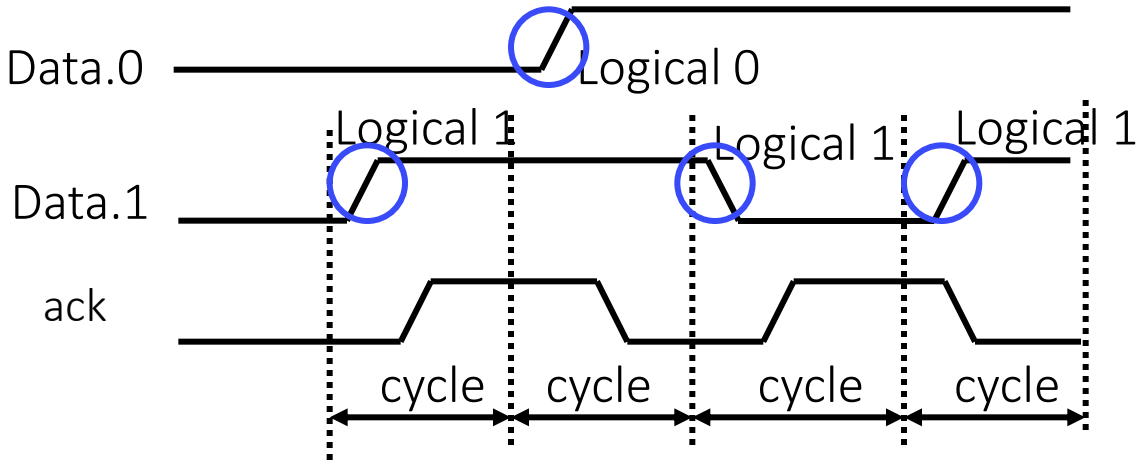


RTZ:



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

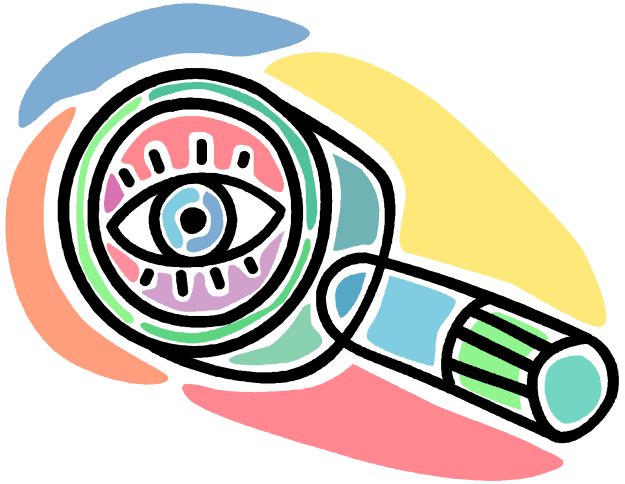
NRZ:



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

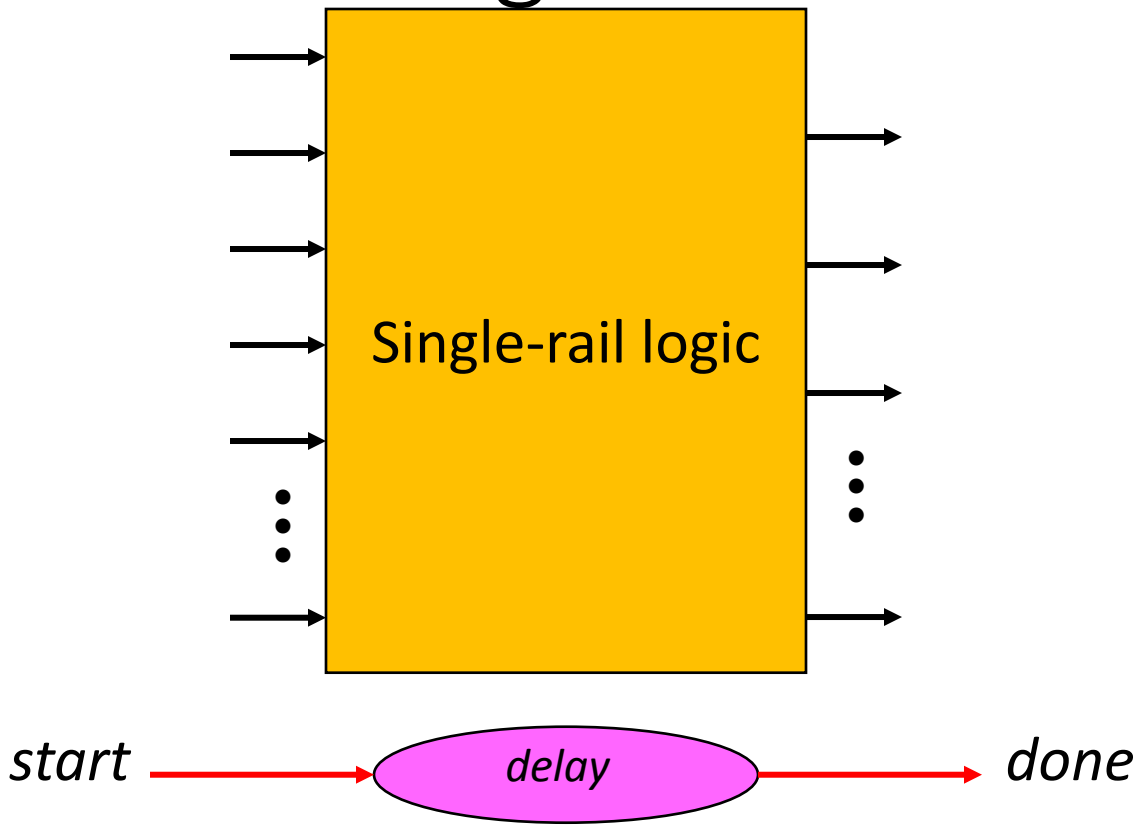
- 1-of-4:
  - 0001=> 00, 0010=>01, 0100=>10, 1000=>11
- 2-of-4:
  - 1100, 1010, 1001, 0110, 0101, 0011 – total 6 combinations (cf. 2-bit dual-rail – 4 comb.)
- 3-of-6:
  - 111000, 110100, …, 000111 – total 20 combinations (can encode 4 bits + 4 control tokens)
- 2-of-7:
  - 1100000, 1010000, …, 0000011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

Why and what is completion detection?



Signalling that the Transients are over

Bundled-data logic blocks

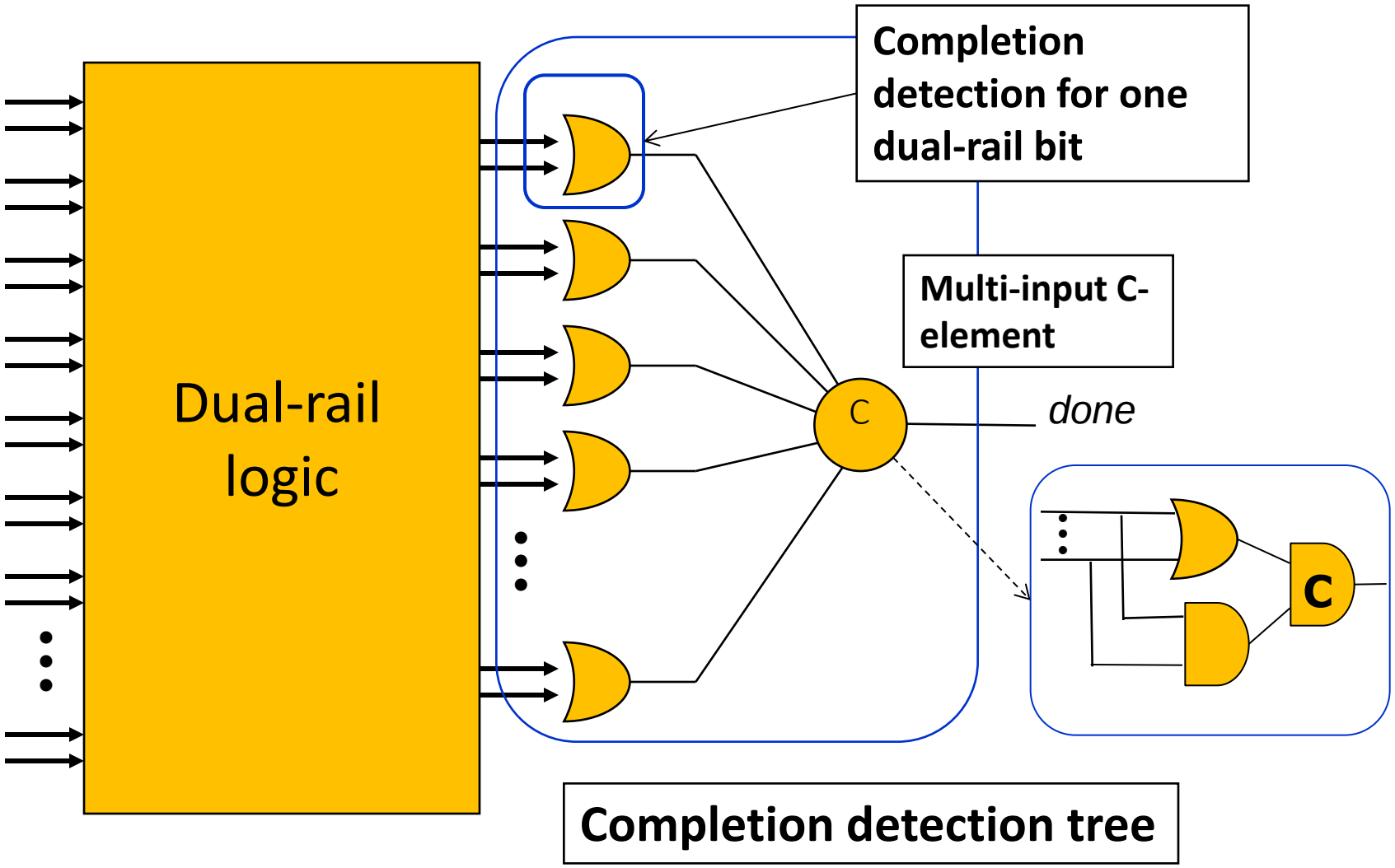


Conventional logic + matched delay

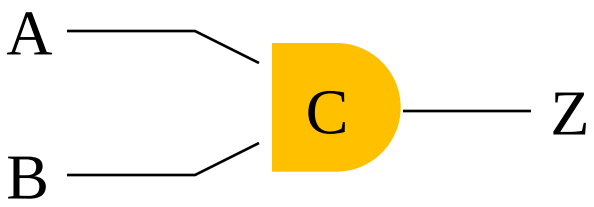
Completion is implicit: by done signal

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

True completion detection

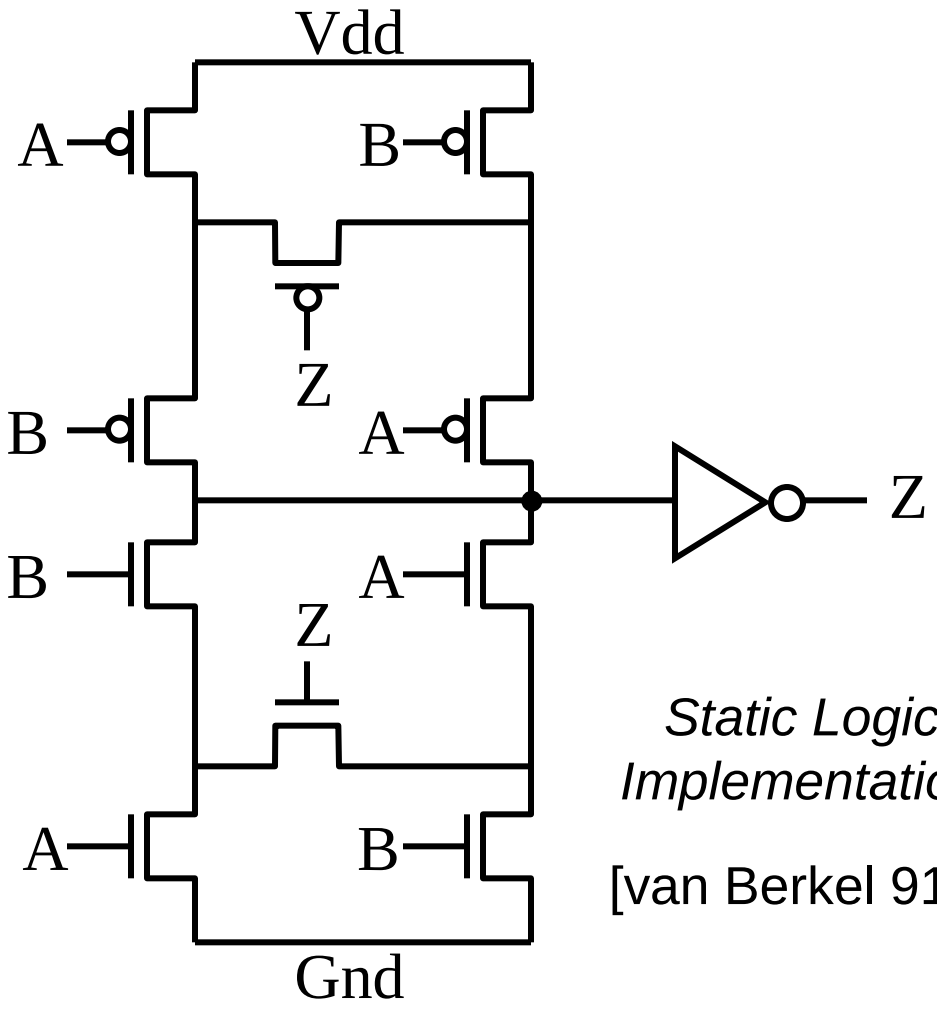


The Muller C element



C

A	B	Z^+
0	0	0
0	1	Z
1	0	Z
1	1	1



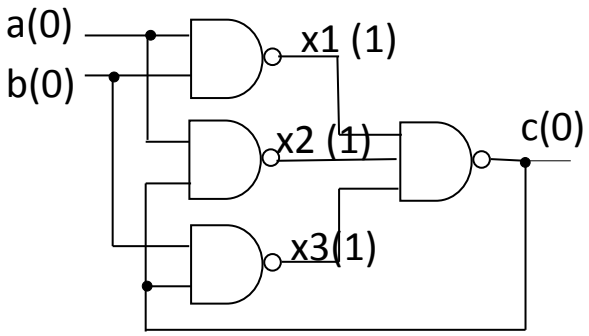
Static Logic Implementation

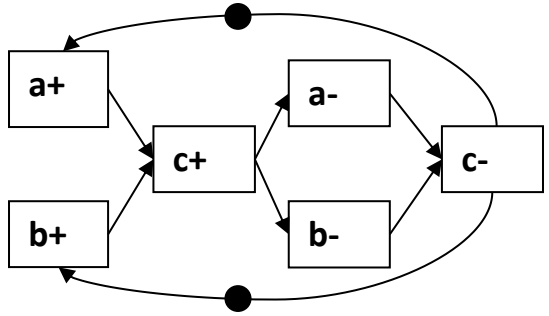
[van Berkel 91]

Why and what is causal acknowledgment?

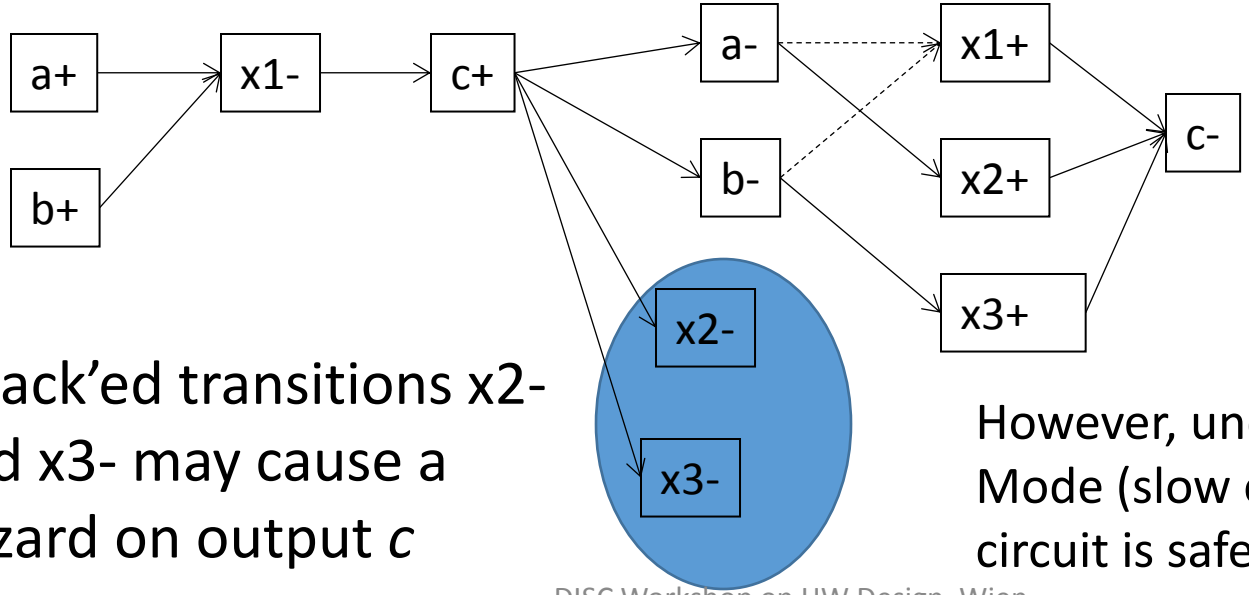
Every signal event must be acknowledged by another event

Causal acknowledgment





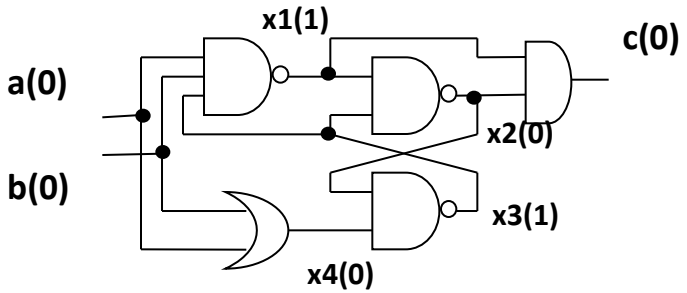
C-element implementation using simple gates

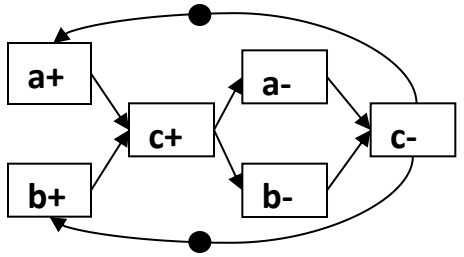


Unack'ed transitions x2- and x3- may cause a hazard on output *c*

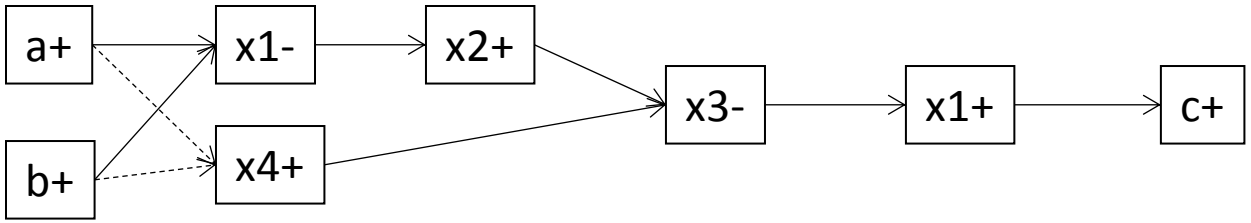
However, under Fundamental Mode (slow environment) the circuit is safe

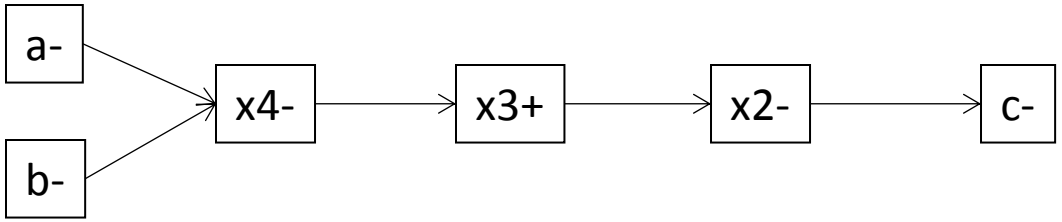
Principle of causal acknowledgement





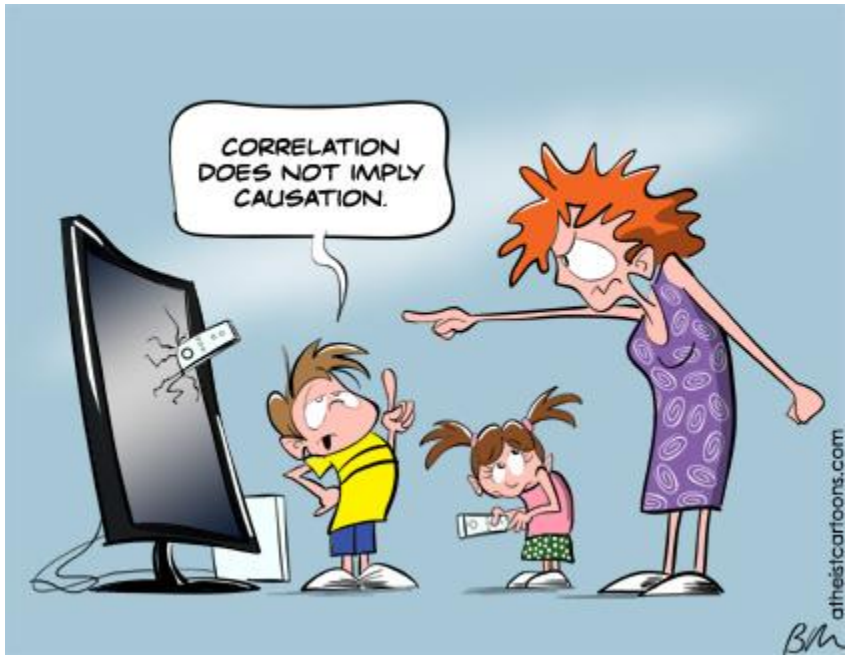
C-element implementation using simple gates





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

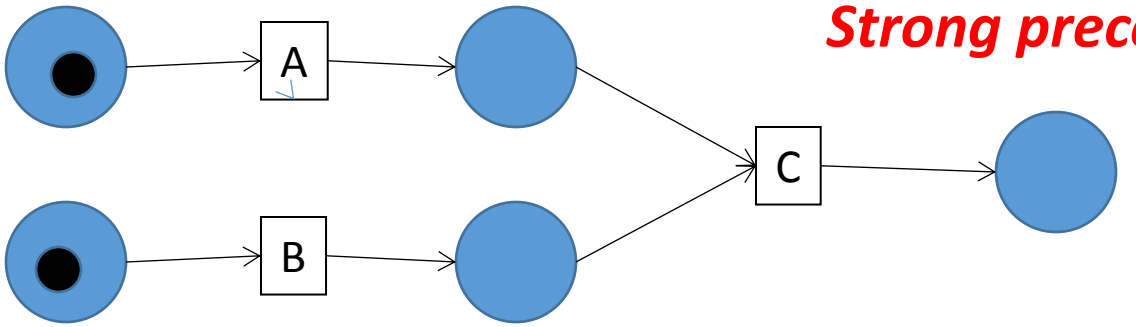
Why and what are strong and weak causality ?



Degree of necessity of precedence of some events for other events

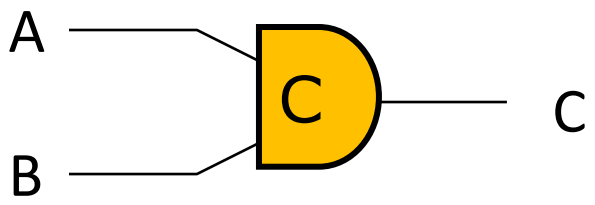
Strong Causality

- Petri net transitions synchronising as rendez-vous



Strong precedence

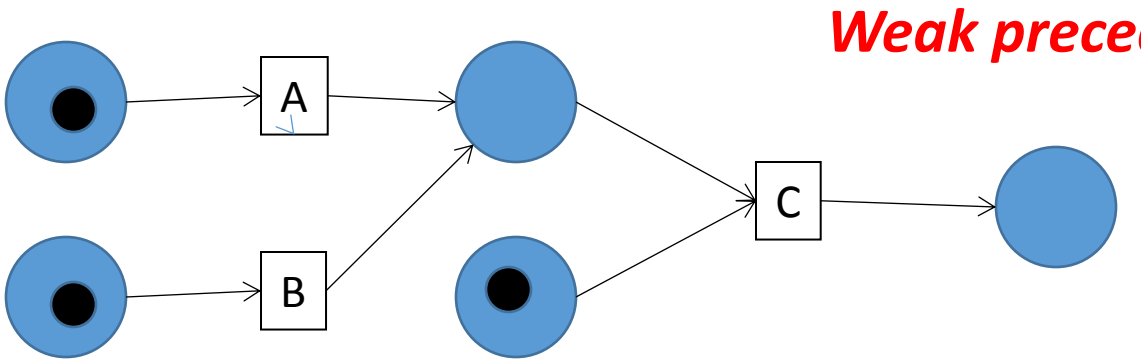
- **Logic circuits: Muller C-element (in 0-1 and 1-0 transitions), AND gate (in 0-1 transitions), OR gate (in 1-0 tranisitions)**



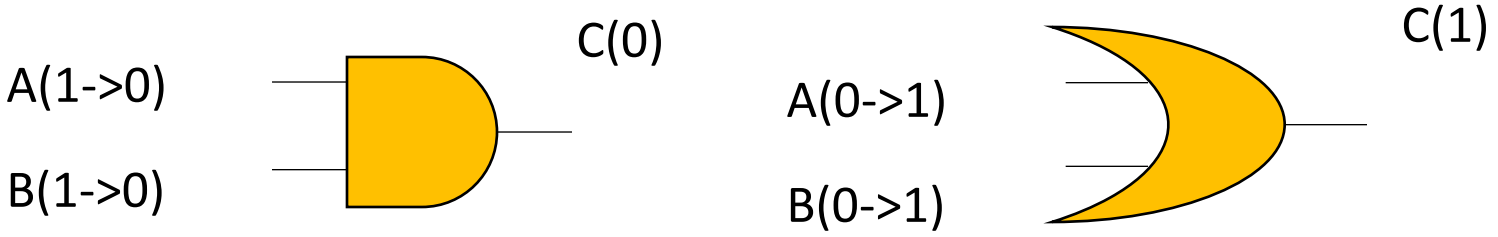
A	B	C^+
0	0	0
0	1	C
1	0	C
1	1	1

Weak Causality

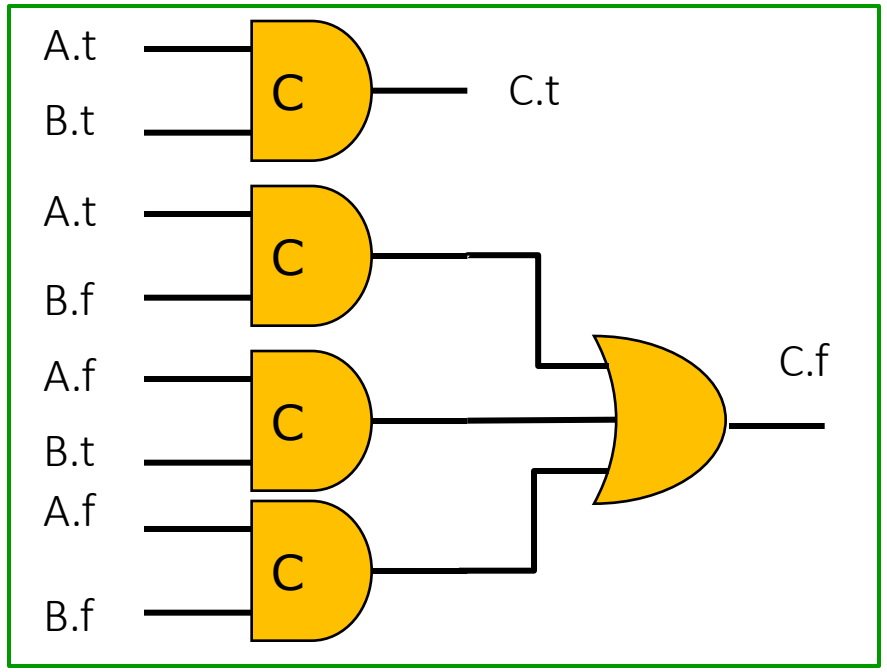
- Petri net transitions communicating via places



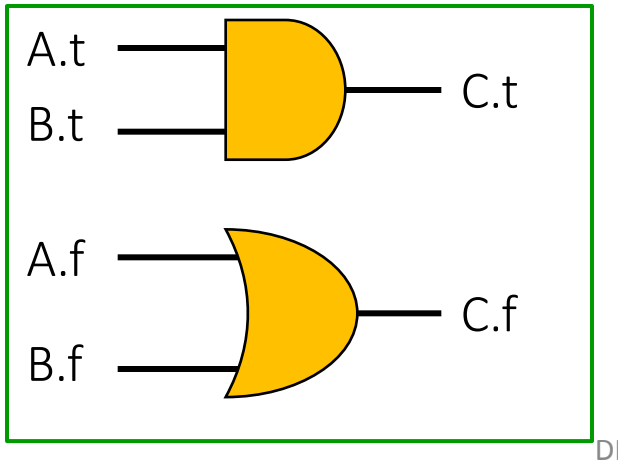
- **Logic circuits: AND gate (in 1-0 transitions), OR gate (in 0-1 transitions)**



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)

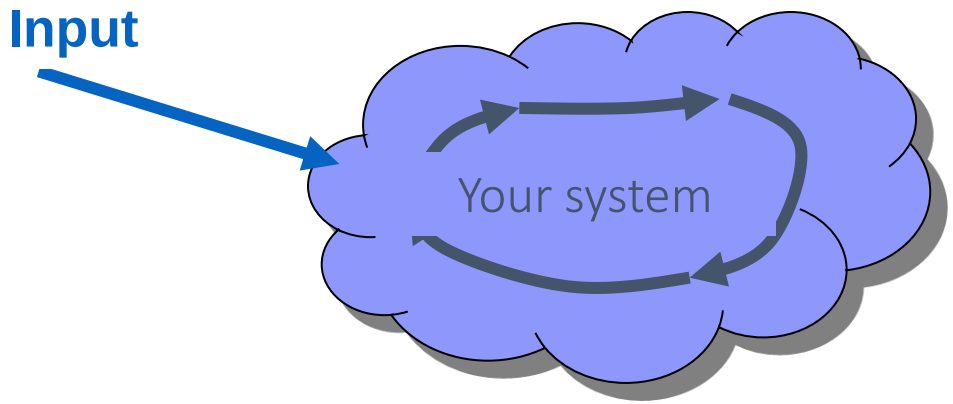
Why and what is timing comparison?

Telling if some event happened before another event

Synchronizers and arbiters

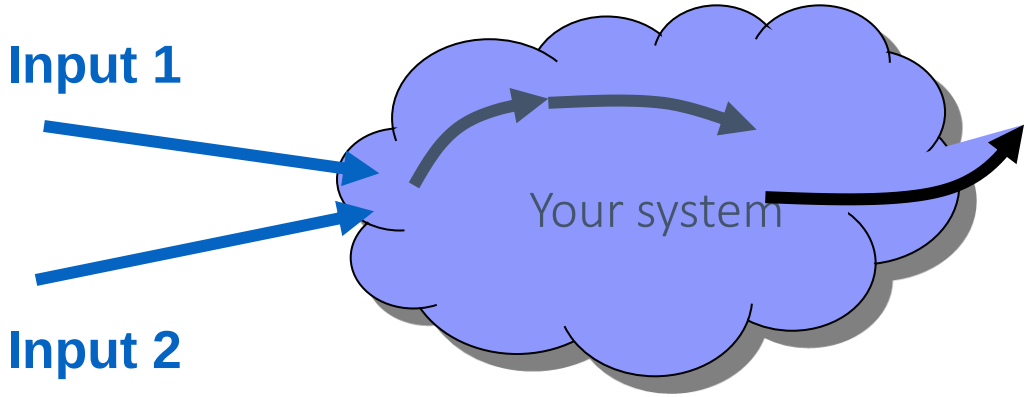
- Synchronizer

Decides which clock cycle to use for the input data

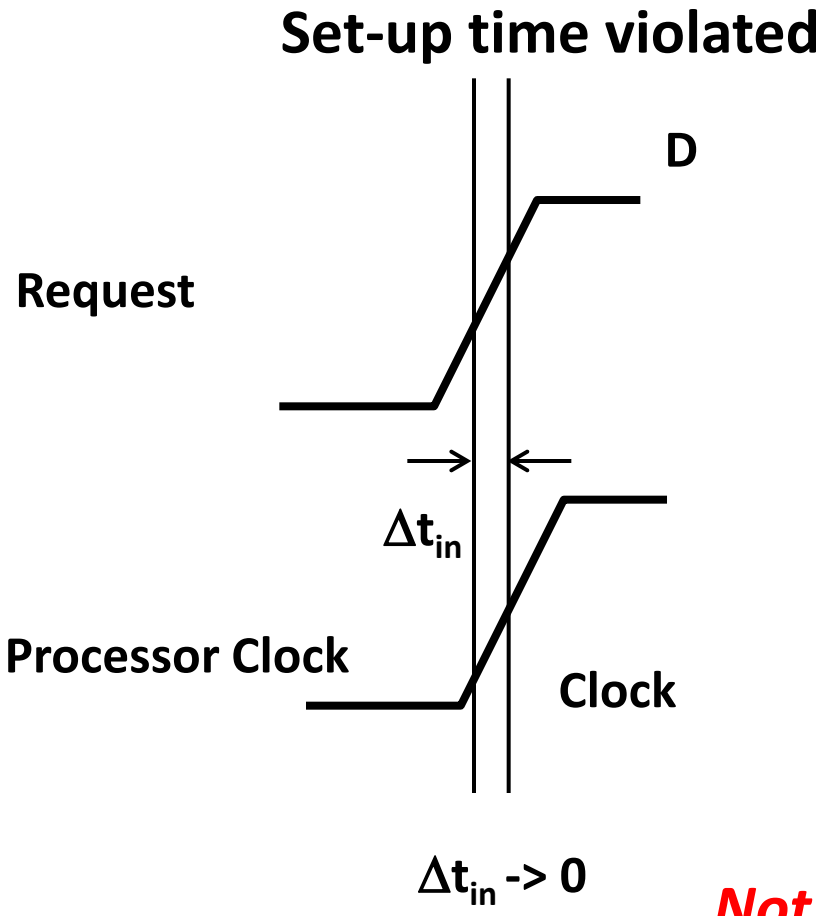


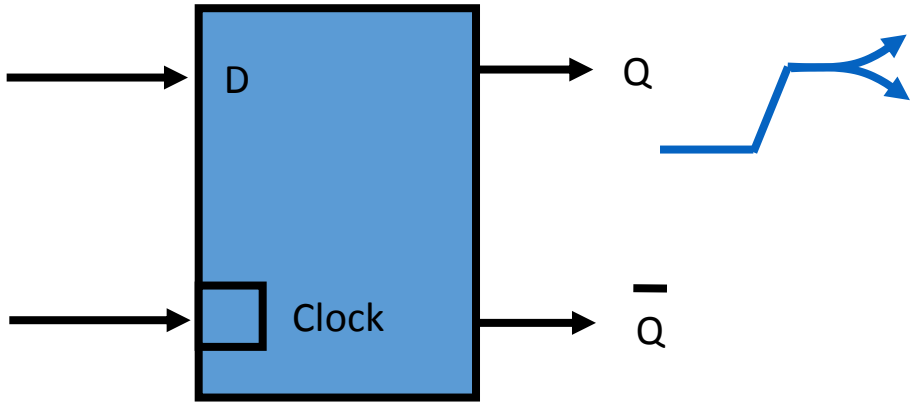
- Asynchronous arbiter

Decides the order of inputs



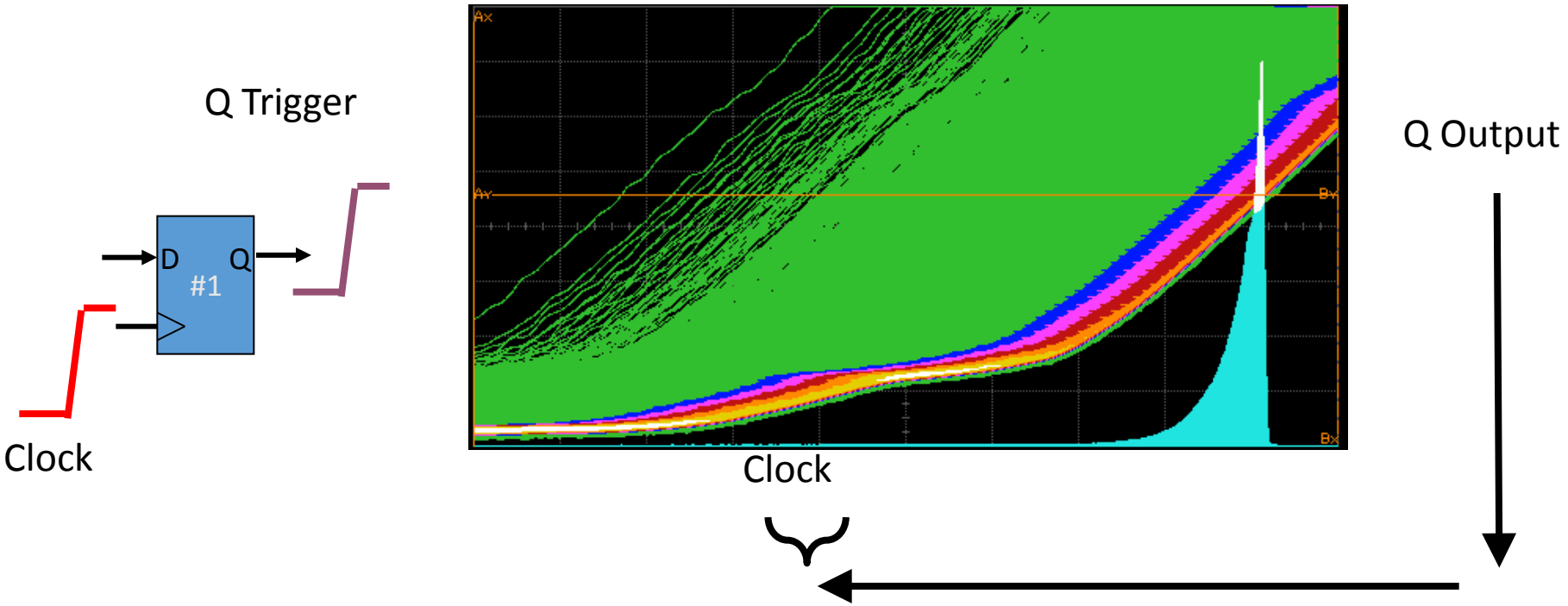
Metastability is....





Not being able to decide…

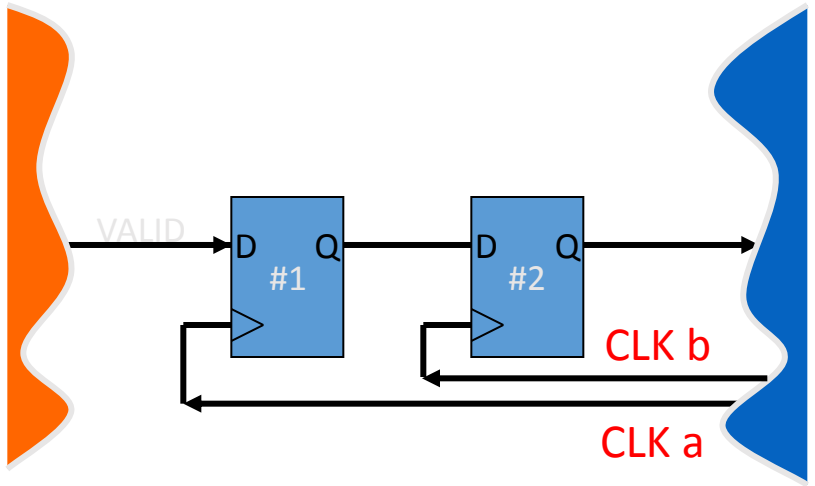
Typical responses



- 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

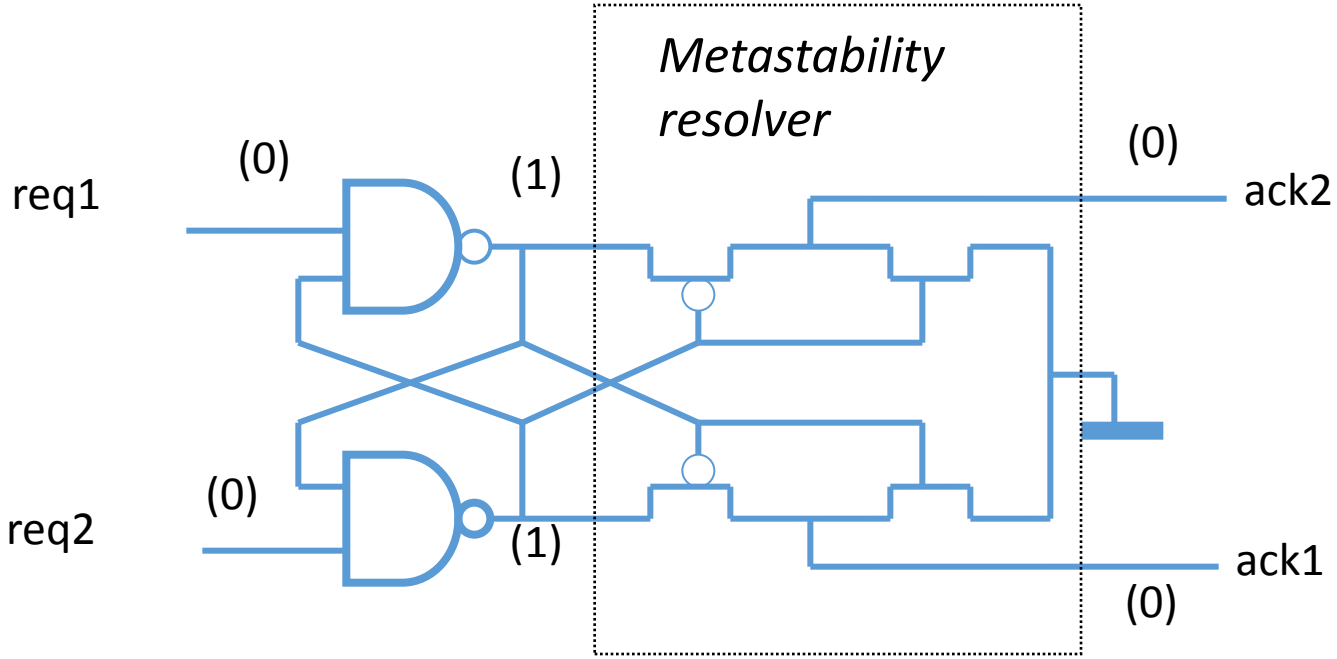
- t is time allowed for the Q to change between CLK a and CLK b
- τ is the recovery time constant, usually the gain-bandwidth of the circuit
- T_w is the "metastability window" (aperture around clock edge in which the capture of data edge causes a delay that is greater than normal propagation delay of 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)

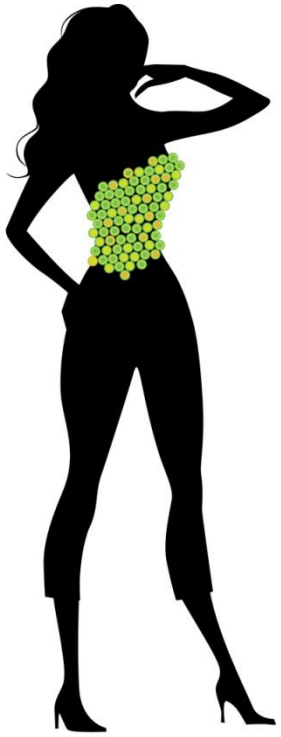


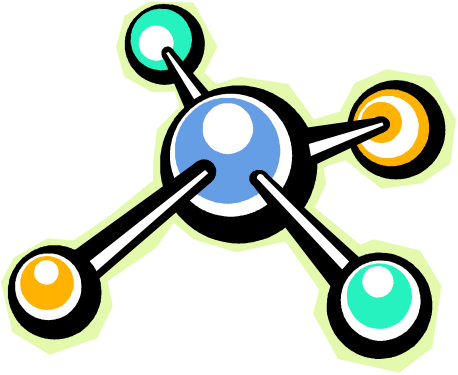
An asynchronous data latch with metastability resolver can be built similarly

Importance of Timing Comparison

- Understanding metastability is becoming very important as analogue and digital domains get closer, and timing uncertainty and PVT variations increase
- Arbitration and synchronization are increasing their importance due to many-core, timing domains, NoCs, GALS
- Design automation for metastability and synchronization is turning from research to practice (Blendix)

Models and techniques for design



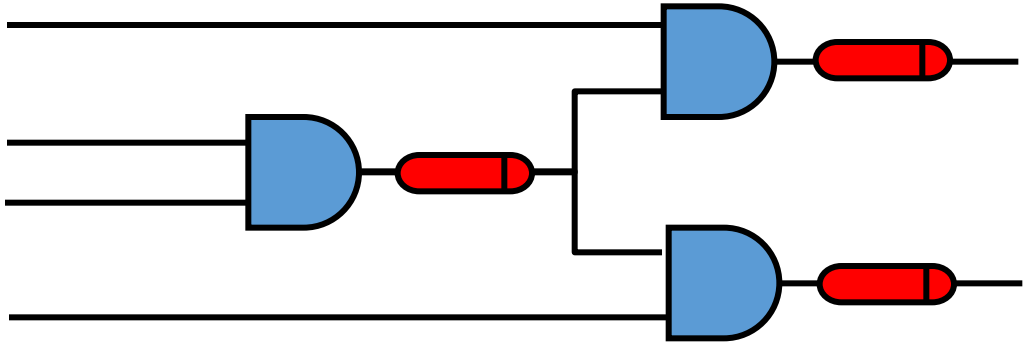


Models and techniques for asynchronous design

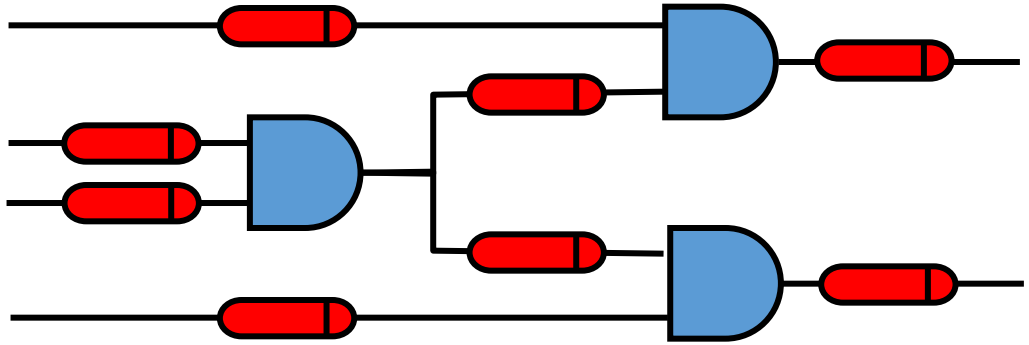
- Nature of Models:
  - Delay model (inertial, pure, gate delay, wire delay, bounded and unbounded delays)
  - Models of environment (fundamental mode, input-output)
  - Models of switching behaviour (state-based, event-based, hybrid)
- RTL level:
  - Data and control paths separate (data flow graphs, FSMs, Signal Transition Graphs, 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

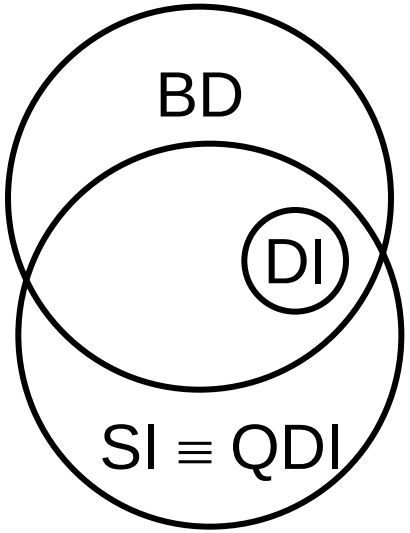


- Wire delay model: delays in gates and wires



Delay models for async. 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

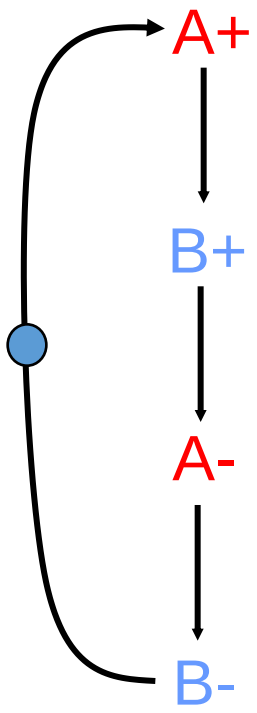


Control Logic

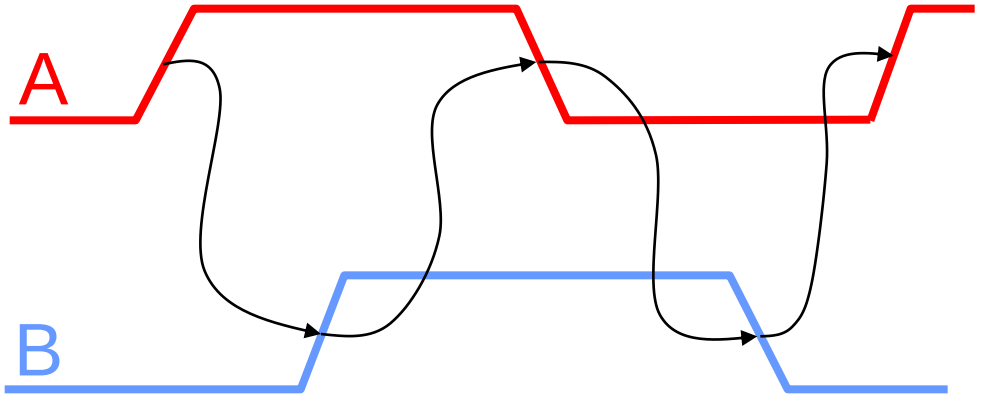
- Control specification based on Petri nets (Signal Transition graphs)

Control specification

Signal Transition Graph (STG)

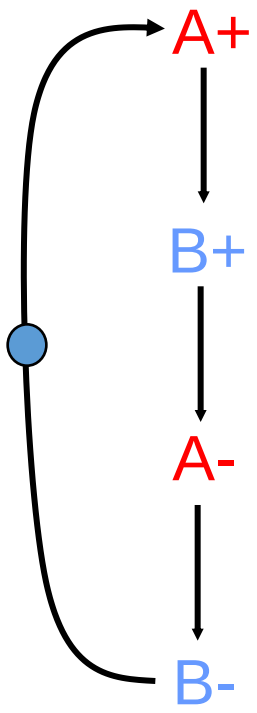


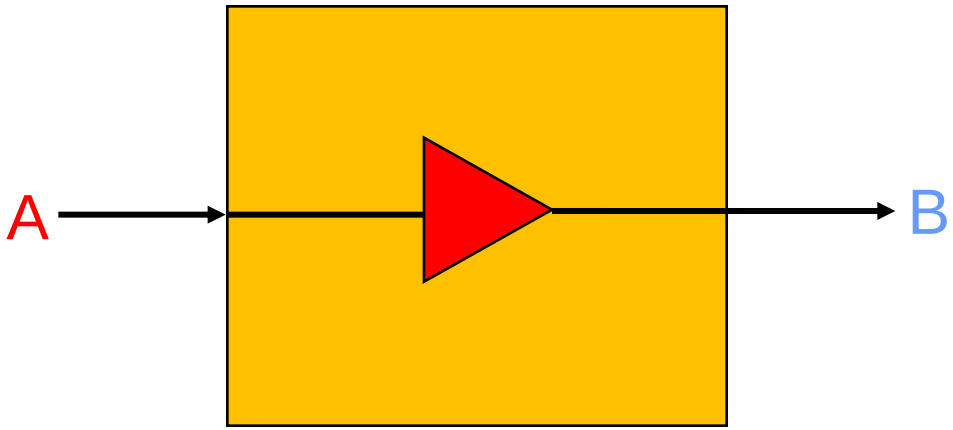
Timing Diagram



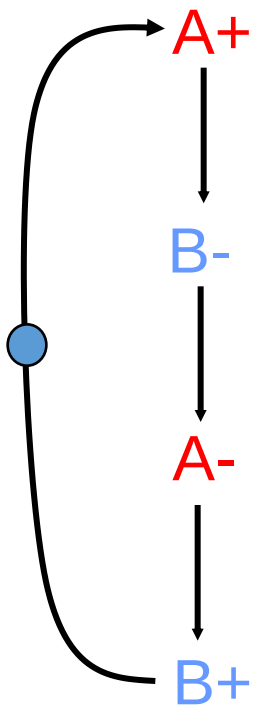
A input
B output

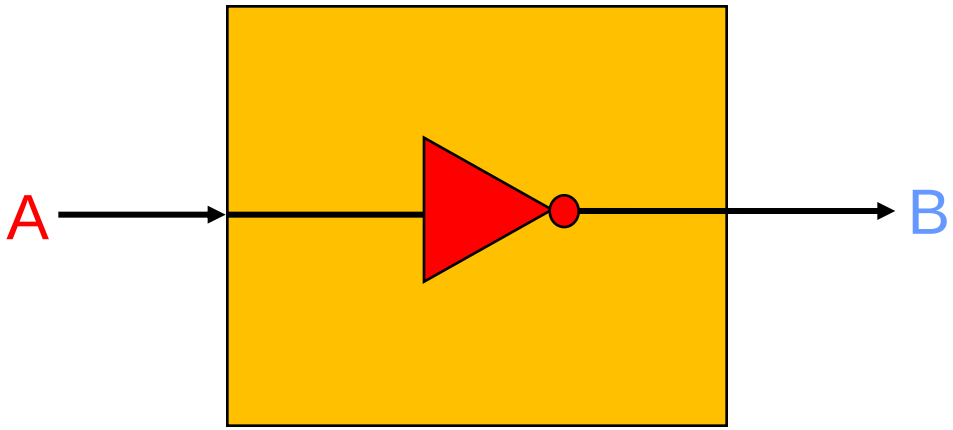
Control specification



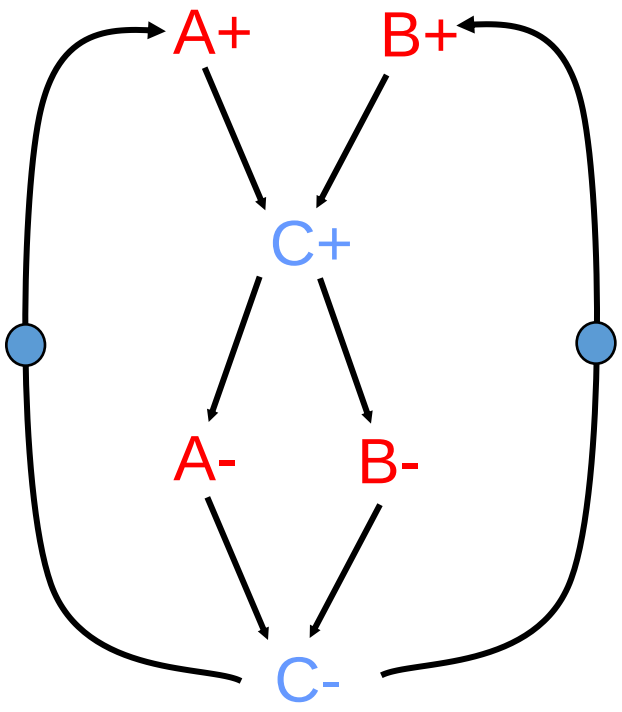


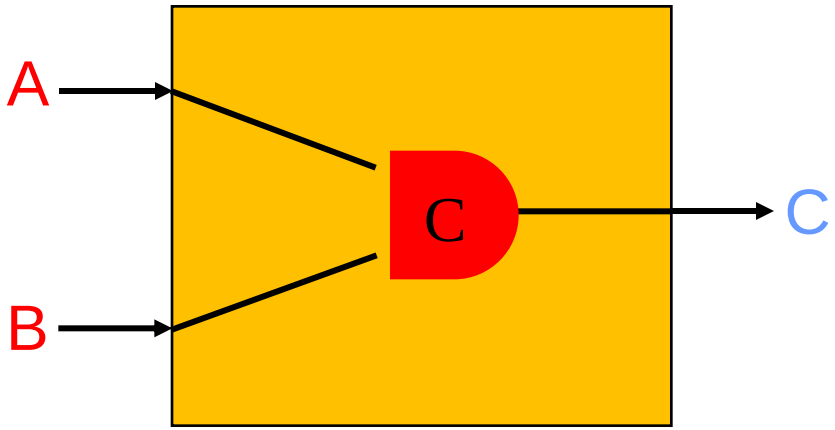
Control specification



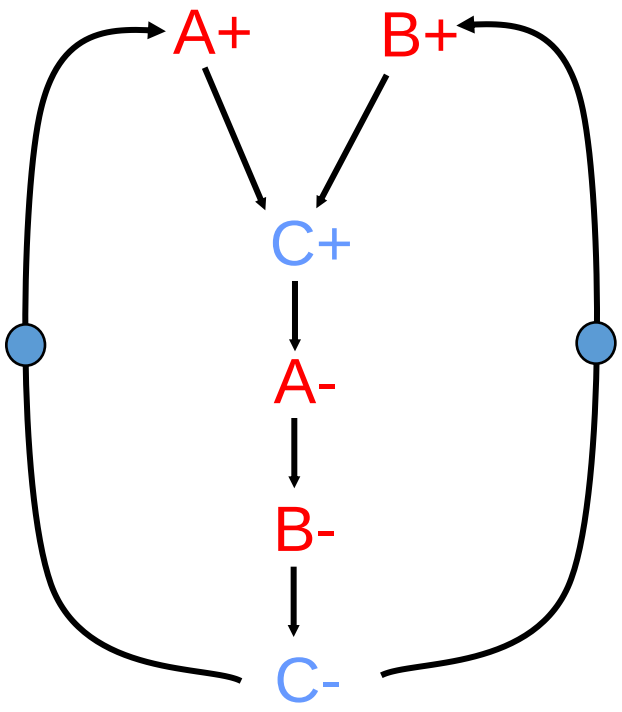


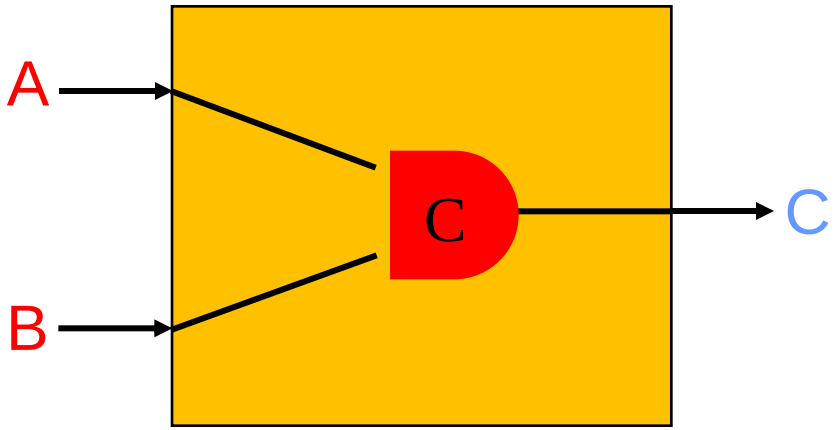
Control specification



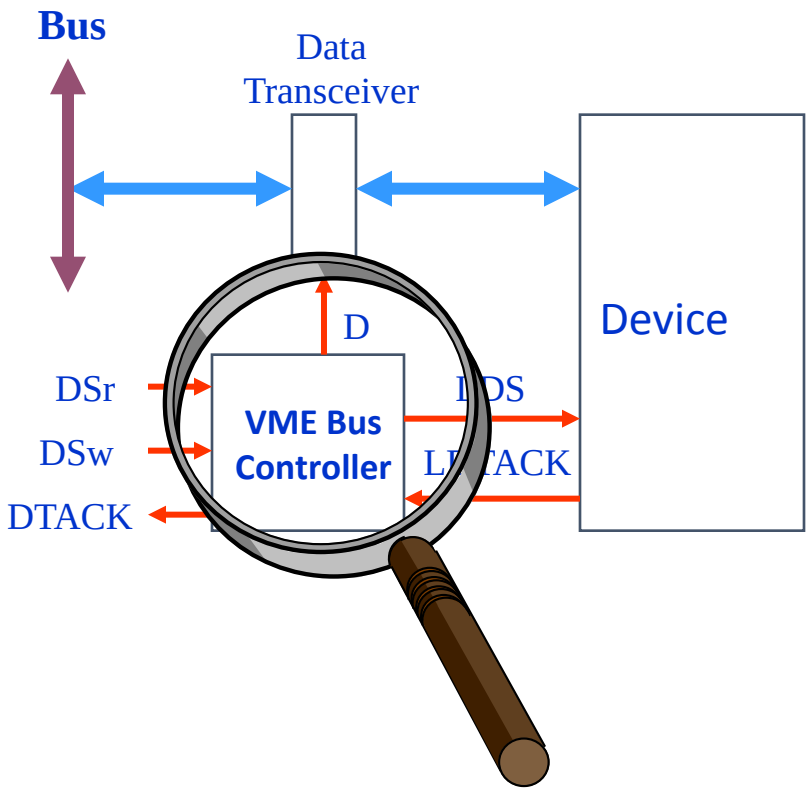


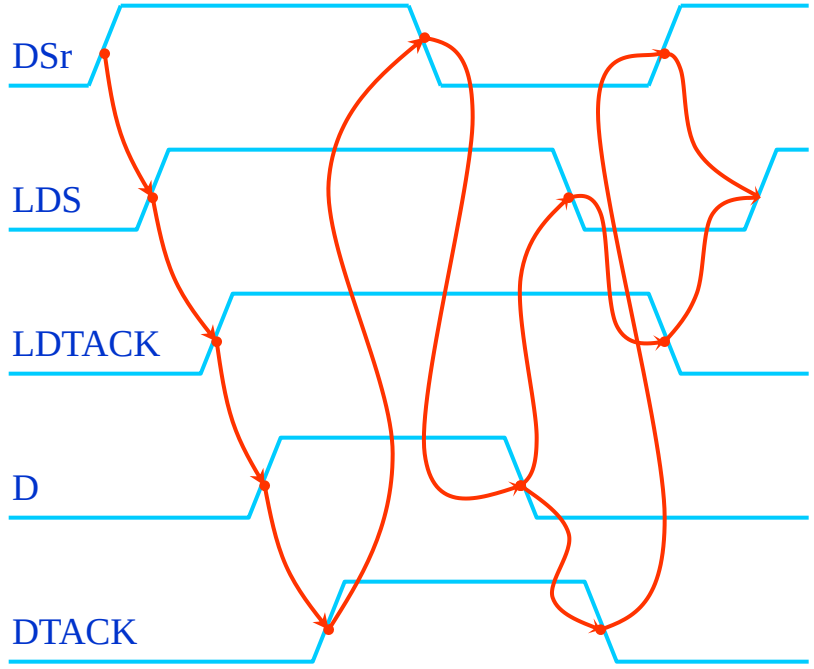
Control specification





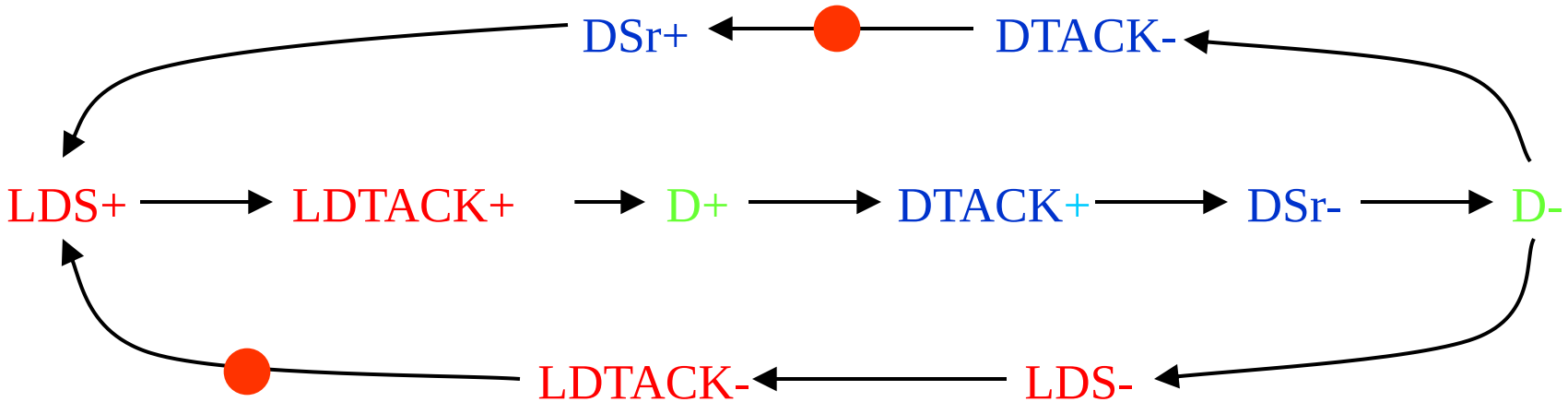
VME bus example using Petri nets

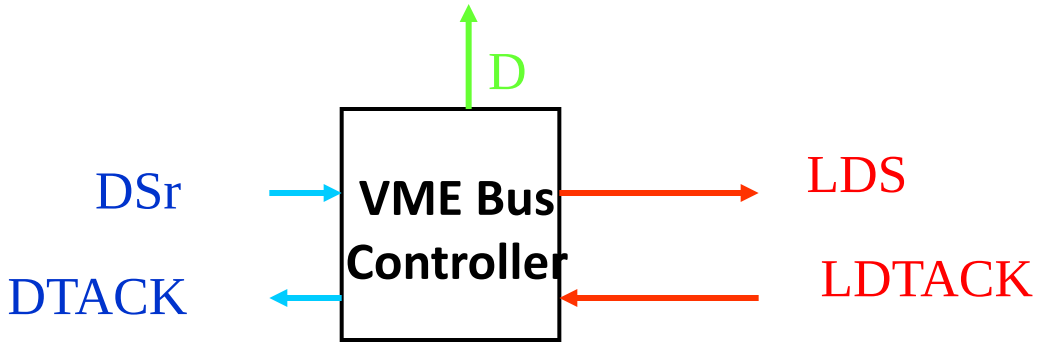




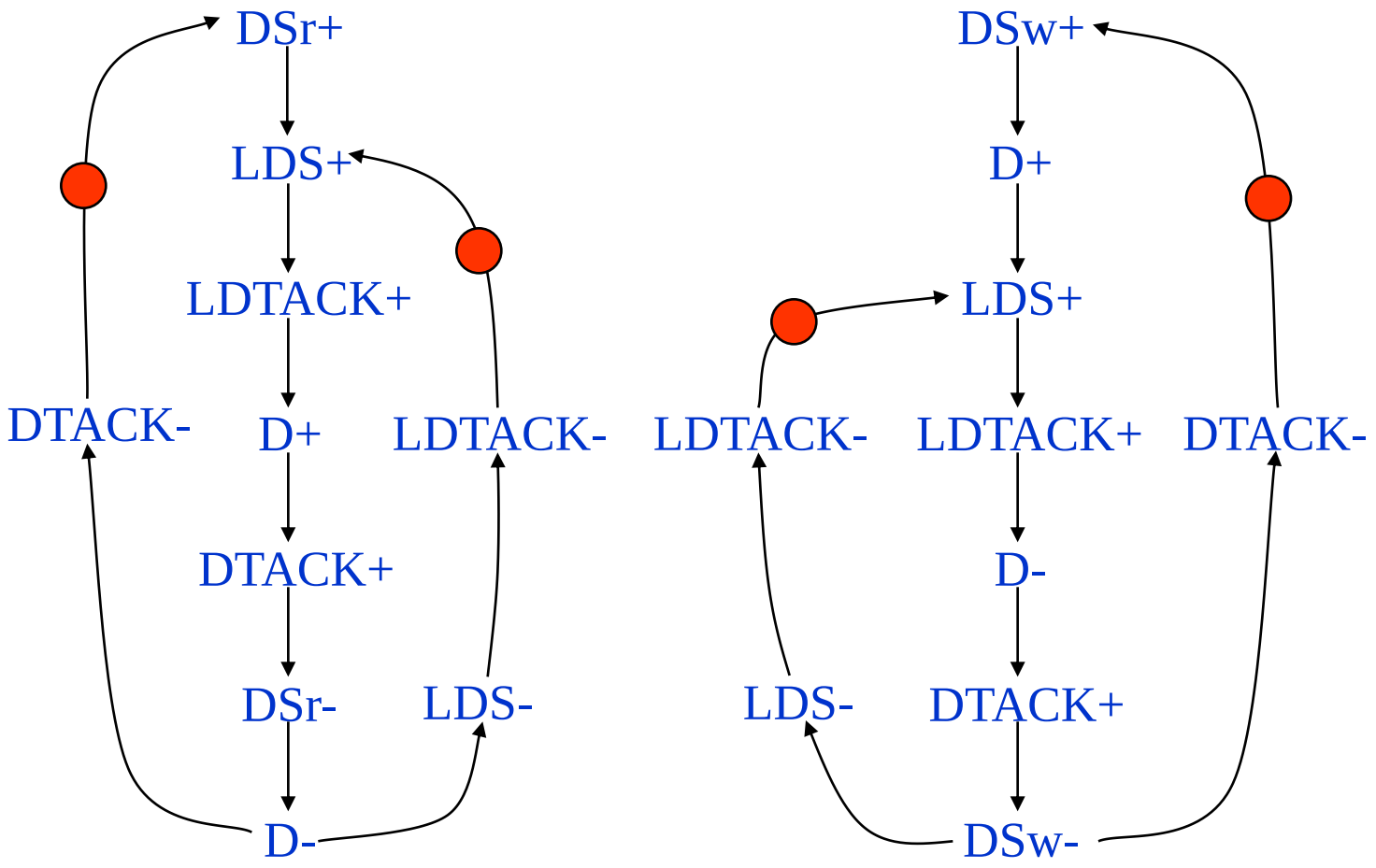
Read Cycle

STG for the READ cycle

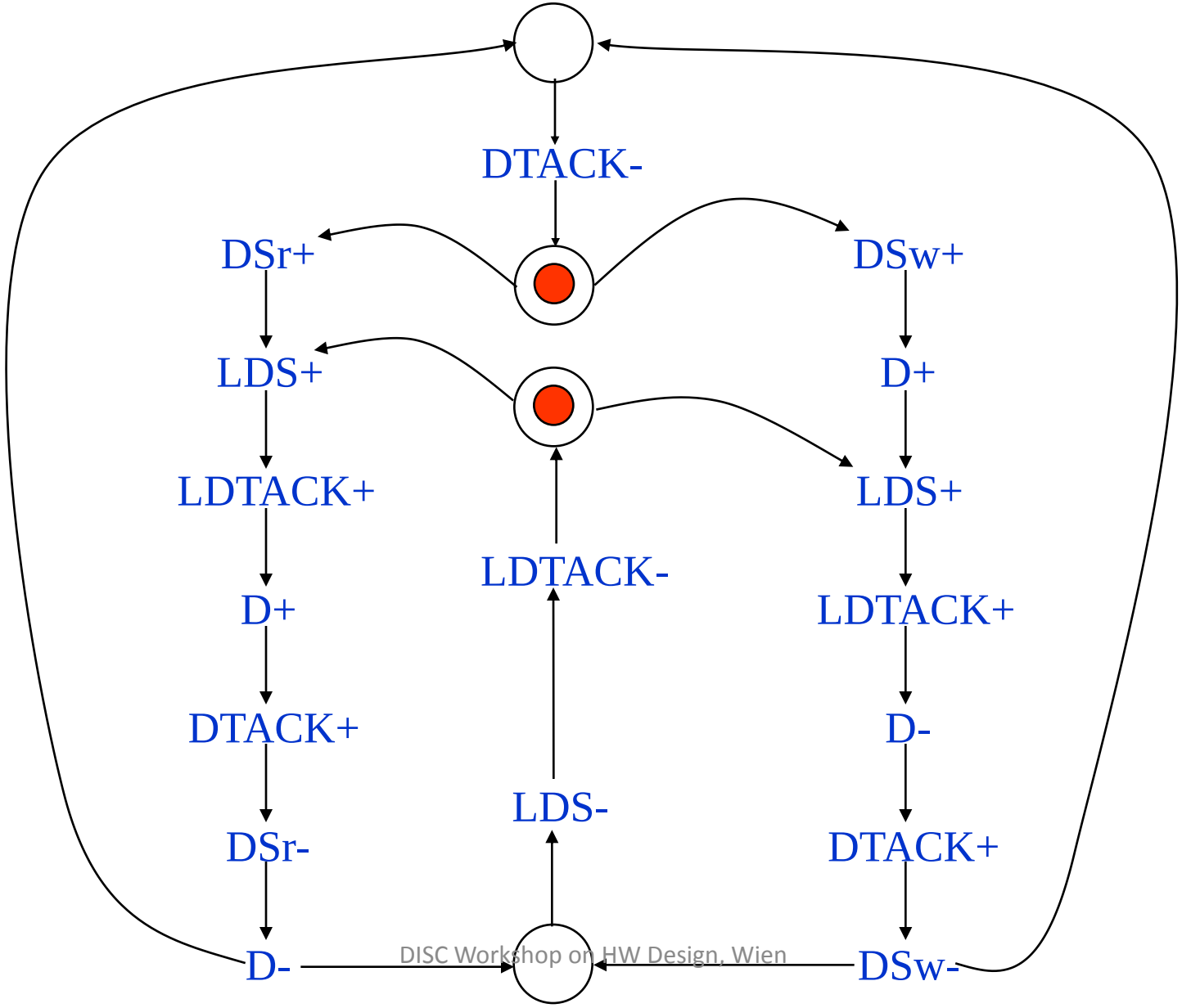




Choice: Read and Write cycles



Choice: Read and Write cycles



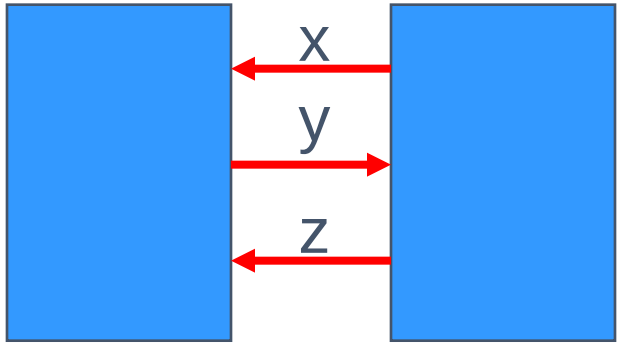
Workcraft tool

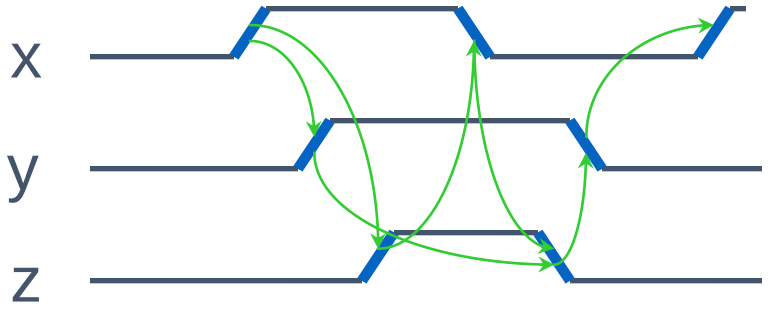
- Workcraft is a software package for graphical edit, analysis, synthesis and visualisation of asynchronous circuit behaviour
- Petrify plus a few other tools are part of it as plug-ins
- It is based in Java tools
- Can be downloaded from http://workcraft.org/
- And installed in few minutes
- There is a simple to use tutorial for that
- Many other tutorials on various aspects of Petri nets modelling, STG synthesis and analysis, circuit verification, visualisation etc.

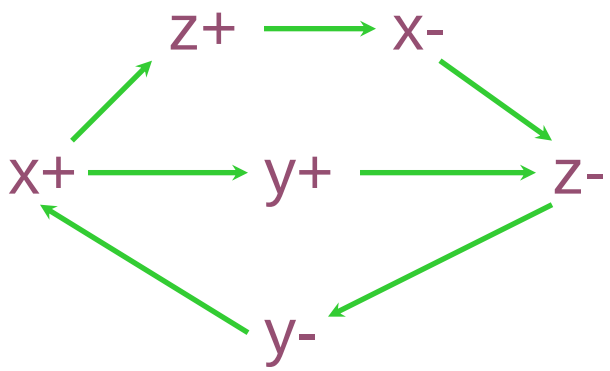
Some references

- **General Async Design**: J. Sparsø and S.B. Furber, editors. *Principles of Asynchronous Circuit Design*, Kluwer Academic Publishers, 2001. (electronic version of a tutorial based on this book can be found on: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/855/pdf/imm855.pdf
- **Async Control Synthesis**: 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/)
- **Arbiters and Synchronizers**: D.J. Kinniment, Synchronization and Arbitration in Digital Systems, Wiley and Sons, 2007 (a tutorial on arbitration and synchronization from ASYNC/NOCS 2008 can be found: http://async.org.uk/async2008/async-nocs-slides/Tutorial-Monday/Kinniment-ASYNC-2008-Tutorial.pdf)

xyz-example: Specification

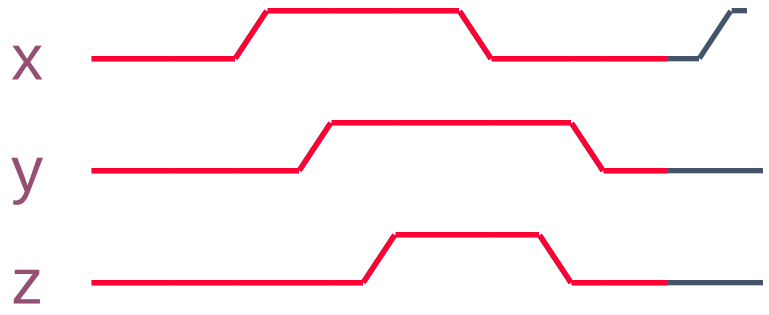


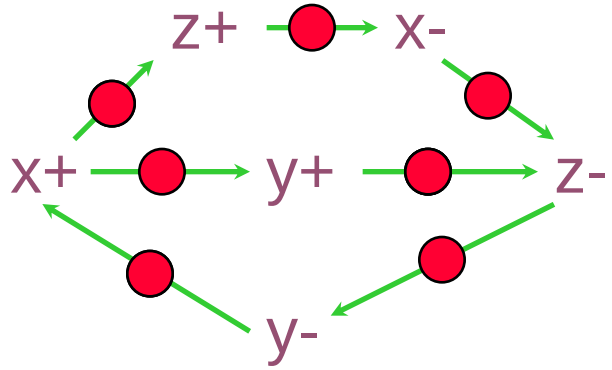




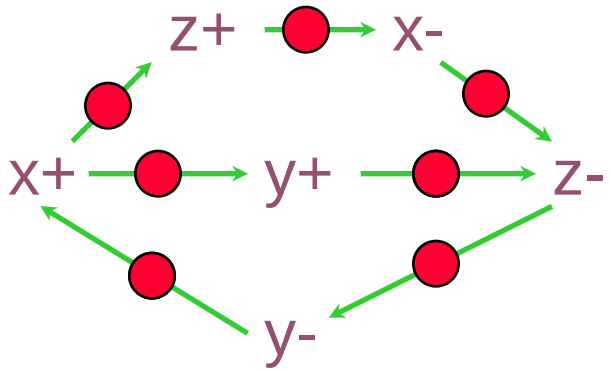
Signal Transition Graph (STG)

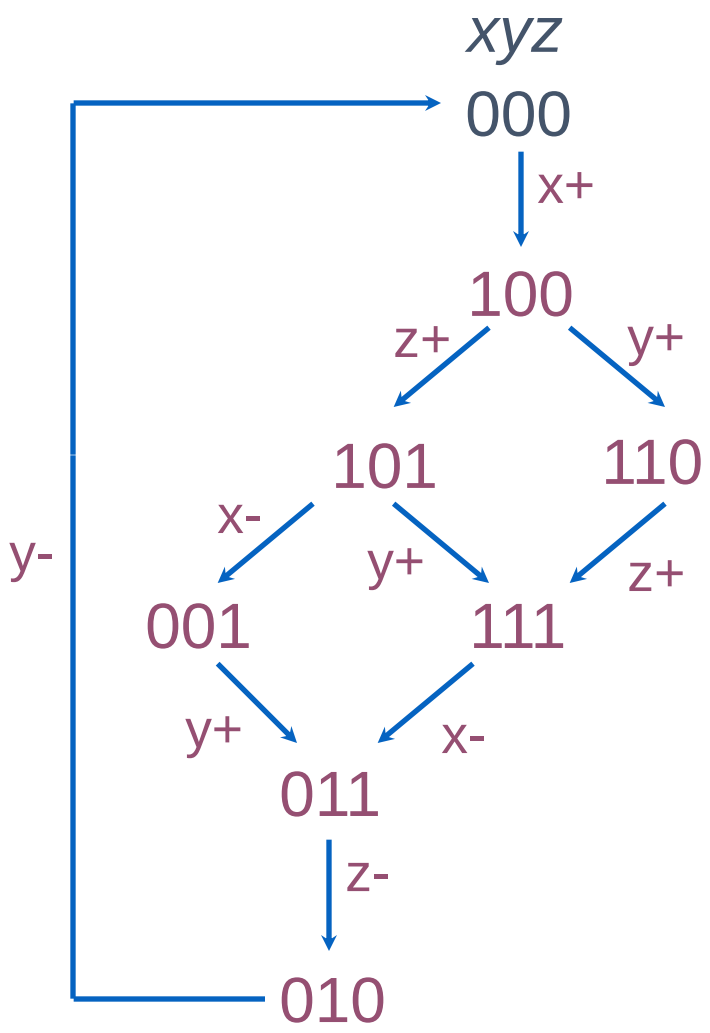
Token flow





State graph



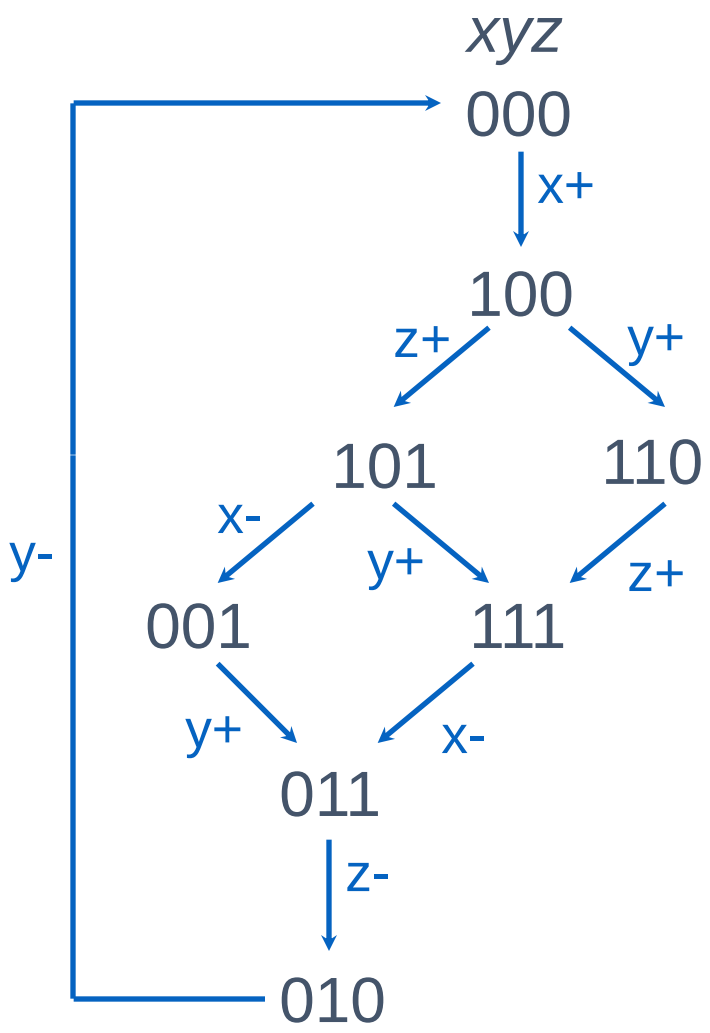


Next-state functions

$$x = \bar{z} \cdot (x + \bar{y})$$

$$y = z + x$$

$$z = x + \bar{y} \cdot z$$

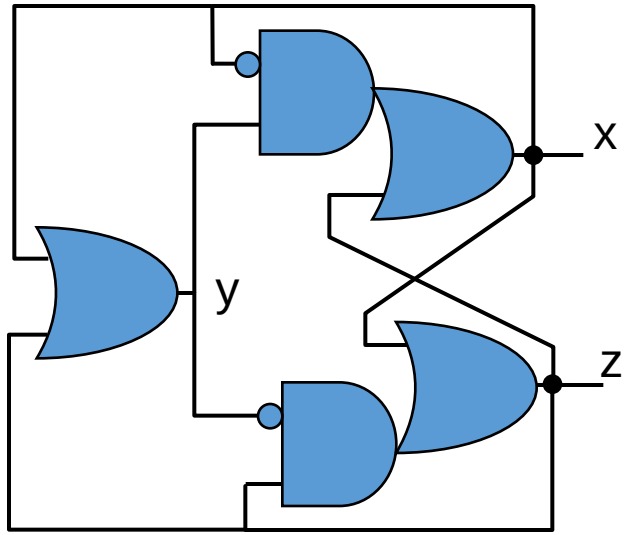


Complex Gate netlist

$$x = \bar{z} \cdot (x + \bar{y})$$

$$y = z + x$$

$$z = x + \bar{y} \cdot z$$



Circuit synthesis

- Goal:
  - Derive a hazard-free circuit under a given delay model and mode of operation

Speed independence

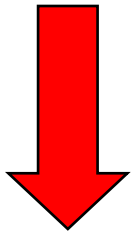
- Delay model
  - Unbounded gate / environment delays
  - Certain wire delays shorter than certain paths in the circuit
- Conditions for implementability:
  - Consistency
  - Complete State Coding
  - Persistency

Implementability conditions

- Consistency
  - Rising and falling transitions of each signal alternate in any trace
- Complete state coding (CSC)
  - Next-state functions correctly defined
- Persistency
  - No event can be disabled by another event (unless they are both inputs)

Implementability conditions

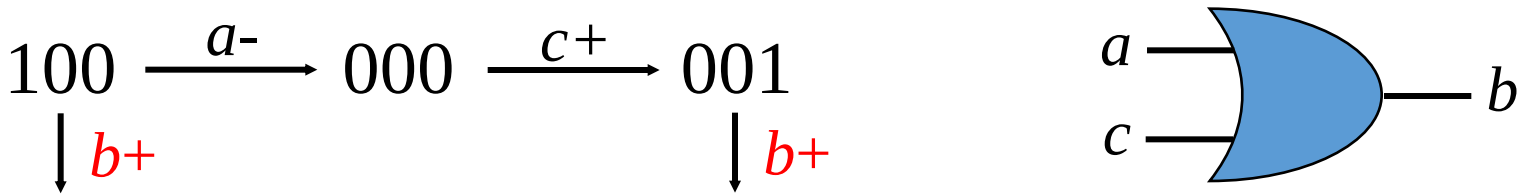
- Consistency + CSC + persistency

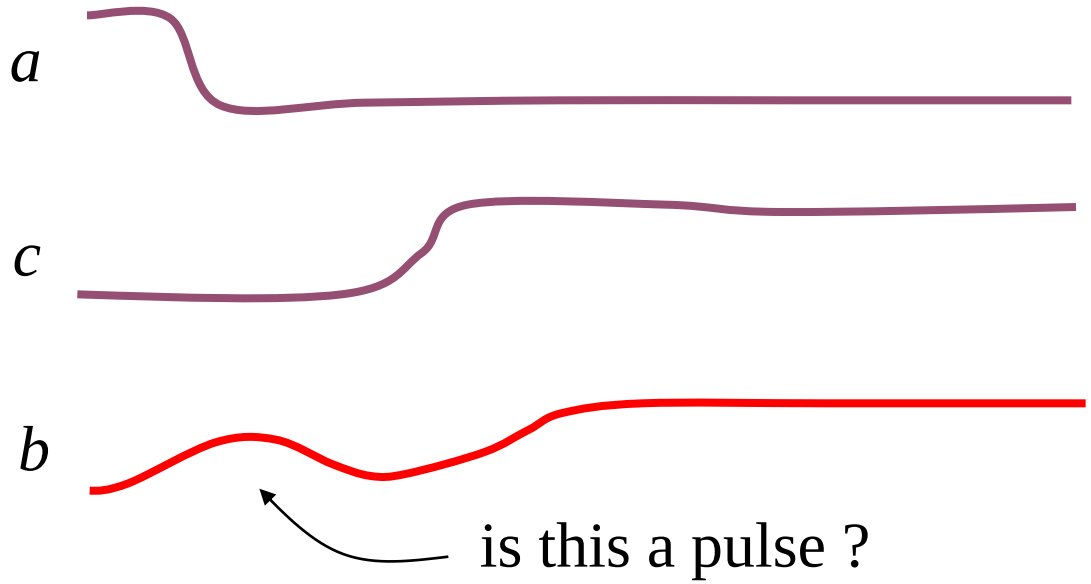


- There exists a speed-independent circuit that implements the behavior of the STG

  (under the assumption that ay Boolean function can be implemented with one complex gate)

Persistency





Speed independence ⇒ glitch-free output behavior under any delay

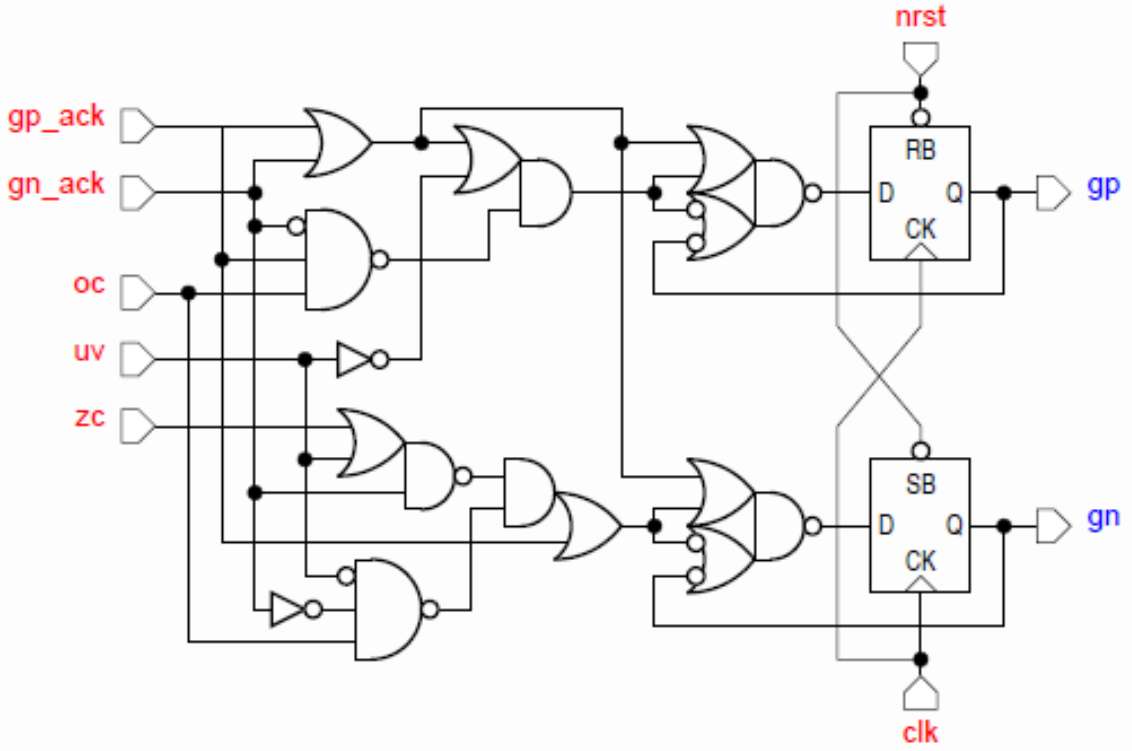
CASE Study: Buck converter controller

Case study: Buck converter - synchronous control

Behavioural Verilog specification:

```
module control (clk, nrst, oc, uv, zc, gp_ack, gn_ack, gp, gn);
  input clk, nrst, uv, oc, zc, gp_ack, gn_ack;
  output reg gp, gn;
  always @(posedge clk or negedge nrst) begin
    if (nrst == 0) begin
      gp <= 0; gn <= 1;
    end else case ({gp_ack, gn_ack})
      2'b00: if (uv == 1) gp <= 1; else if (oc == 1) gn <= 1;
      2'b10: if (oc == 1) gp <= 0;
      2'b01: if (uv == 1 || zc == 1) gn <= 0;
    endcase
  end
endmodule
```

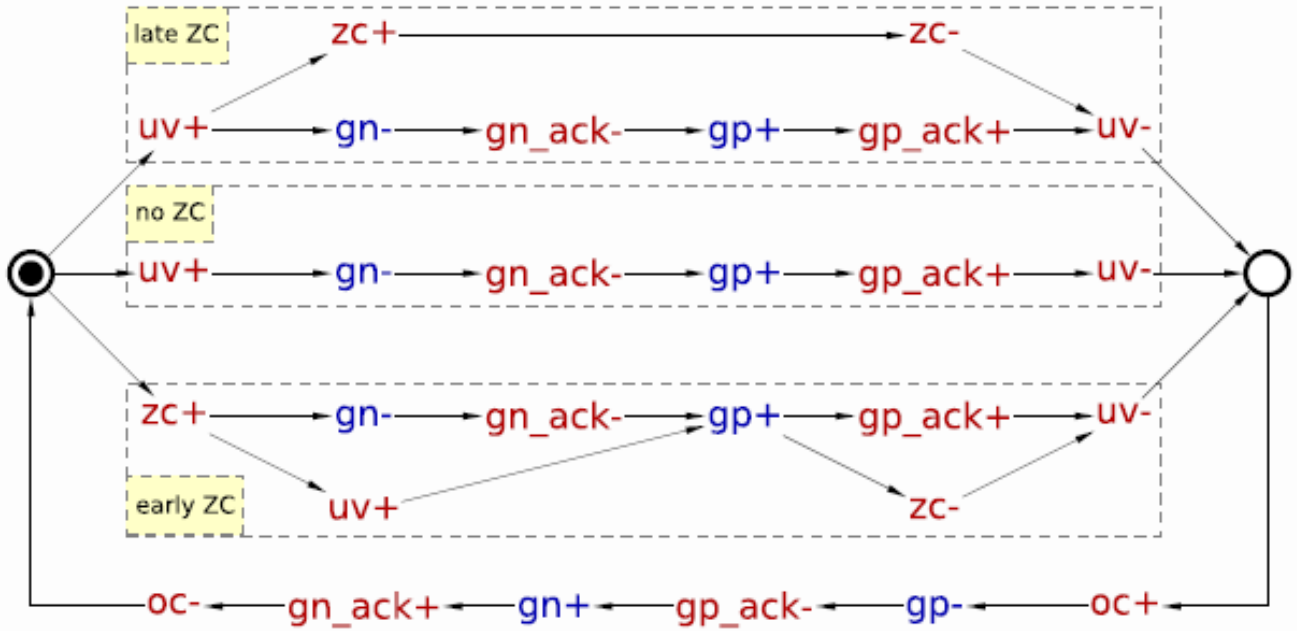
Design Compiler synthesis result:



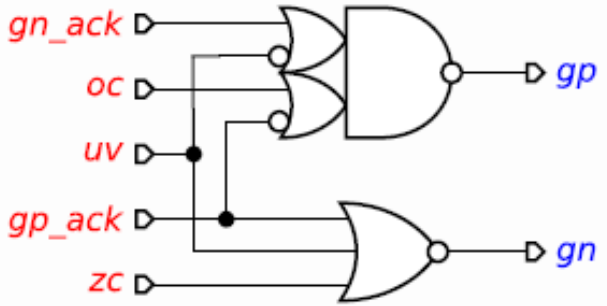
- If clock is slow, the control is unresponsive to the buck changes
- If clock is fast, it burns energy when the buck is inactive

Buck converter – asynchronous control

STG specification:

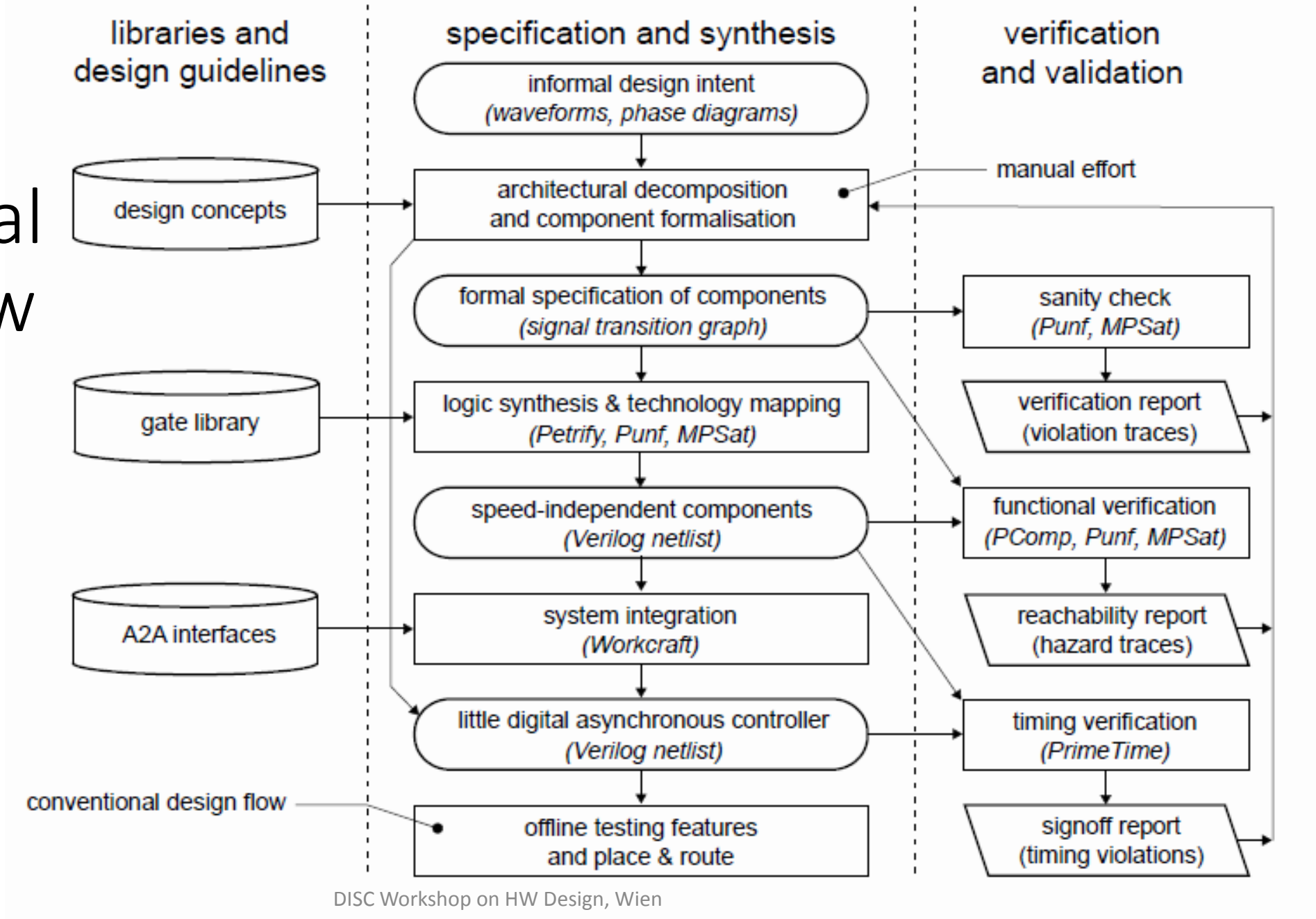


SI implementation:



- Formal specification using Signal Transition Graph (STG) model – similar to Petri nets
- Verifiable speed-independent (SI) implementation – hazard-free for any gate delays
- Prompt reaction time to the buck changes – latency of a complex gate

Little digital design flow

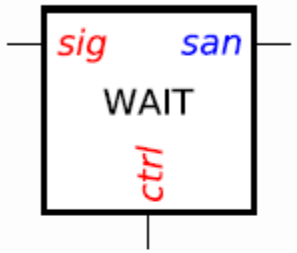


Analog to Async (A2A) components

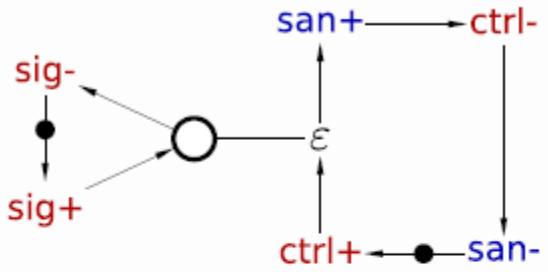
- Interface analog world of *dirty* signals
- Provide hazard-free *sanitised* digital signals
- Basic A2A components
  - **WAIT** / **WAIT0** – wait for analog input to become high / low and latch it until explicit release signal
  - **RWAIT** / **RWAIT0** – modification of WAIT / WAIT0 with a possibility to persistently cancel the waiting request
  - **WAIT01** / **WAIT10** – wait for a rising / falling edge
- Advanced A2A components
  - **WAIT2** – combination of WAIT and WAIT0 to wait for high and low input values, one after the other.
  - **WAITX** – arbitrate between two non-persistent analog inputs
  - **WAITX2** – behaves as WAITX in the rising phase and as WAIT0 in the falling phase

Interfacing analog to async: WAIT element

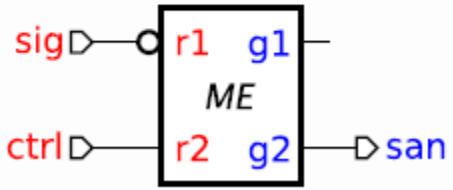
- Component interface:



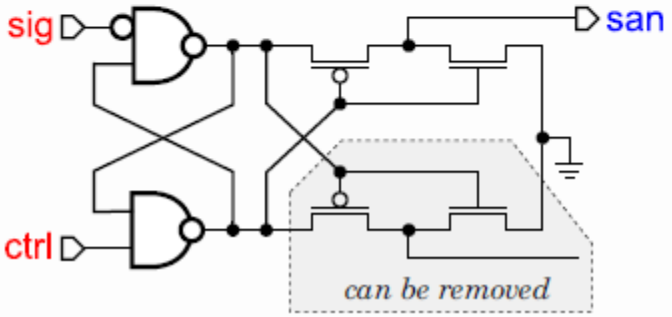
- STG specification:



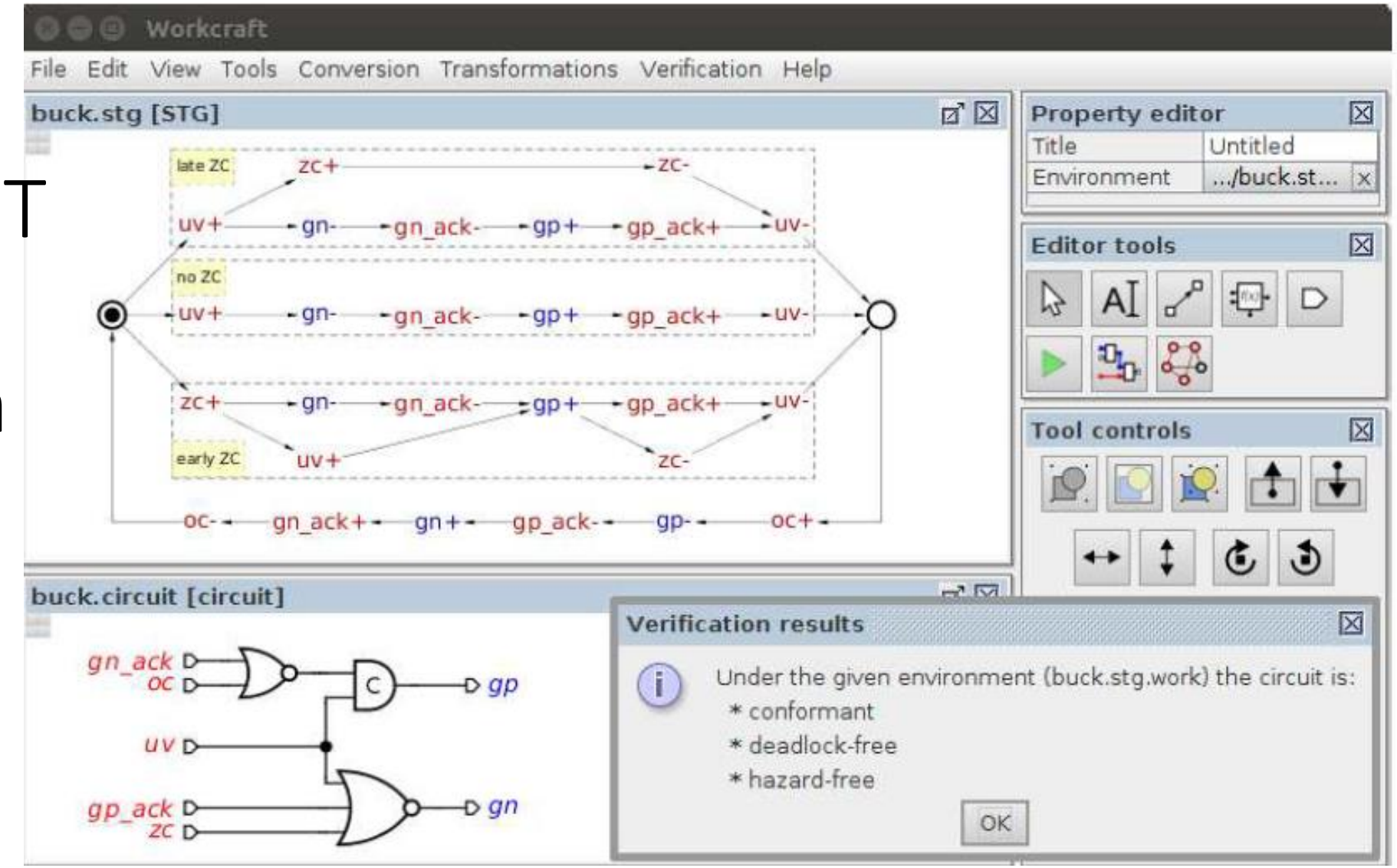
- ME-based implementation:



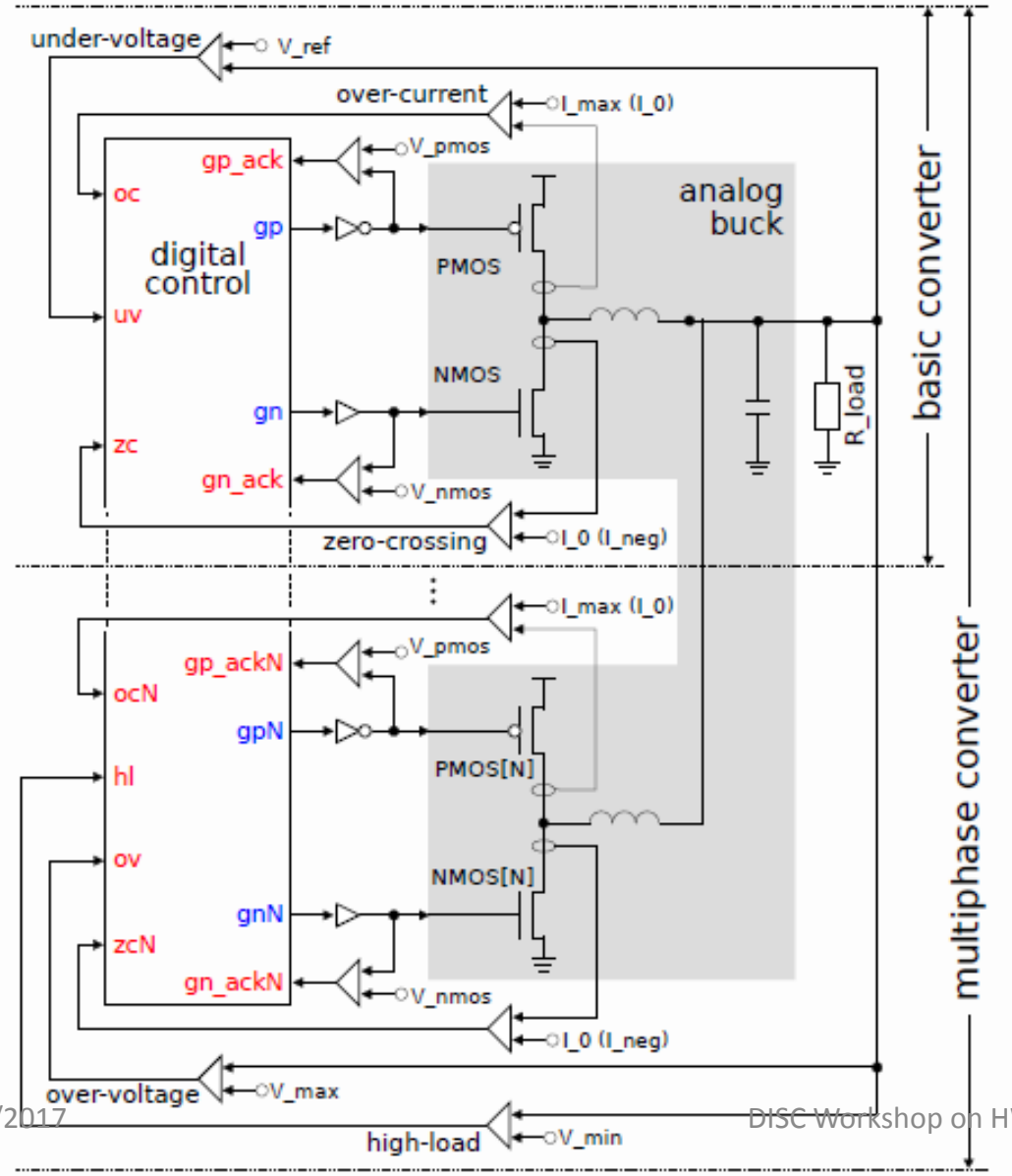
- Gate-level implementation:



WORKCRAFT design automation

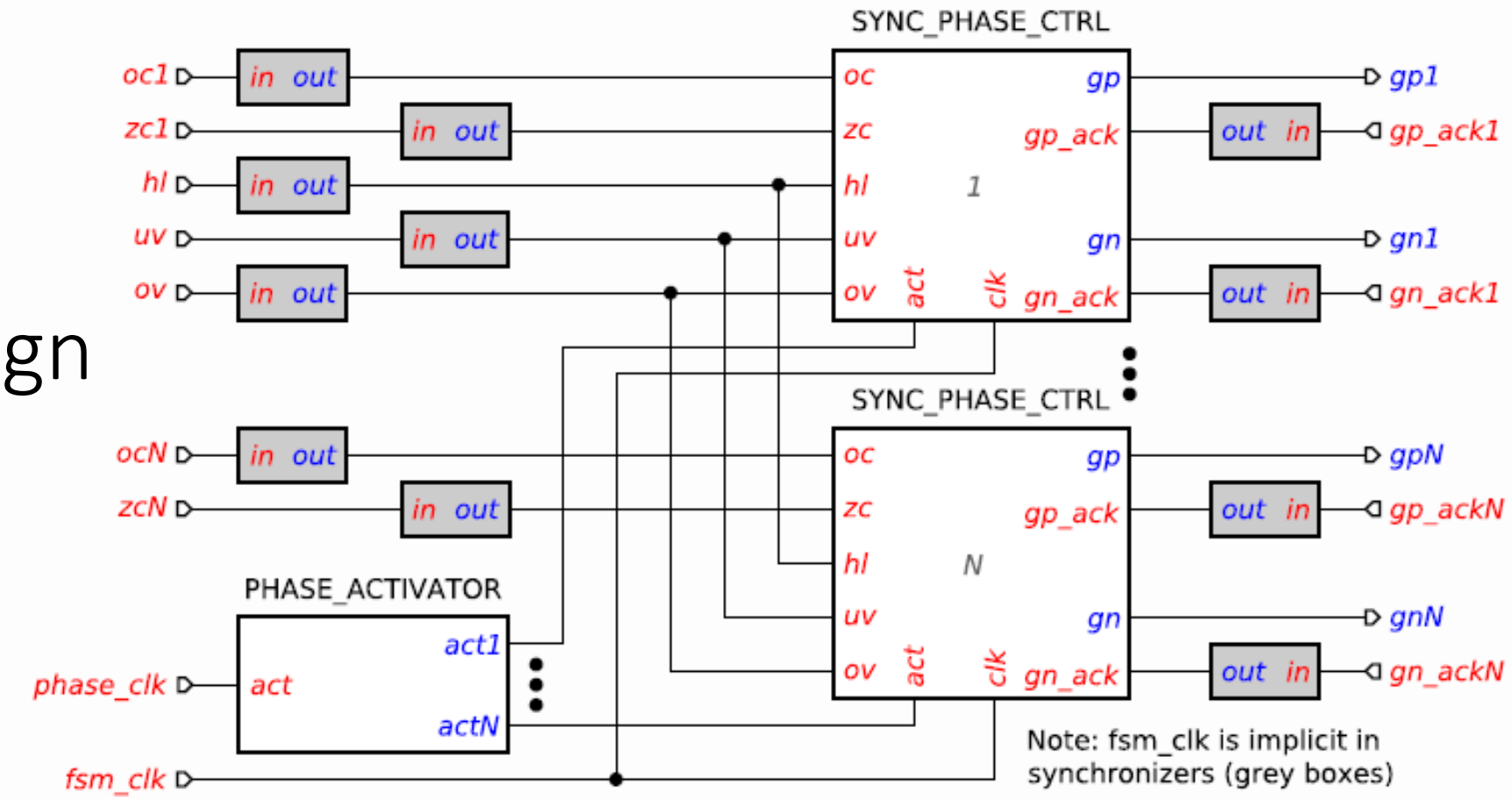


Synthesis example: multiphase Buck



- Activation of phases
  - Sequential
  - May overlap
- More operating modes
  - High-load (HL)
  - Over-voltage (OV)
- Transistor min ON times
  - PMIN delay for PMOS switch
  - NMIN delay for NMOS switch
  - PMIN+PEXT for PMOS at first cycle

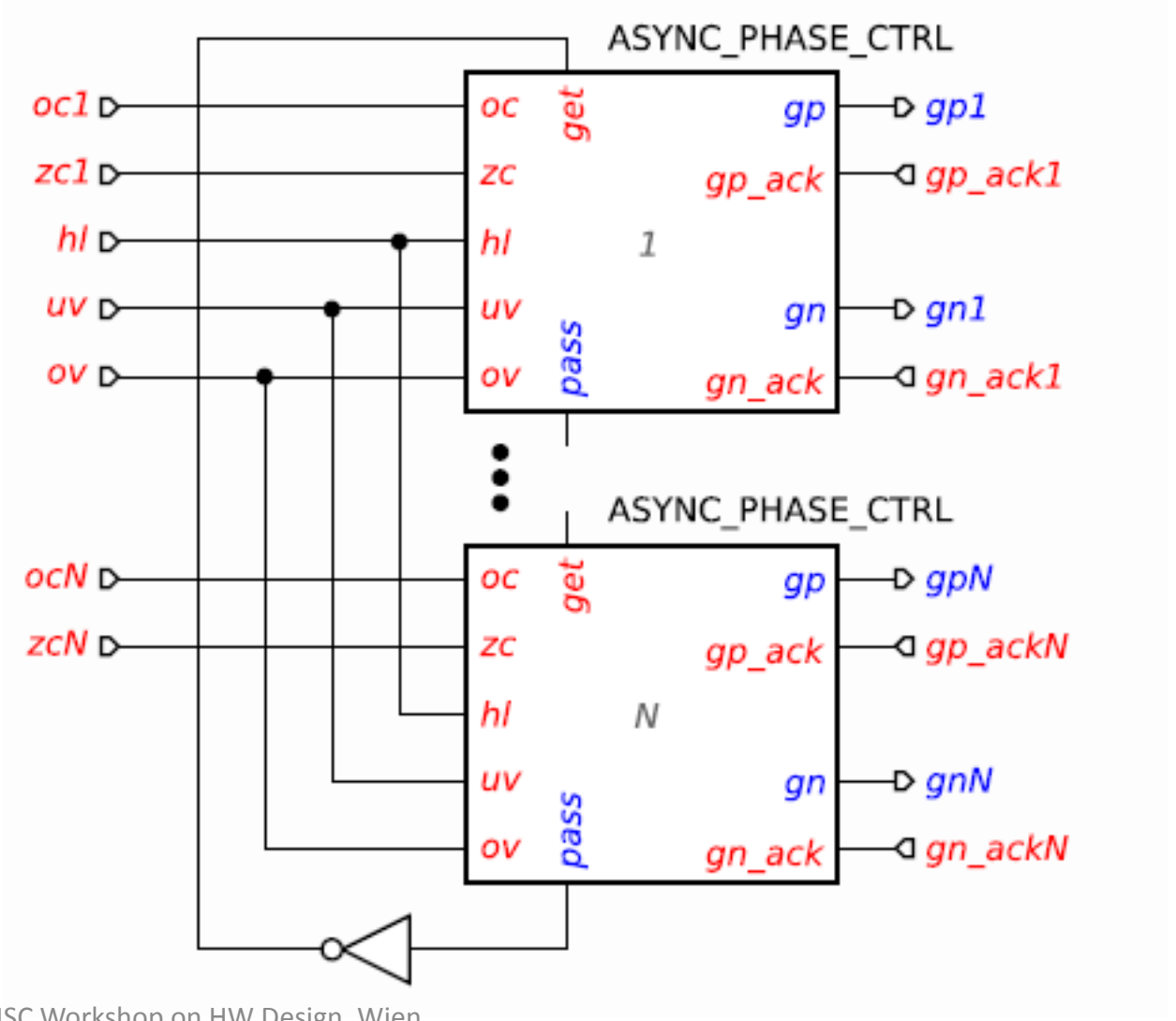
Multiphase buck: synchronous design



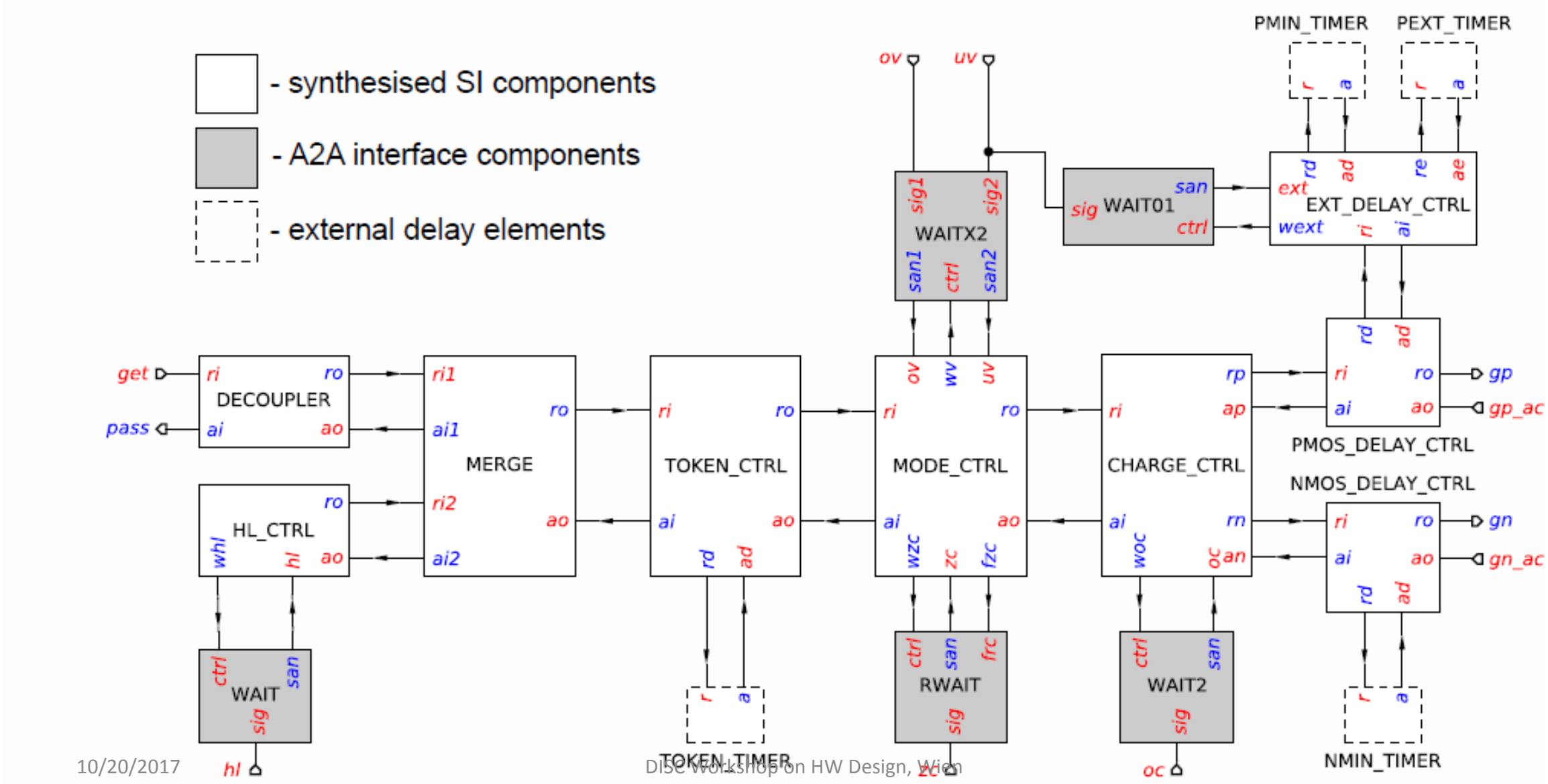
- Two clocks: phase activation (slow) and sampling (fast)
- Conventional RTL design flow for phase control
- Need for multiple synchronizers (grey boxes)

Multiphase buck: async design

- **Token ring architecture, no need for phase activations clock**
- **No need for synchronisers**
- **A4A design flow for phase control**

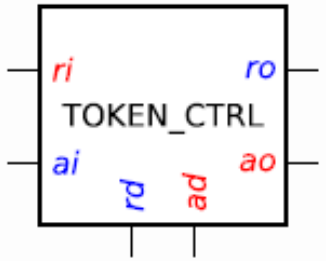


Multiphase buck: async phase control

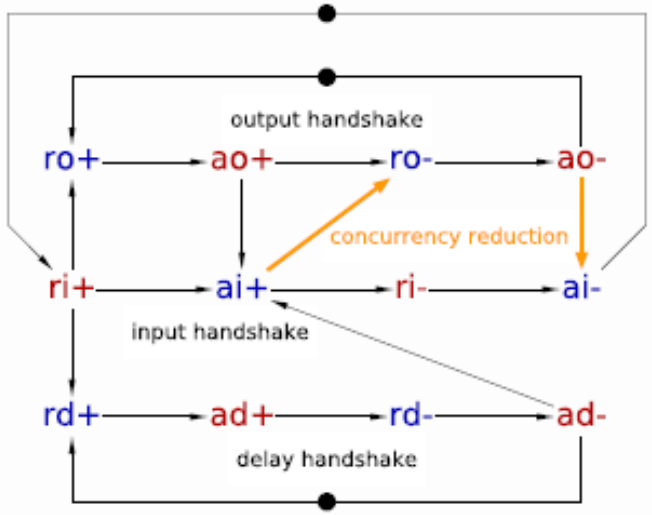


Synthesis of async control components

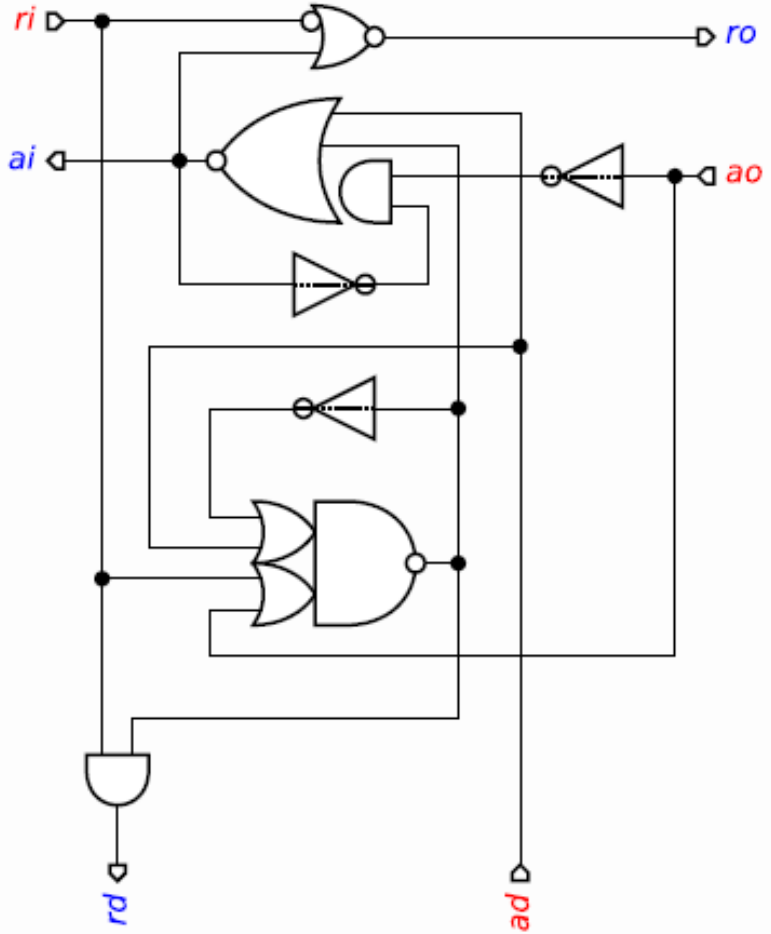
- Token control



- STG specification



- Speed-independent implementation



References on asynchronous little digital design

1. **D. Sokolov, V. Dubikhin, V. Khomenko, D. Lloyd, A. Mokhov, and A. Yakovlev. Benefits of asynchronous control for analog electronics: multiphase buck case study. In Proc. Design, Automation & Test in Europe (DATE), Lausanne, Switzerland, March 2017.**
2. **V. Khomenko, D. Sokolov, A. Mokhov, and A. Yakovlev. WAITX: An arbiter for non-persistent signals. In Proc. IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC), San Diego, CA., May 2017.**
3. **V. Dubikhin, D. Sokolov, A. Yakovlev, and C. J. Myers. Design of mixed-signal systems with asynchronous control. IEEE Design & Test, 33(5):44--55, 2016.**
4. **S. Mileiko, A. Kushnerov, D. Sokolov, and A. Yakovlev. Self-timed control of two-phase switched capacitor converters. In IEEE International Conference on the Science of Electrical Engineering (ICSEE), Eilat, Israel, November 2016.**
5. **A. Mokhov, D. Sokolov, V. Khomenko and A. Yakovlev. Asynchronous Arbitration Primitives for New Generation of Circuits and Systems. In IEEE New Generation of Circuits and Systems (NGCAS), Genoa, Italy, September 2017.**

Messages to take away

- Little digital circuits can be highly concurrent!
- Asynchronous circuits began their life (in the 50s) for ‘little digital’ and today is the right time for them
- Analog and mixed-signal is a good application – it combines:
  - Need for low latency and high range of feedback types
  - Analog designers are more inclined towards async than digital designers
- Design tools are (slowly) coming up and industry is a good drive!
- Interesting research problems are there – tech mapping, holistic analog-mixed signal verification, behavioural mining, dealing with complexity
- In particular, extending the notion of speed-independence into the world of relative timing, circuits with time comparison (arbitration), with analog components
- Where else do we have little digital? … Plastic electronics?