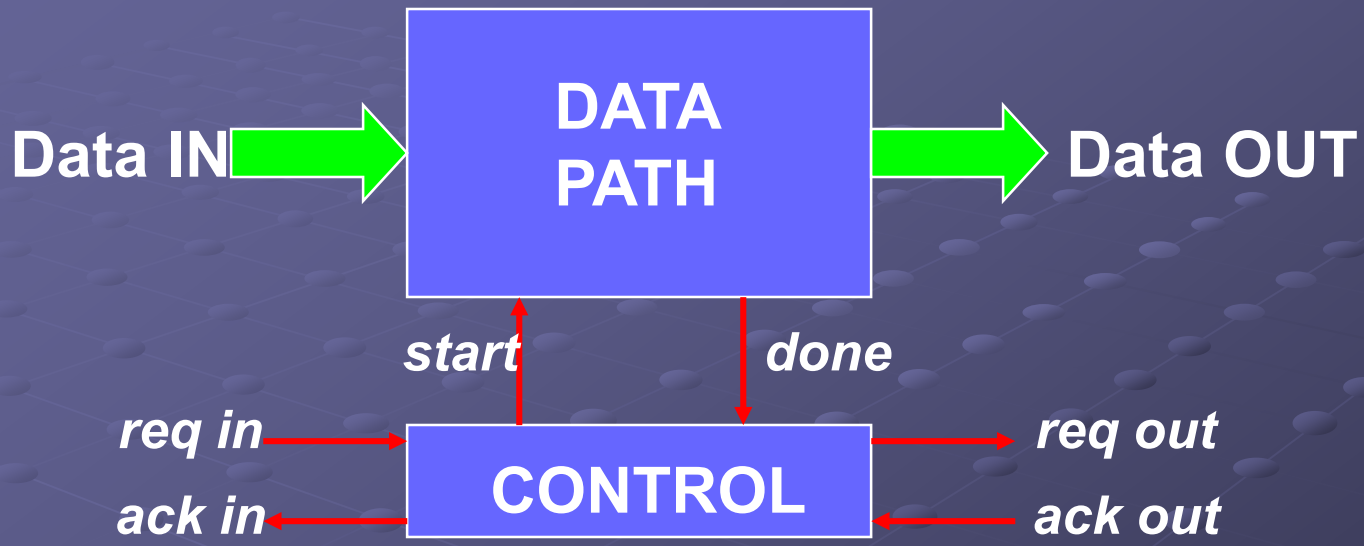
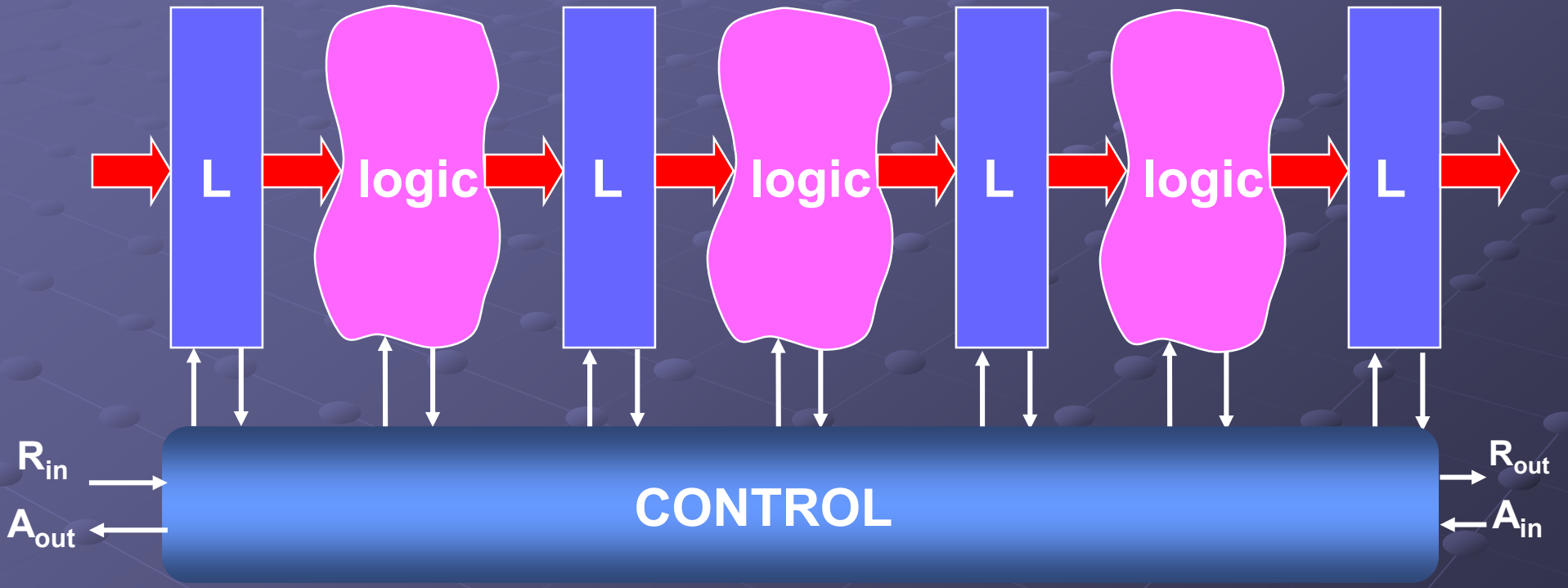
Asynchronous modules



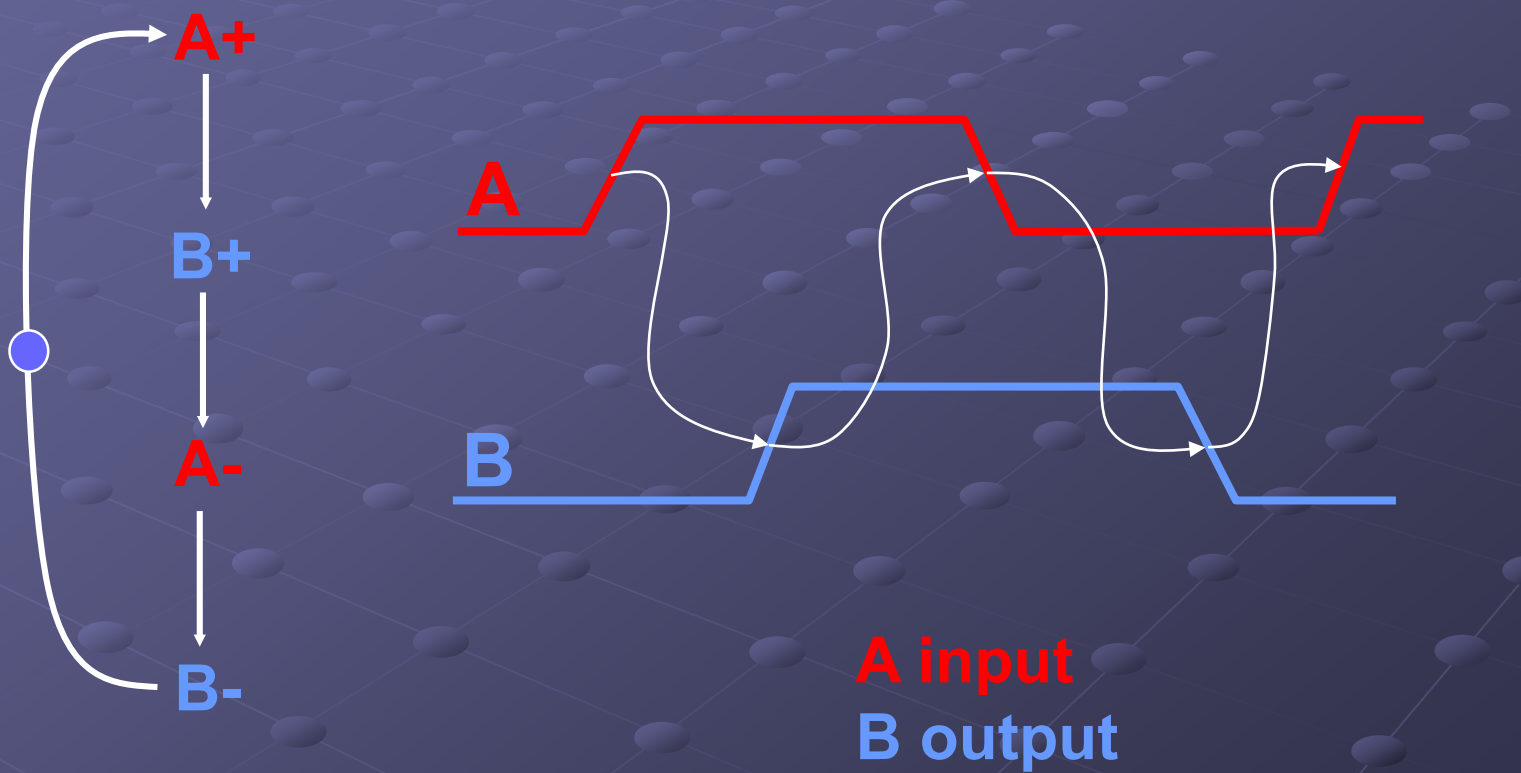
- **Signaling protocol:**

  reqin+ start+ [*computation*] done+ reqout+ ackout+ ackin+
  reqin- start- [*reset*] done- reqout- ackout- ackin-
  (more concurrency is also possible)

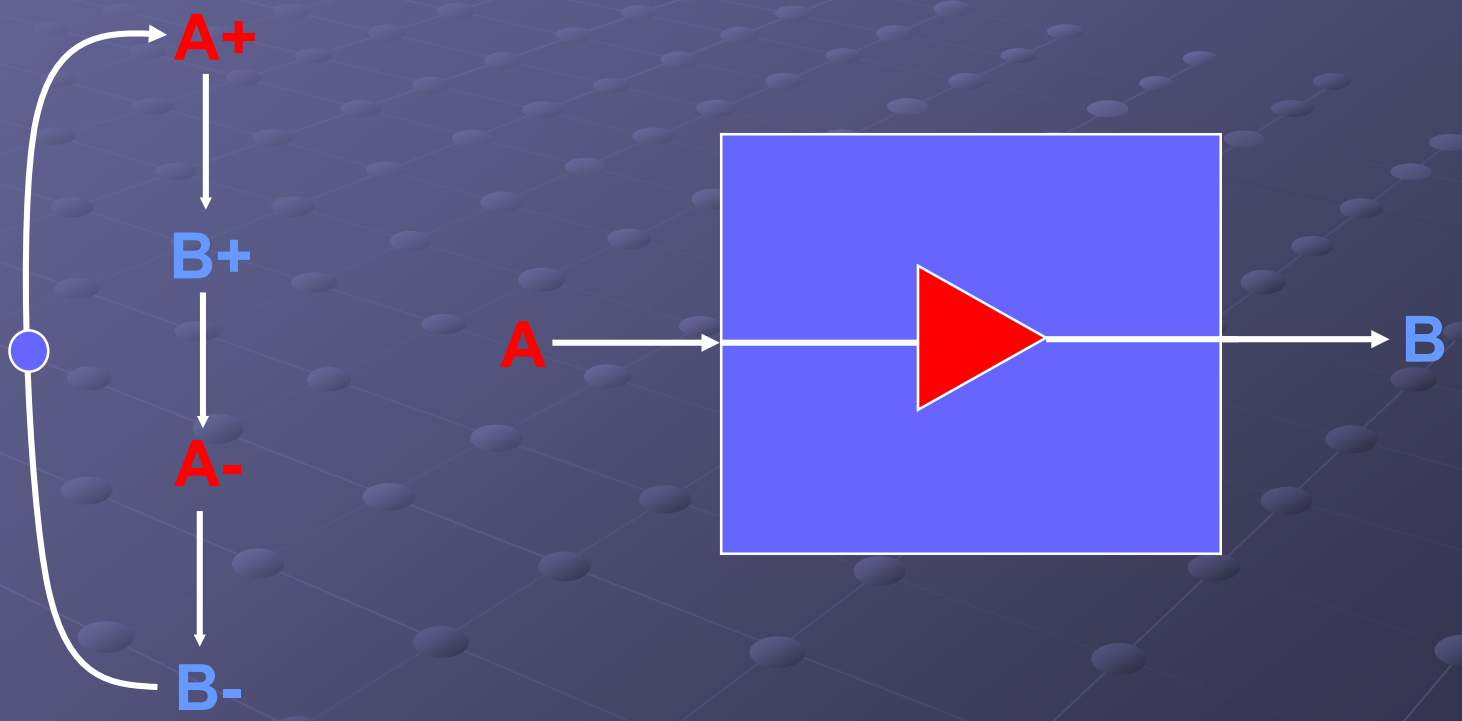
Data-path / Control



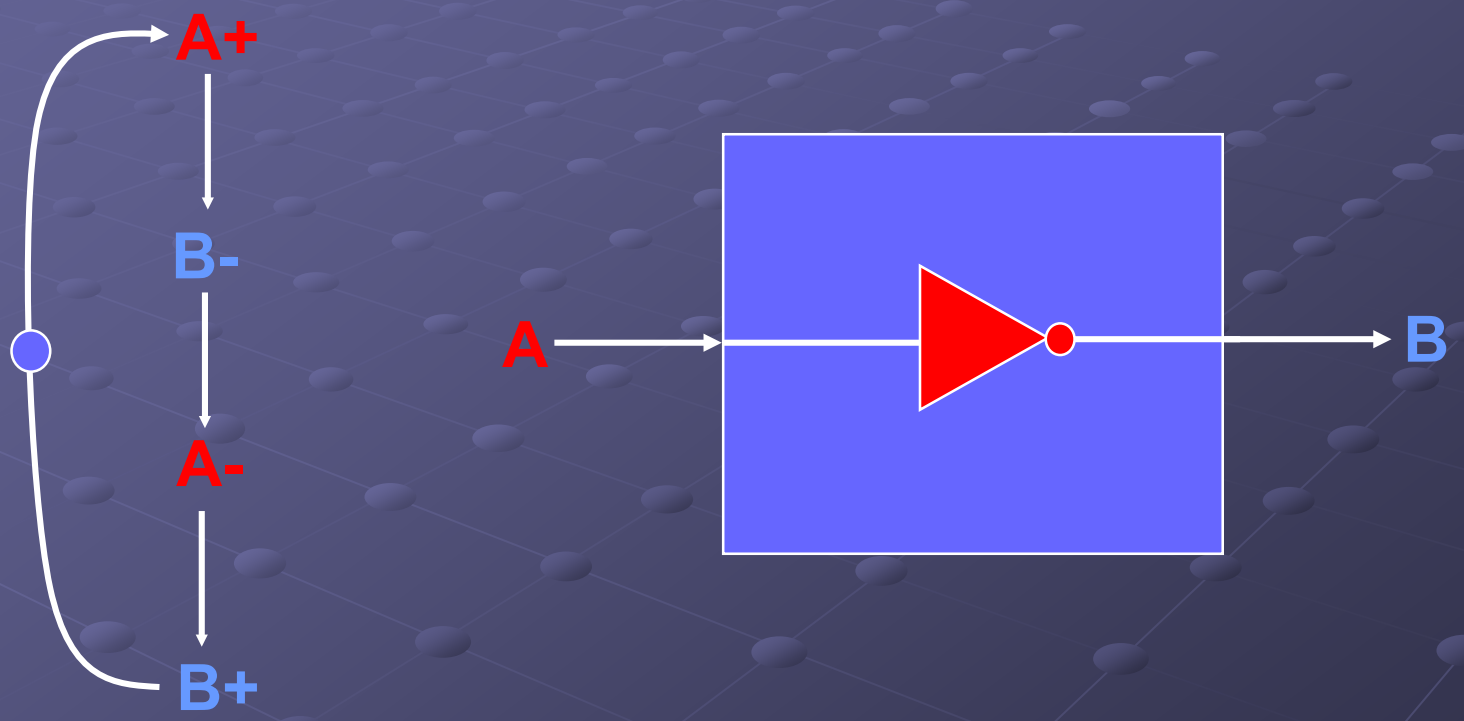
Control specification



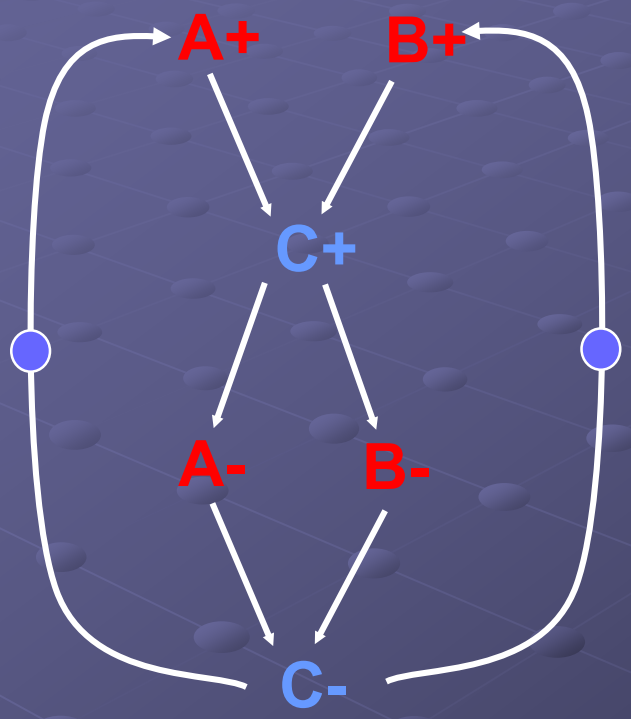
Control specification

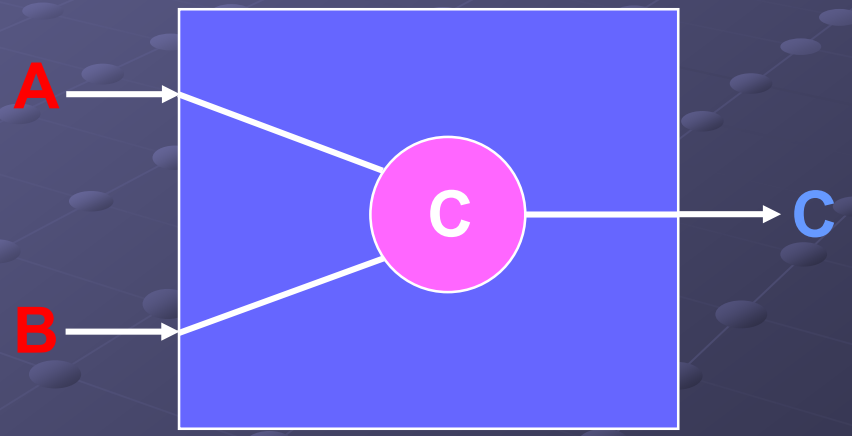


Control specification

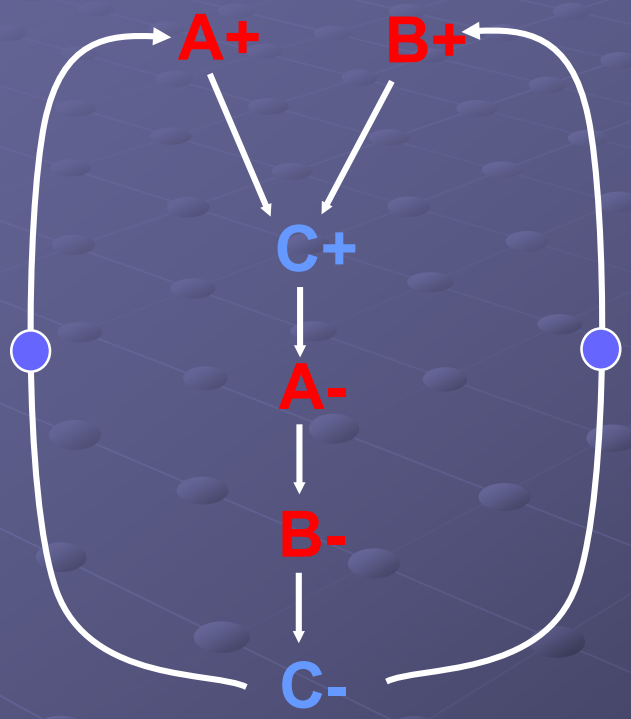


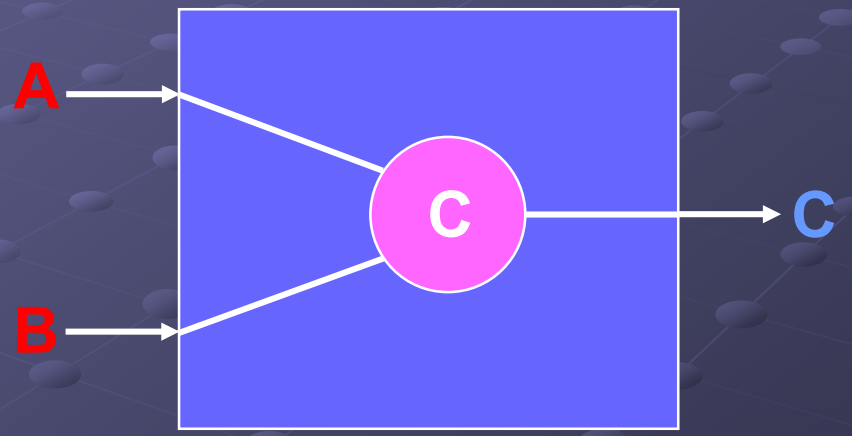
Control specification



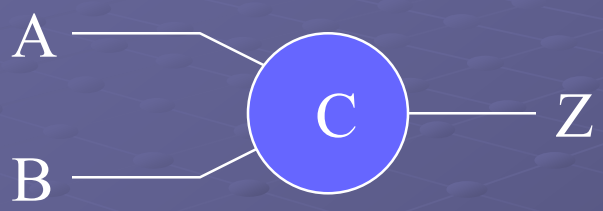


Control specification



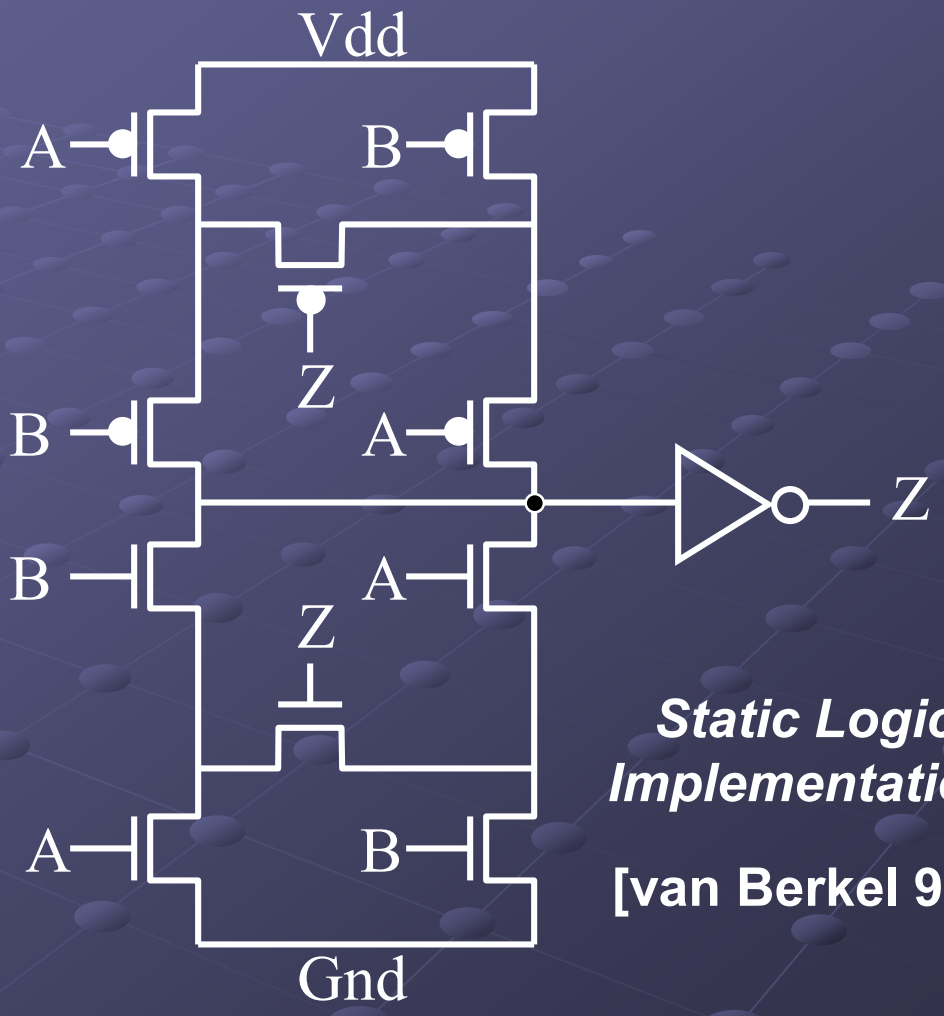


Asynchronous latches: C element



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

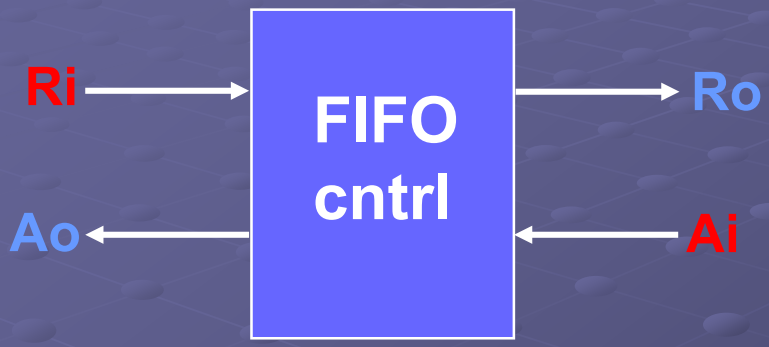
$Z=AB+(A+B)Z$

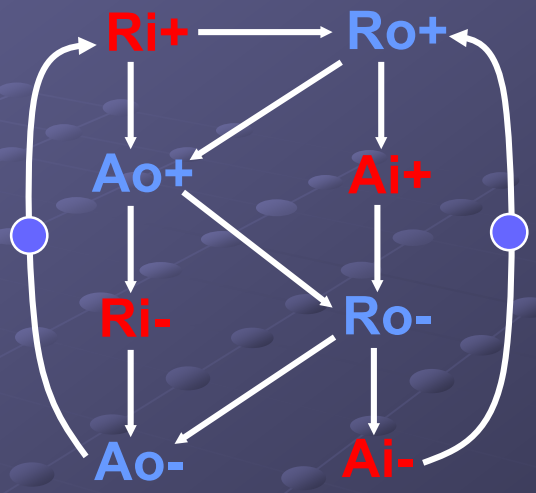


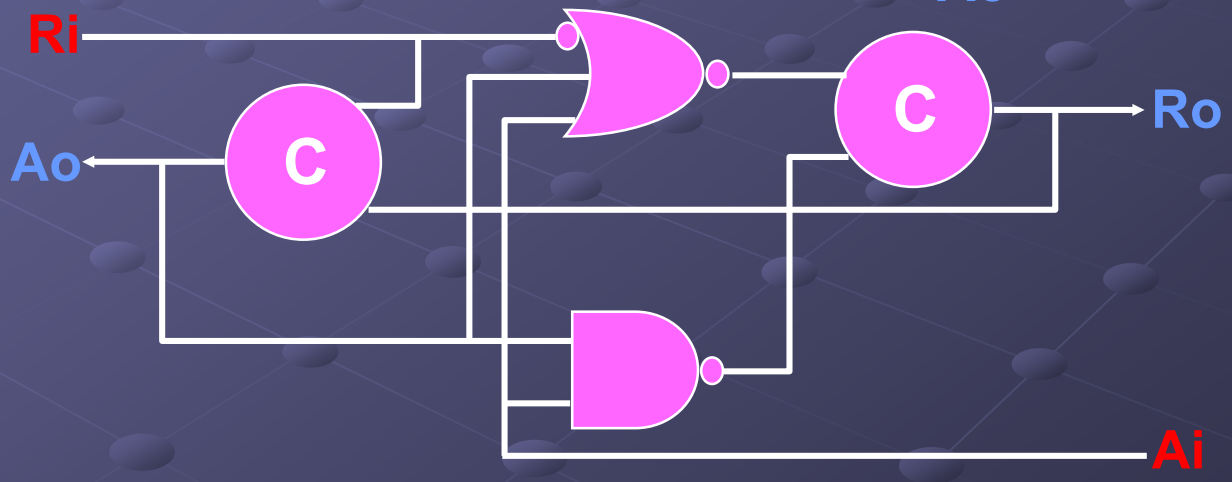
Static Logic Implementation

[van Berkel 91]

Control specification



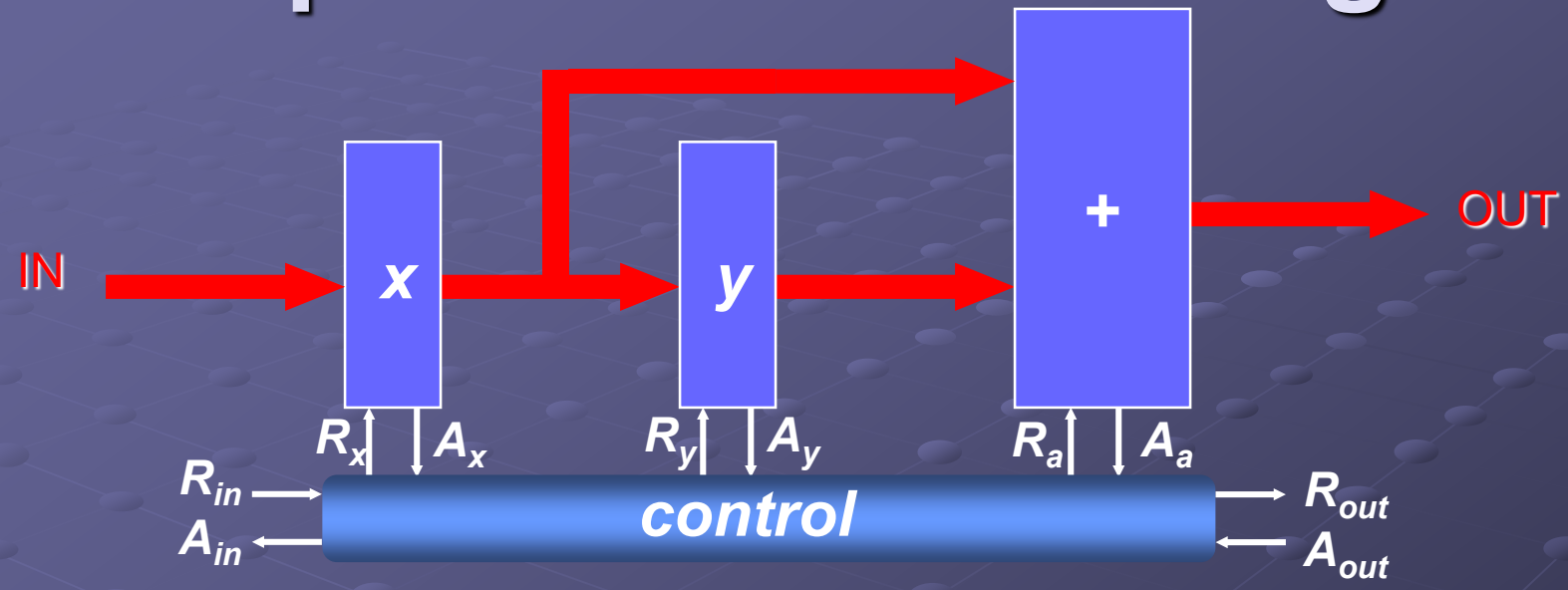




A simple filter: specification

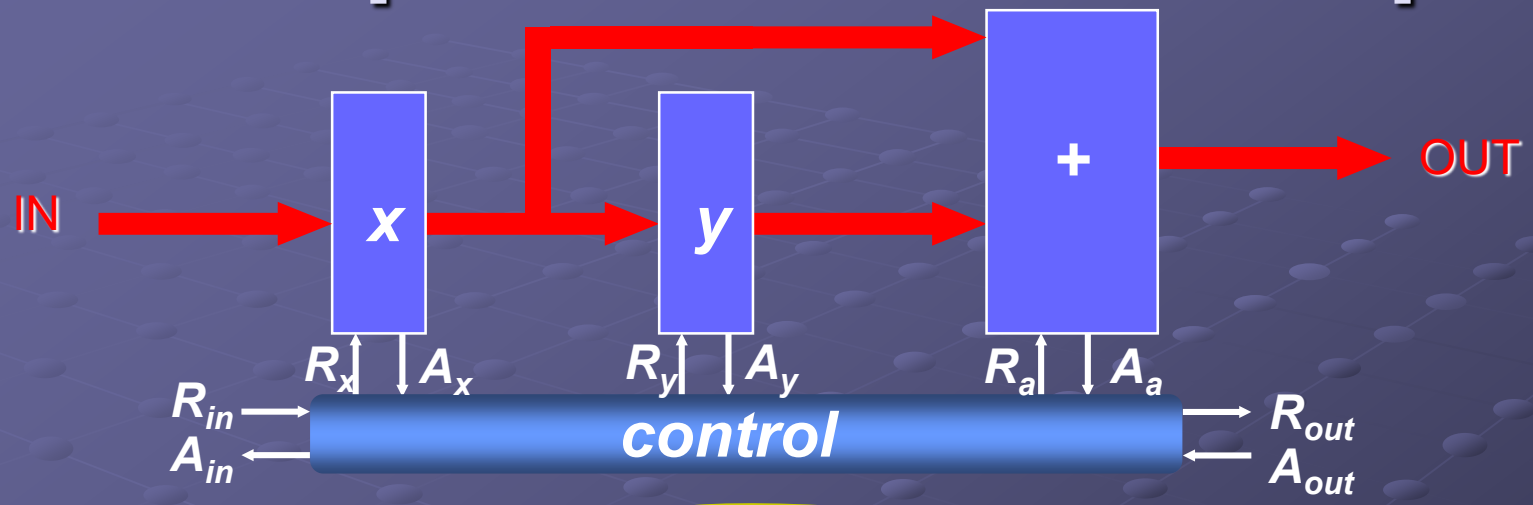
```
y := 0;
loop
    x := READ (IN);
    WRITE (OUT, (x+y)/2);
    y := x;
end loop
```

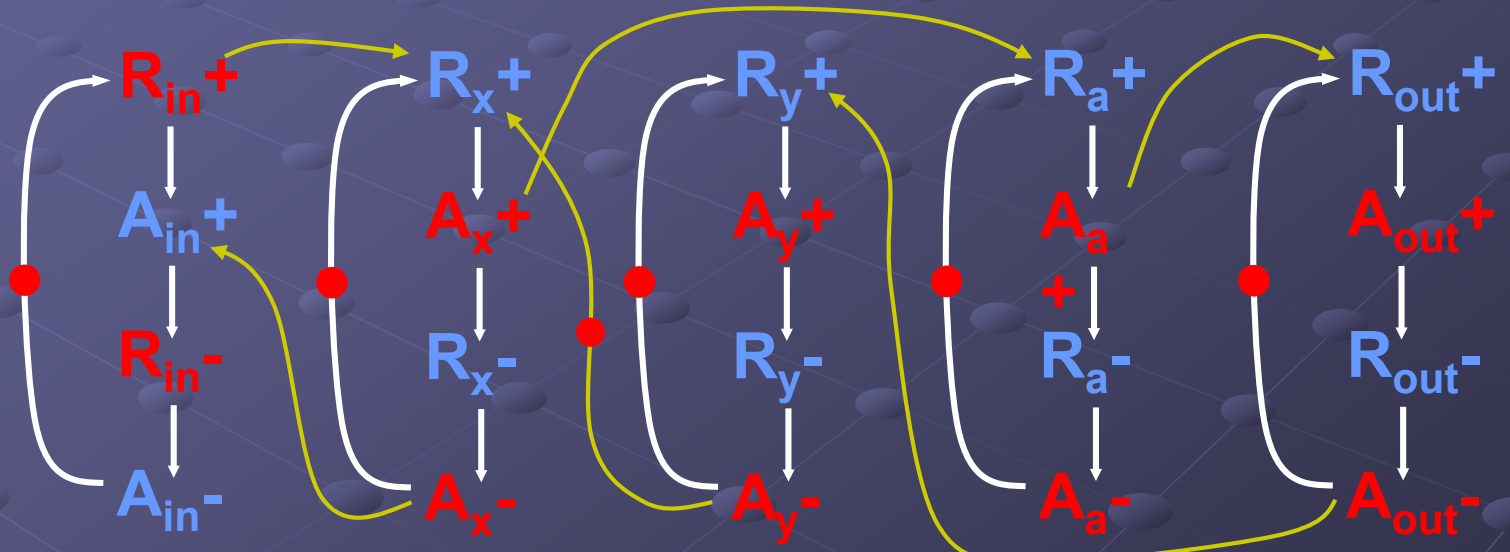
A simple filter: block diagram



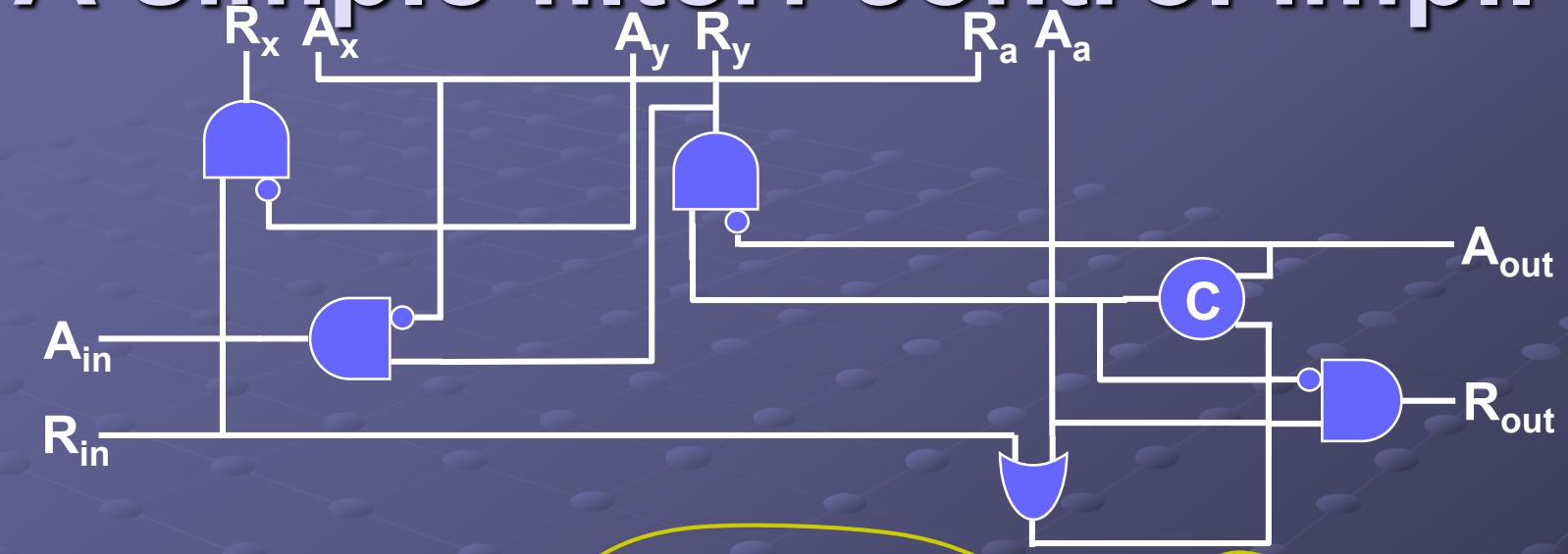
- *x* and *y* are level-sensitive latches (transparent when R=1)
- + is a bundled-data adder (matched delay between R_a and A_a)
- R_{in} indicates the validity of IN
- After A_{in}+ the environment is allowed to change IN
- (R_{out},A_{out}) control a level-sensitive latch at the output

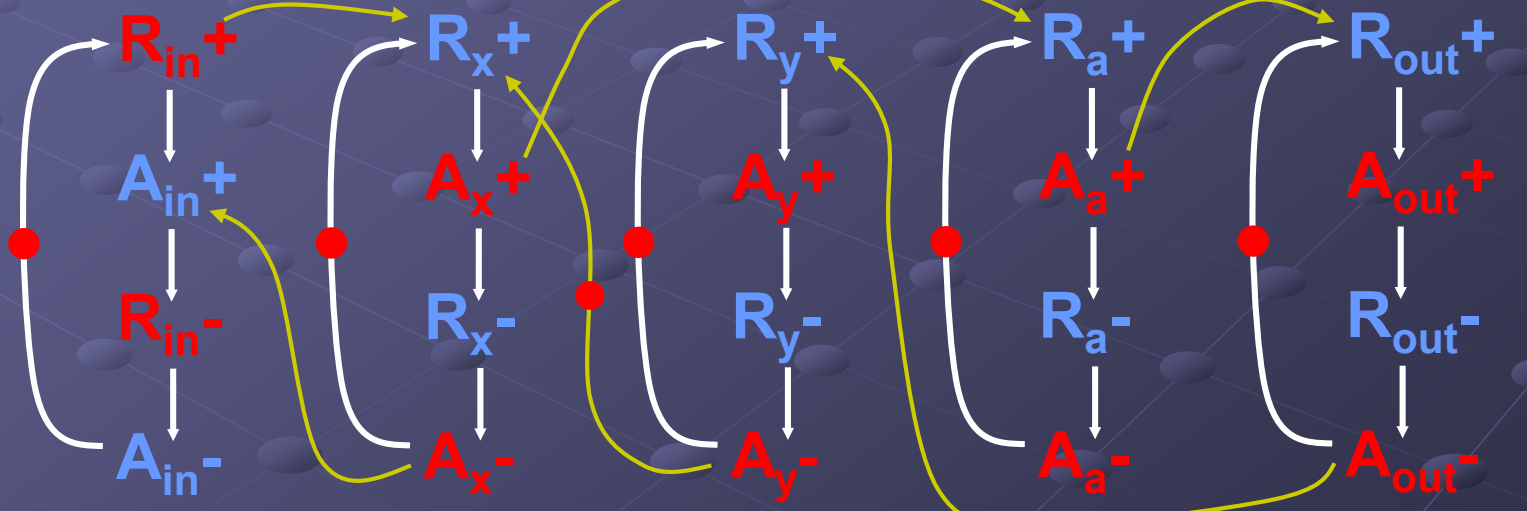
A simple filter: control spec.



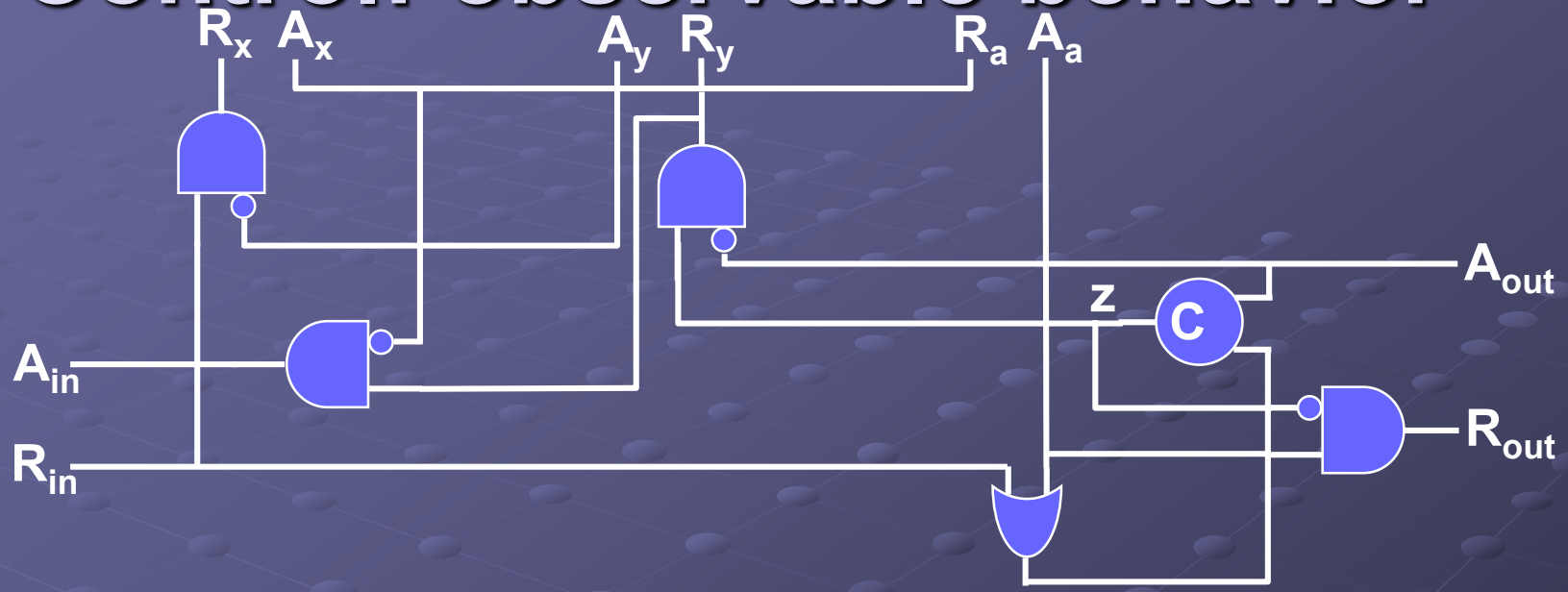


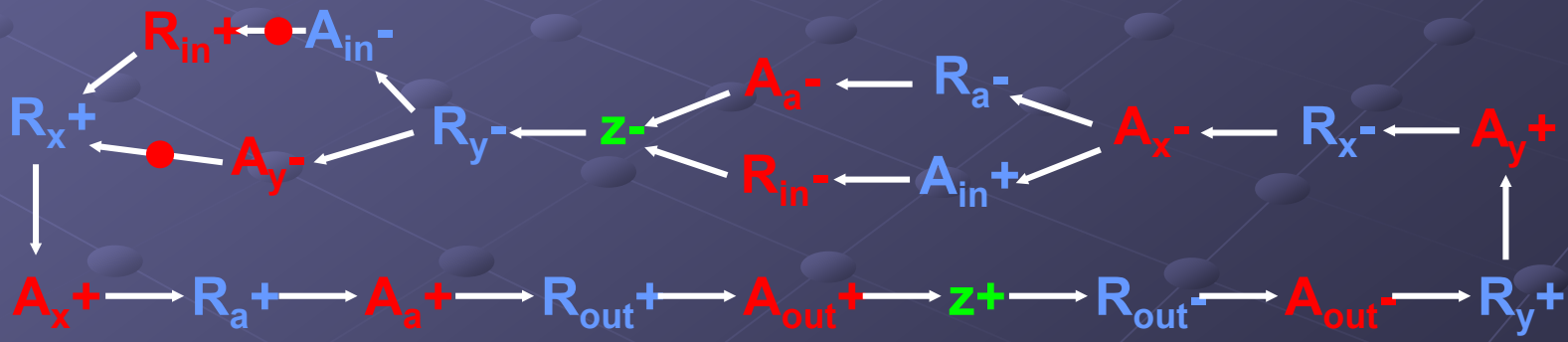
A simple filter: control impl.



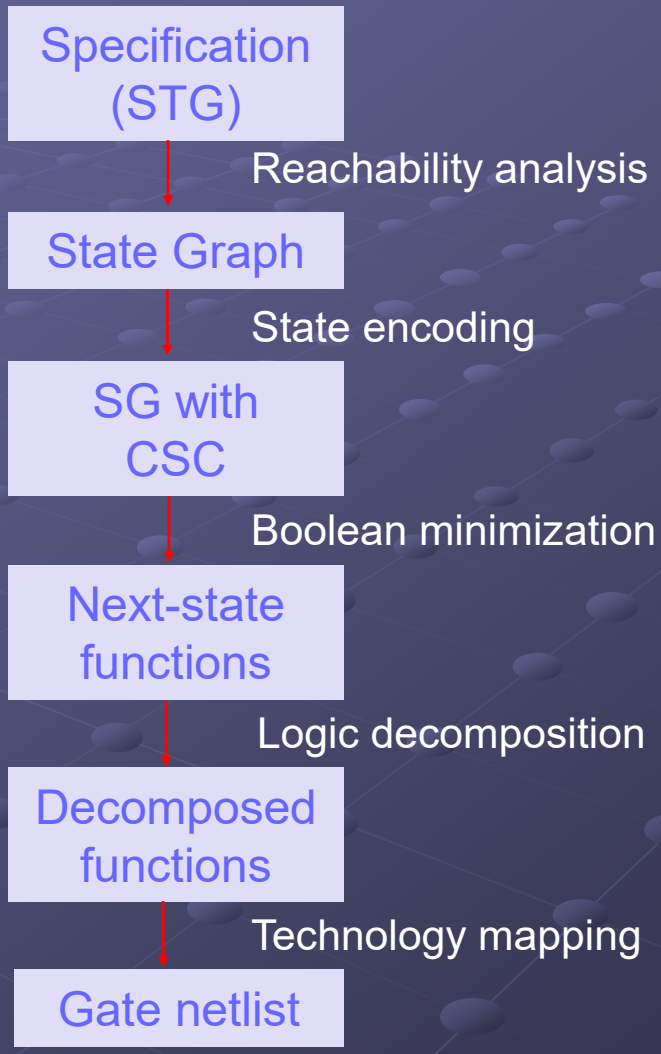


Control: observable behavior

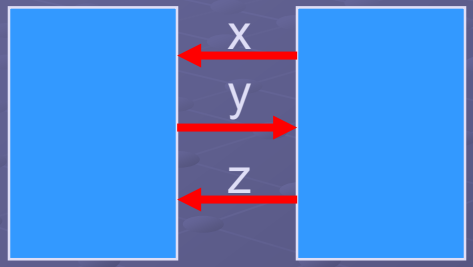


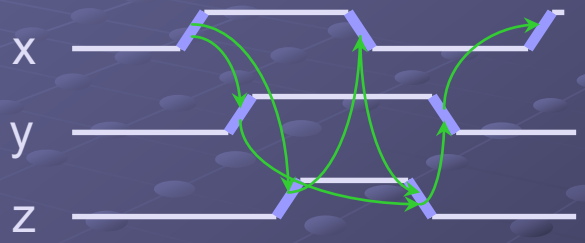


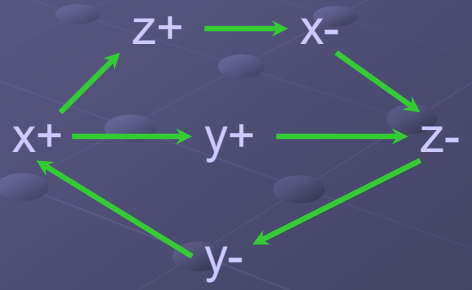
Basic synthesis flow



xyz-example: Specification

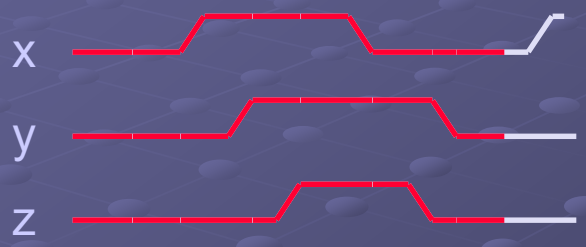


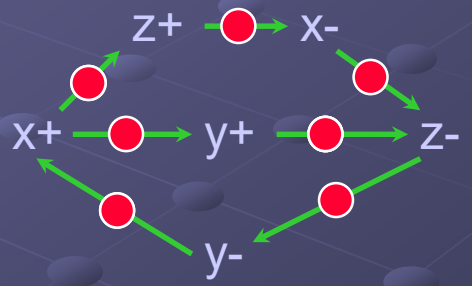




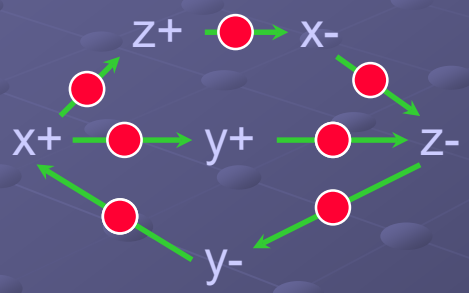
Signal Transition Graph (STG)

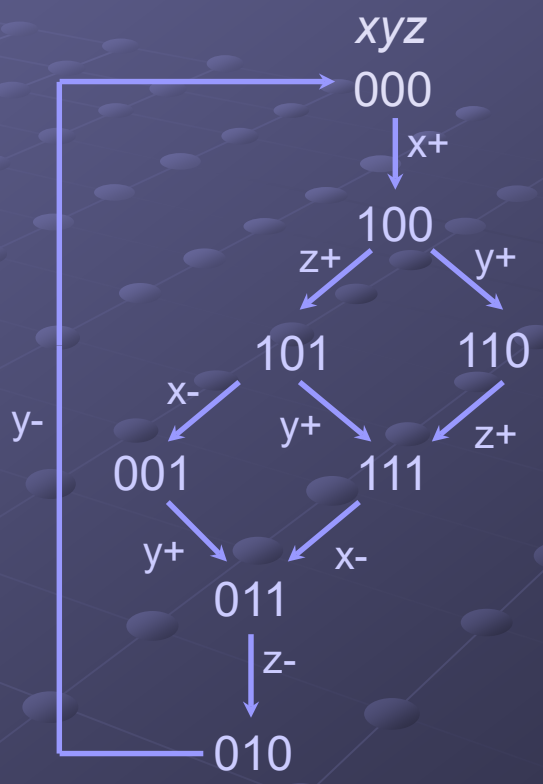
Token flow





State graph



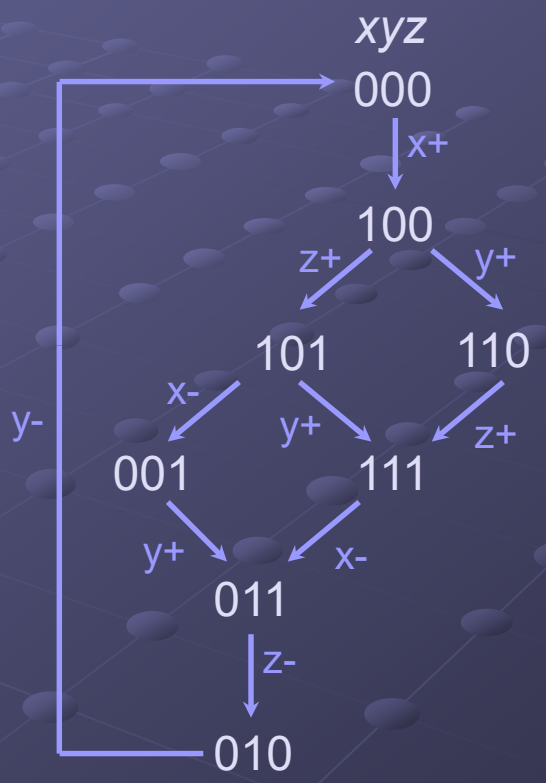


Next-state functions

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

$$y = z + x$$

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



Deriving next state functions

1) **Truth Table**

Previous state	Next State
0*0 0	1 0 0
1 0*0*	1 1 1
0 1*0	0 0 0
1 1 0*	1 1 1
0 0*1	0 1 1
1*0*1	0 1 1
0 1 1*	0 1 0
1*1 1	0 1 1

2) **Boolean Minimization**

xy / z	00	10	11	01
0	1	1	1	0
1	0	0	0	0

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

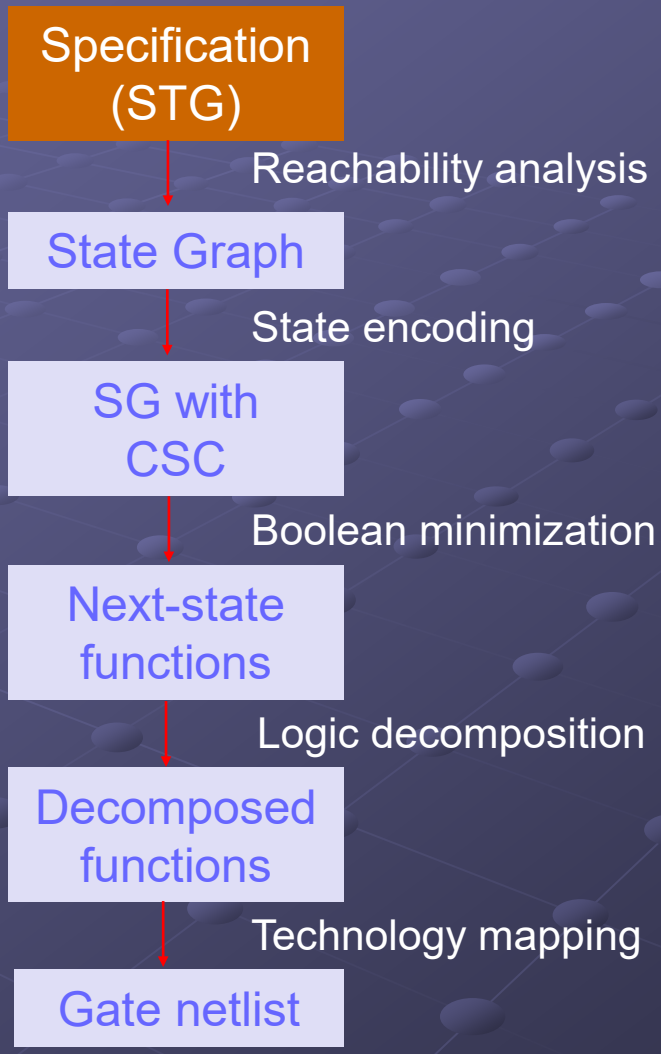
Observations in this example:

1) All combinations are used as states

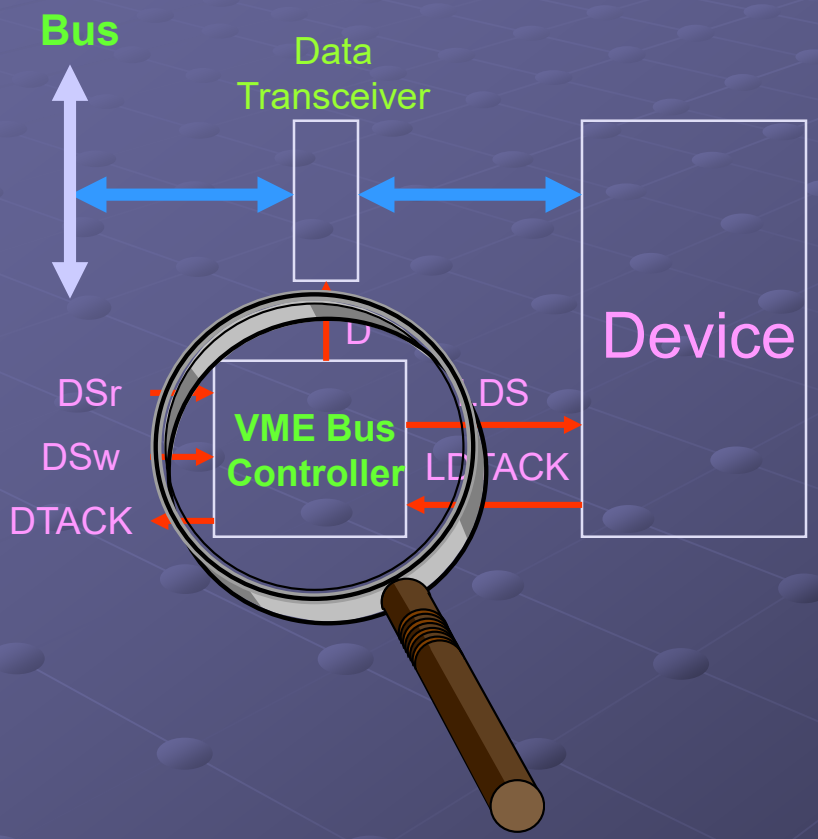
2) All states appear uniquely

Generally, this is not always the case!

Design flow

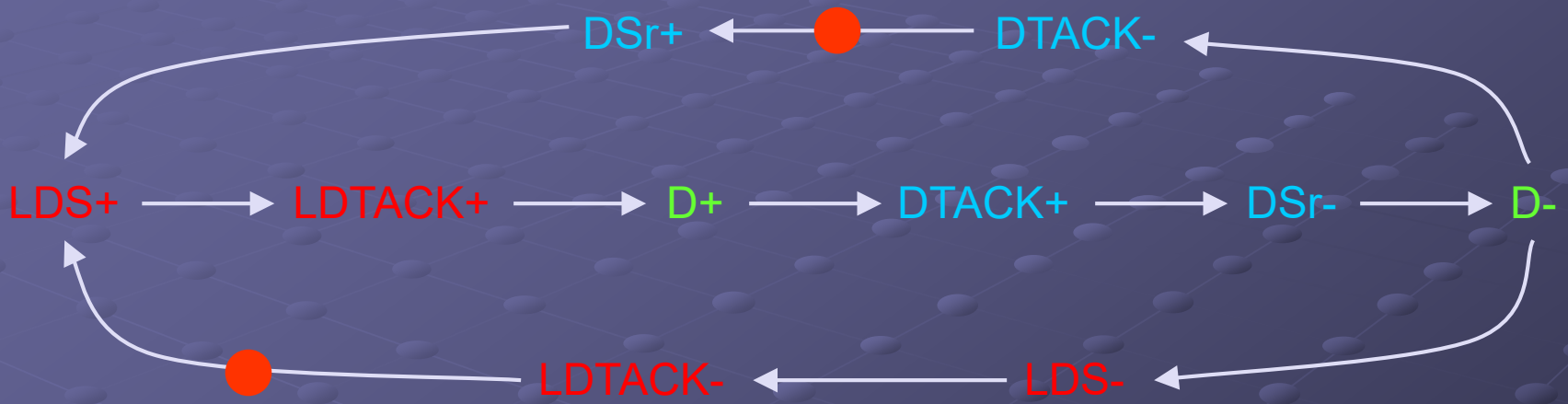

Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

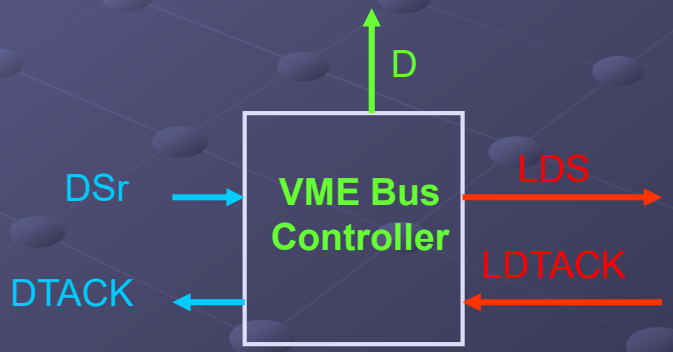
VME bus



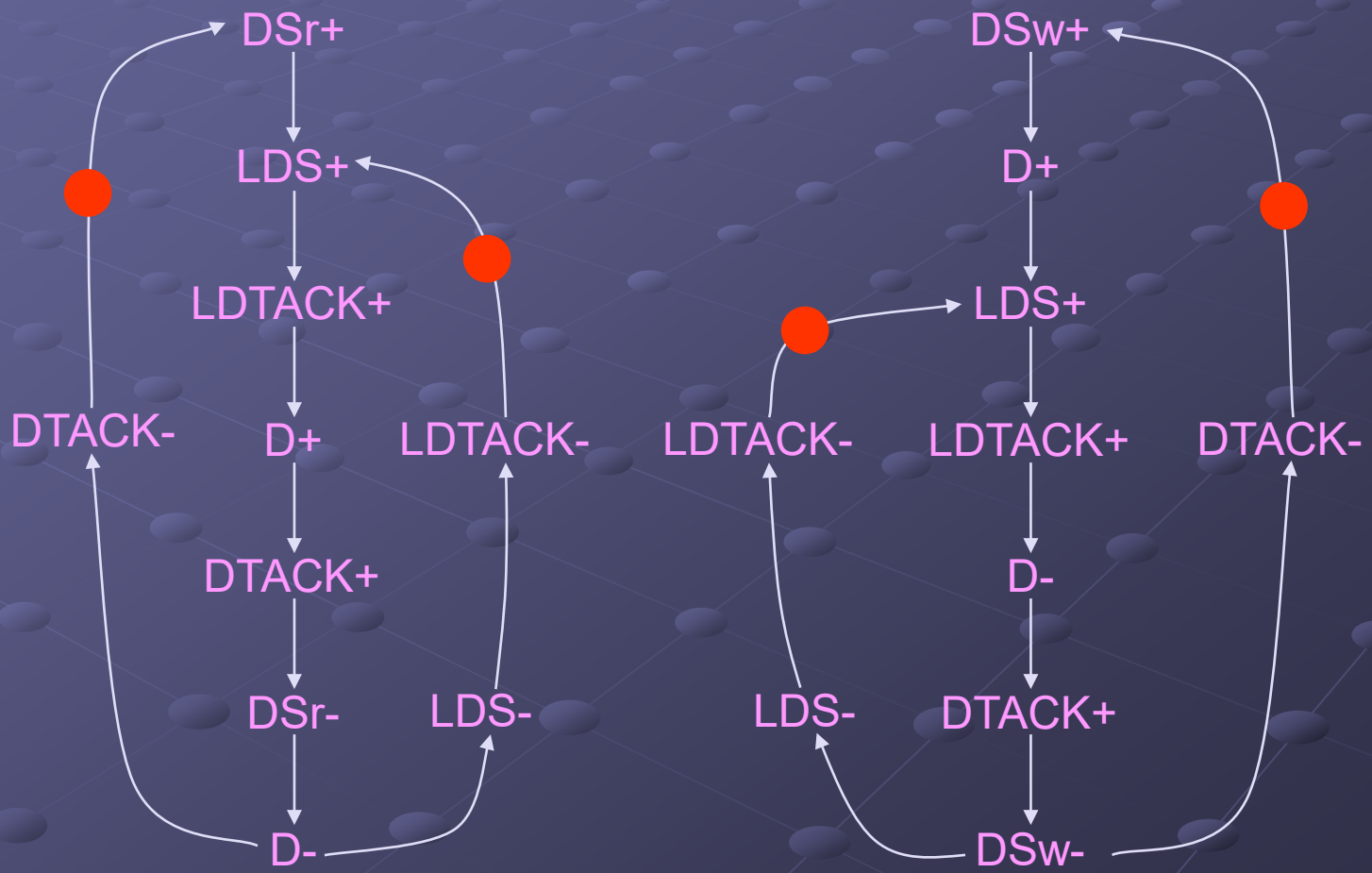
Read Cycle

STG for the READ cycle

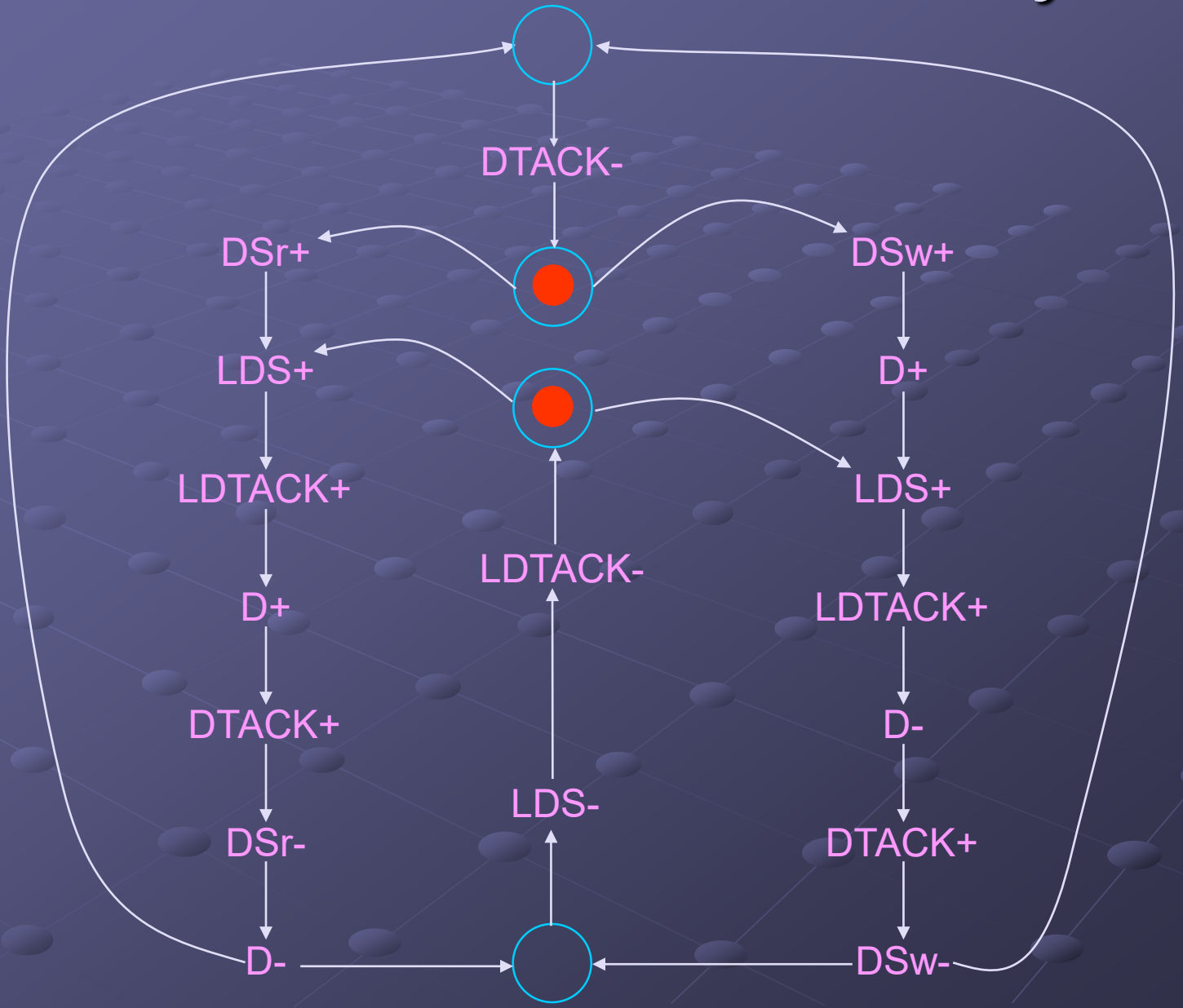




Choice: Read and Write cycles



Choice: Read and Write cycles



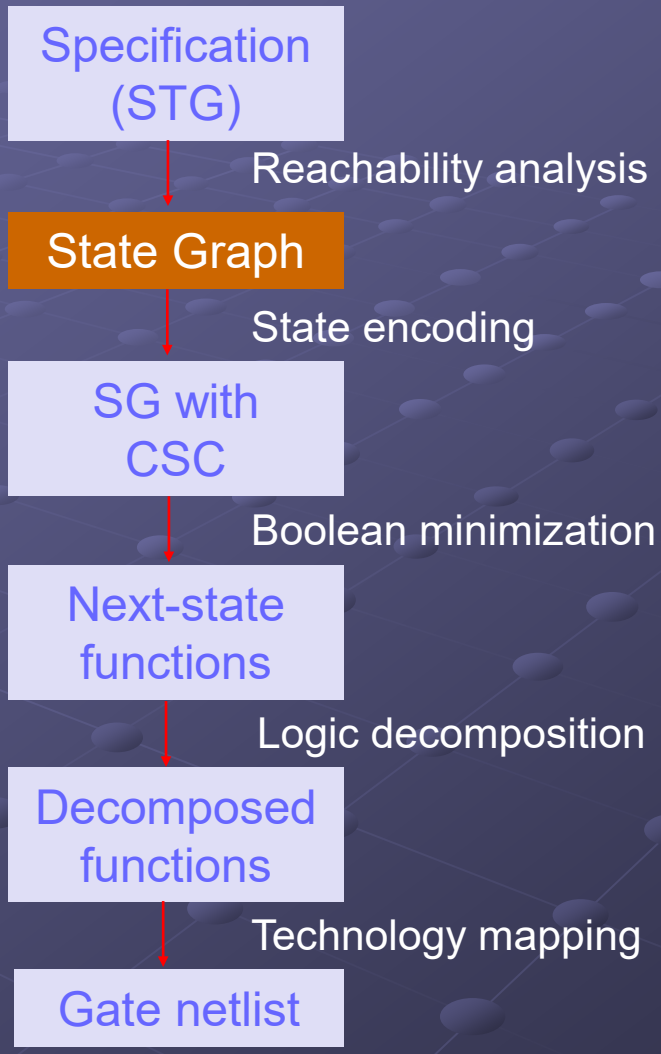
Circuit synthesis

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

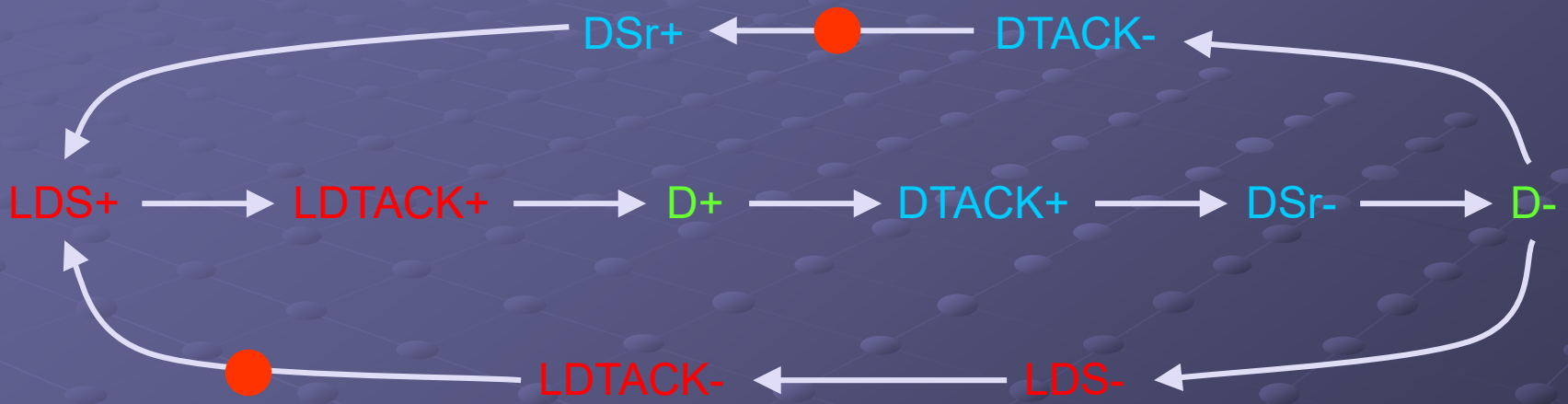
Speed independence

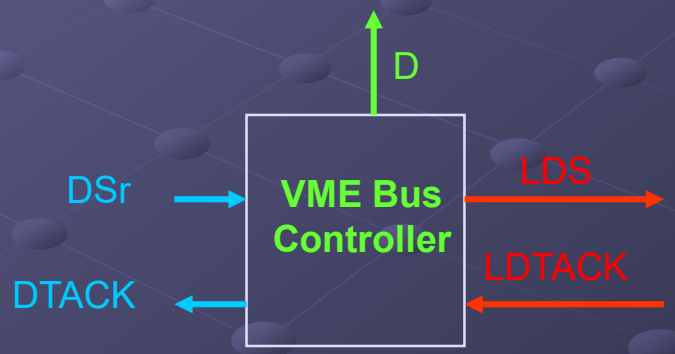
- Delay model
  - Unbounded gate / environment delays
  - Certain wire delays shorter than certain paths in the circuit
- Conditions for implementability:
  - Consistency
  - Complete State Coding
  - Persistency

Design flow

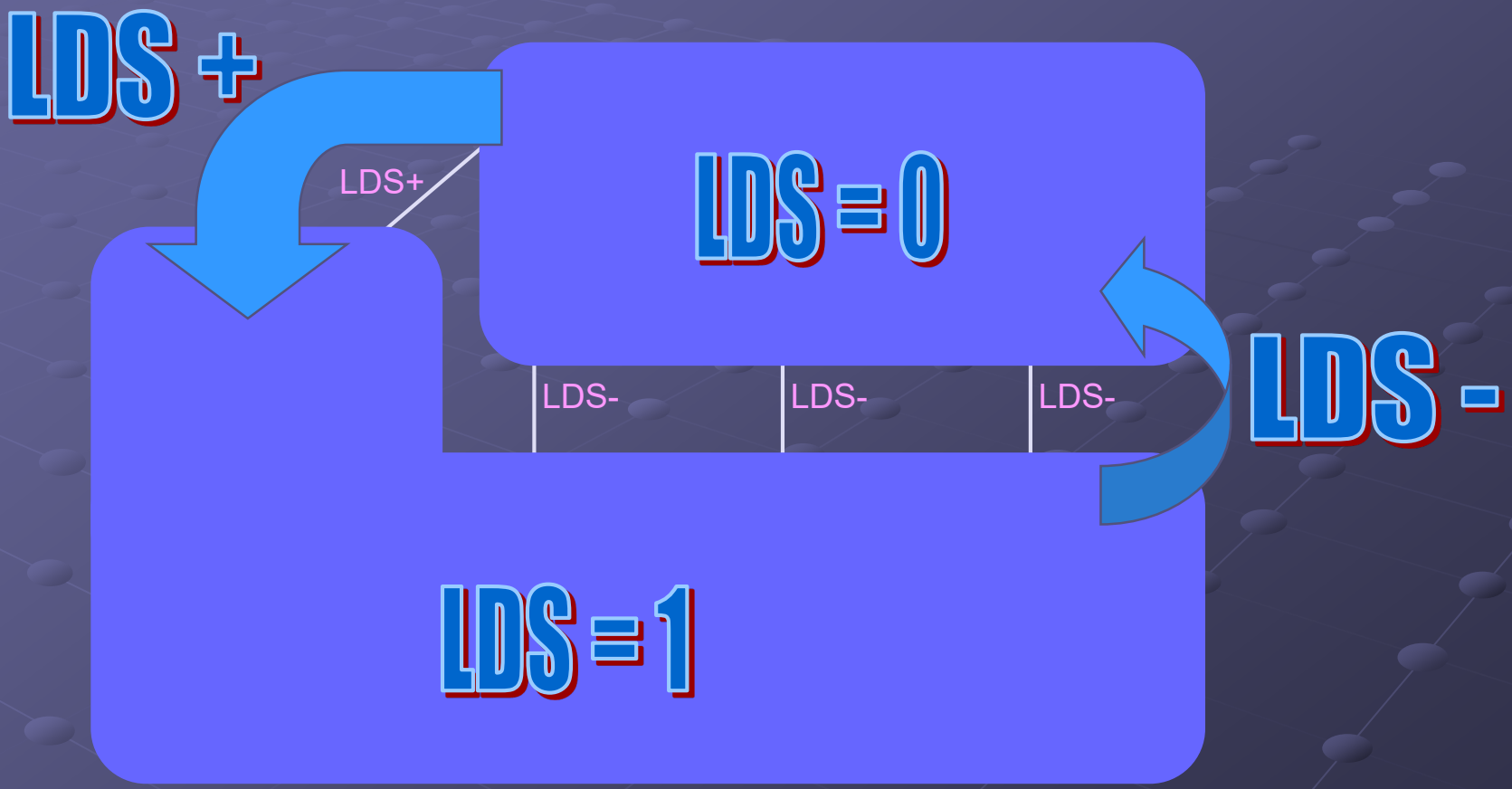

Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

STG for the READ cycle

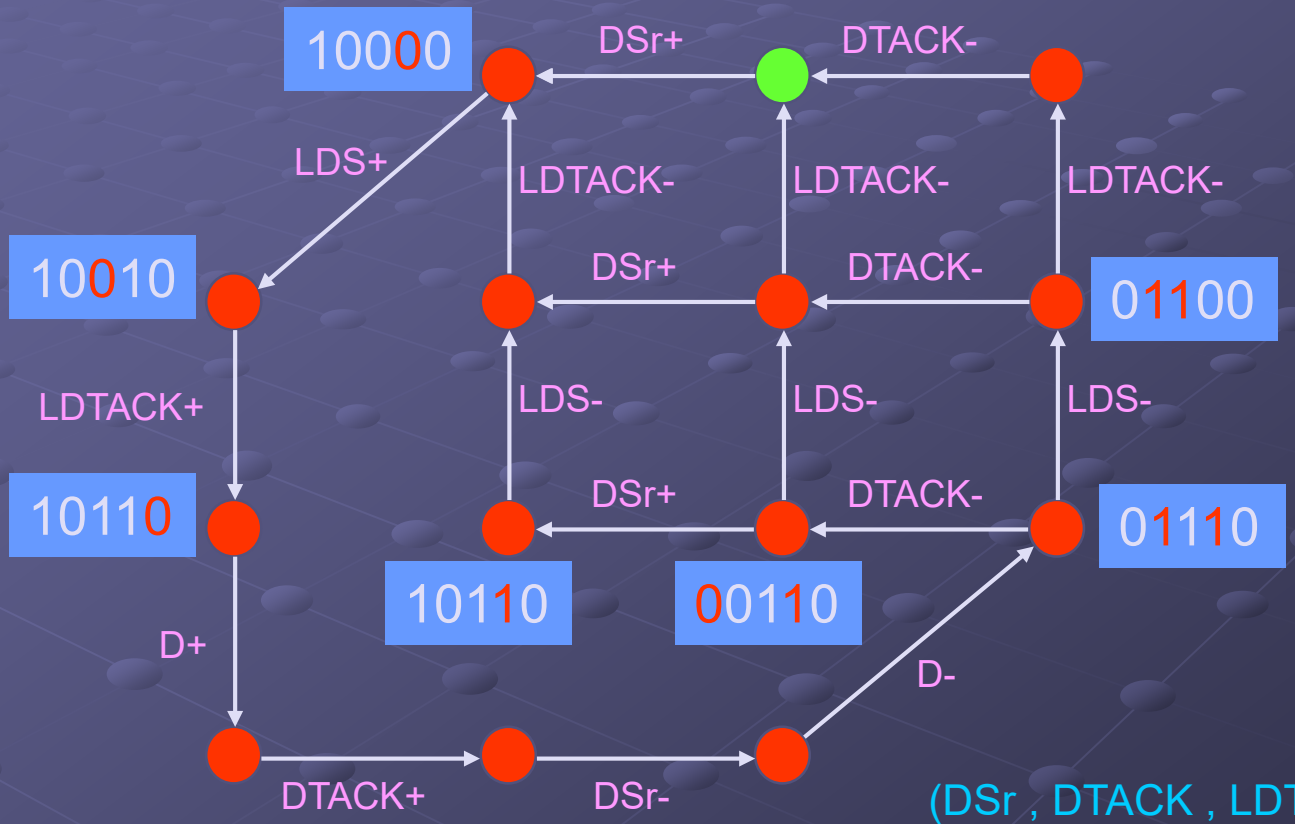




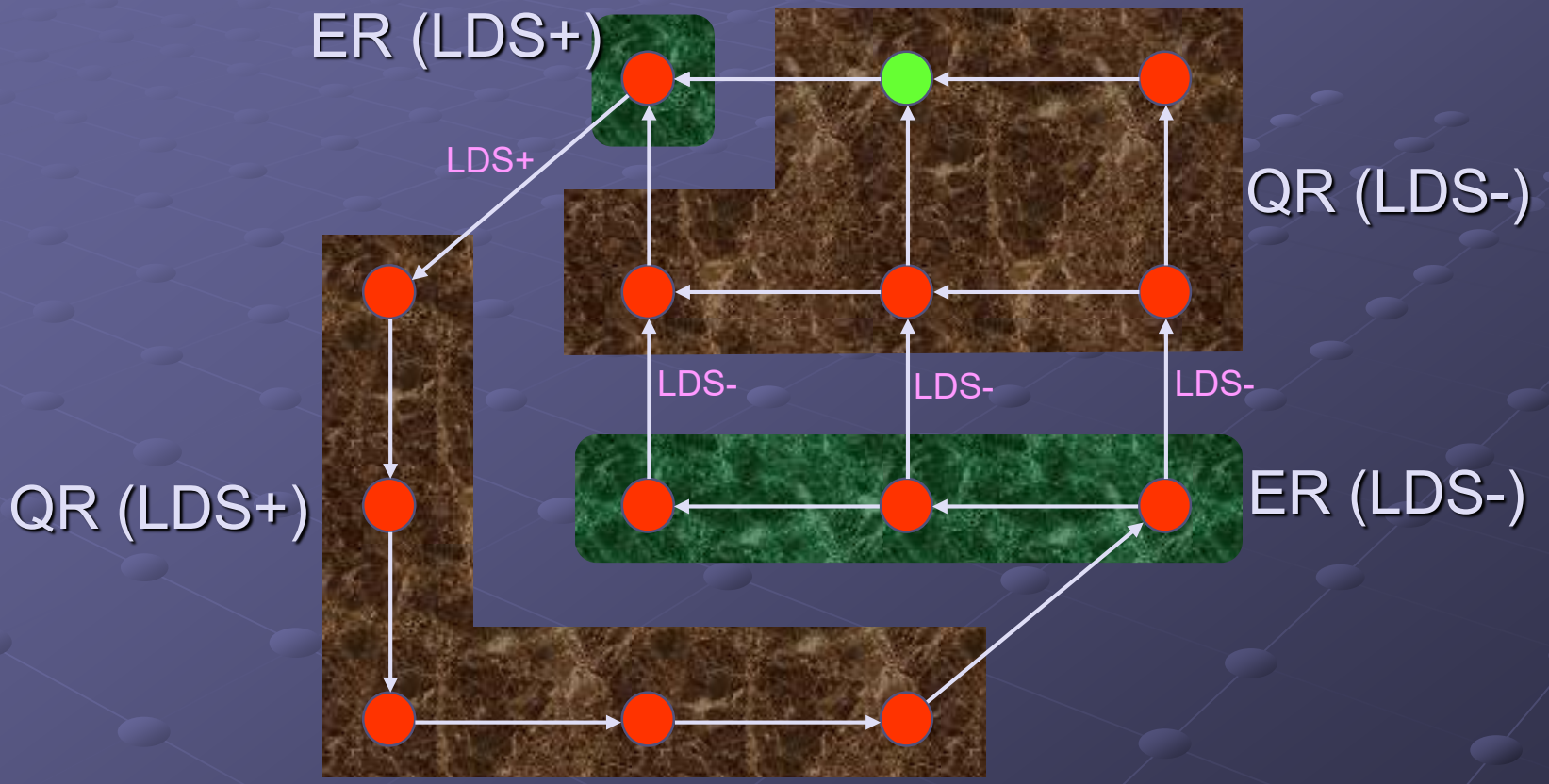
Binary encoding of signals



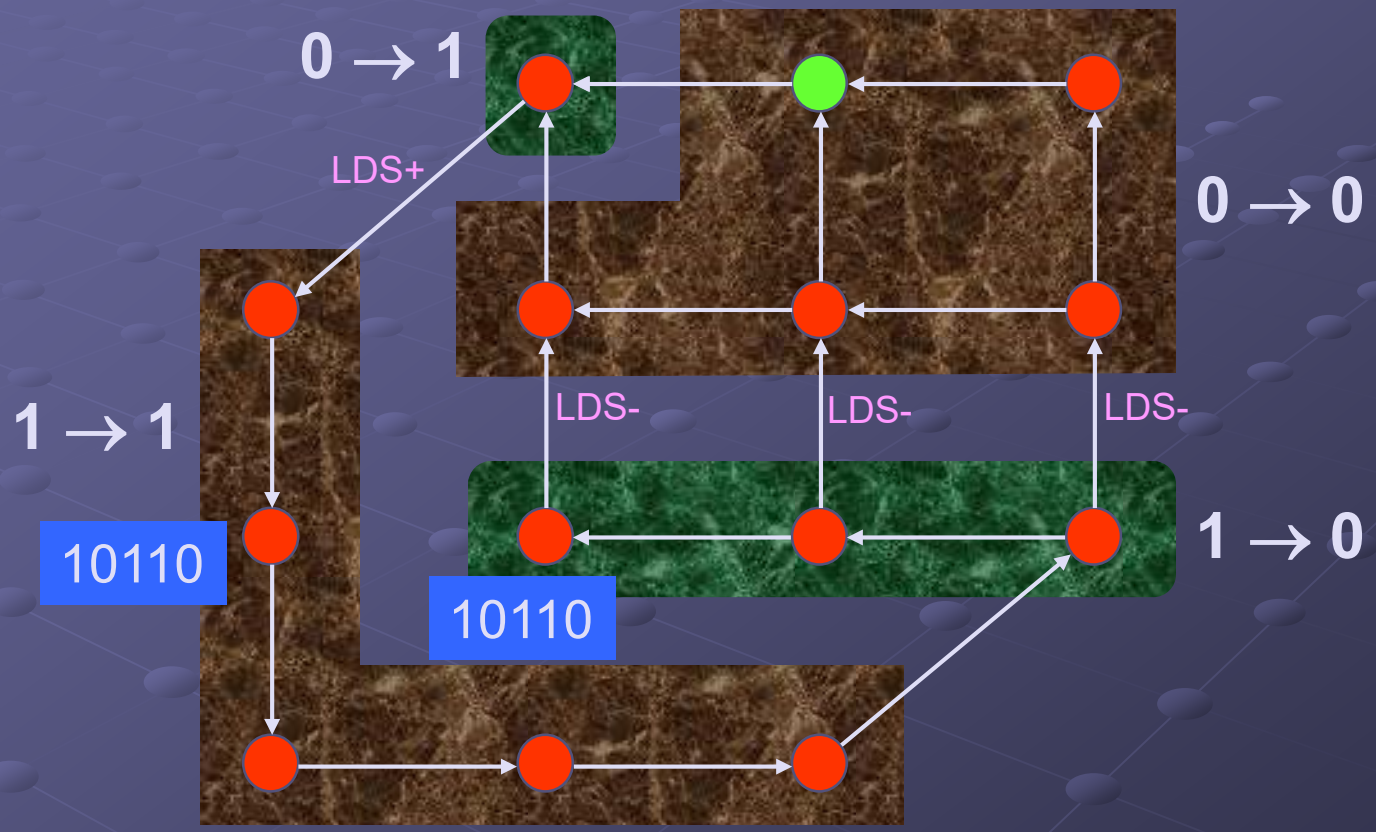
Binary encoding of signals



Excitation / Quiescent Regions

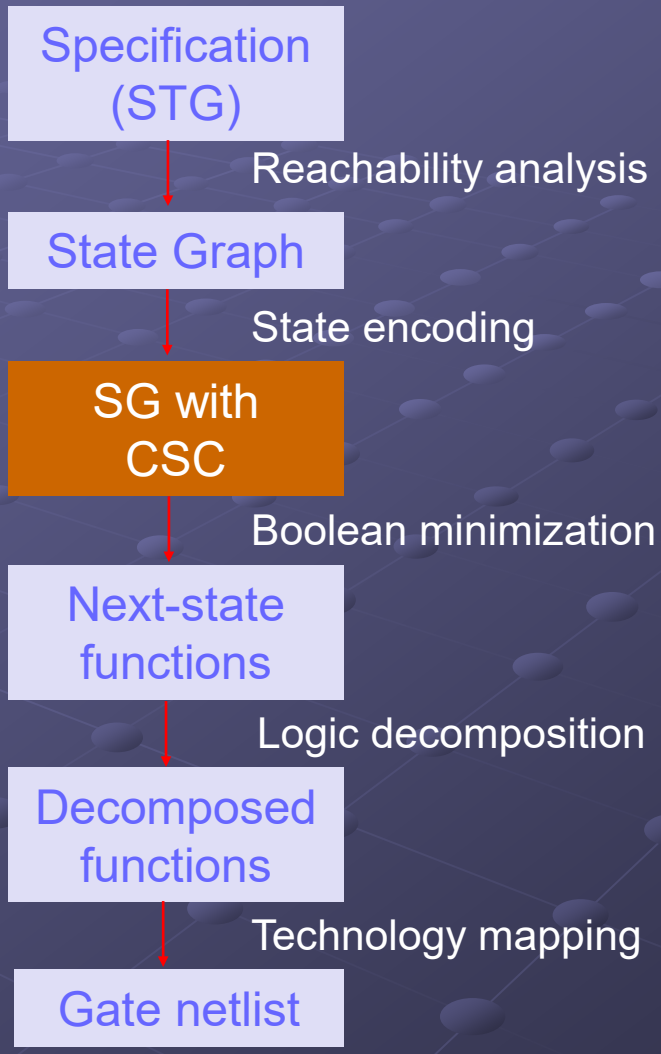


Next-state function

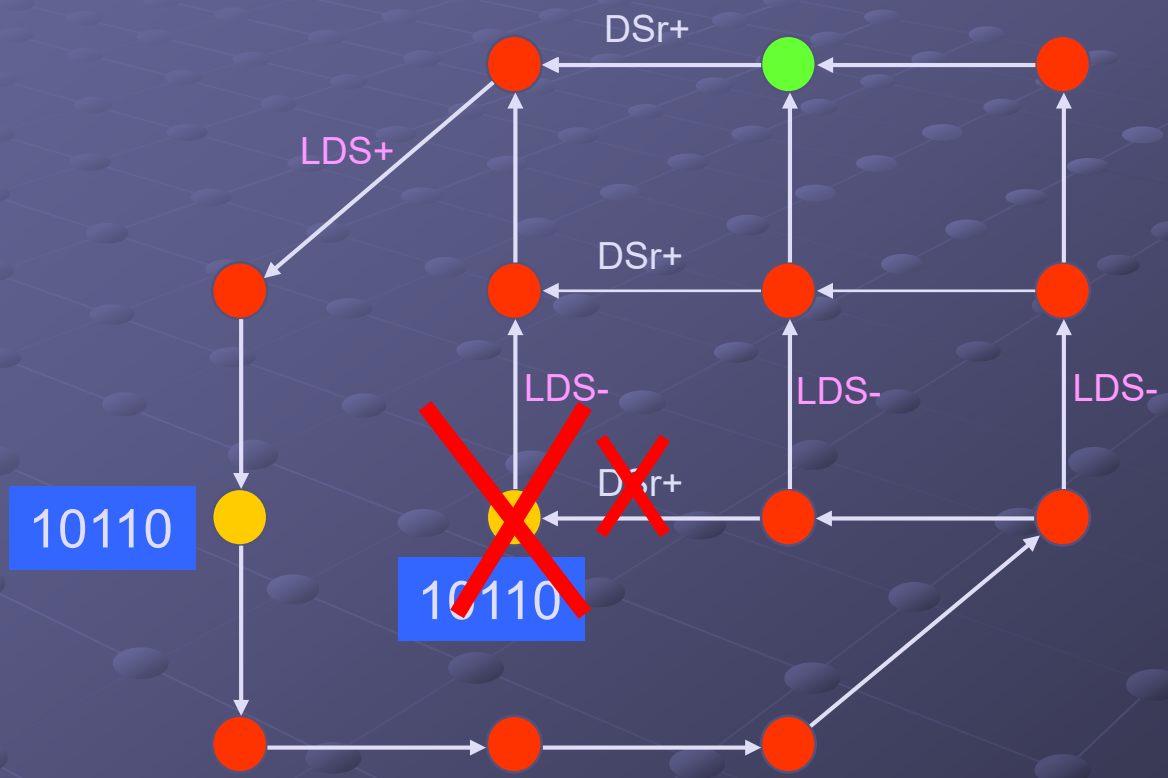


Karnaugh map for LDS

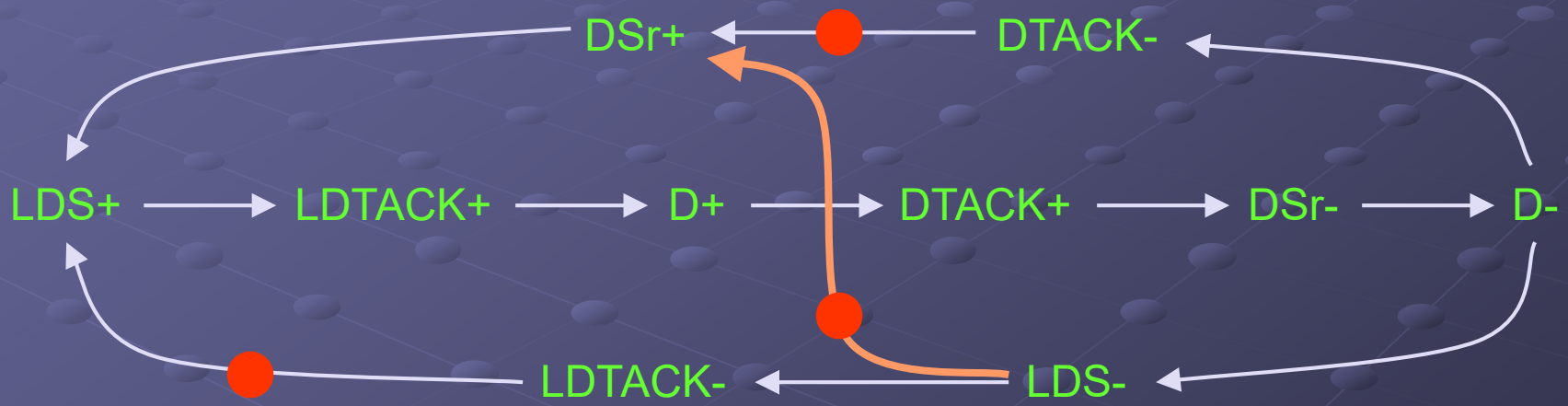
Design flow


Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

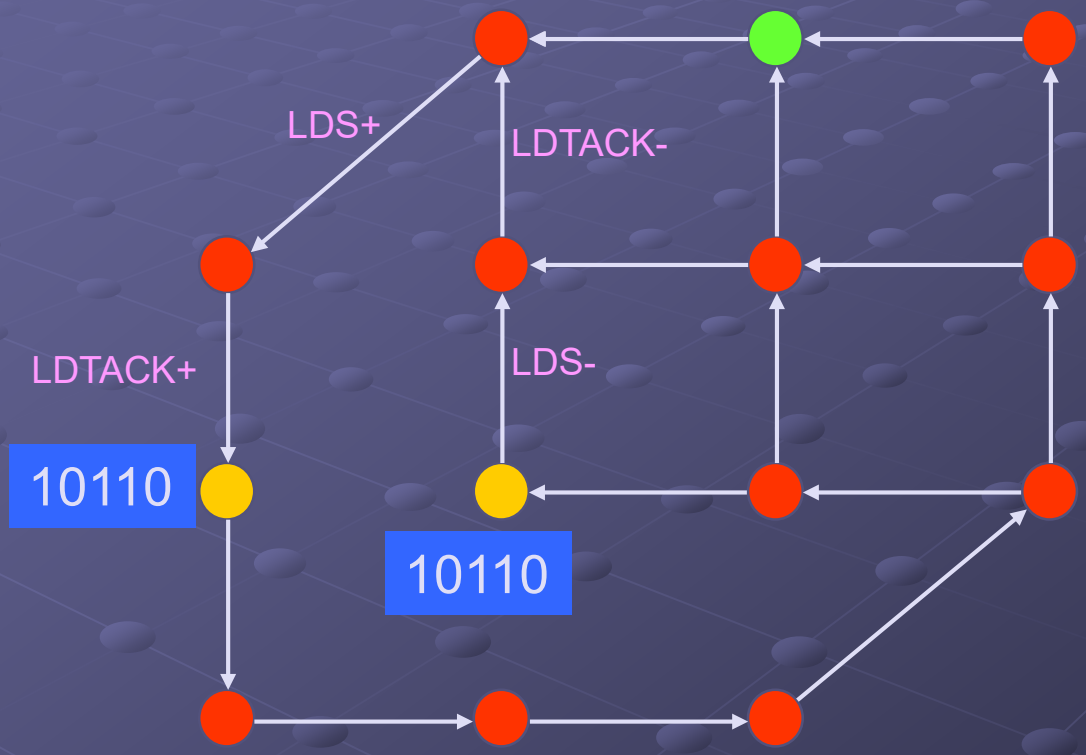
Concurrency reduction



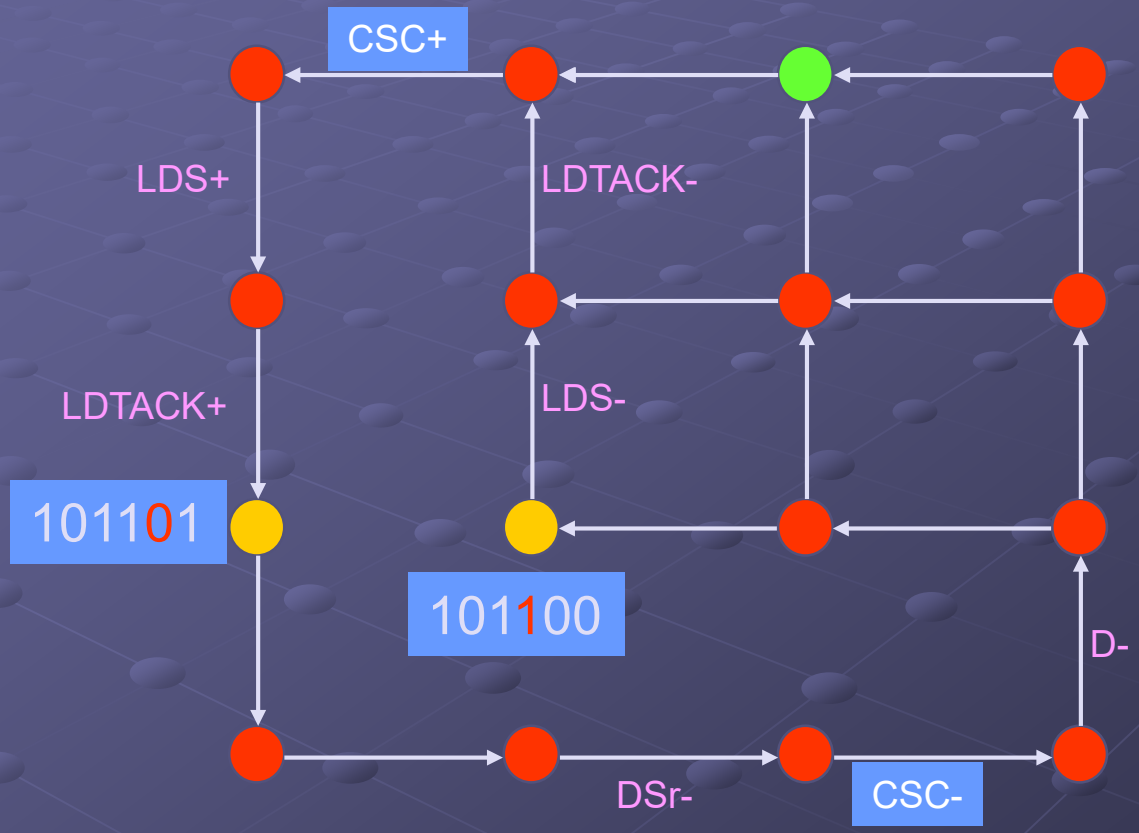
Concurrency reduction



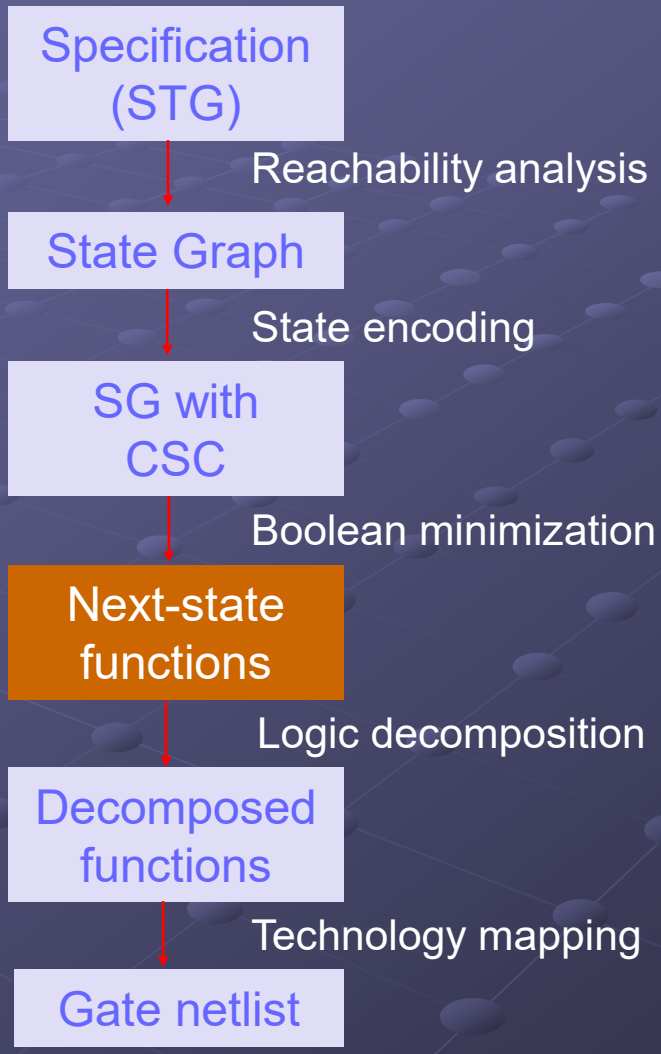
State encoding conflicts



Signal Insertion



Design flow


Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

Complex-gate implementation

$$LDS = D + \mathrm{csc}$$

$$DTACK = D$$

$$D = LDTACK \cdot \mathrm{csc}$$

$$\mathrm{csc} = DSr \cdot (\mathrm{csc} + \overline{LDTACK})$$

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)

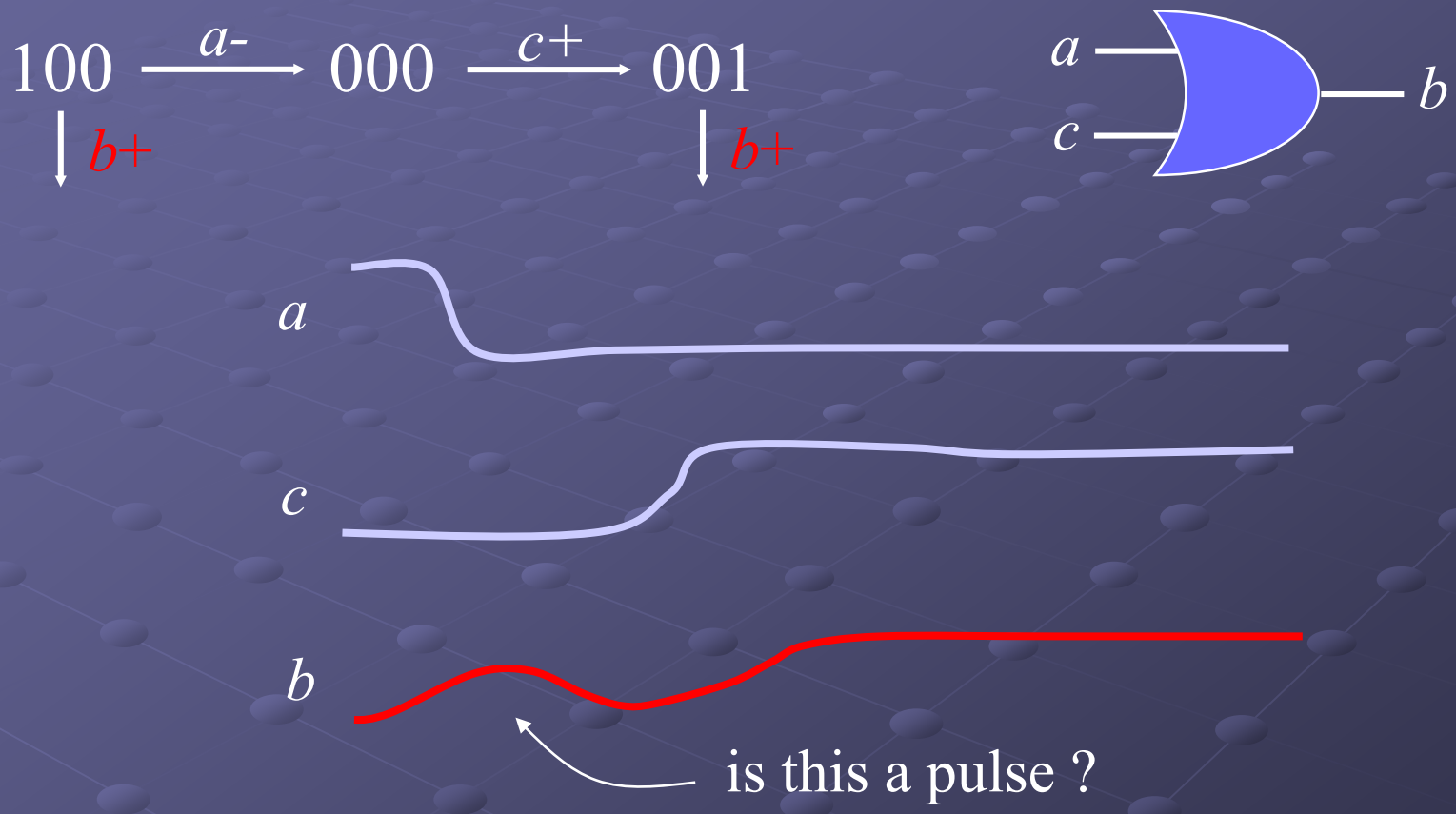
Implementability conditions

- Consistency + CSC + persistency

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

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

Persistency

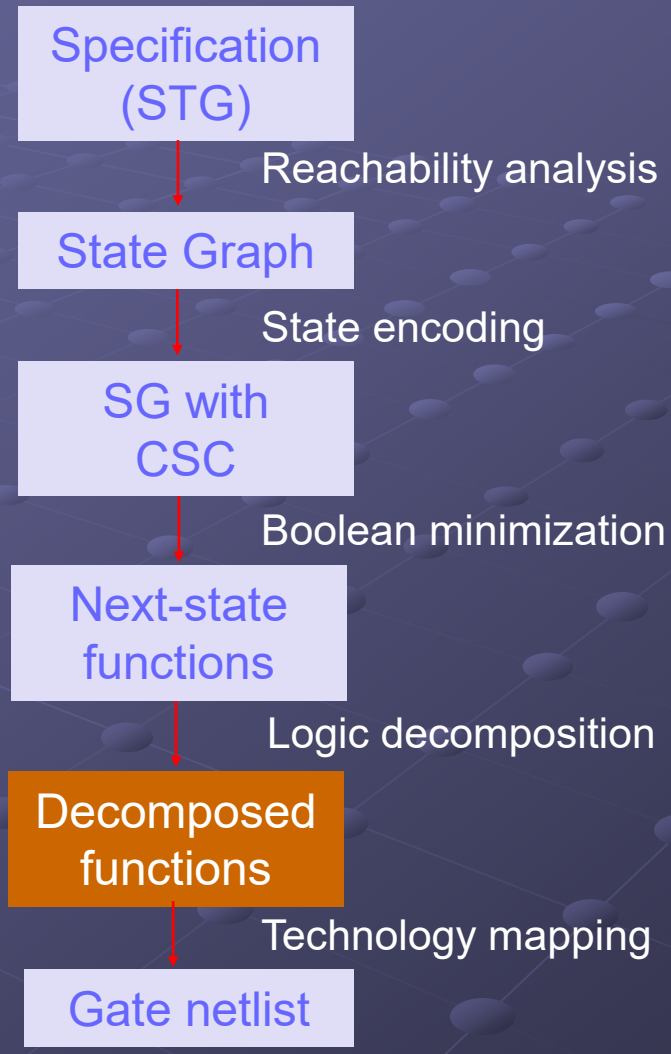


Speed independence ⇒ glitch-free output behavior under any delay

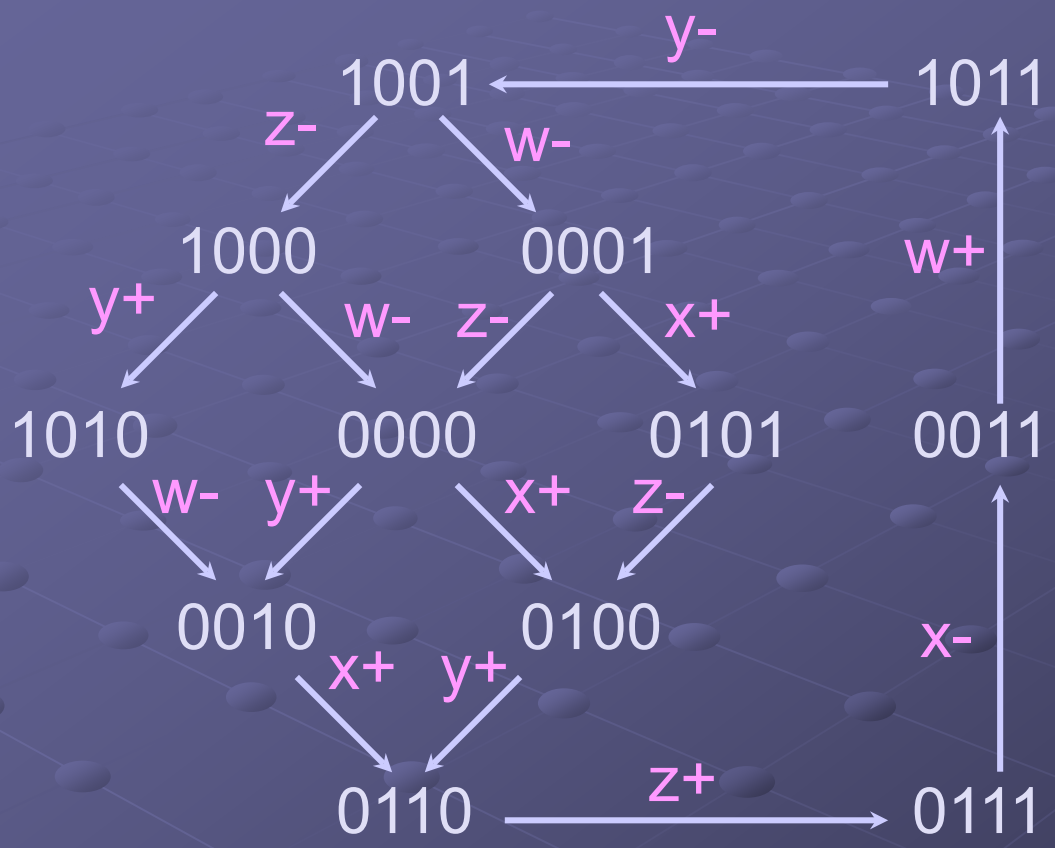
Speed-independent implementations

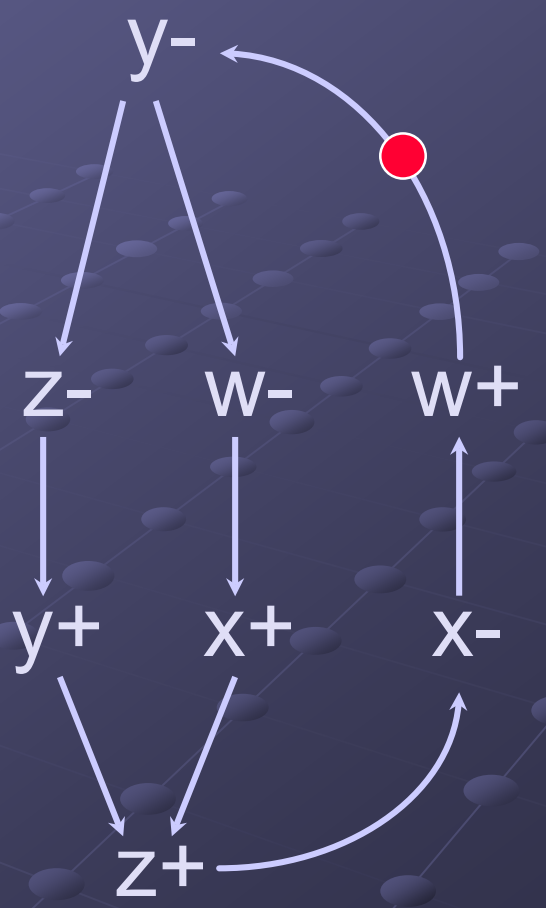
- Implementability conditions
  - Consistency
  - Complete state coding
  - Persistency
- Circuit architectures
  - Complex (hazard-free) gates
  - C elements with monotonic covers
  - ...

Design flow

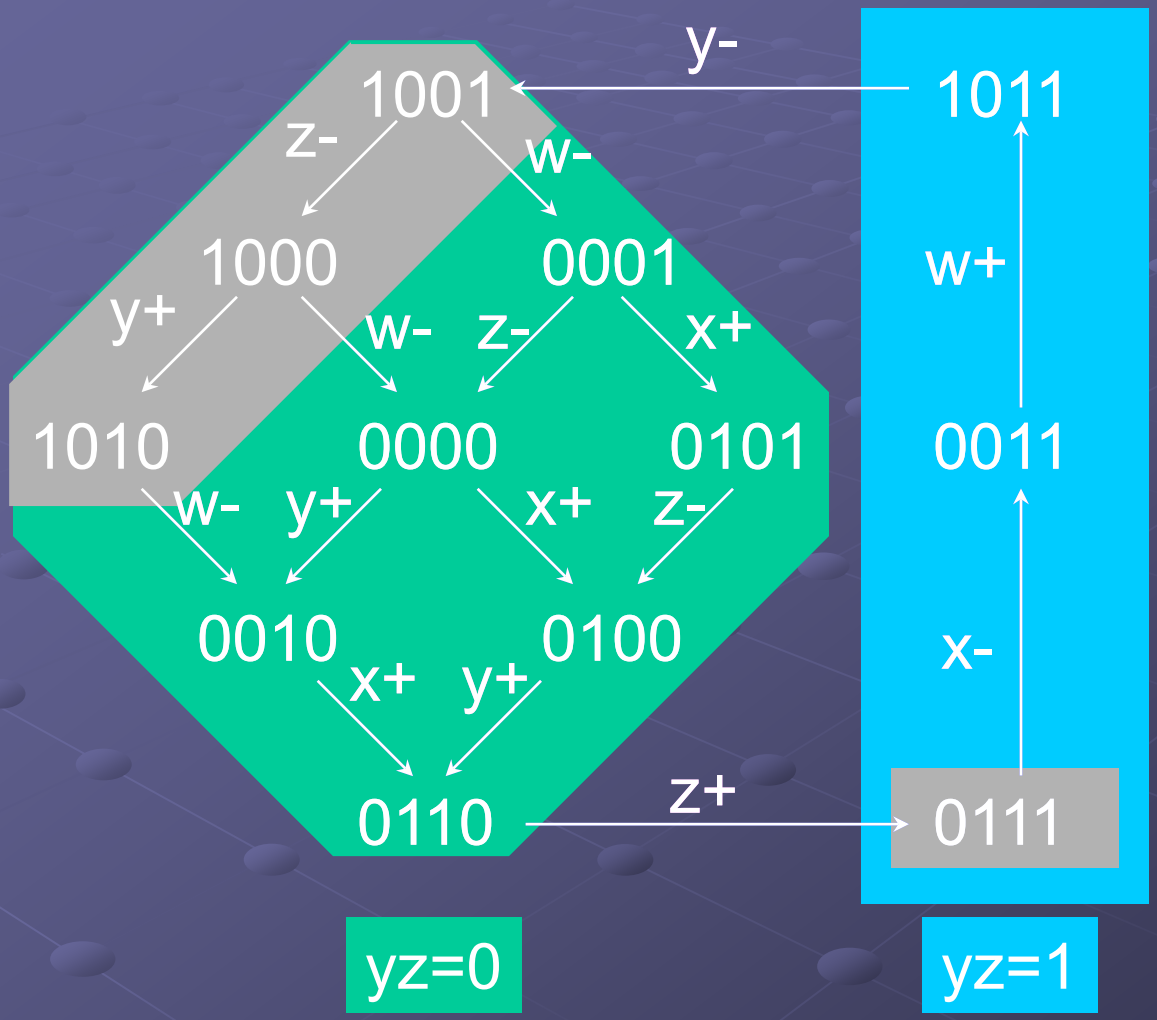

Specification (STG)
Reachability analysis
State Graph
State encoding
SG with CSC
Boolean minimization
Next-state functions
Logic decomposition
Decomposed functions
Technology mapping
Gate netlist

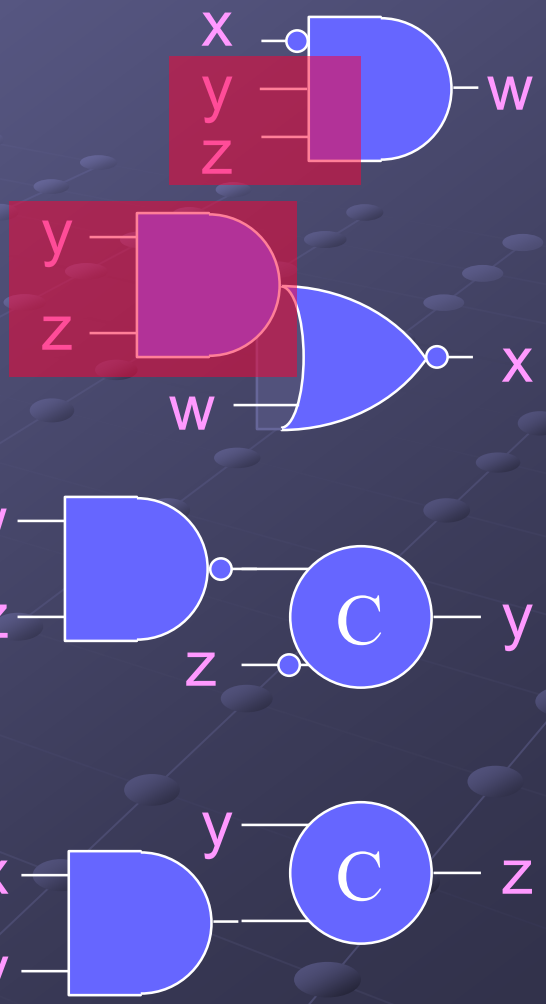
Logic decomposition: example



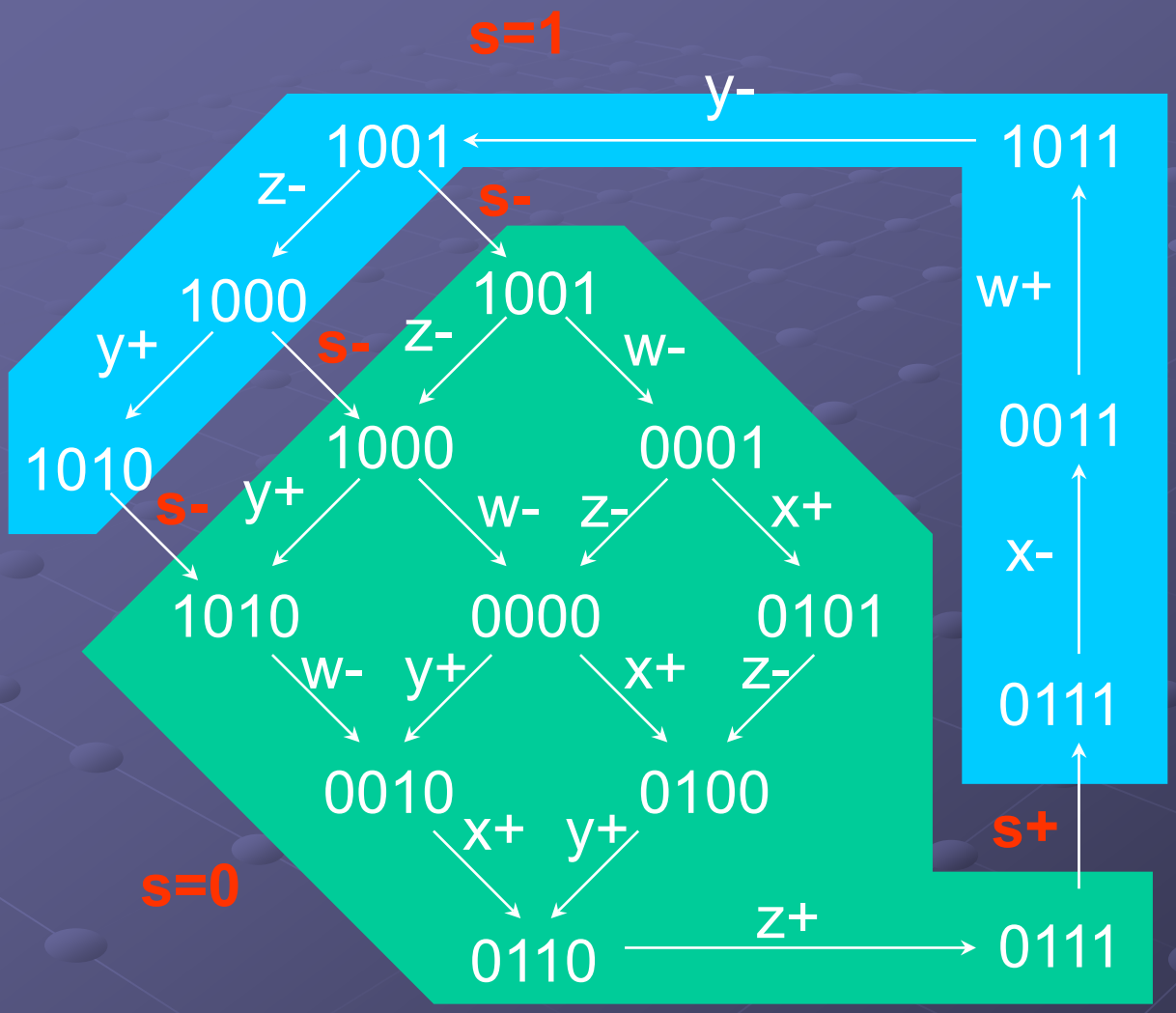


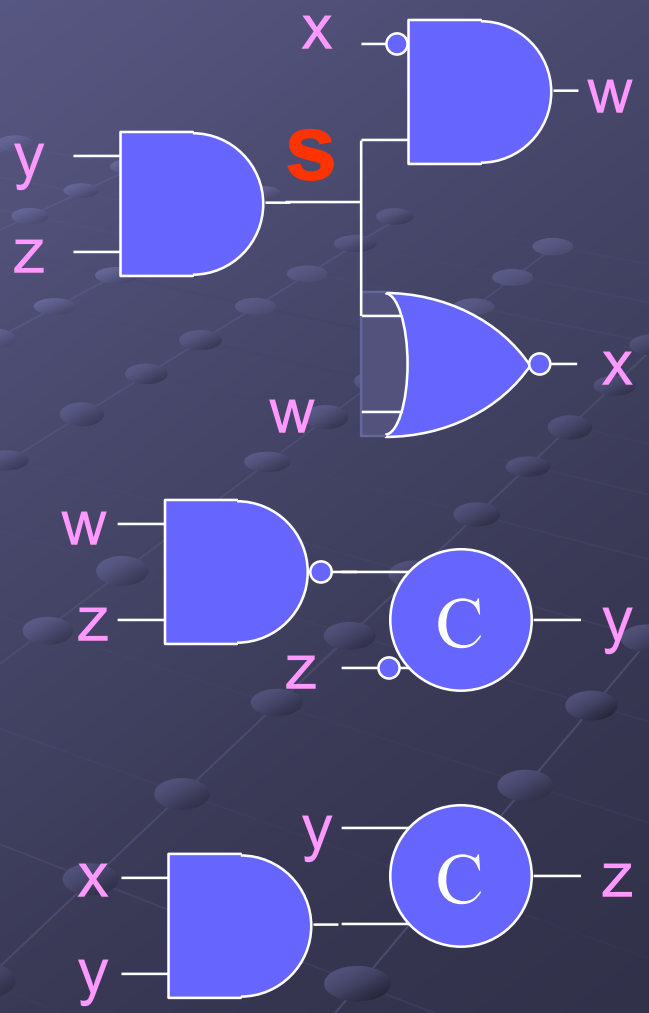
Logic decomposition: example



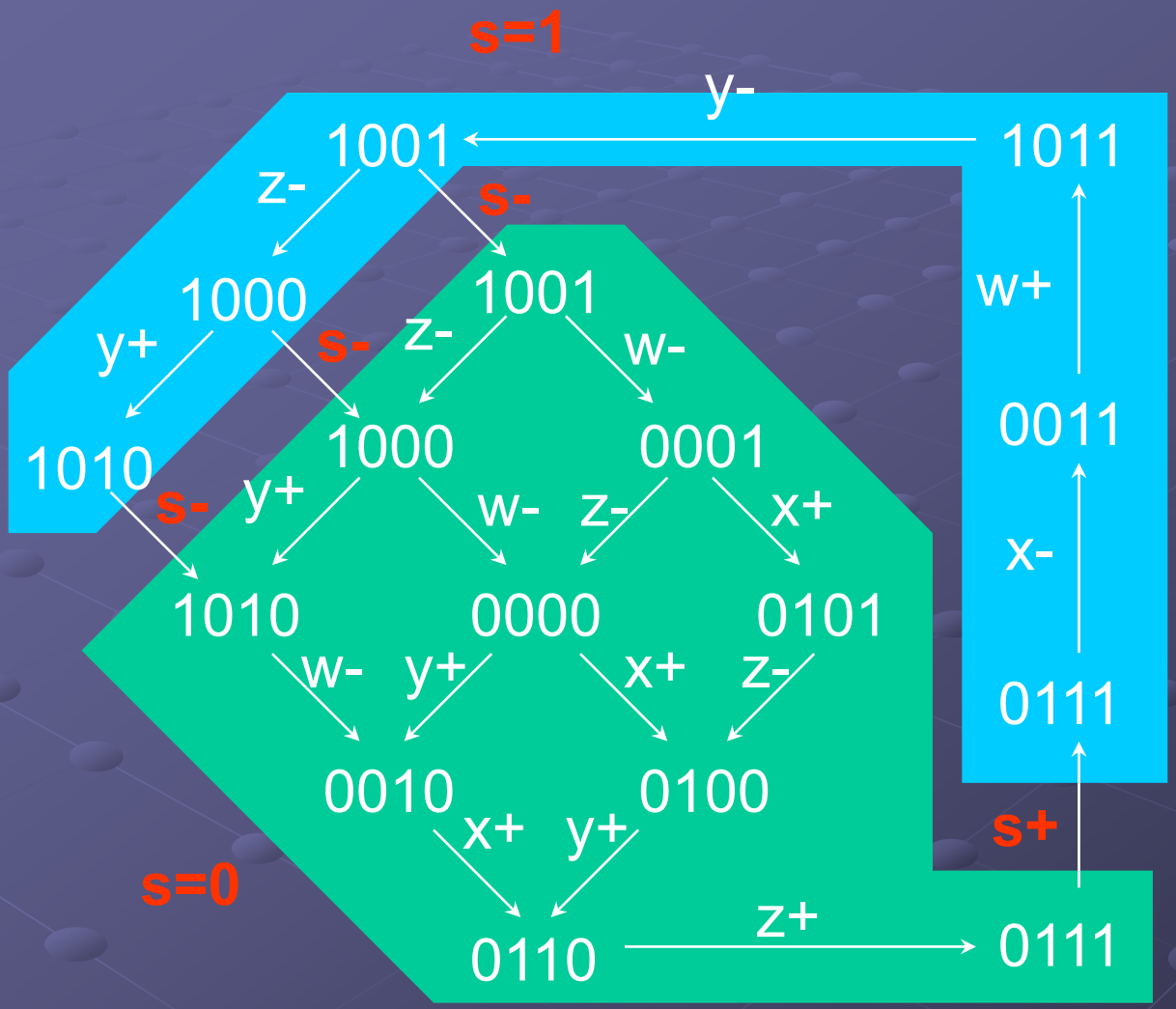


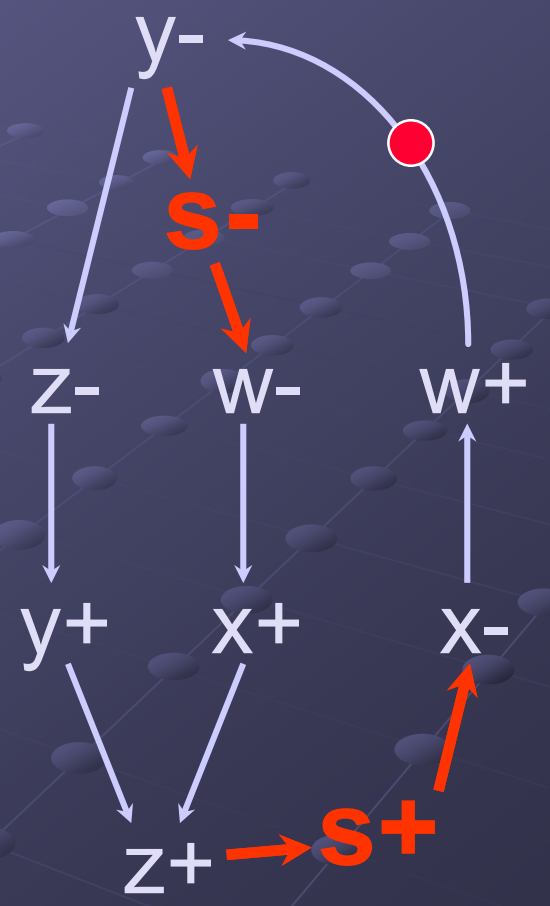
Logic decomposition: example



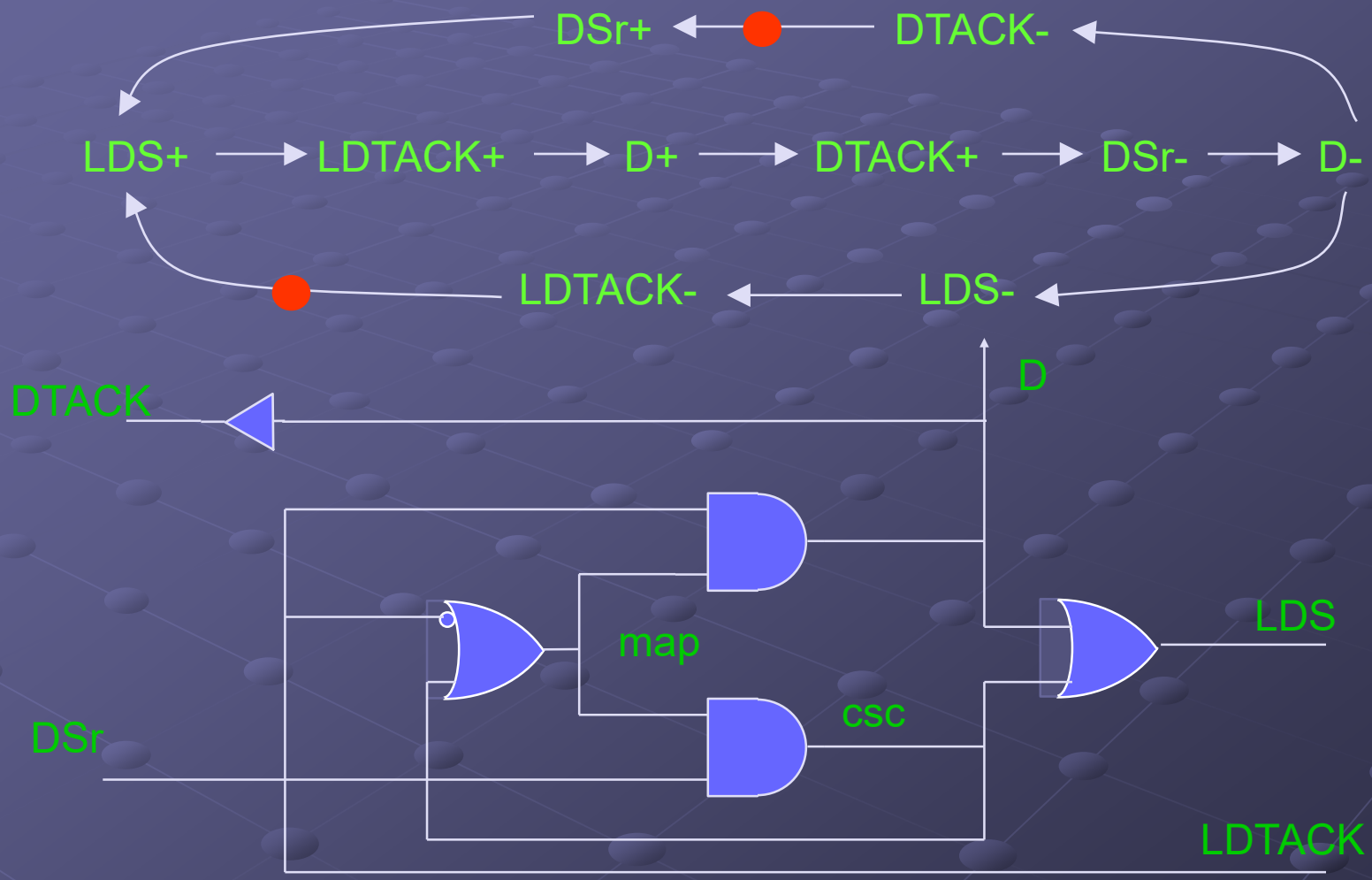


Logic decomposition: example

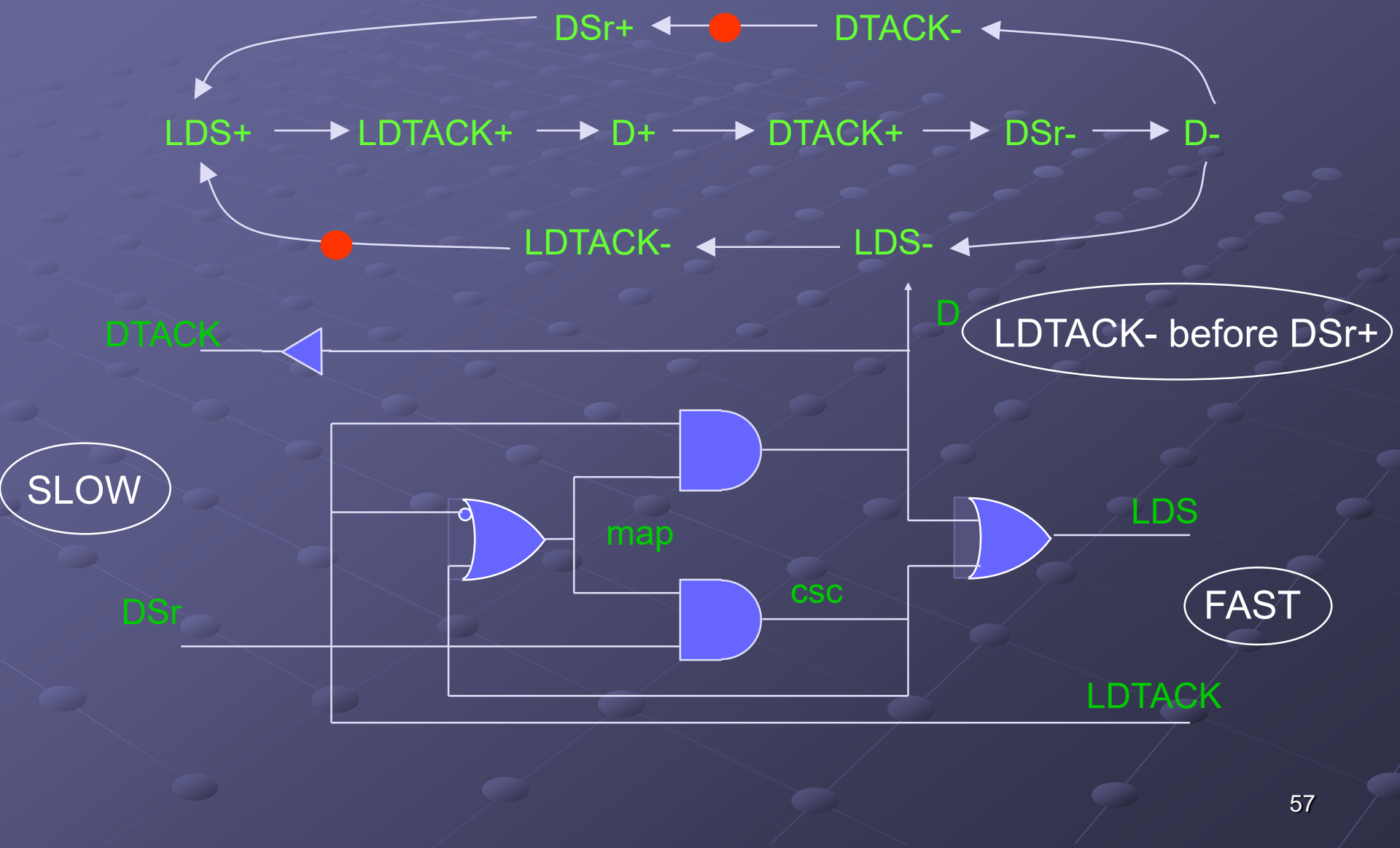




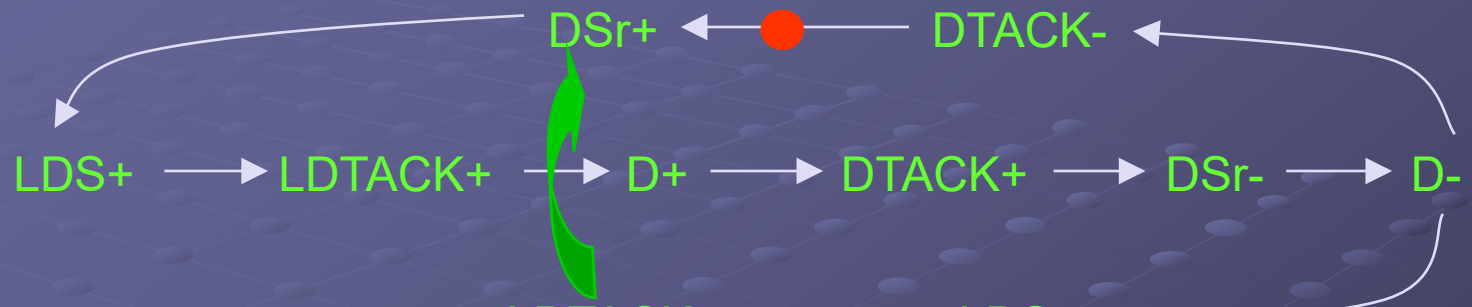
Speed-independent Netlist

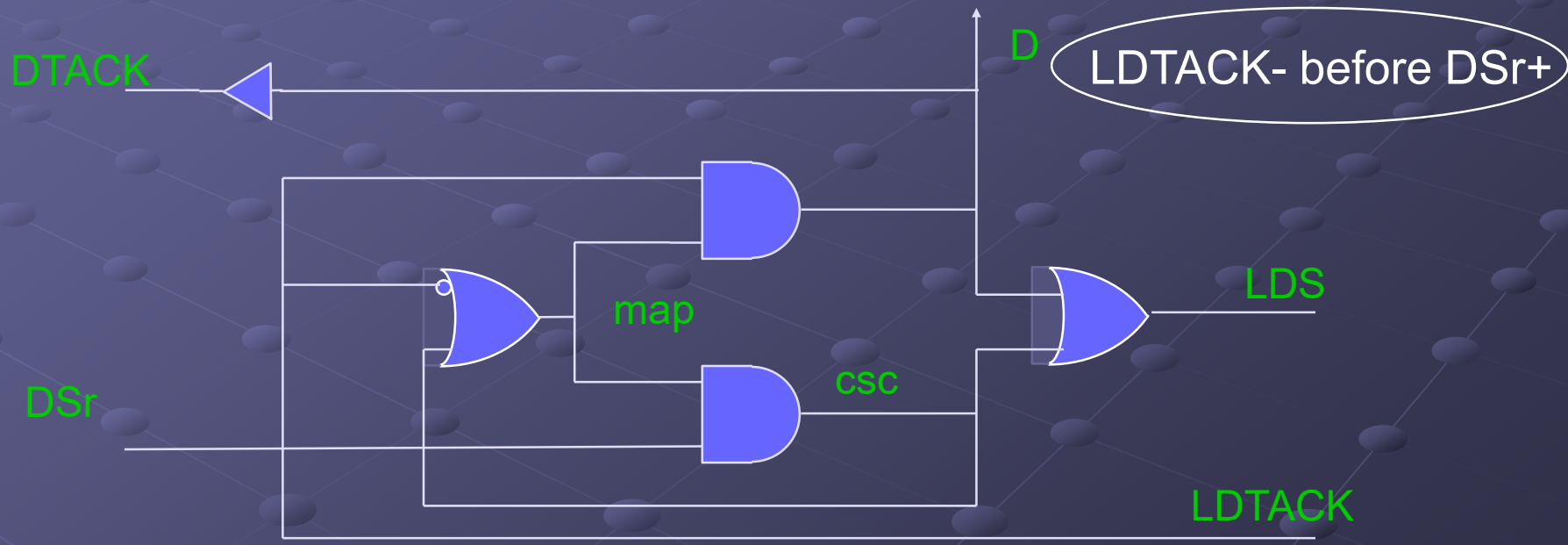


Adding timing assumptions

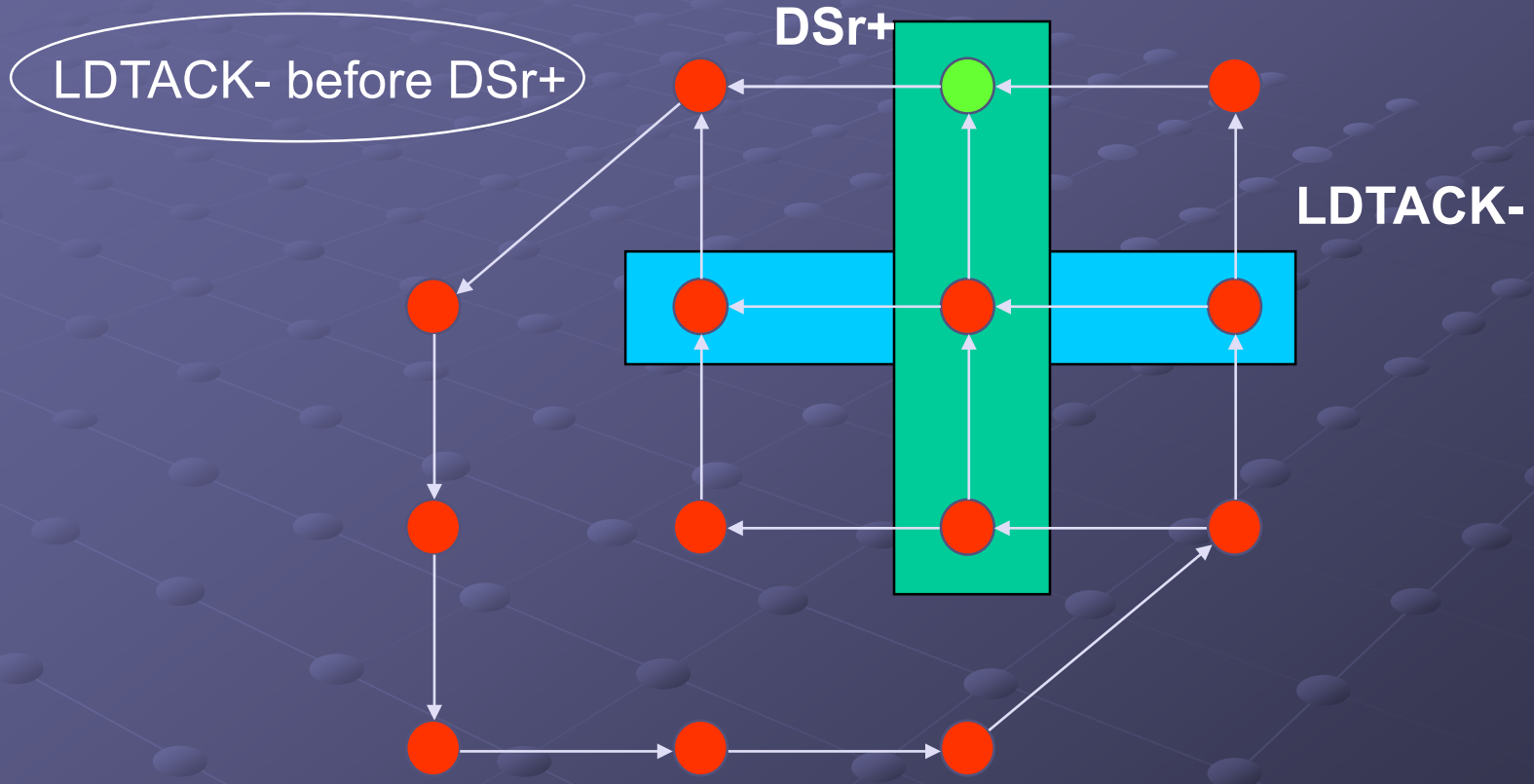


Adding timing assumptions

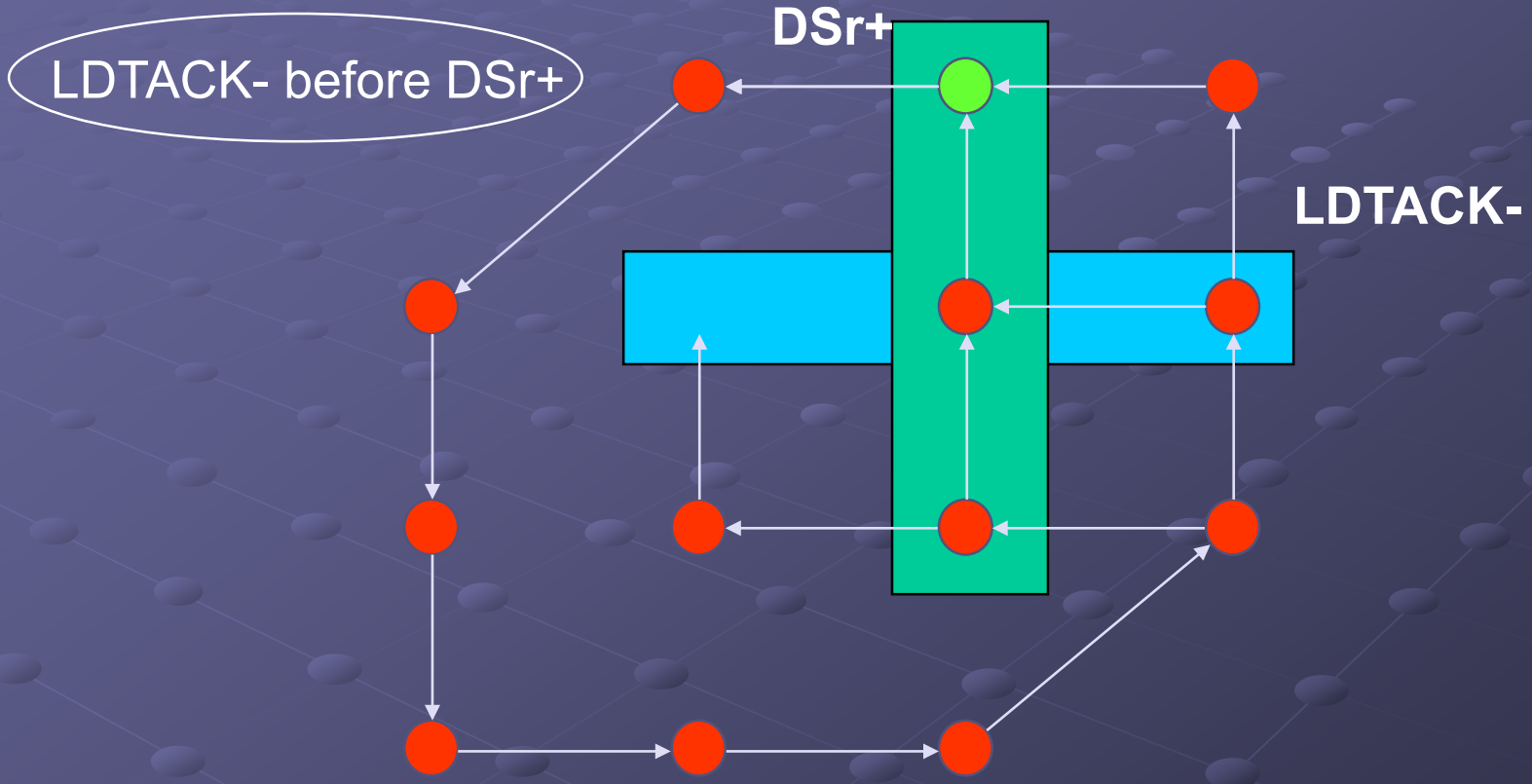




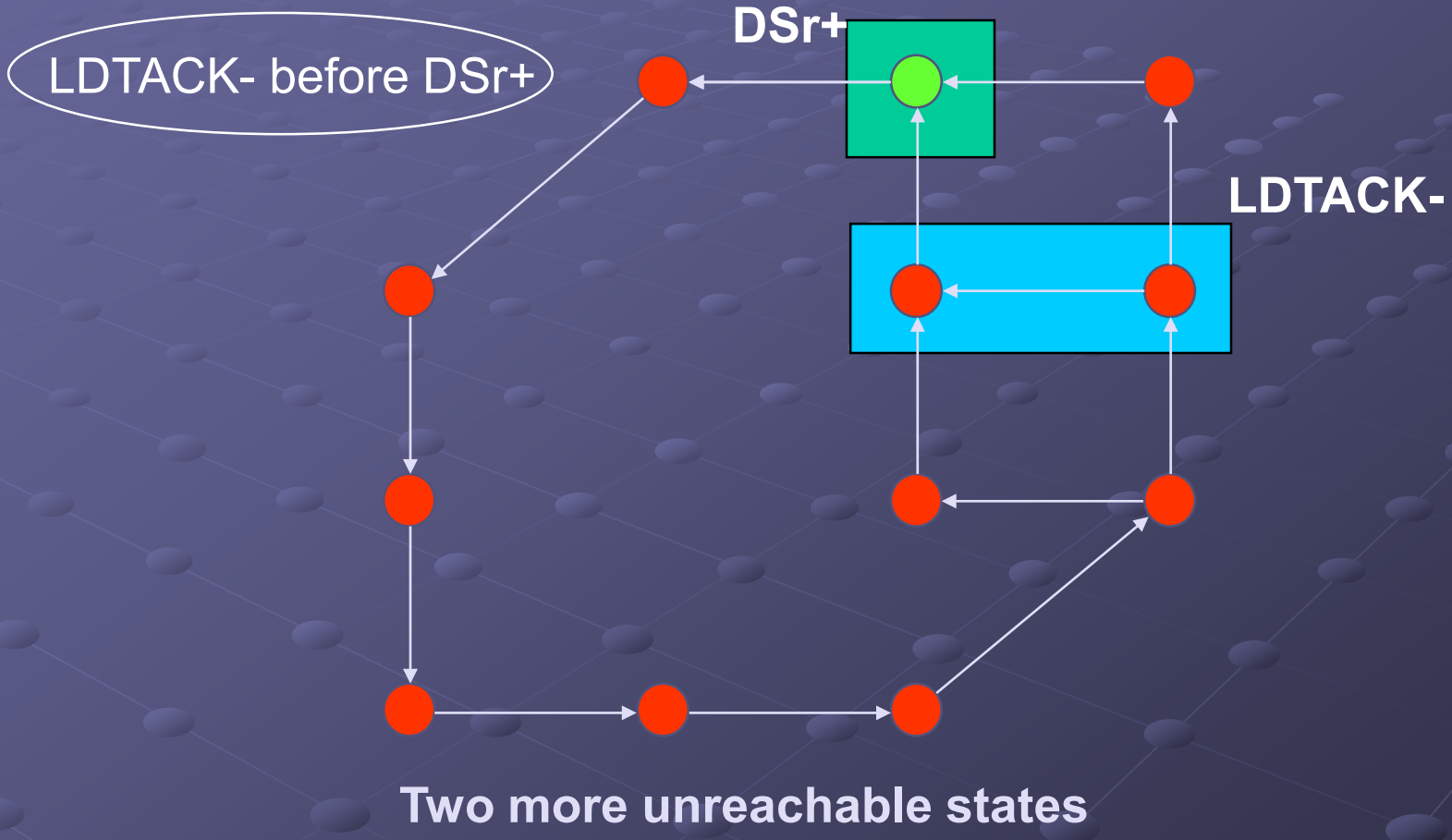
State space domain



State space domain



State space domain



Boolean domain

LDS = 0

D LDTACK \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	0

LDS = 1

D LDTACK \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	0/1?

Boolean domain

LDS = 0

D LDTACK \ DTACK DSr	00	01	11	10
00	0	0	-	1
01	-	-	-	-
11	-	-	-	-
10	0	0	-	-

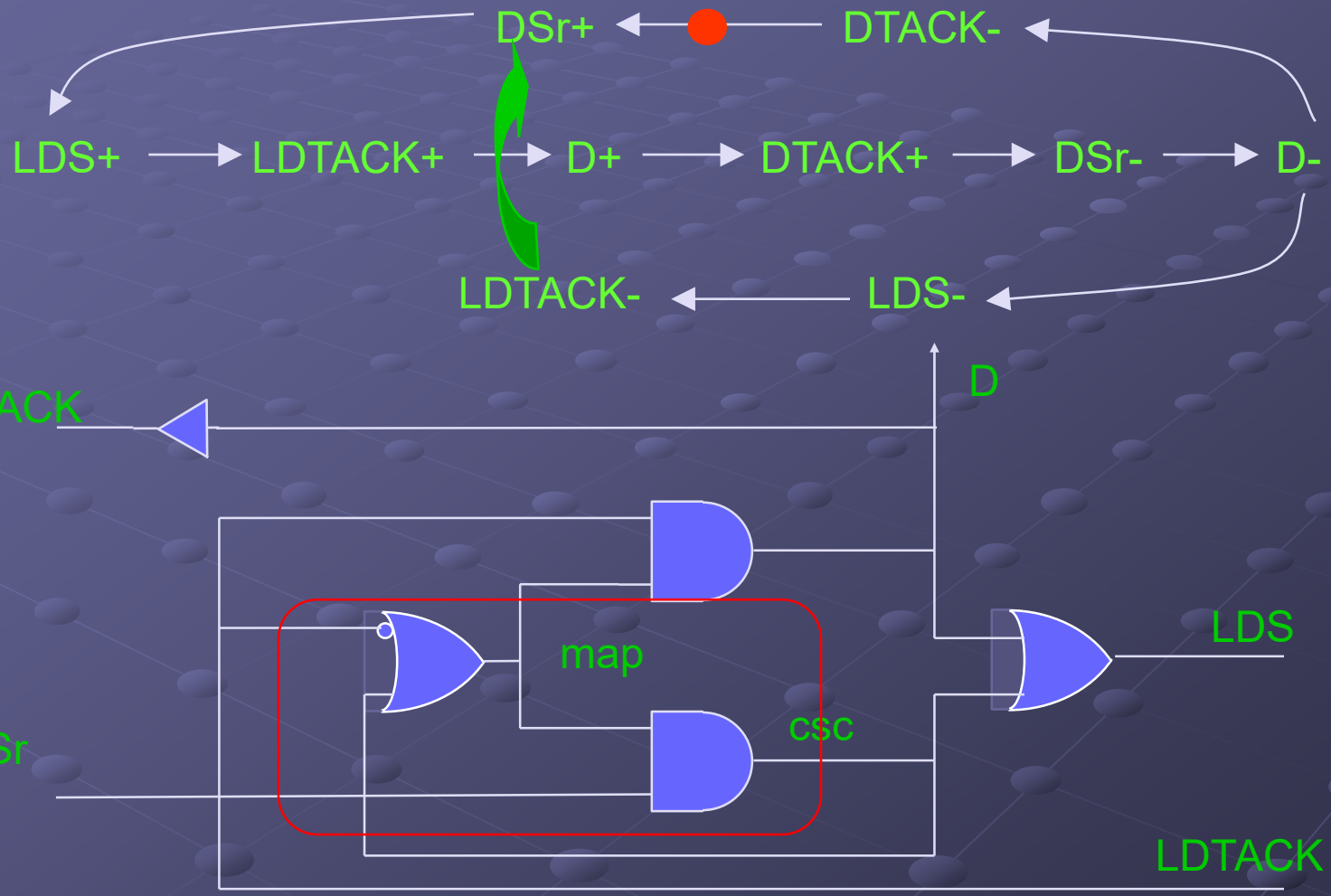
One more DC vector for *all* signals

LDS = 1

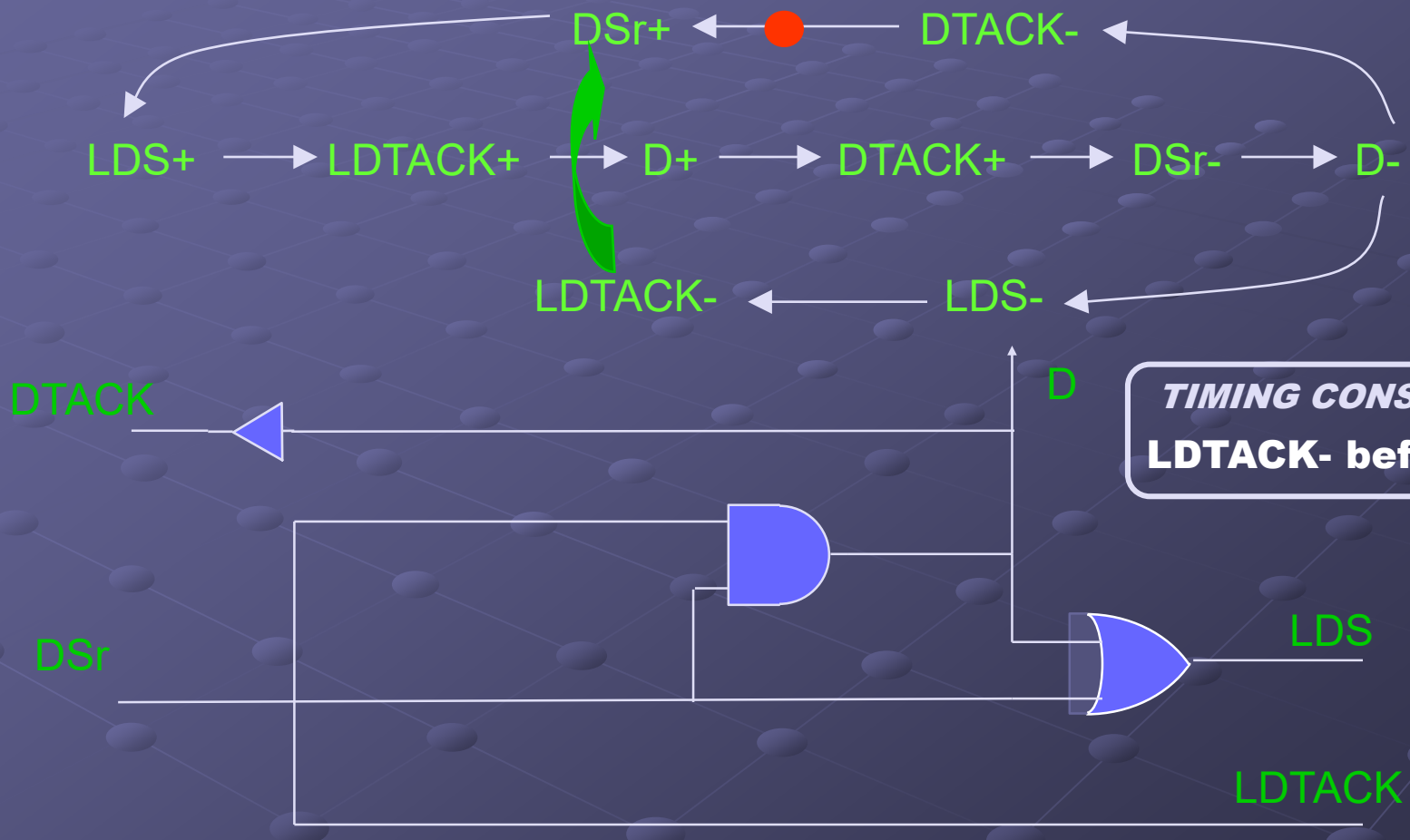
D LDTACK \ DTACK DSr	00	01	11	10
00	-	-	-	1
01	-	-	-	-
11	-	1	1	1
10	0	0	-	1

One state conflict is removed

Netlist with one constraint



Netlist with one constraint



Conclusions

- STGs (which are Interpreted Petri nets) have a high expressiveness power at a low level of granularity (similar to FSMs for synchronous systems)
- Synthesis from STGs is fully automated
- Synthesis tools often suffer from the state explosion problem (symbolic techniques and Petri net unfoldings are used)
- The theory of logic synthesis from STGs can be found in:

J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, *Logic Synthesis of Asynchronous Controllers and Interfaces*, Springer Verlag, 2002.

Application of Petri nets and Asynchronous Design to Analog-Mixed Signal Systems

Analog behaviour

- Typically in continuous time and level
- Can be represented by EM forces/fields, electric charge, magnetic flux, voltage, current
- In various applications can represent mechanical, chemical, thermal etc. forms of information or energy
- Dynamics can be represented in time and frequency domains
- Typically separates signal flows between “data” and power supply but does not always need to
- Applications: sensing, measurement, signal processing, control, power
- Interfaces between digital and analog: analog elements inside digital components, ADC and DACs, digital control of analog

Analog elements

- Amplifiers:
  - Operational amplifiers
  - Low noise amplifiers
  - Sense amplifiers
- Current mirrors
- Bandgap circuits
- Delay elements
- Oscillators
- Transmission lines

Async (digital) behaviour

- Triggered by events (e.g. level-crossing)
- Modelled by
  - cause-effect relations
  - token flow
  - handshakes
  - data-flow
- Power-driven timing
- Applications: interfacing, control, pipeline

Do we need to divide the world into analog and digital?

Async helps us to remove or at least lower the A-to-D and D-to-A walls

Areas for analog for async

- Logic cell design:
  - GasP
- Power supply:
  - IR degradation,
  - Drooping,
  - Subthreshold
- Interconnect:
  - Transmission line models,
  - Cross-talk and noise
- Delay analysis and design:
  - stray, inertial, pure delays

Applying analog knowledge has always helped in:
- Speeding circuits up,
- Reducing power,
- Improving reliability and robustness

What about the other direction: Async for Analog?

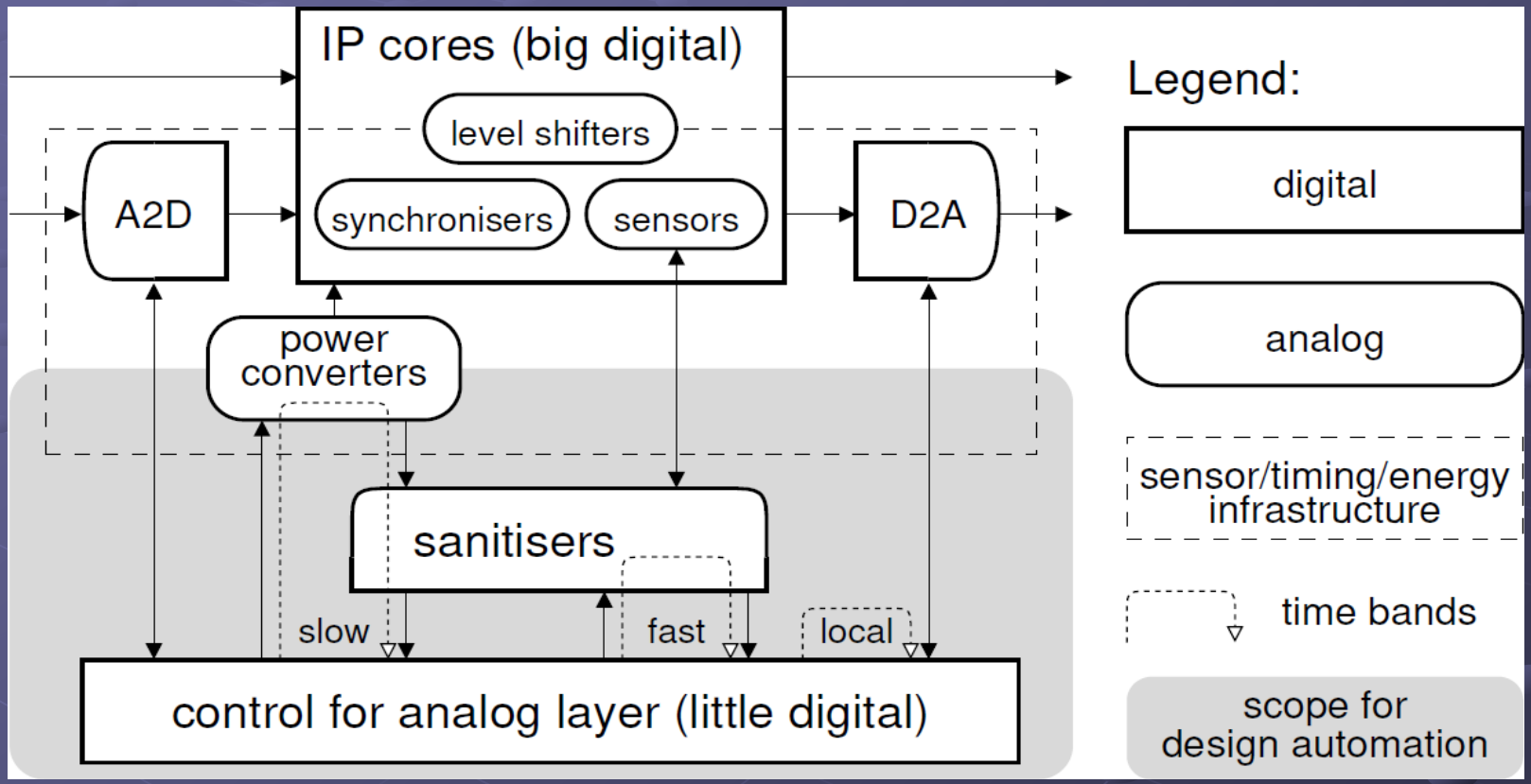
Motivation for Async for Analog

- Analog and Mixed Signal (AMS) design becomes more complex:
  - More functionality
  - Move to deep submicron after all!
- According to Andrew Talbot from Intel (2016) *"transistors are very fast switches, netlists are huge, parasitics are phenomenally difficult to estimate, passives don't follow Moore's law, reliability is a brand new landscape"*

Motivation: power electronics context

- Efficient implementation of power converters is paramount
  - Extending battery life for mobile gadgets
  - Reducing energy bill for PCs and data centres (5% and 3% of global electricity production, respectively)
- Need for responsive and reliable control circuits – *little digital*
  - Millions of control decisions per second for years
  - A wrong decision may permanently damage the circuit (not as fuzzy as genetic circuits!)

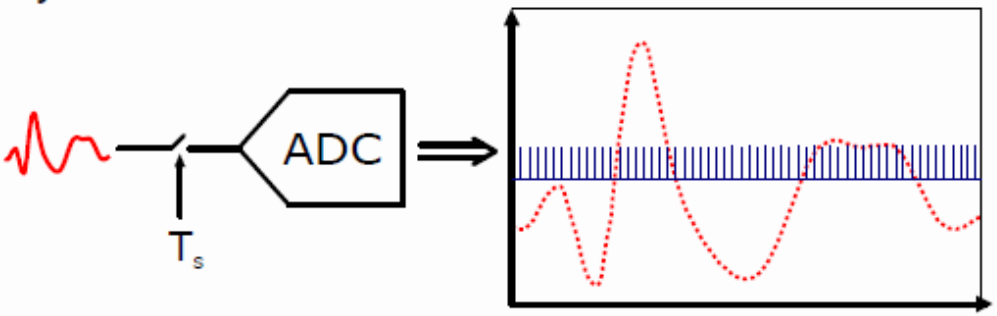
Emergence of “little digital” electronics



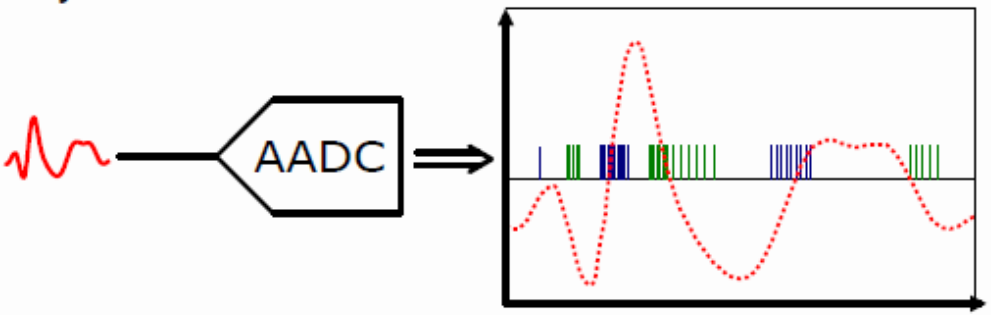
- Analog and digital electronics are becoming more intertwined
- Analog domain becomes complex and needs digital control

Async ADC

- Synchronous

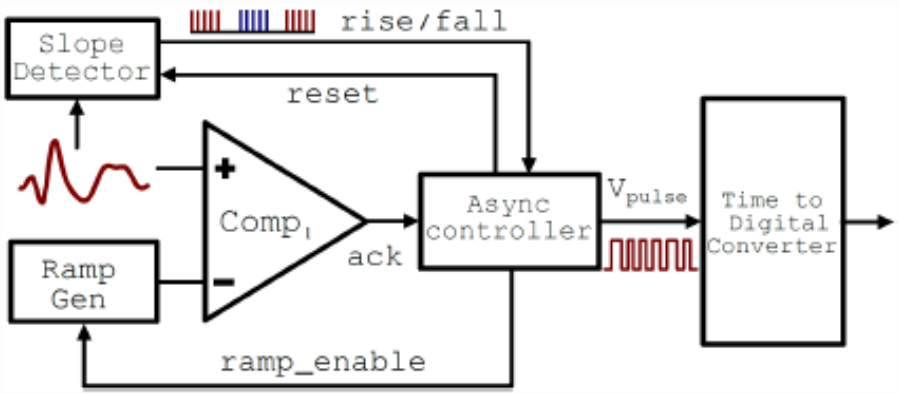


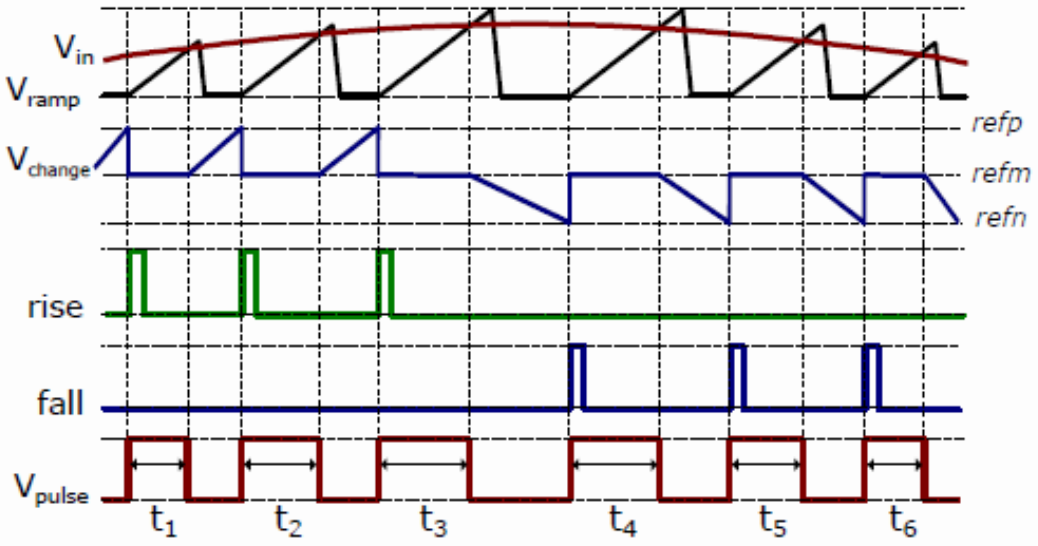
- Asynchronous



A. Ogweno, P. Degenaar, V. Khomenko and A. Yakovlev: *"A fixed window level crossing ADC with activity dependent power dissipation"*, accepted for NEWCAS-2016.

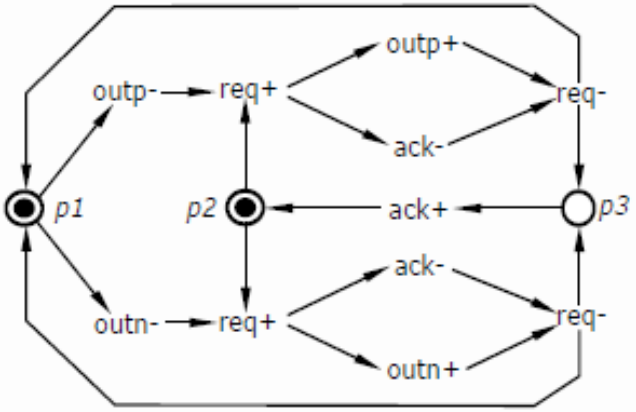
ADC design


rise/fall
Slope Detector
reset
Comp1
ack
Async controller
Vpulse
Time to Digital Converter
Ramp Gen
ramp_enable

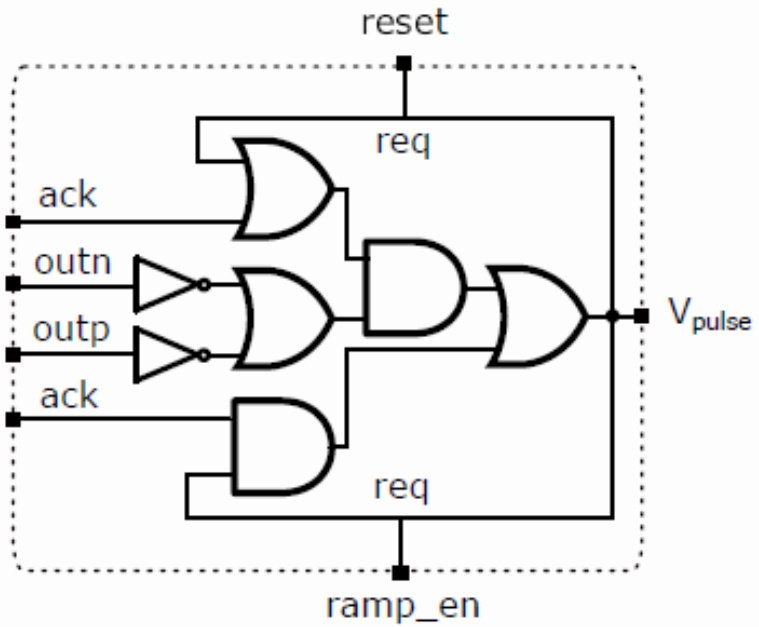

Vin
Vramp
refp
Vchange
refm
refn
rise
fall
Vpulse
t1
t2
t3
t4
t5
t6

Asynchronous controller

- STG specification

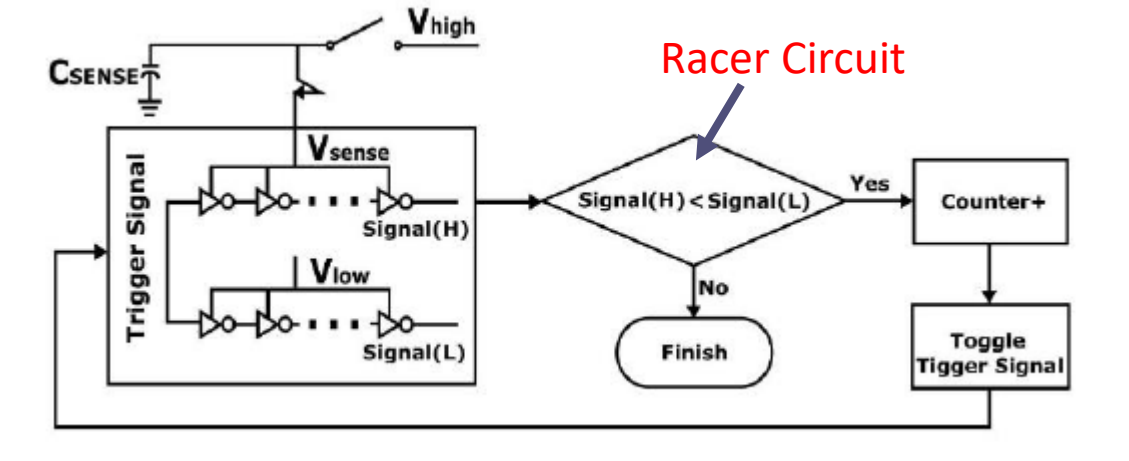


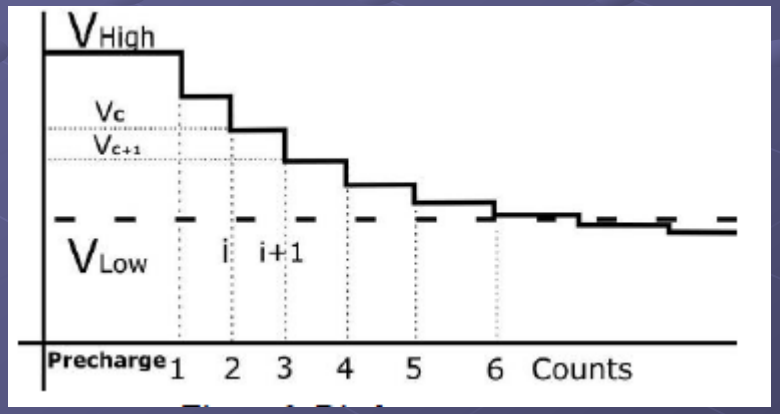
- Speed-independent implementation

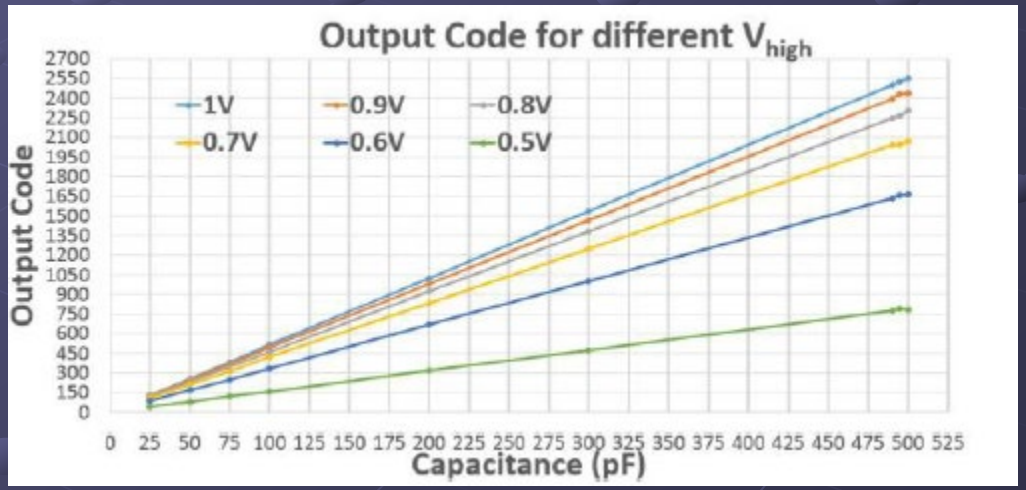


Sensors using asynchronous

Cap-to-digital Conversion (sensing)







Y. Xu et al,ICECS'16

Racer Circuit (Patented)

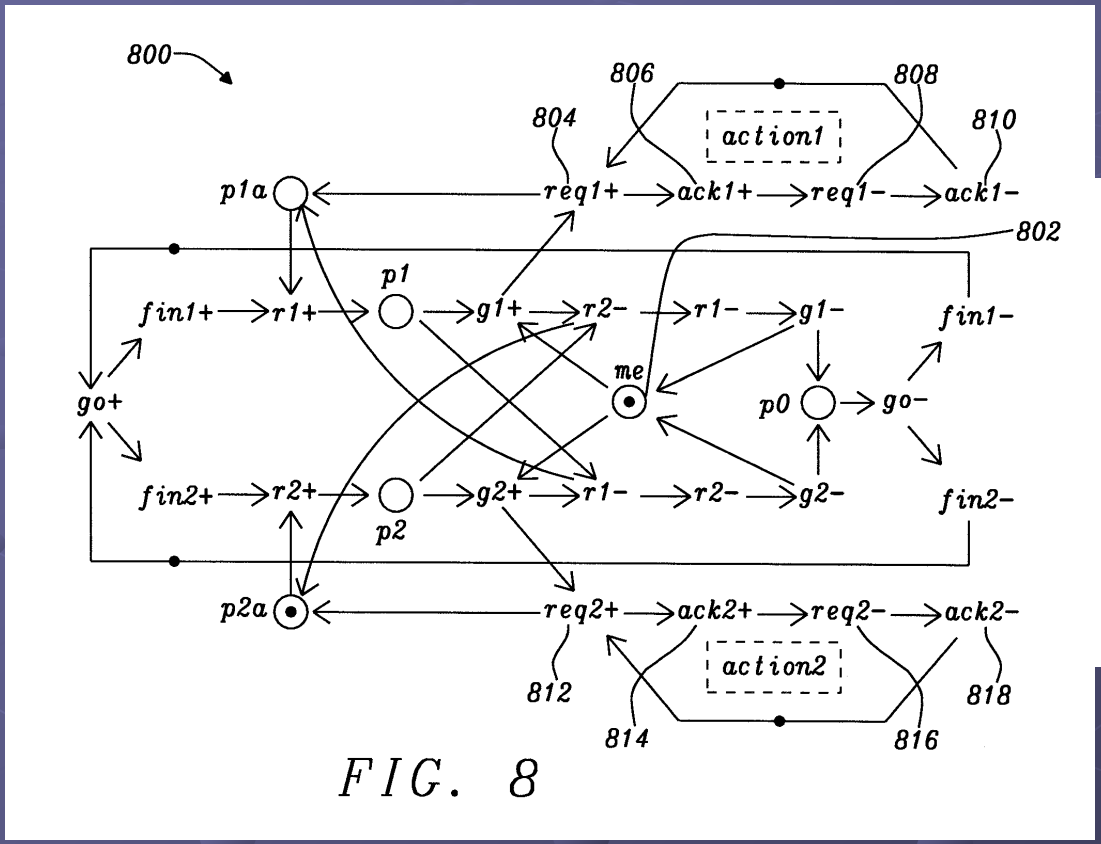


FIG. 8

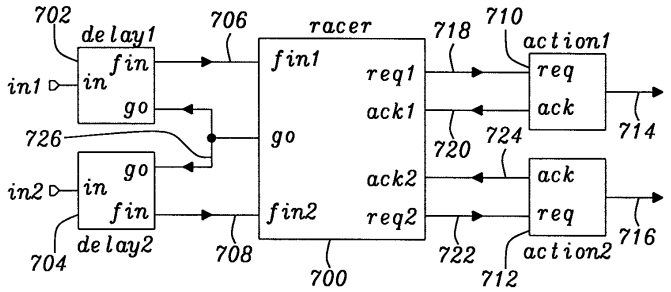


FIG. 7

D. Sokolov et al, US Patent 10,581,435

Can we go further with Async?

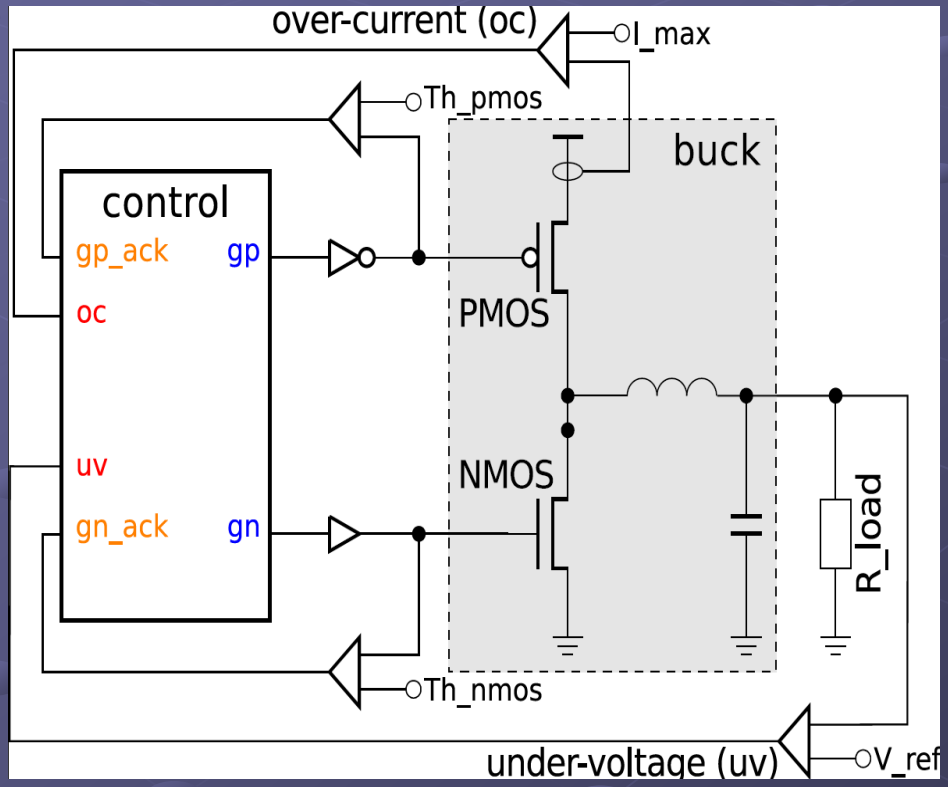
Where we can have a win-win situation!

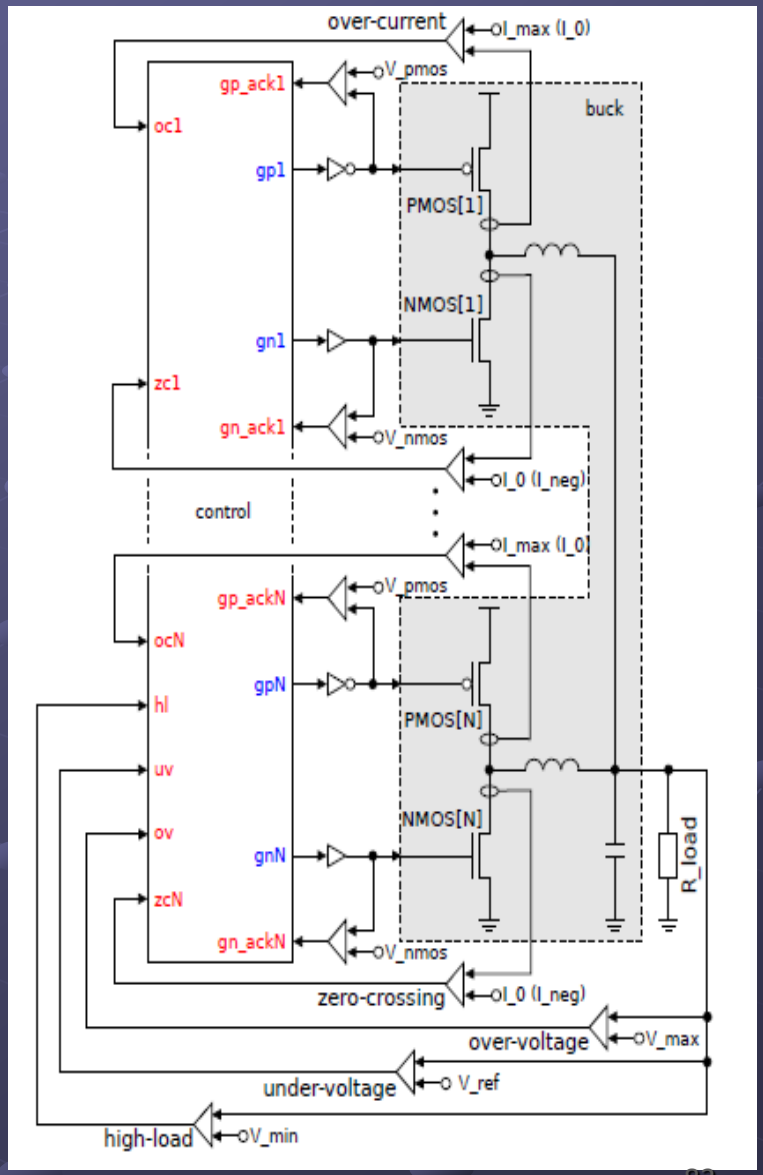
To something bigger … such as

Power electronics

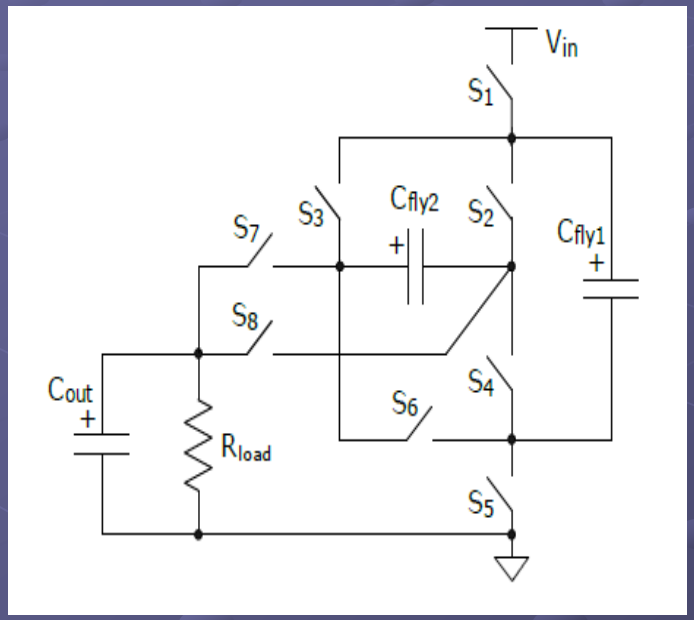
And show impact outside the ‘usual’ digital scope’ ….

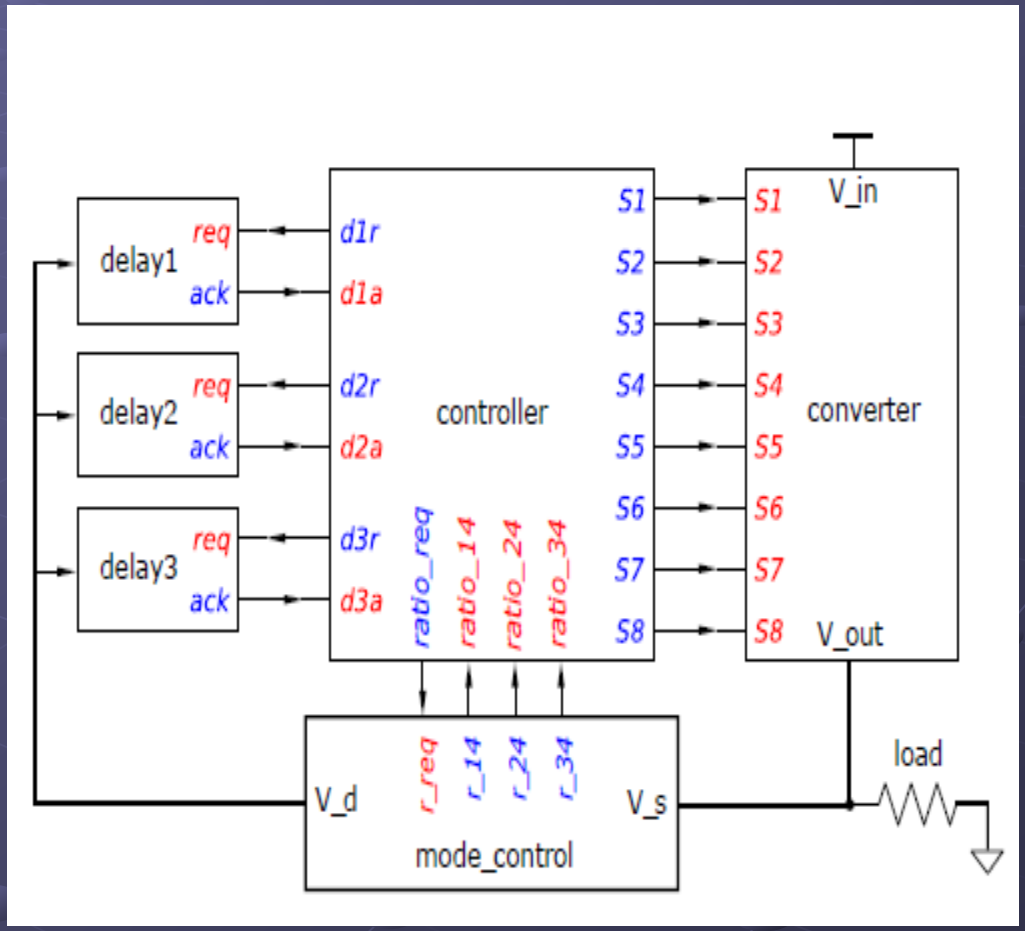
Example: Buck (DC-DC) converter control



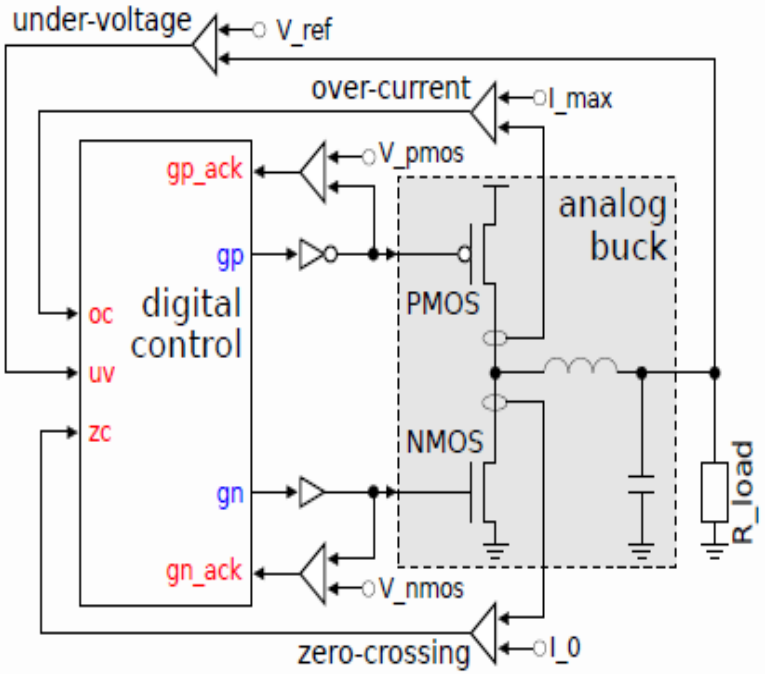


Example: Switched Capacitor (DC-DC) Converter control

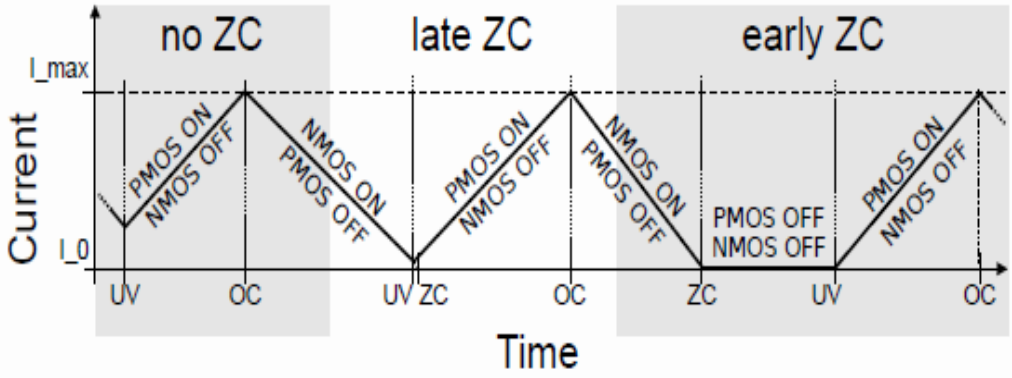




Example: Buck converter



Phase diagram specification:



Building asynchronous circuits in Analog-Mixed Signal 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

Analog design in digital context is hard

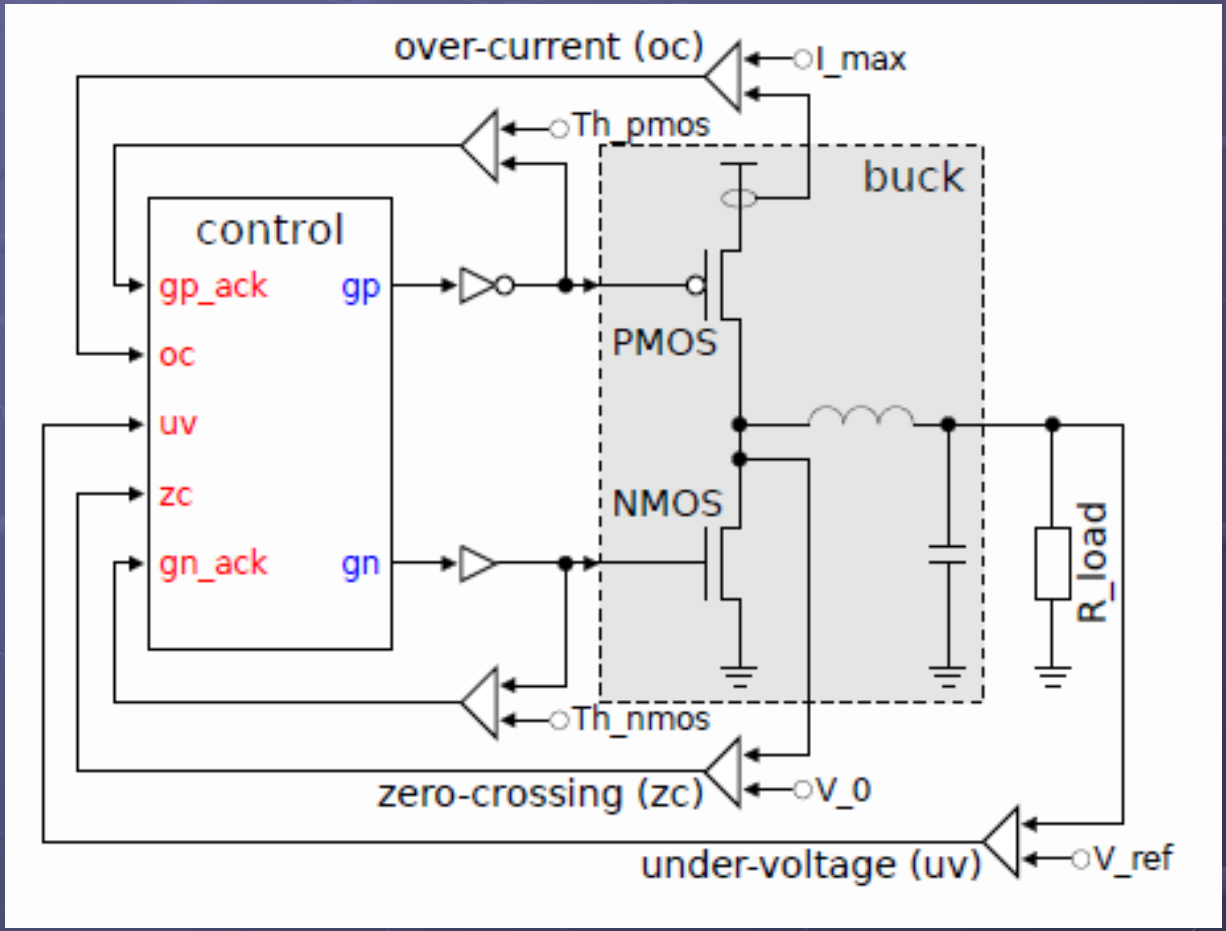
- If digital parts don't use clock, they are normally designed by hand and require massive simulations:
  - E.g. analog designers cannot afford simulating power converters from start-up; Instead they force it into known state
  - More specifically: 50 us of Spectre simulation time takes approx. 10 hours using 8 CPU cores
  - Hence they can only verify cherry-picked corners of digital functionality

(from Dialog Semiconductor, 2016)

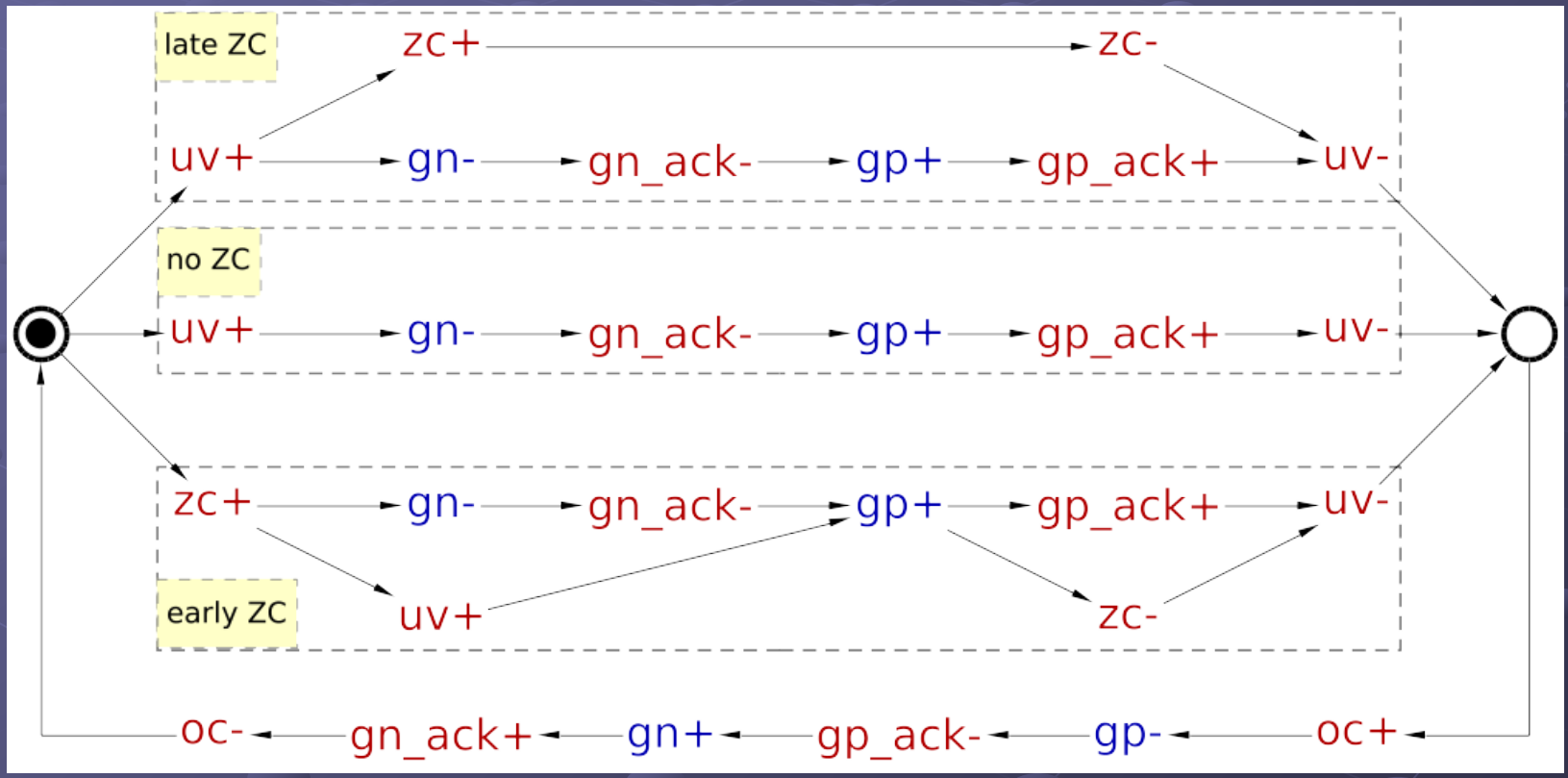
Towards Async Design for Analog

- Asynchronous design offers many advantages for AMS control
- Challenges:
  - It requires behavioural capture and synthesis but commercial EDA tools don't support it
  - Verification of asynchronous designs as part of AMS
  - How to provide non-invasiveness with existing design practices – we need to work with SVA and SPICE simulation traces

Buck example



STG Specification of buck controller



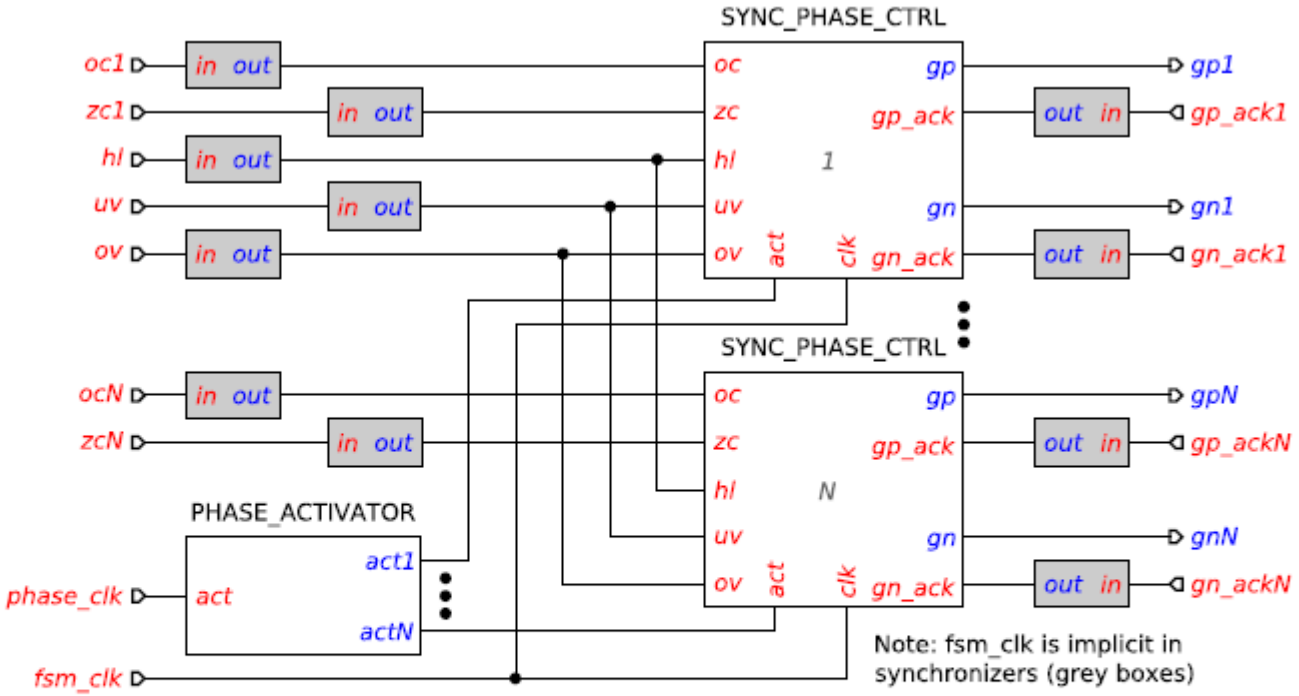
Synchronous design

- Two clocks: phase activation (~5MHz) and sampling (~100MHz)
  - ☺ Easy to design (RTL synthesis flow)
  - ☹ Response time is of the order of clock period
  - ☹ Power consumed even when idle
  - ☹ Non-negligible probability of a synchronisation failure
- Manual ad hoc design to alleviate the disadvantages
  - ☹ Verification by exhaustive simulation

Asynchronous design

- Event-driven control decisions
  - ☺ Prompt response (a delay of few gates)
  - ☺ No dynamic power consumption when the buck is inactive
  - ☺ Other well known advantages
  - ☹ Insufficient methodology and tool support
- Our goals
  - Formal specification of power control behaviour
  - Reuse of existing synthesis methods
  - Formal verification of the obtained circuits
  - Demonstrate new advantages for power regulation (power efficiency, smaller coils, ripple and transient response)

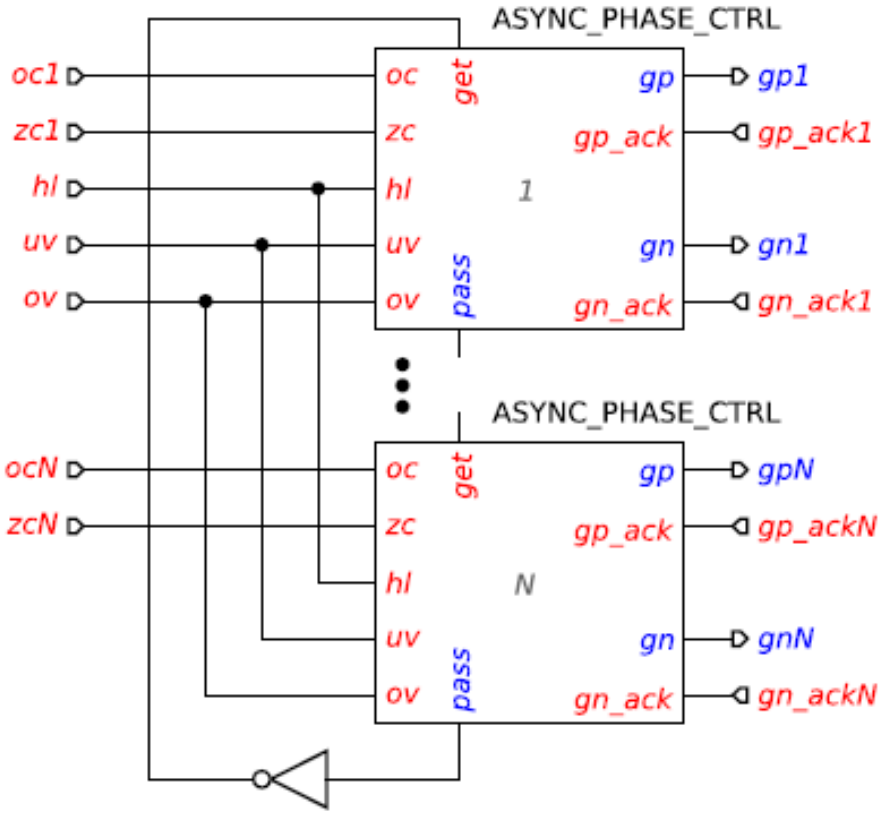
Multiphase Buck: Sync Control



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

Multiphase Buck: Async Control

Source: D. Sokolov, V. Khomenko, A. Mokhov, V. Dubikhin, D. Lloyd and A. Yakovlev, "Automating the Design of Asynchronous Logic Control for AMS Electronics," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 952-965, May 2020, doi: 10.1109/TCAD.2019.2907905.

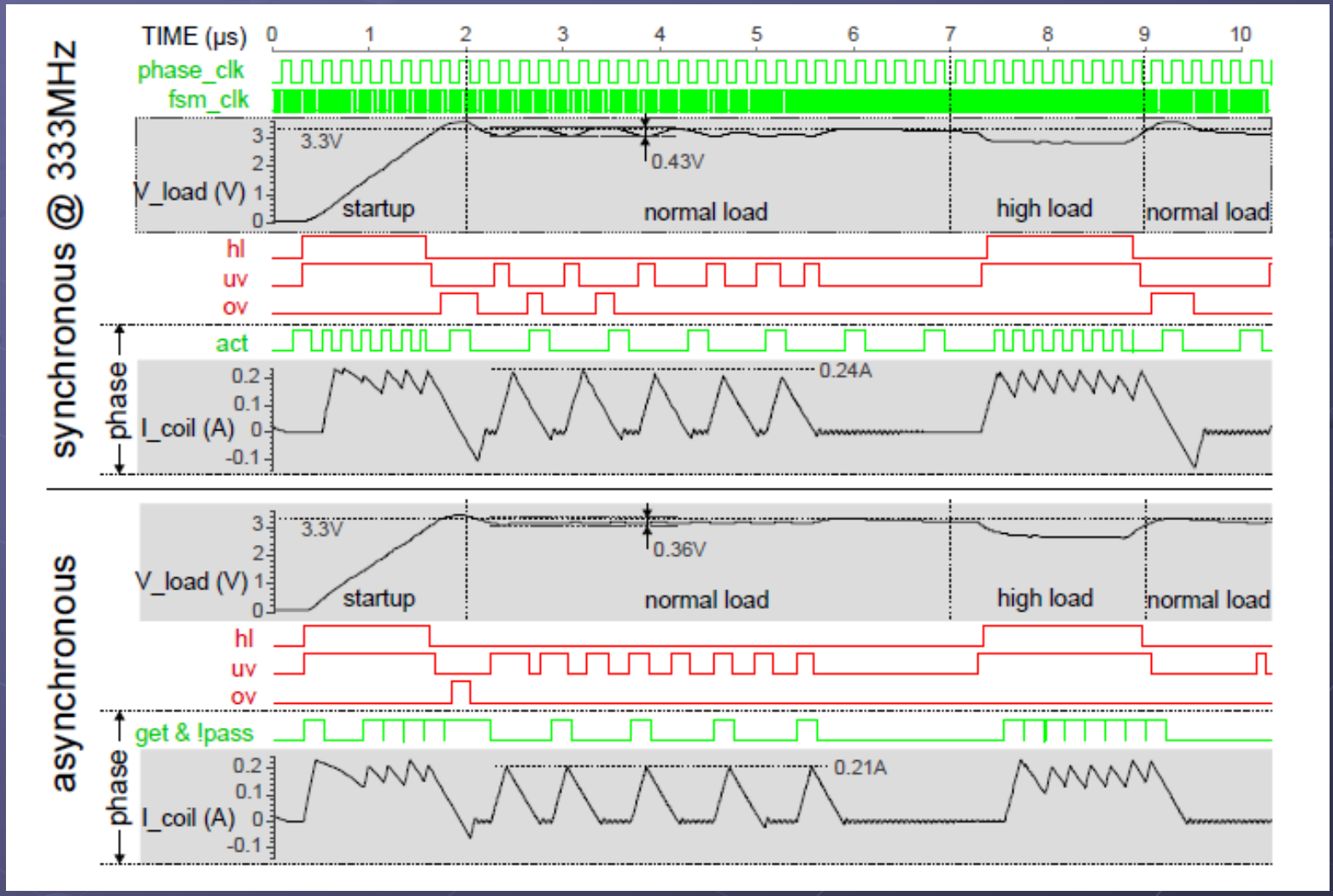


- Token ring architecture, no need for phase activation clock
- No need for synchronisers - all signals are asynchronous
- A4A design flow

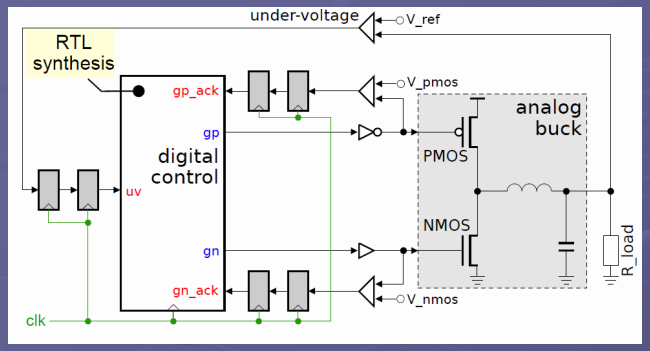
Simulation results: Comparison

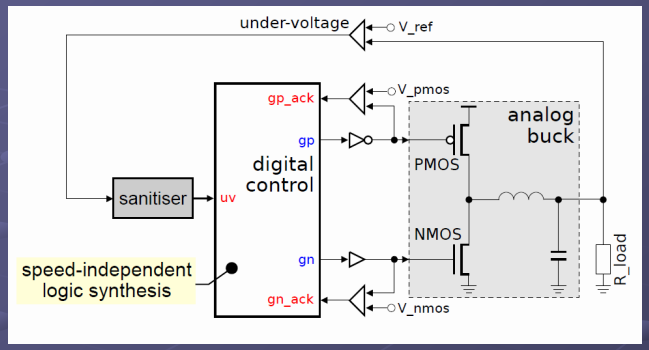
- Verilog-A model of the 3-phase buck
- Control implemented in TSMC 90nm
- AMS simulation in CADENCE NC-VERILOG
- Synchronous design
  - Phase activation clock – 5 MHz
  - Clocked FSM-based control – 100 MHz
  - Sampling and synchronisation
- Asynchronous design
  - Phase activation - token ring with 200 ns timer (= 5 MHz)
  - Event-driven control (input-output mode)
  - Waiting rather than sampling (A2A components)

Simulation results



A2A components



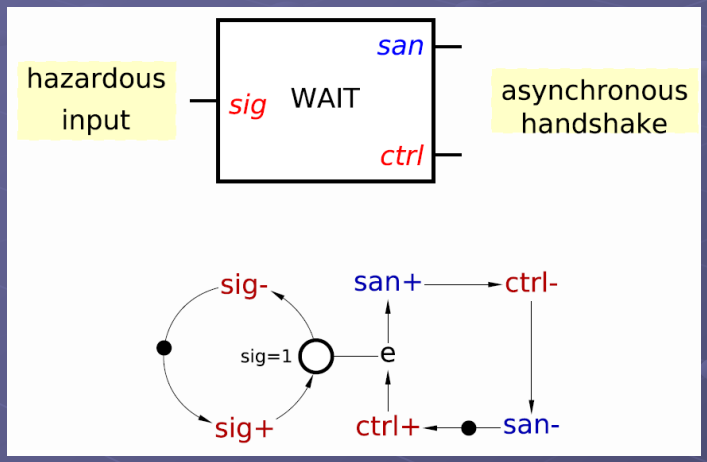


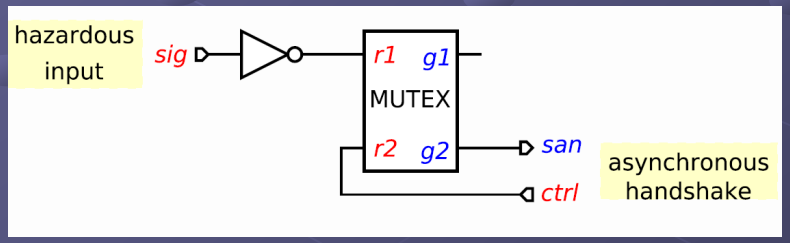
Analog-to-Async (A2A) elements:

- Synchronisation
  - WAIT: synchronise with high level of hazardous input
  - RWAIT: WAIT that can be with released/cancellation
  - WAIT01: synchronise with hazardous rising edge
  - WAIT2: synchronise with both phases of a hazardous input
- Decision-making
  - WAITX: arbitrate between two hazardous inputs
  - SAMPLE: sample a hazardous input

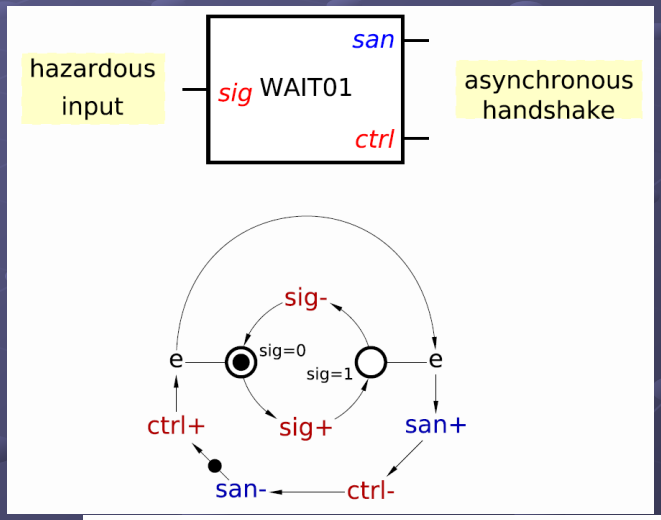
A2A elements (cont.)

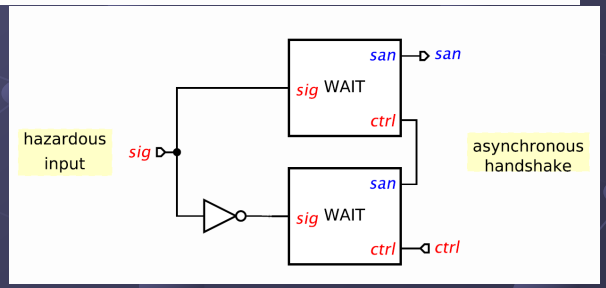
Wait: Synchronize handshake with the high level of hazardous input





Wait01: Synchronize handshake with the rising edge of hazardous input





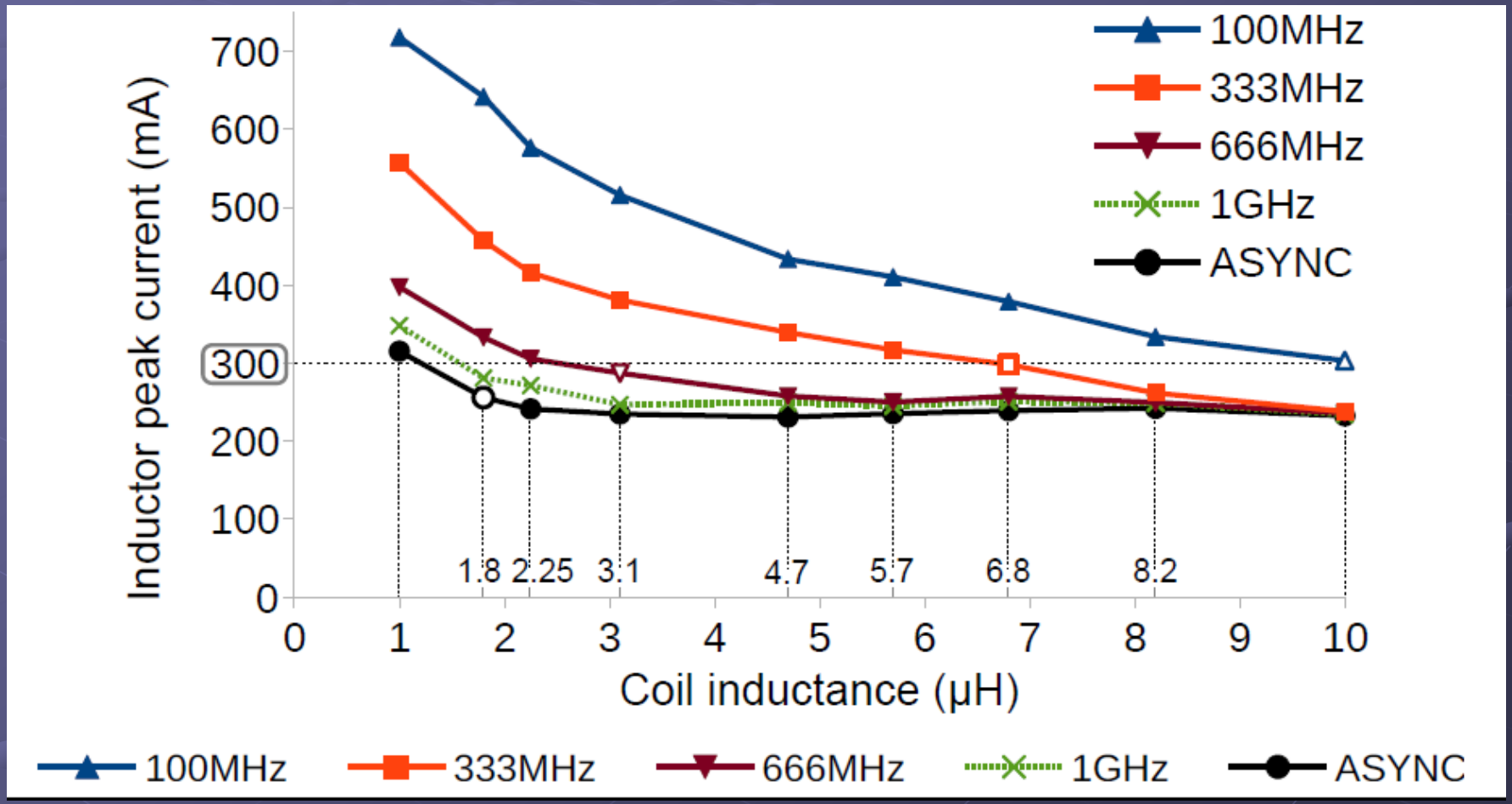
The full list of A2A components can be found here: https://workcraft.org/a2a/start

Reaction time

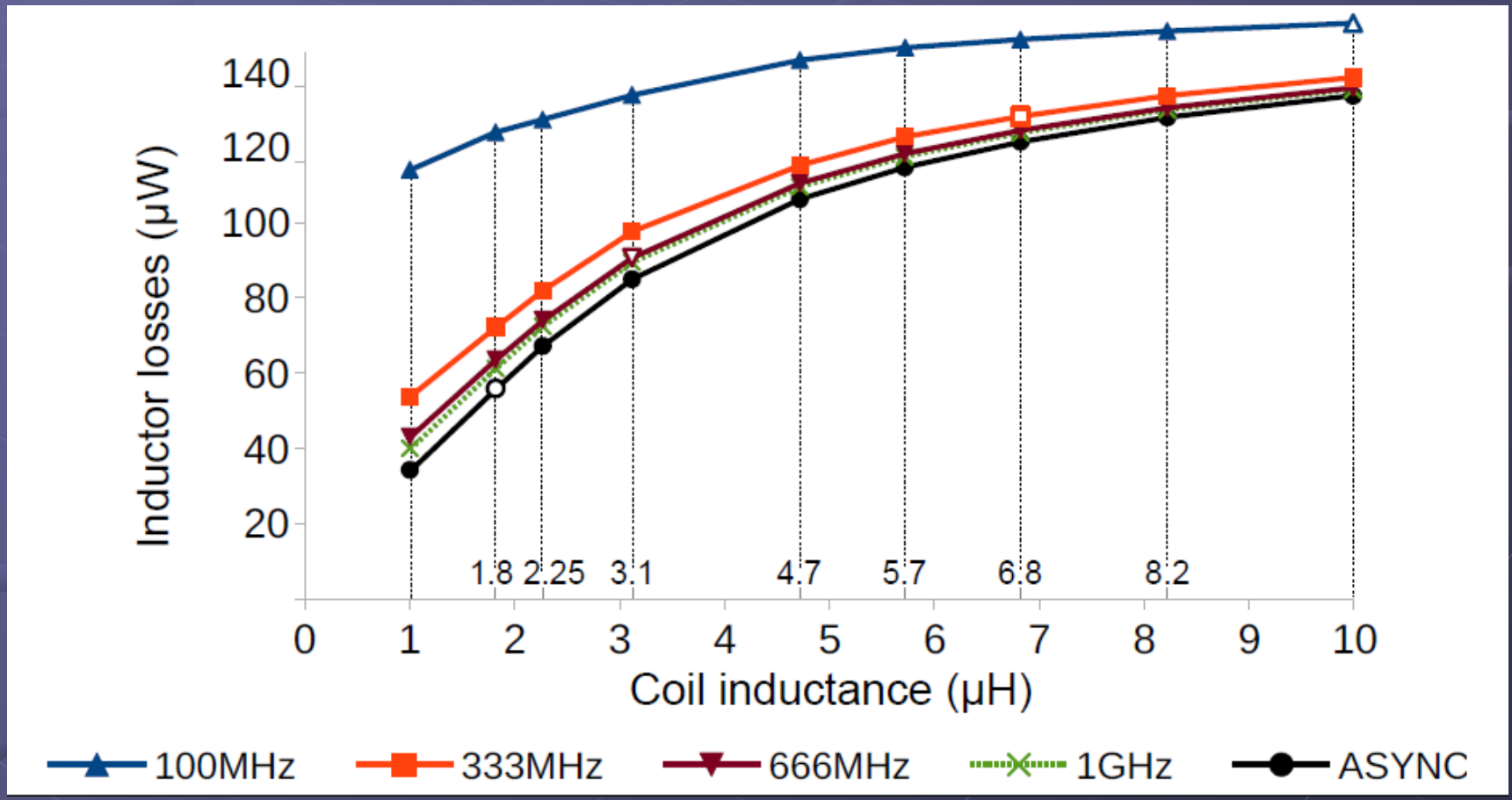
Buck controller	HL (ns)	UV (ns)	OV (ns)	OC (ns)	ZC (ns)
SYNC @ 100MHz	25.00	25.00	25.00	25.00	25.00
SYNC @ 333MHz	7.50	7.50	7.50	7.50	7.50
SYNC @ 666MHz	3.75	3.75	3.75	3.75	3.75
SYNC @ 1GHz	2.50	2.50	2.50	2.50	2.50
ASYNC	1.87	1.02	1.18	0.75	0.31
Improvement over 333MHz	4x	7x	6x	10x	24x

Synchronous buck controllers exhibit latency of 2.5 clock cycles.

Peak current



Inductor losses



Design Results

Design flow is automated to large extent

- Library of A2A components
- Automatic logic synthesis
- Formal verification at the STG and circuit levels

Benefits of asynchronous multiphase buck controller

- Reliable, no synchronization failures
- Quick response time (few gate delays)
- Reaction time can be traded off for smaller coils
- Lower voltage ripple and peak current

More reading

D. Sokolov, V. Khomenko, A. Mokhov, V. Dubikhin, D. Lloyd and A. Yakovlev, "Automating the Design of Asynchronous Logic Control for AMS Electronics," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 5, pp. 952-965, May 2020, doi: 10.1109/TCAD.2019.2907905.

and

https://workcraft.org/