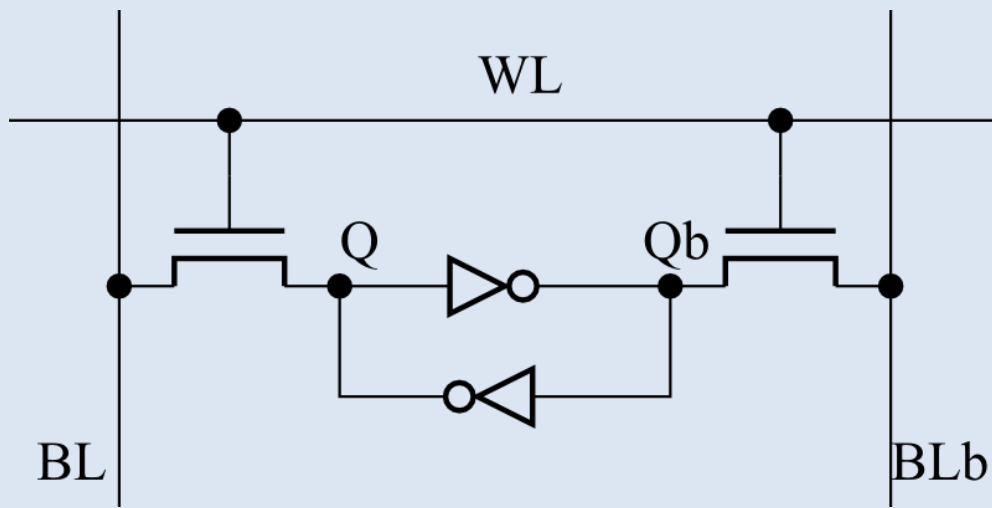


# Design examples: SRAM, buck, ADC

Andrey Mokhov, Alex Yakovlev  
Danil Sokolov, Victor Khomenko

Newcastle University, UK  
[async.org.uk](http://async.org.uk); [workcraft.org](http://workcraft.org)

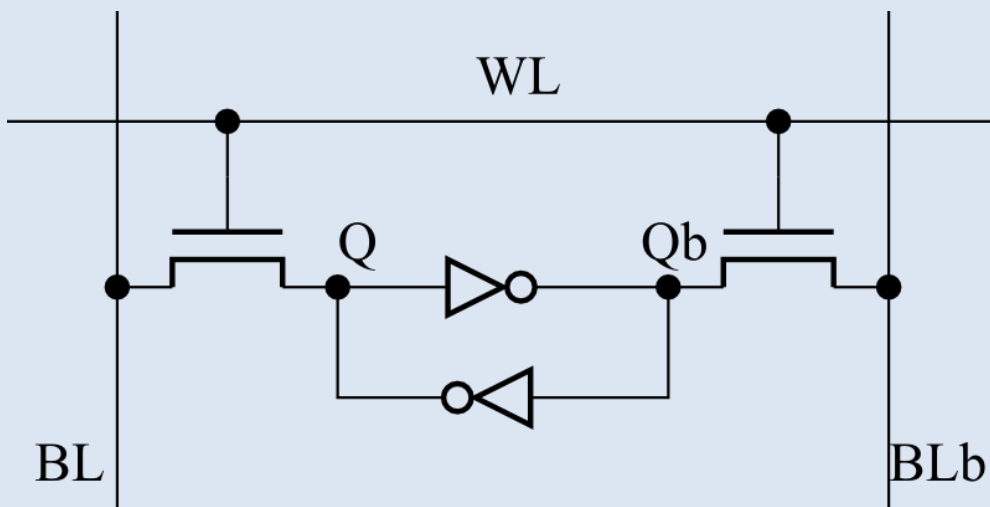
# Conventional 6T SRAM



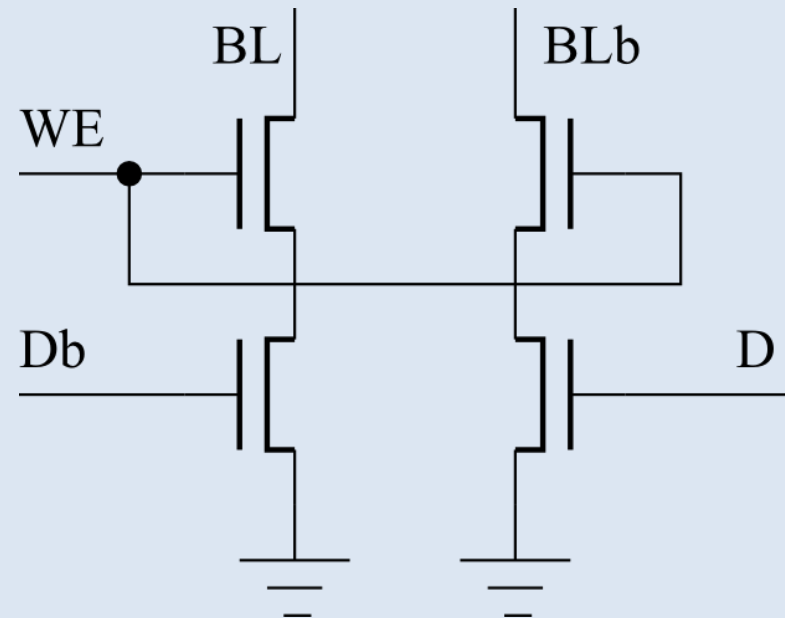
Bit cell

**Reading:** precharge bit lines, assert WL, **sense bit line changes**

# Conventional 6T SRAM



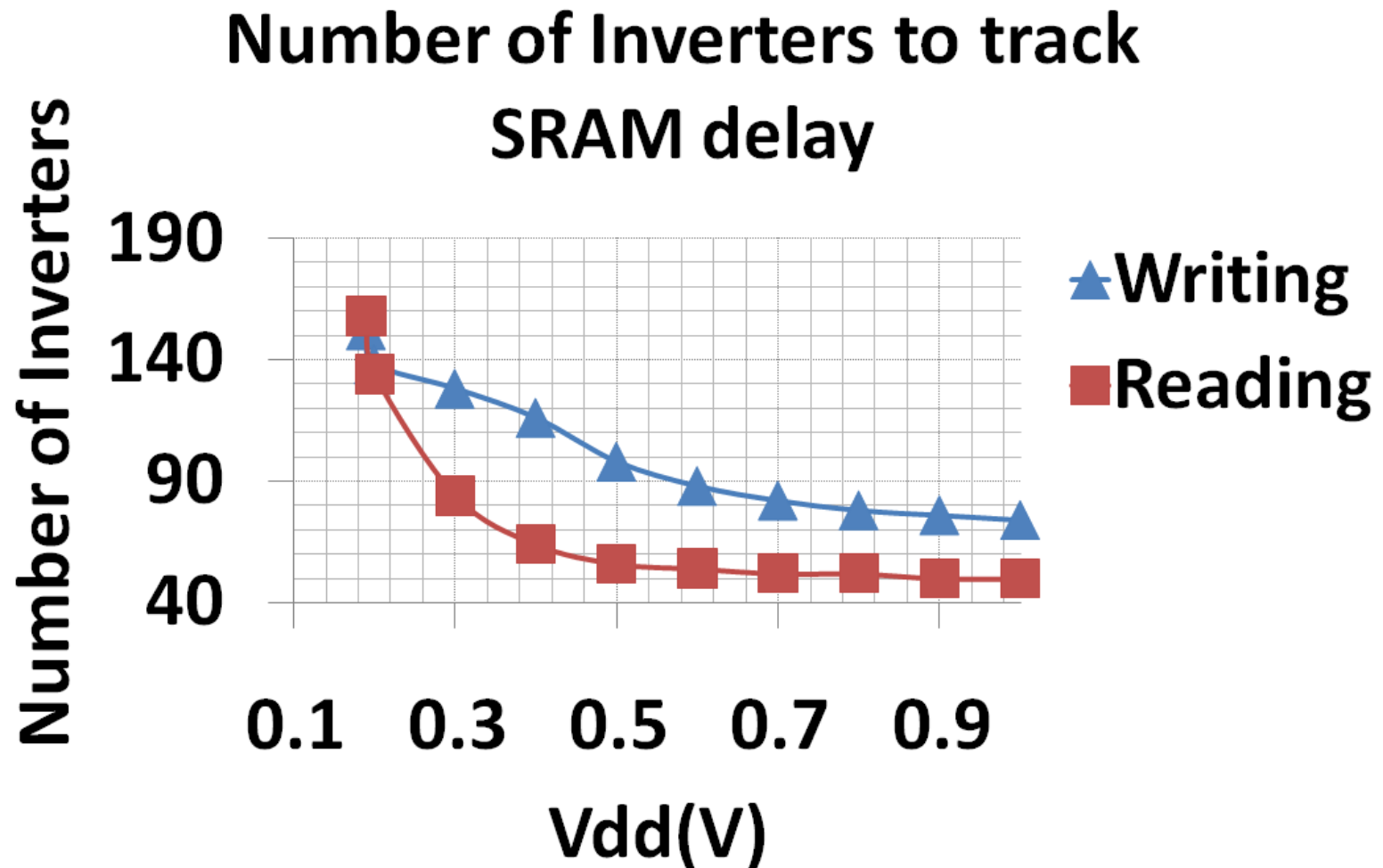
Bit cell



Write driver

**Writing:** set data lines, assert WL and WE, **wait for a while...**

# Problem: how long to wait?



# Problem: how long to wait?

SRAM 6T cell delays are difficult to match accurately

- When  $V_{dd} = 1V$ , SRAM read delay is  $\approx 50$  inverters
- When  $V_{dd} = 190mV$ , SRAM read delay is  $\approx 158$  inverters
- Read and write delays scale differently with  $V_{dd}$

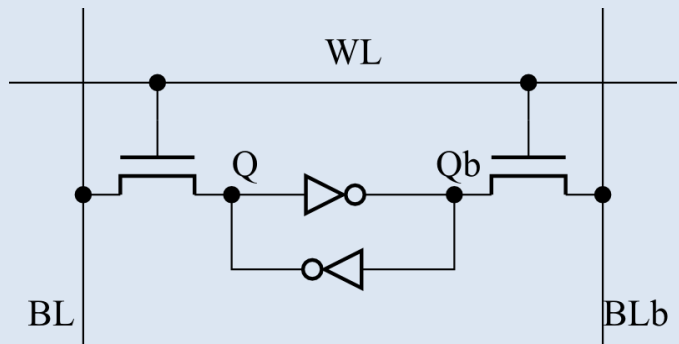
Conventional solutions

- Use different delay lines for different ranges of  $V_{dd}$
- Duplicate an SRAM line to act as a reference delay line
- Need voltage references, costly in area and energy

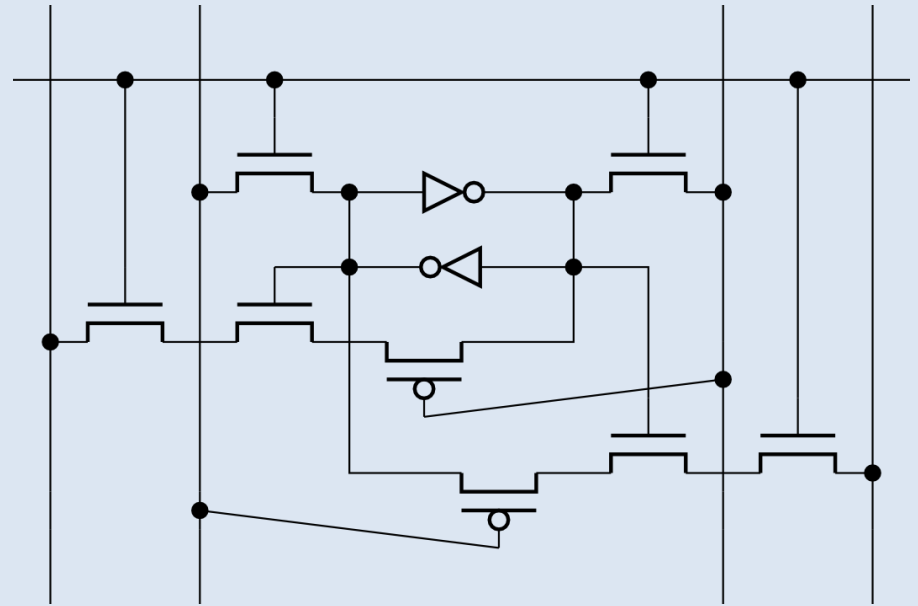
Asynchronous solution with completion detection

- Speed-independent, free from voltage references
- Developed by A. Baz *et al.* (PATMOS 2010, JOLPE 2011)

# Low-level completion detection



Bit cell

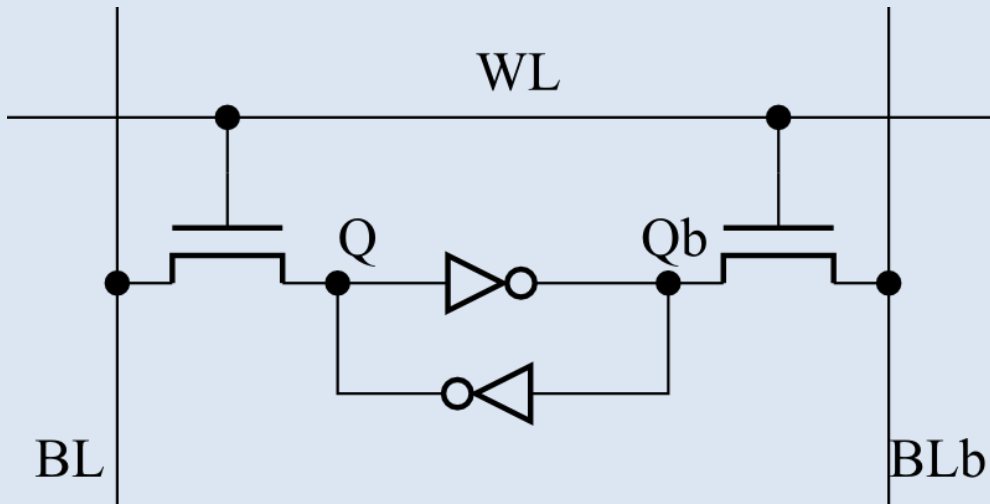


Bit cell with CD

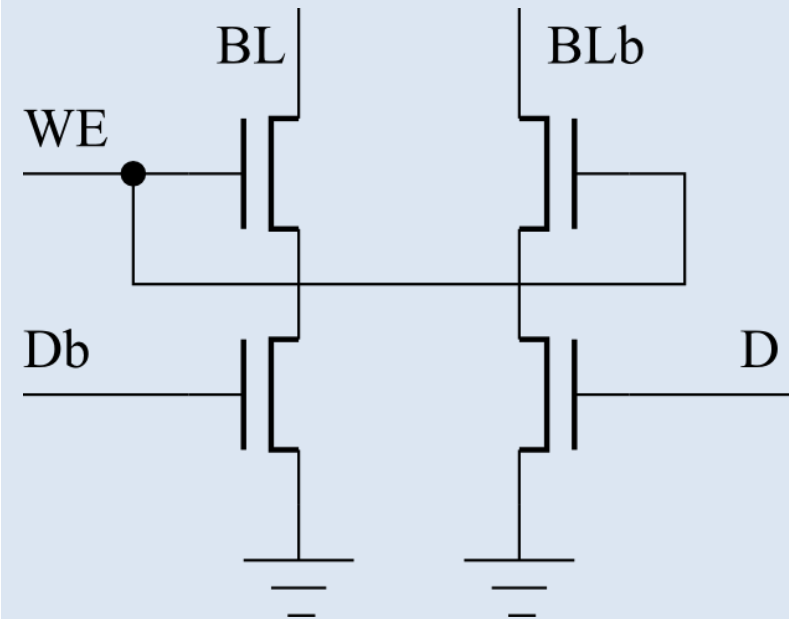
True completion detection both for reading and writing

Too costly!

# Back to conventional 6T SRAM



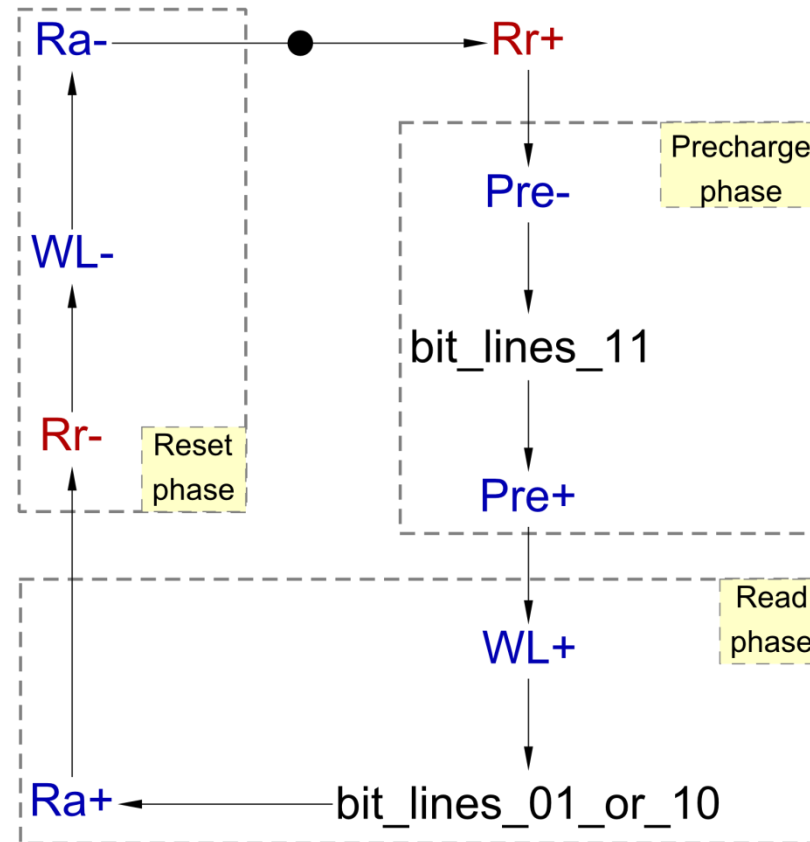
Bit cell



Write driver

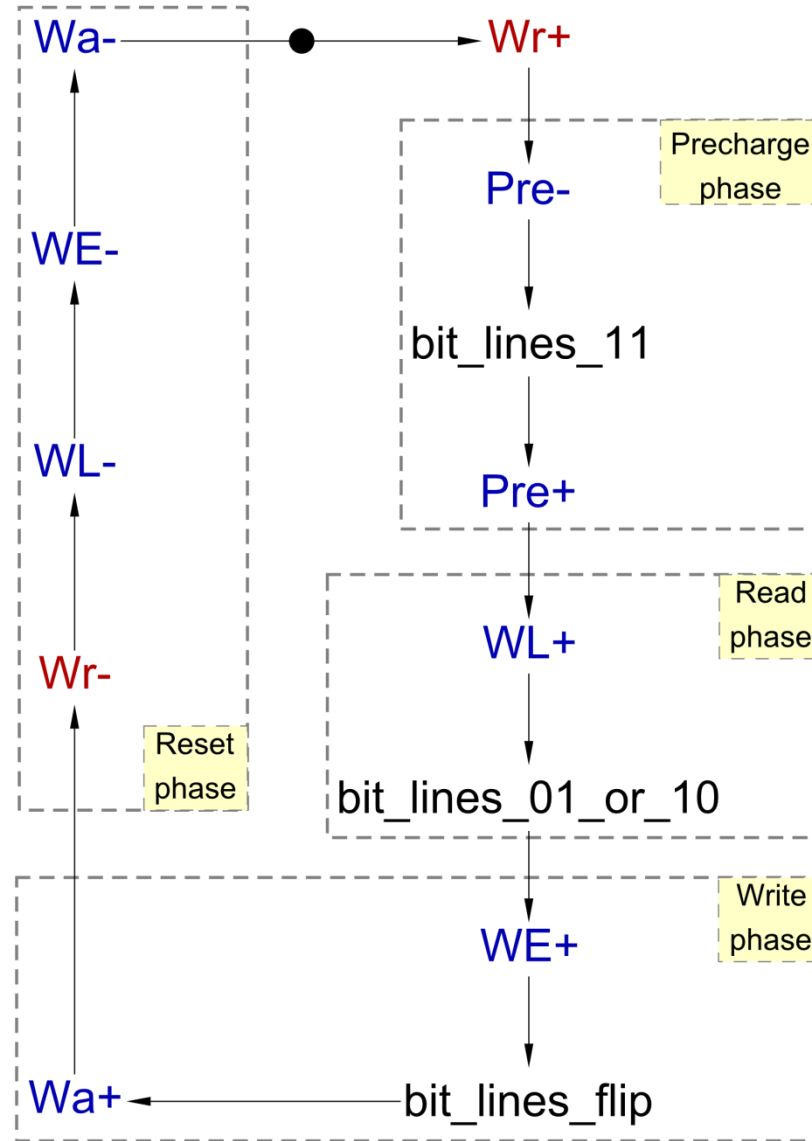
Idea 1: completion detection is possible when the bit is flipped  
Idea 2: read before writing to check if the bit will be flipped

# Specification: read scenario

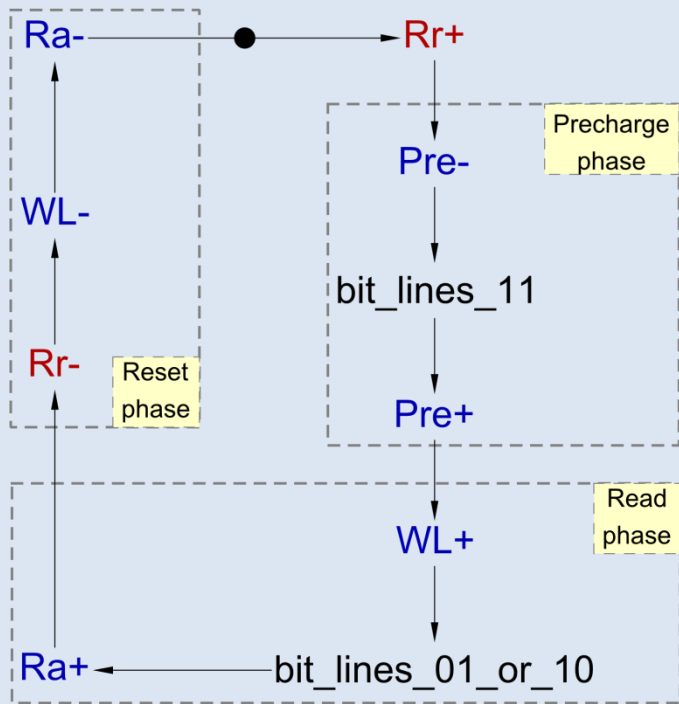




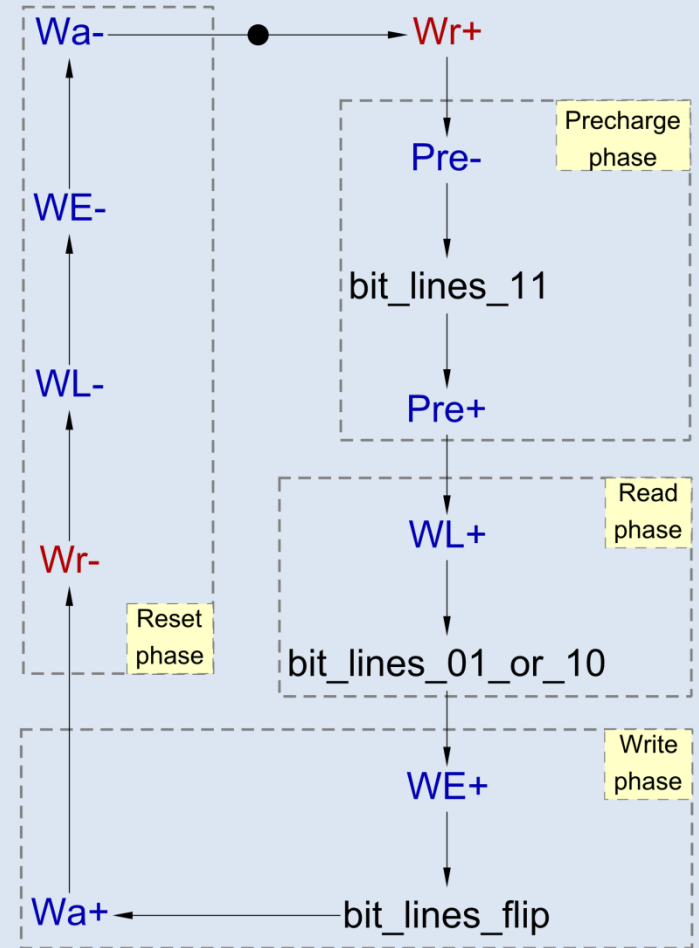
# Specification: write scenario



# Specification: composing scenarios

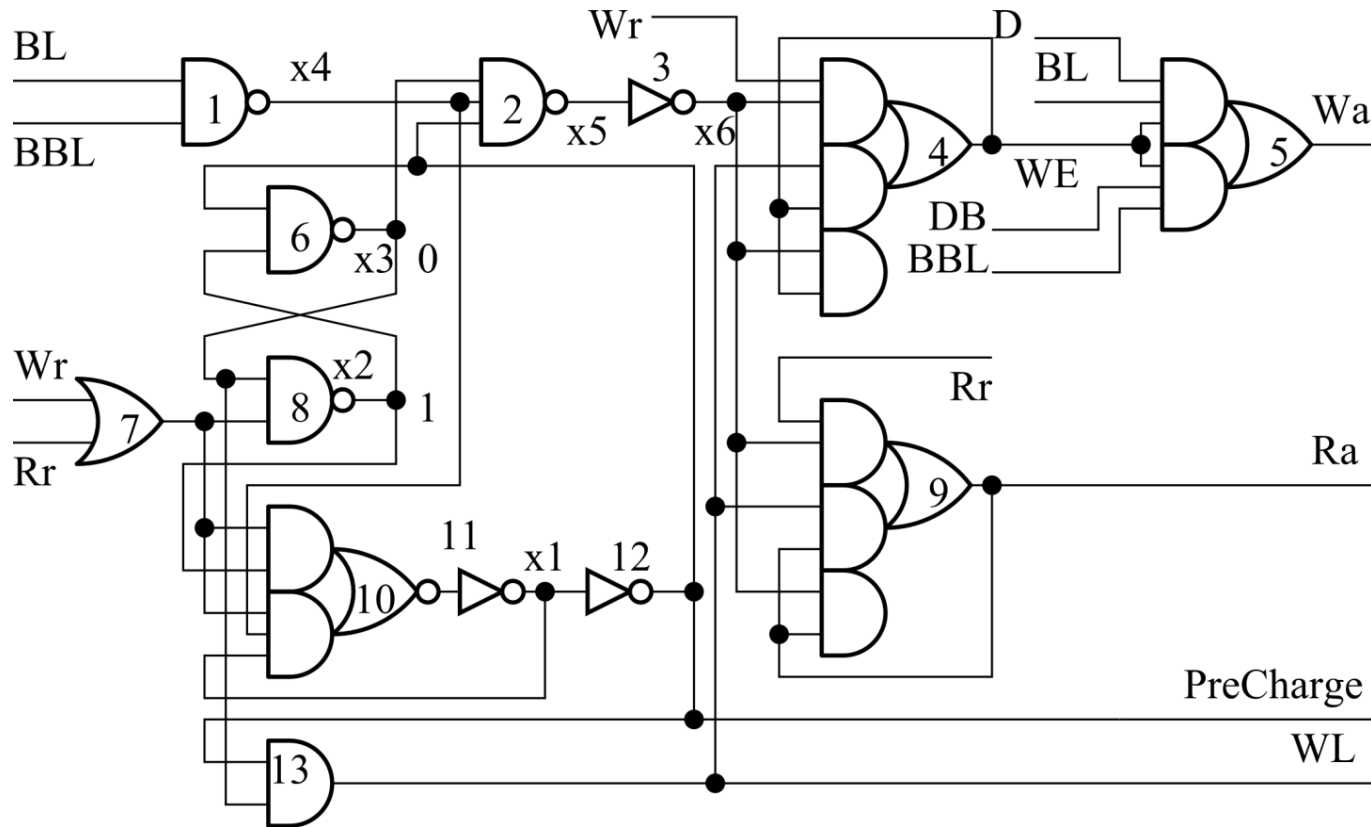


Read



Write

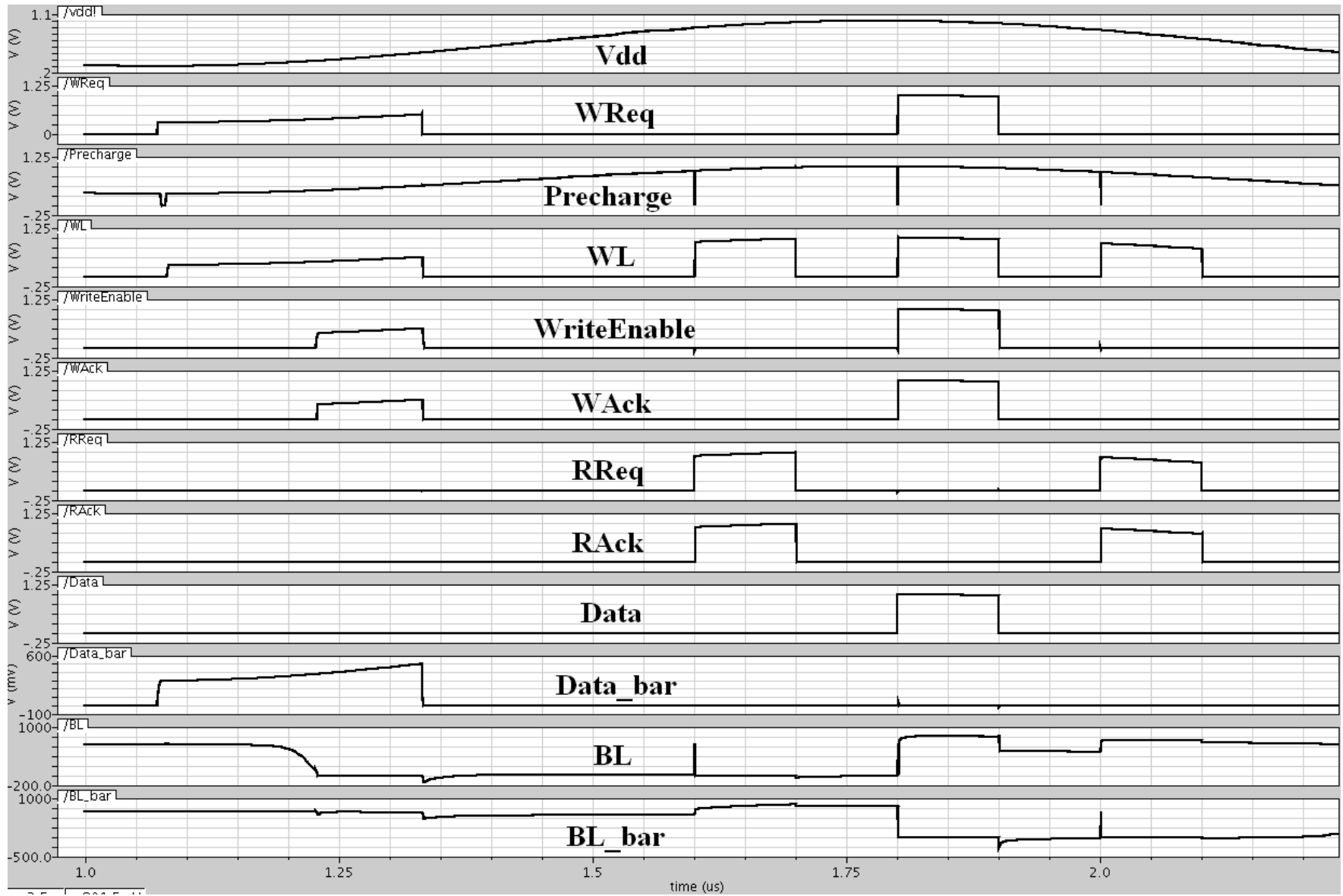
# Asynchronous SRAM controller



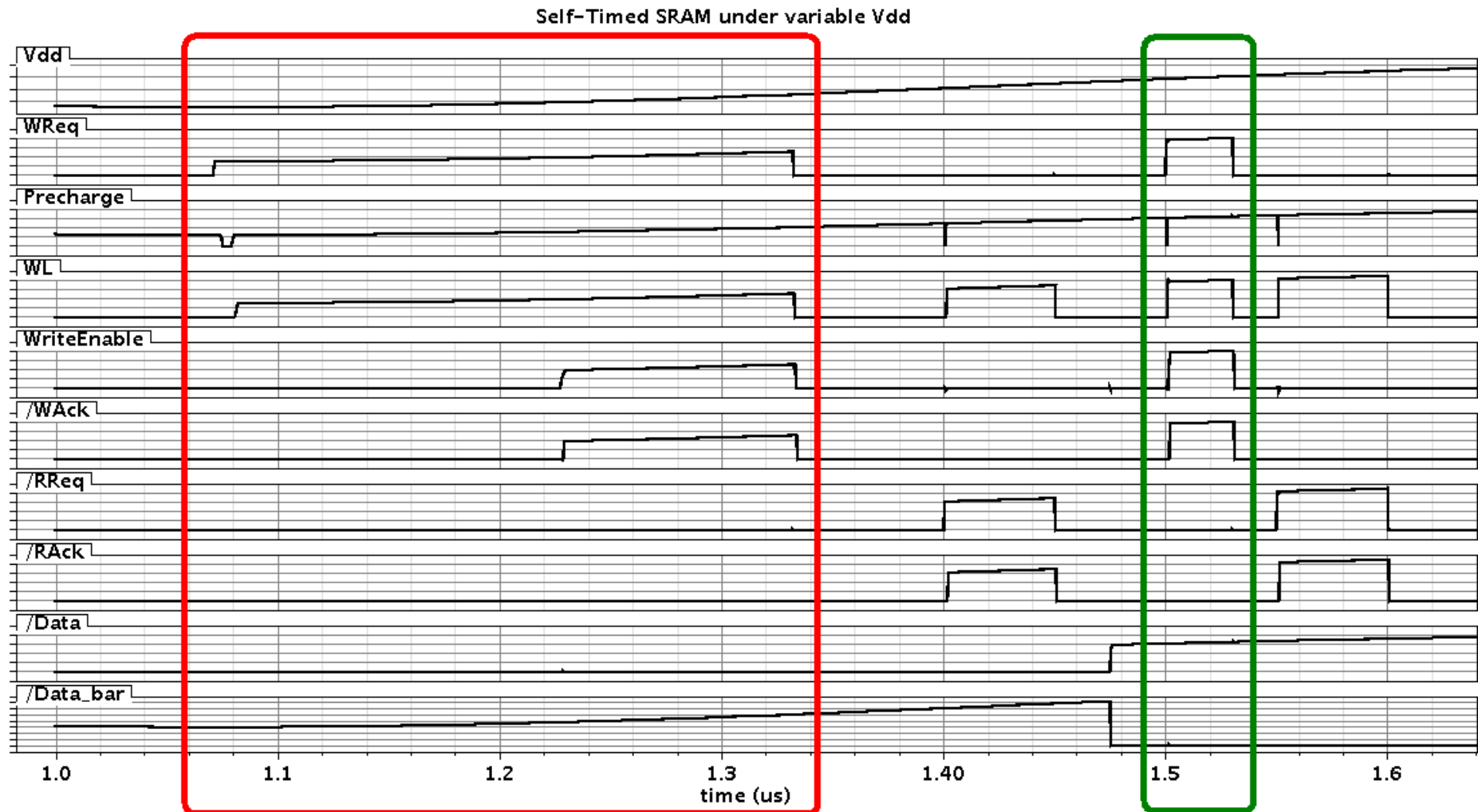
Hand made, hence not guaranteed to be speed-independent

We will design a provably correct implementation in Part II

# Tolerating variable voltage supply



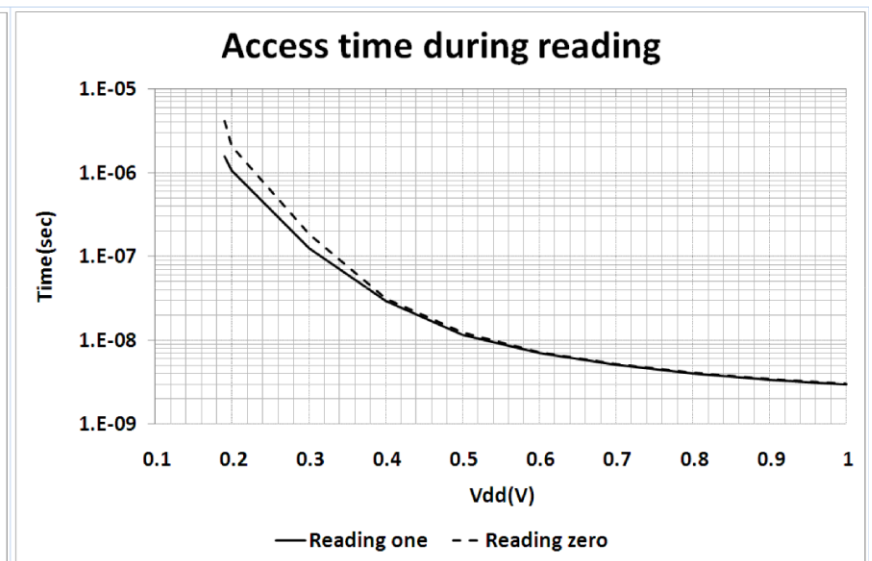
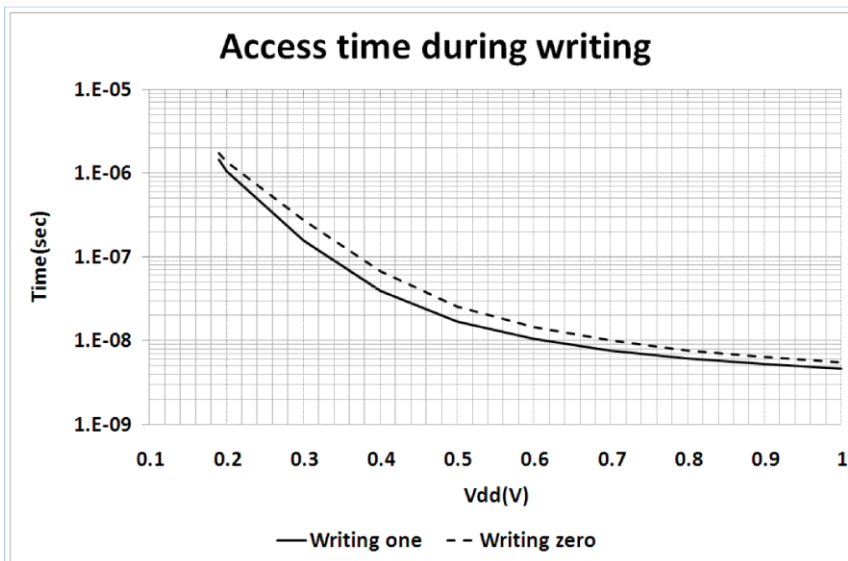
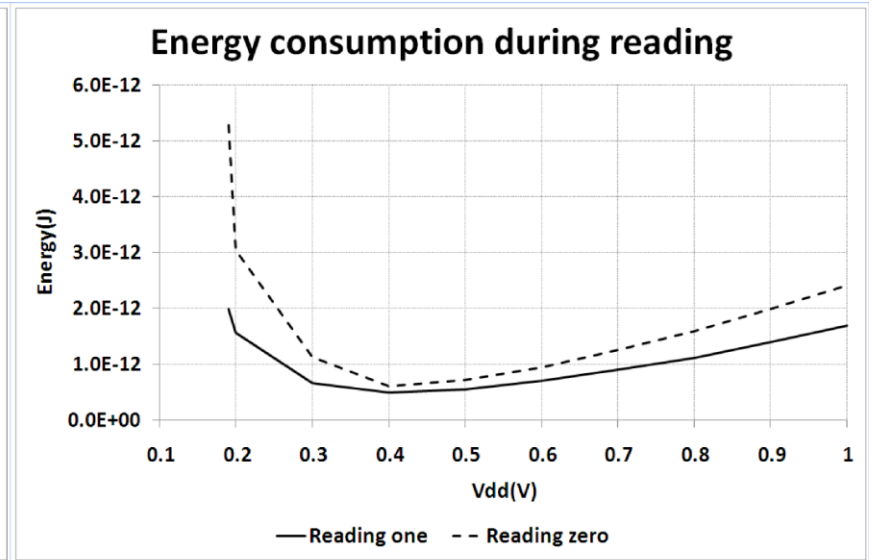
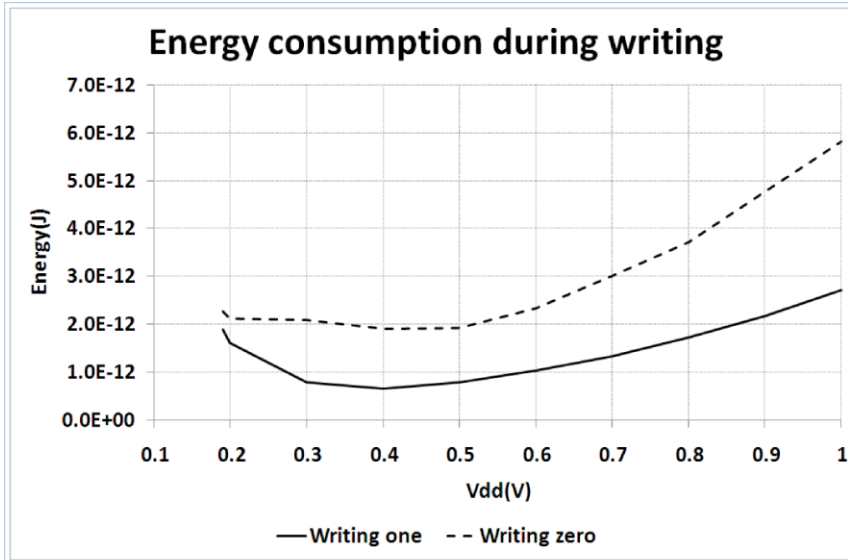
# Tolerating variable voltage supply



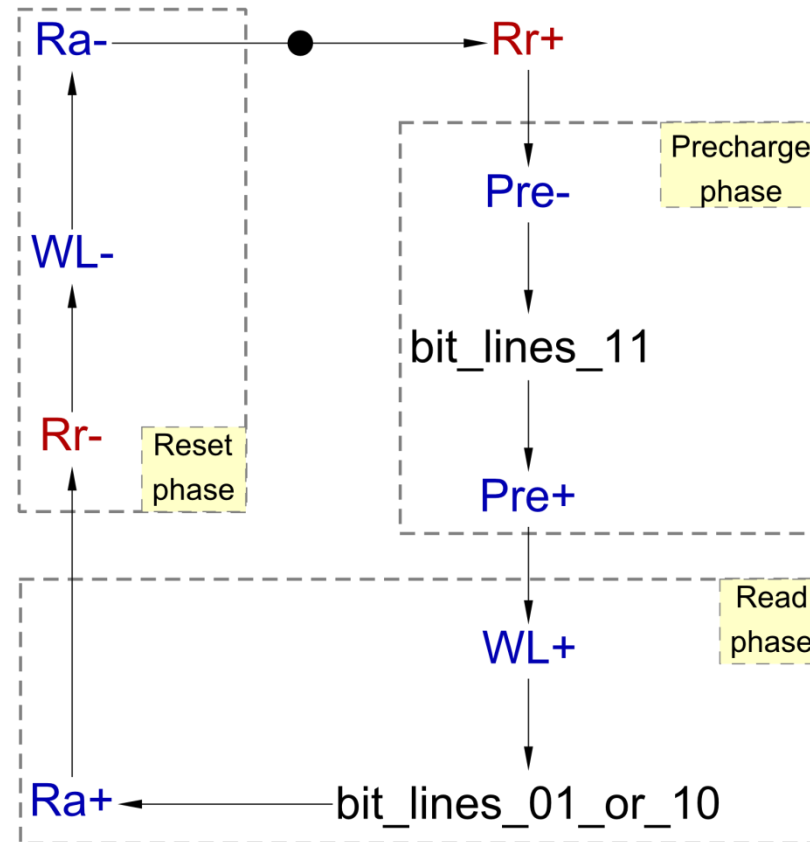
Low voltage, slow response

High voltage, fast response

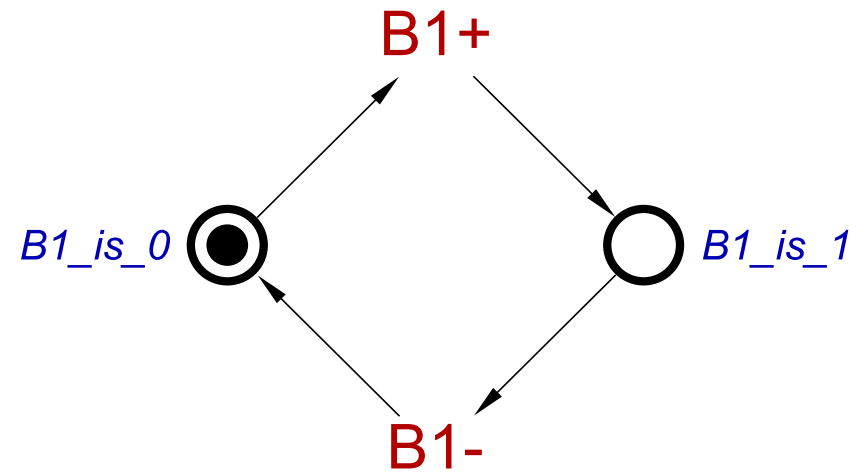
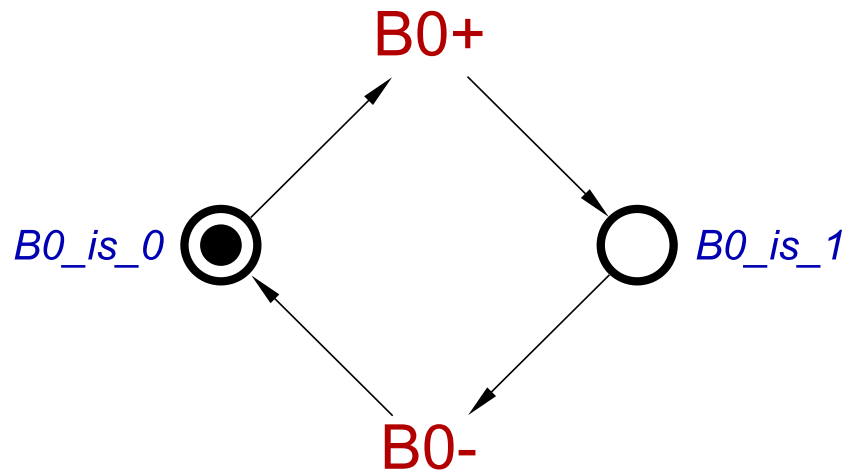
# Trading energy for performance



# Specification: read scenario

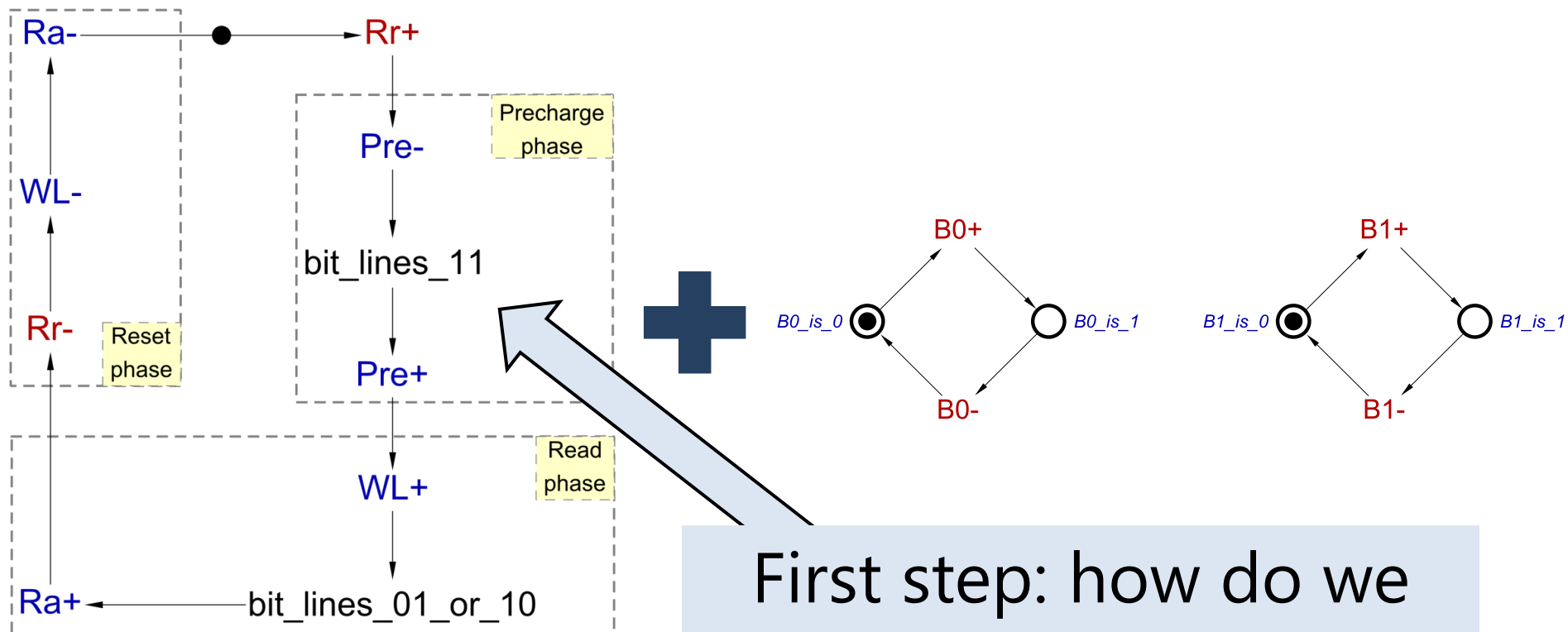


# Modelling bit line signals B0 and B1



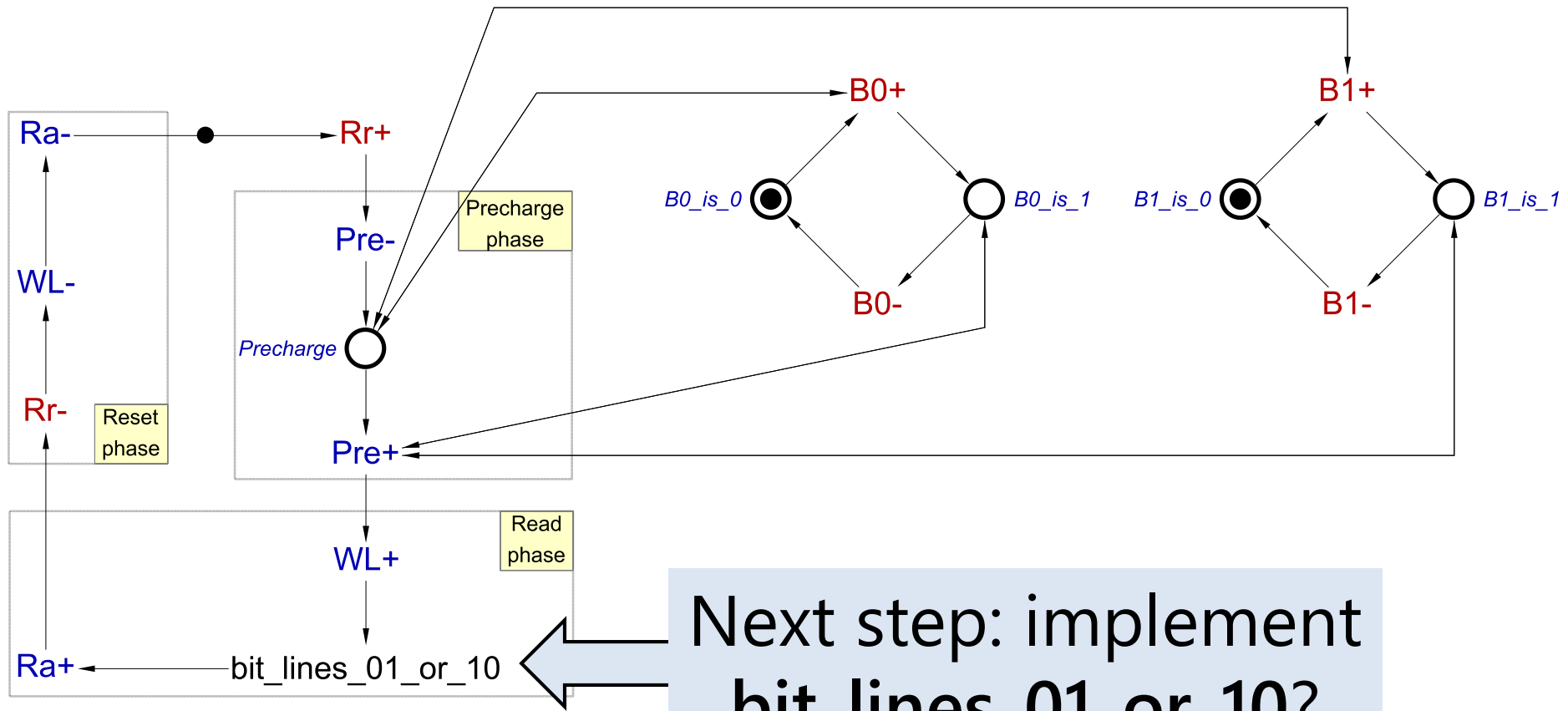


# Adding bit line signals to read scenario

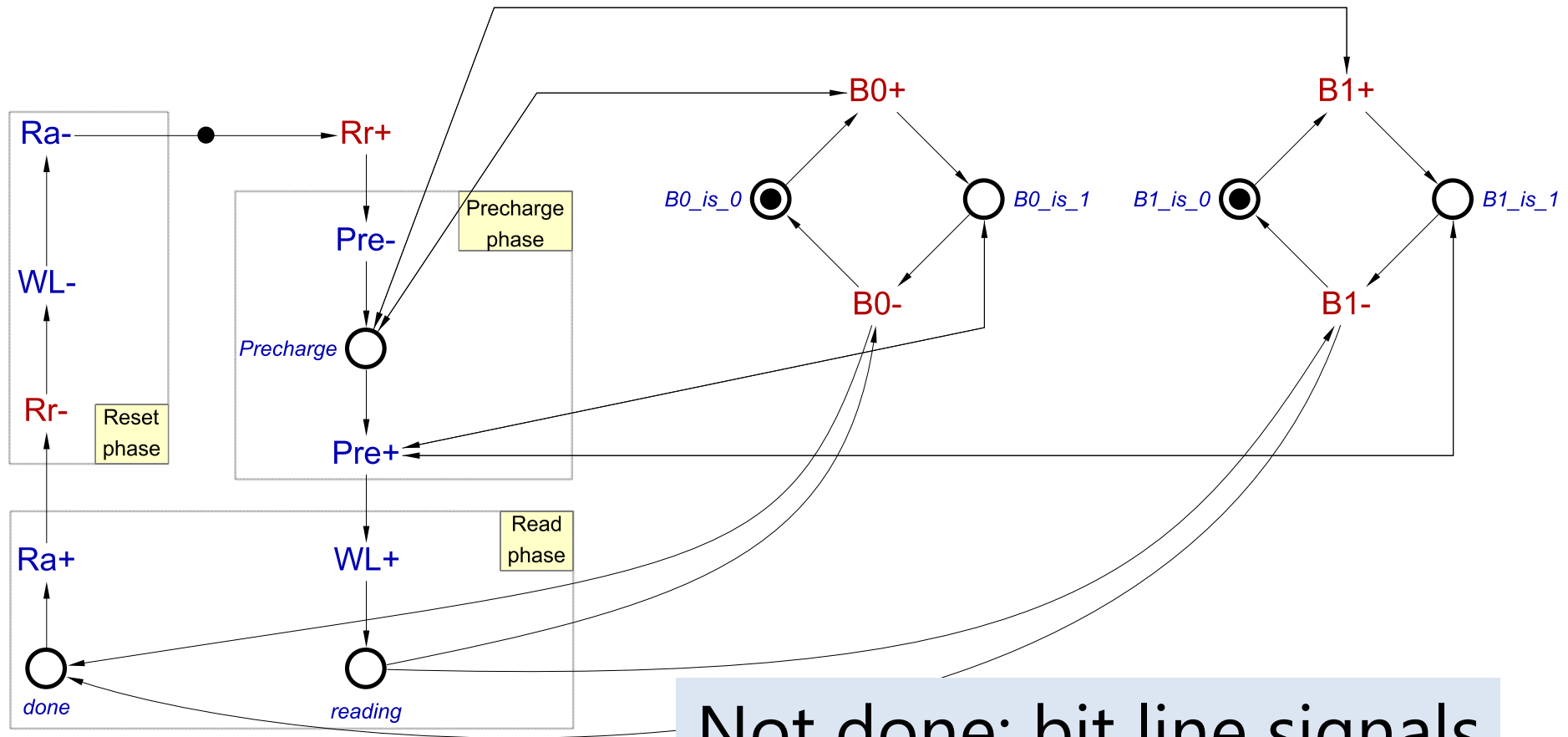


First step: how do we implement **bit\_lines\_11**?

# Adding bit line signals to read scenario



# Adding bit line signals to read scenario



Not done: bit line signals may change when  $Rr=0$

# Summary

Memory is inherently asynchronous

- Read & write completion can be reliably detected
- Conventional synchronous 'handcuffs' (matching delay lines) are clumsy and costly

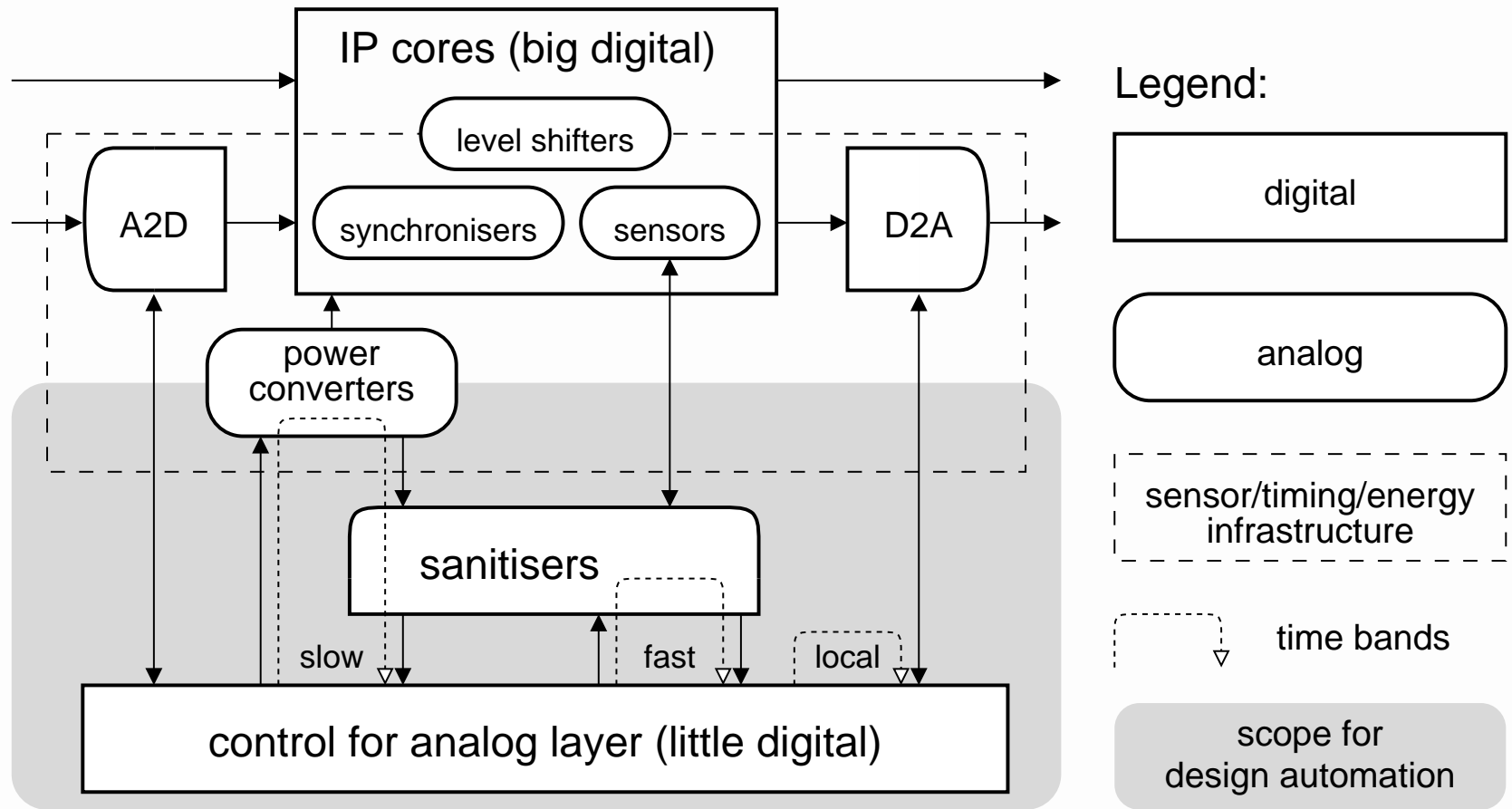
Typical 'little digital' control, fully supported by Workcraft design, synthesis and verification flow

Ongoing and future work – **you can contribute!**

- Integrate asynchronous SRAM into a real system
- Opportunity for new memory architectures

# Multiphase Buck Converter

# Motivation



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

# Power electronics context

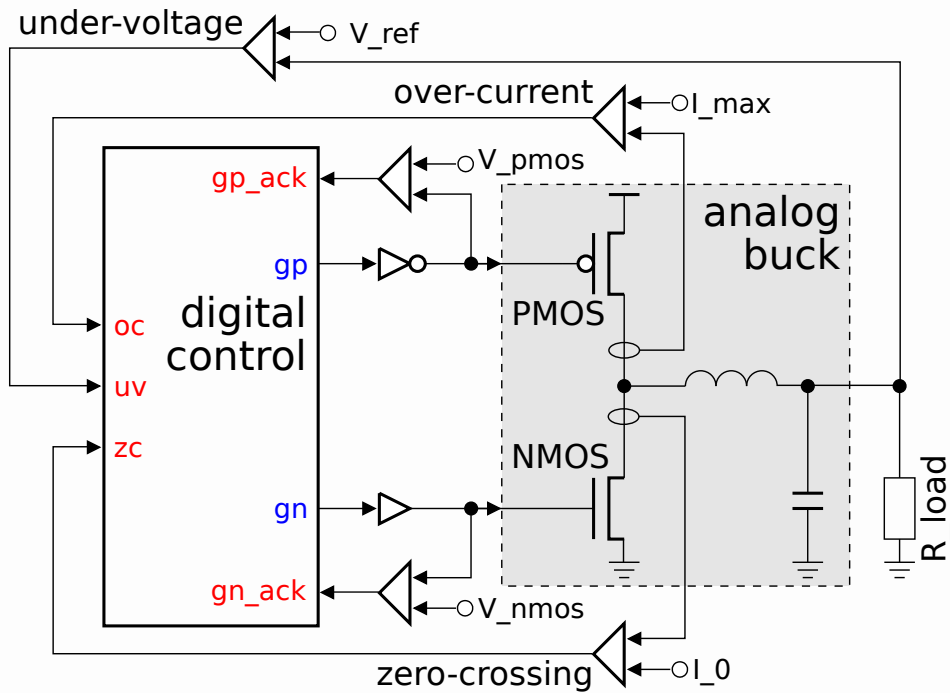
- Efficient implementation of power converters is paramount
  - Extending the battery life of mobile gadgets
  - Reducing the energy bill for PCs and data centres (5% and 3% of global electricity production respectively)
- Need for responsive and reliable control circuitry
  - Millions of control decisions per second for years
  - An incorrect decision may permanently damage the circuit
- Need for EDA (*little digital vs big digital* design flow)
  - RTL flow is optimised for synchronous data processing
  - *Ad hoc* asynchronous solutions are prone to errors and cannot be verified

# Synchronous vs asynchronous control

- 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

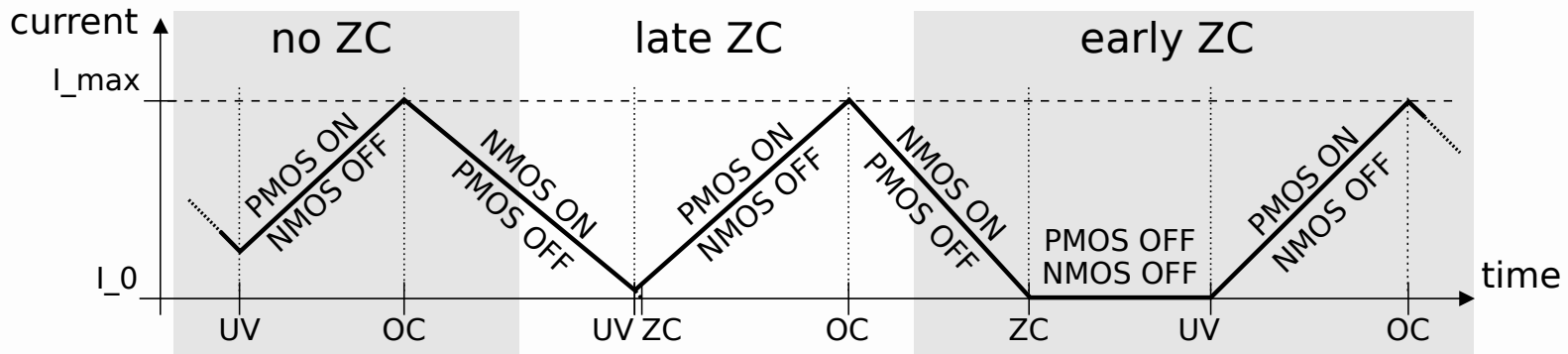


# Basic buck converter



## Operating modes

- Under-voltage (UV)
- Over-current (OC)
- Zero-crossing (ZC)



# Synchronous design

```
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;
```

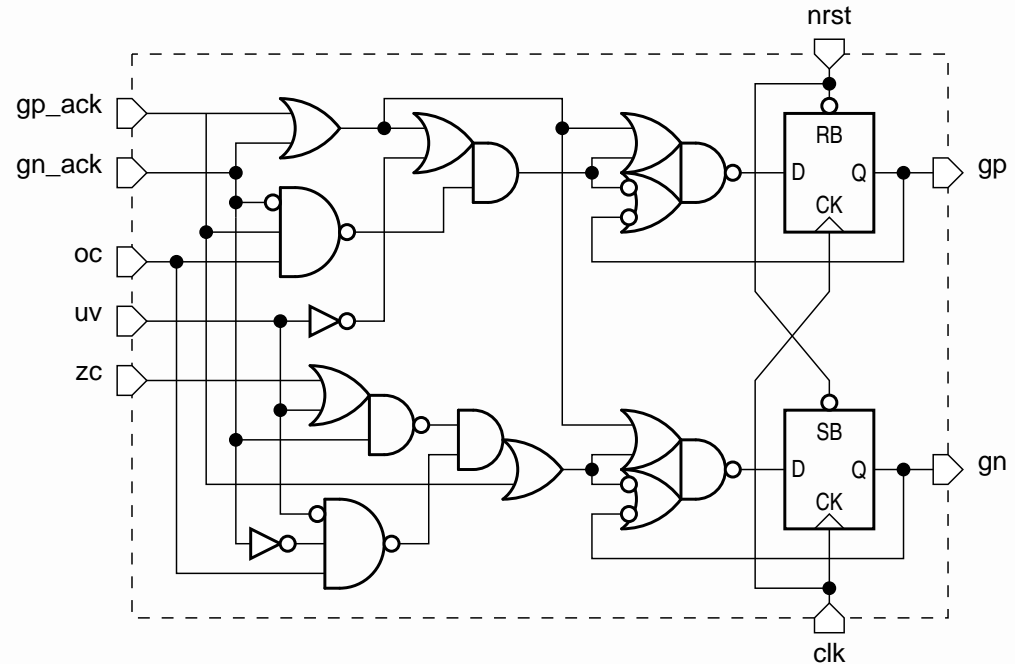
```
2'b10: if (oc == 1) gp <= 0;
```

```
2'b01: if (uv == 1 || zc == 1) gn <= 0;
```

```
endcase
```

```
end
```

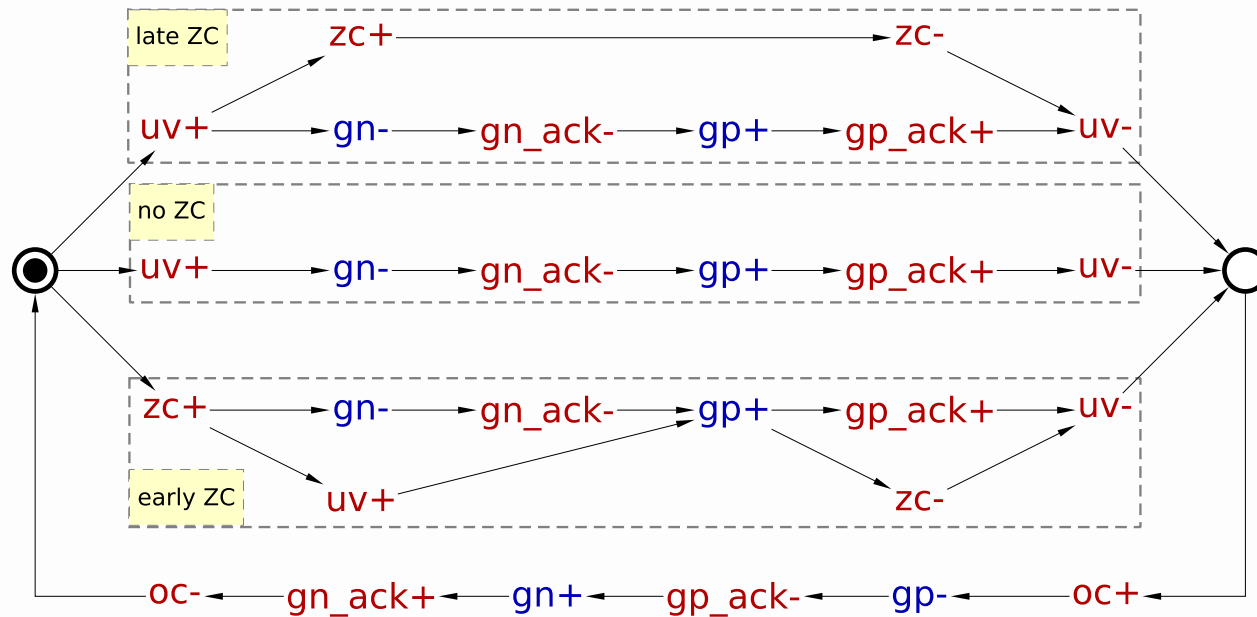
```
endmodule
```



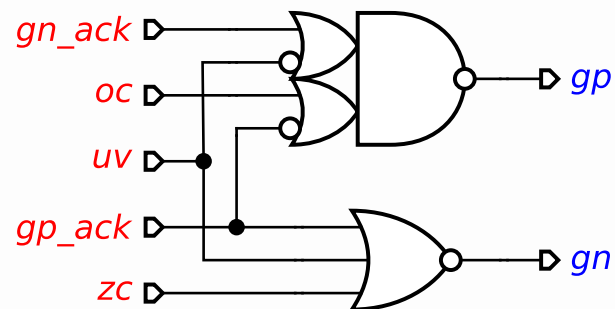
- If clock is slow, the control is unresponsive to the buck changes
- If clock is fast, it burns energy when the buck is inactive

# Asynchronous design

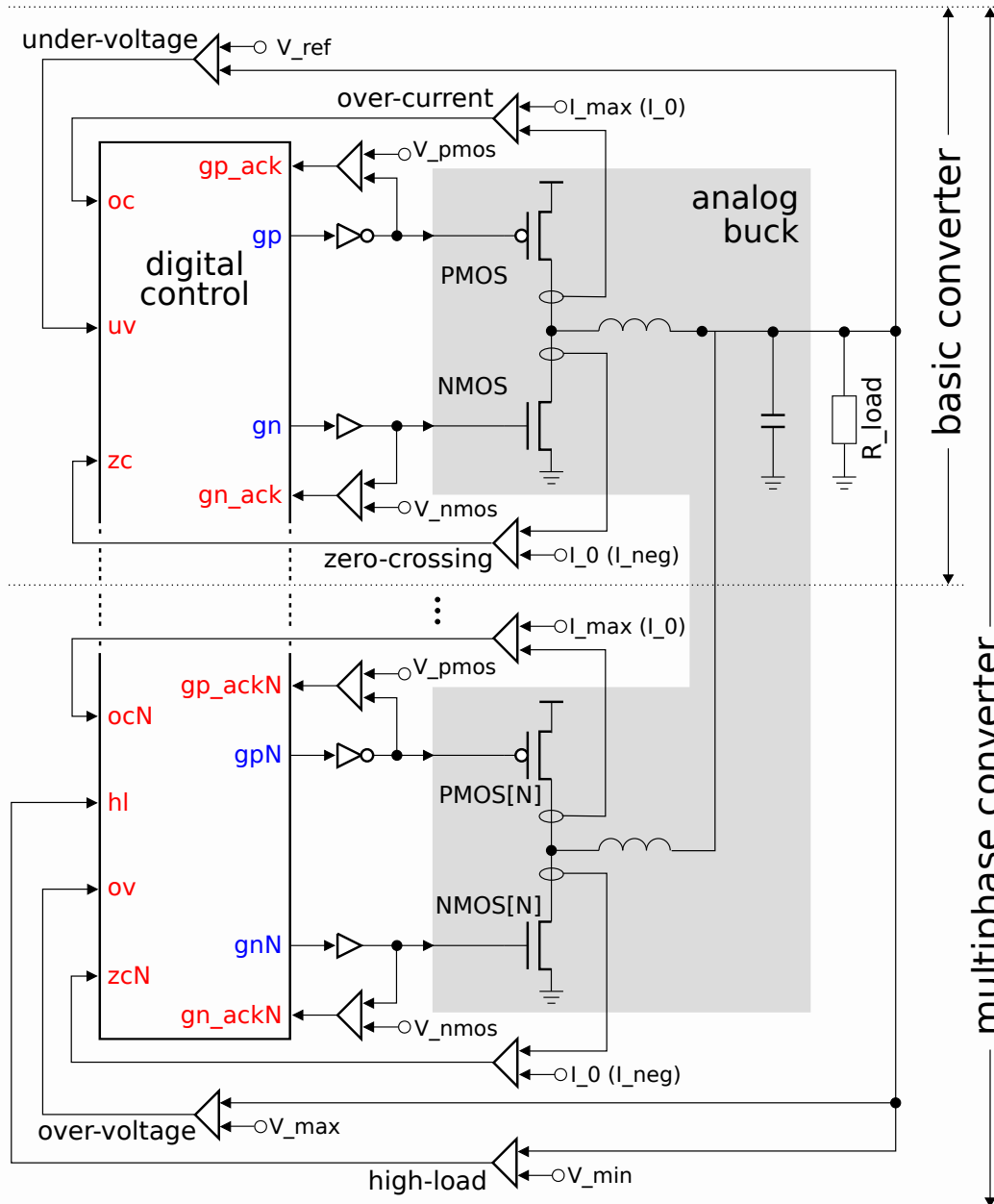
- STG specification



- Speed-independent implementation

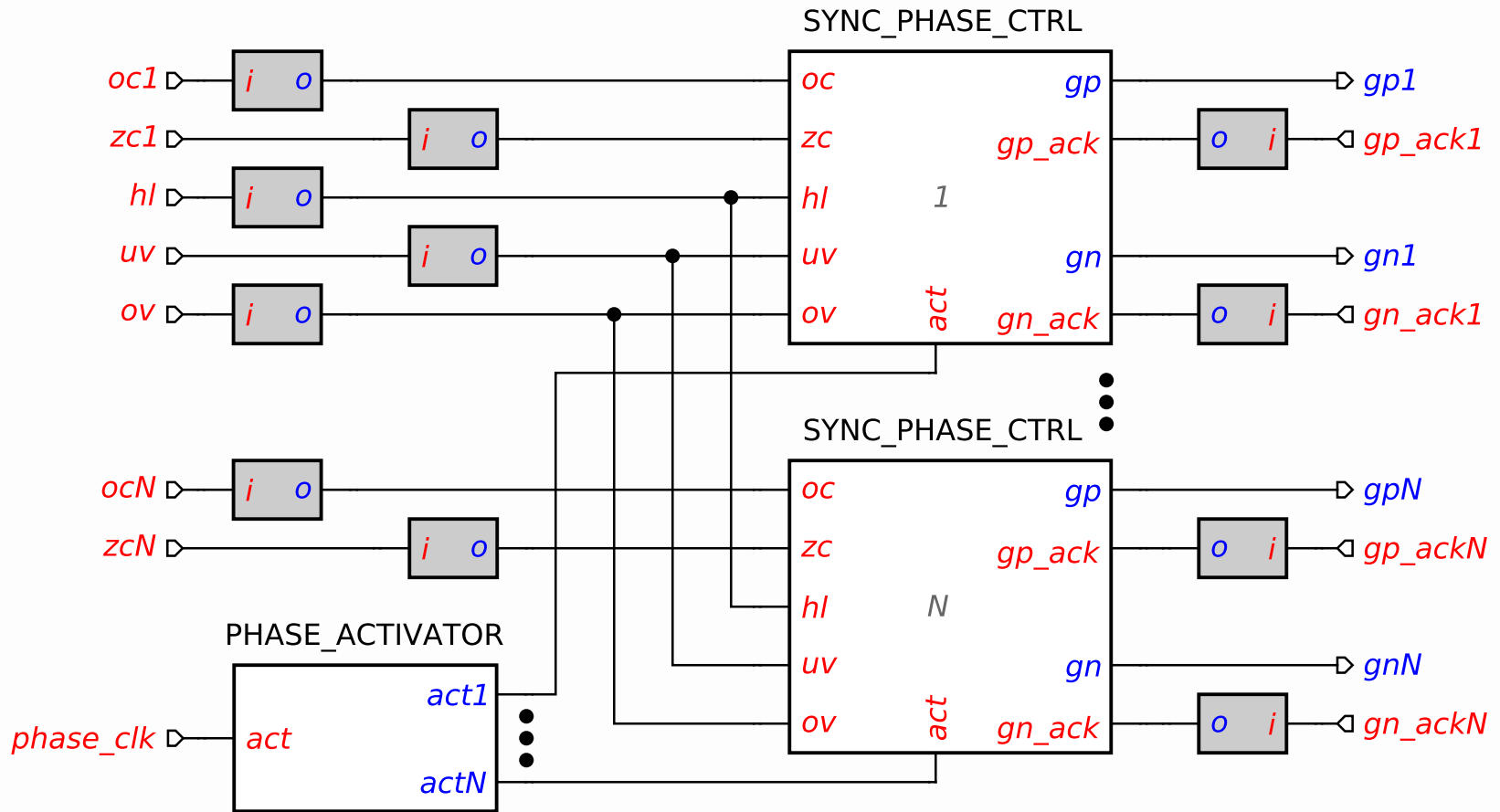


# Multiphase buck converter



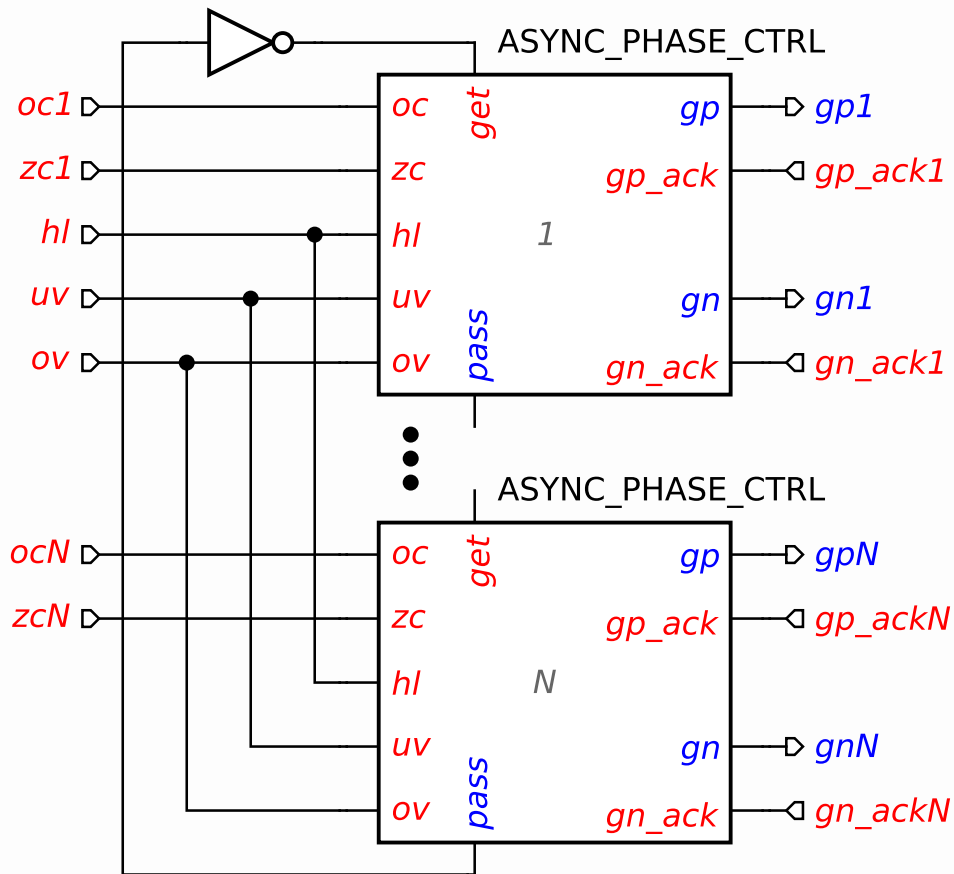
- Activation of phases
  - Sequential
  - May overlap
- More operating modes
  - High-load (HL)
  - Over-voltage (OV)
- Transistor min ON times
  - P<sub>MIN</sub> for PMOS
  - N<sub>MIN</sub> for NMOS
  - P<sub>MIN</sub>+P<sub>EXT</sub> for PMOS at first cycle

# Synchronous design



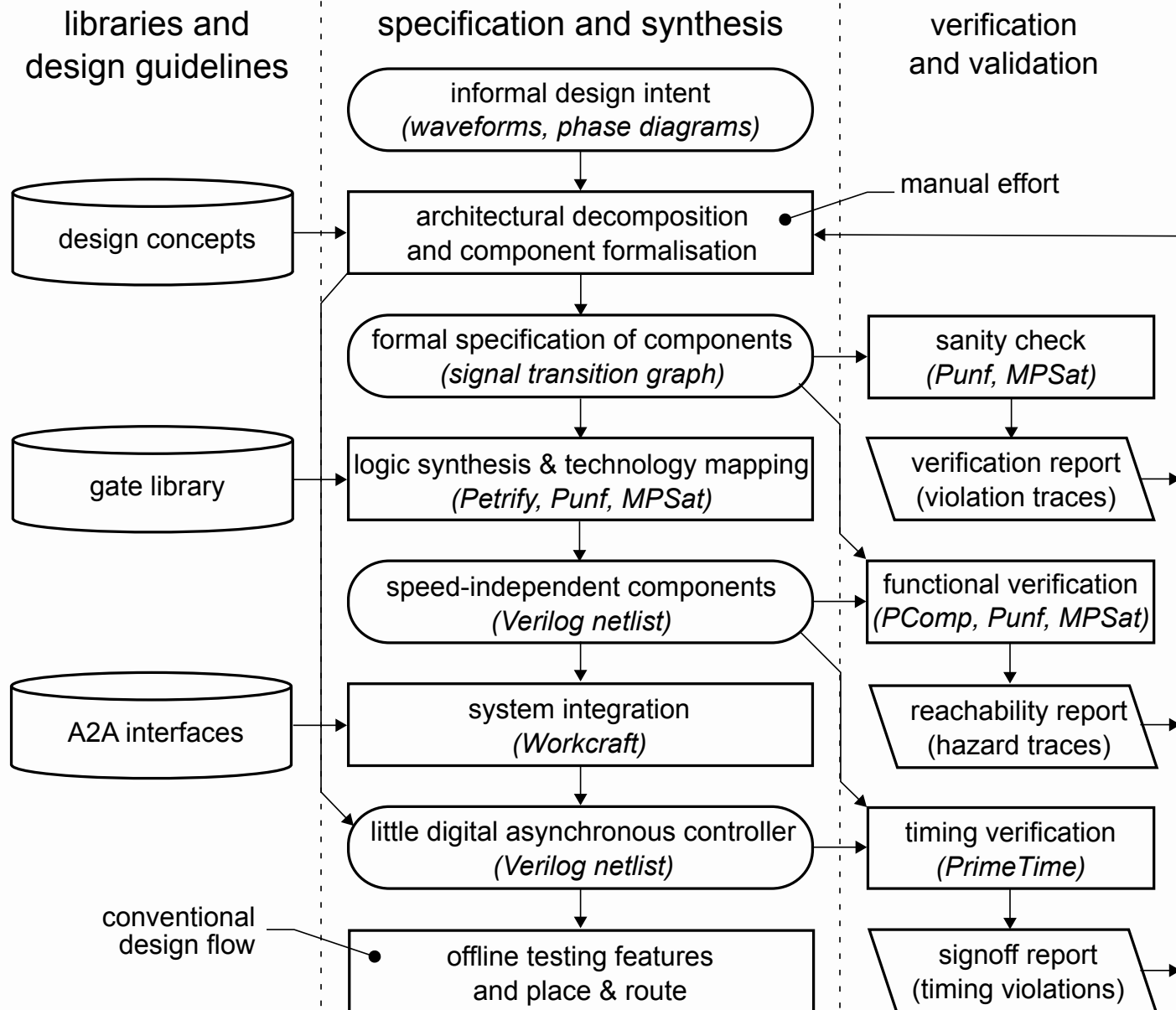
- Two clocks: phase activation (slow) and sampling (fast)
- Conventional RTL design flow for phase control
- Need for multiple synchronizers (grey boxes)

# Asynchronous design



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

# A4A design flow



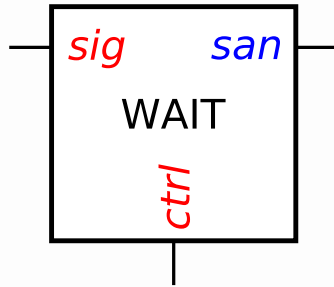
# A2A components

- 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

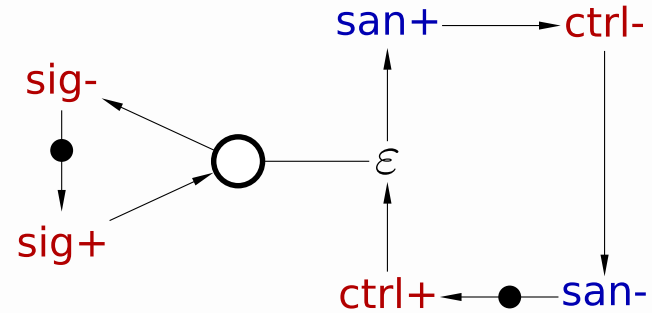


# WAIT element

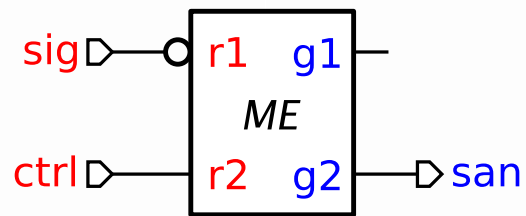
- Interface



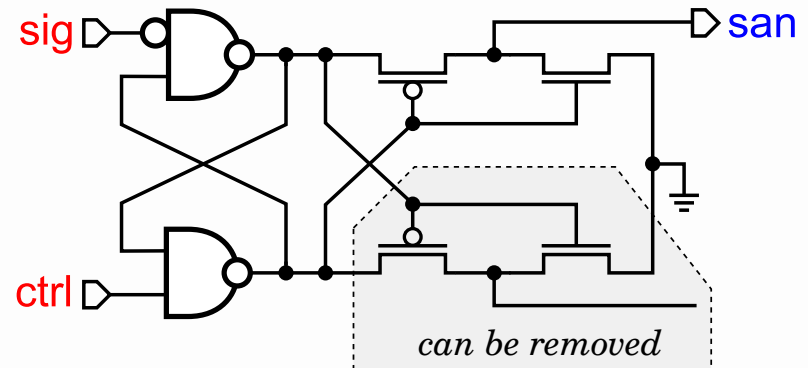
- STG specification



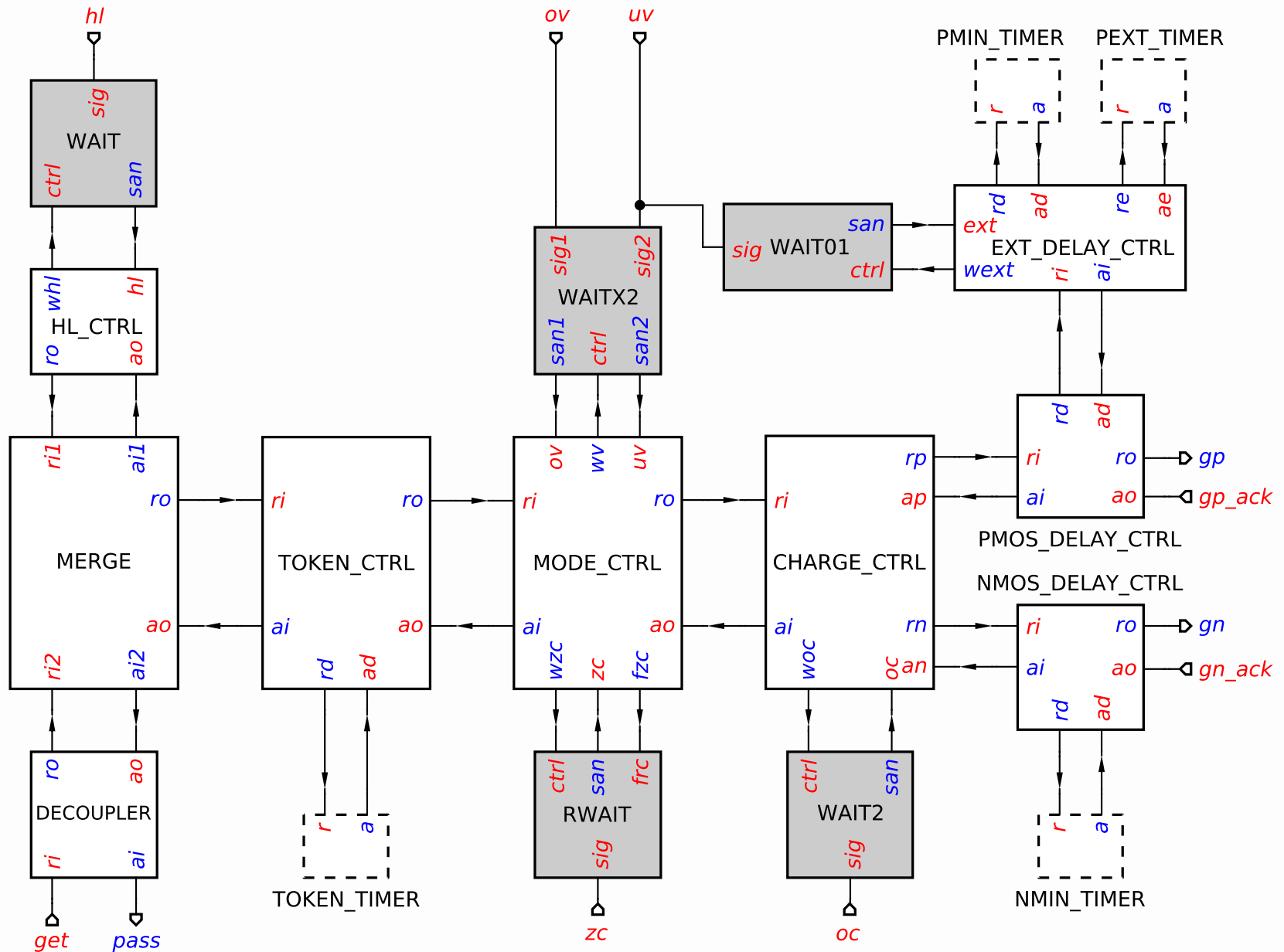
- ME-based solution



- Gate-level implementation

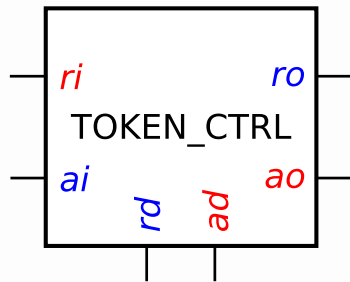


# Asynchronous phase control

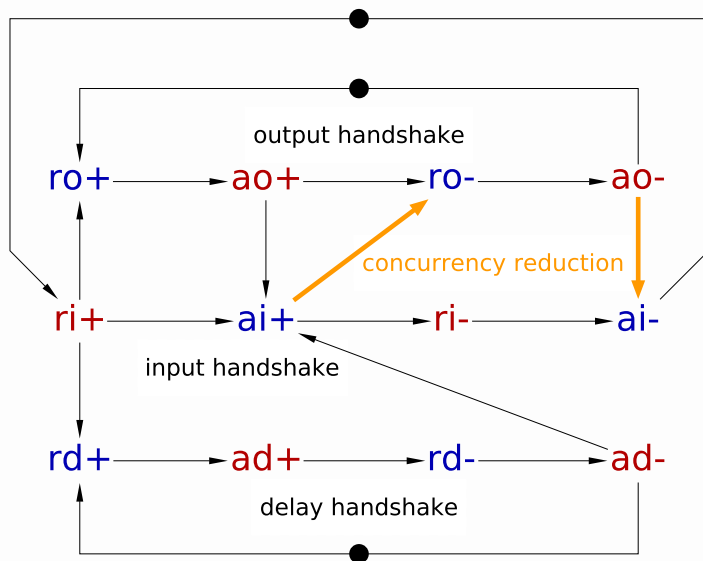


# Design of asynchronous components

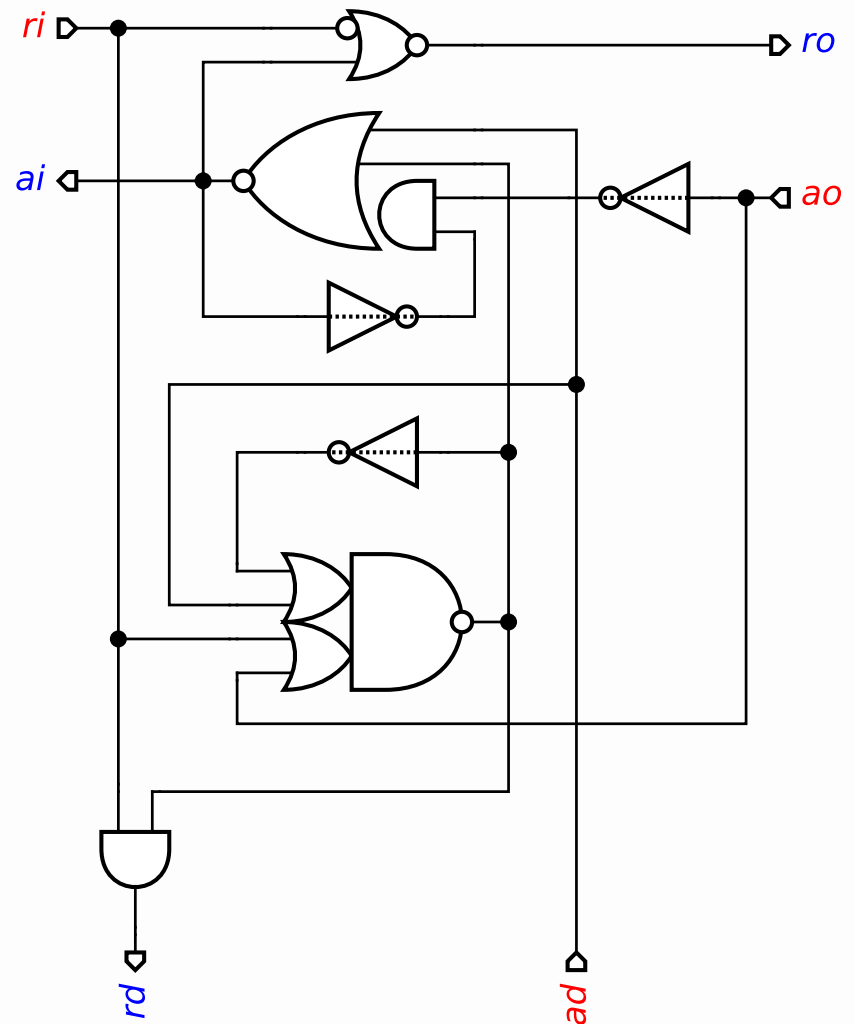
- Token control



- STG specification



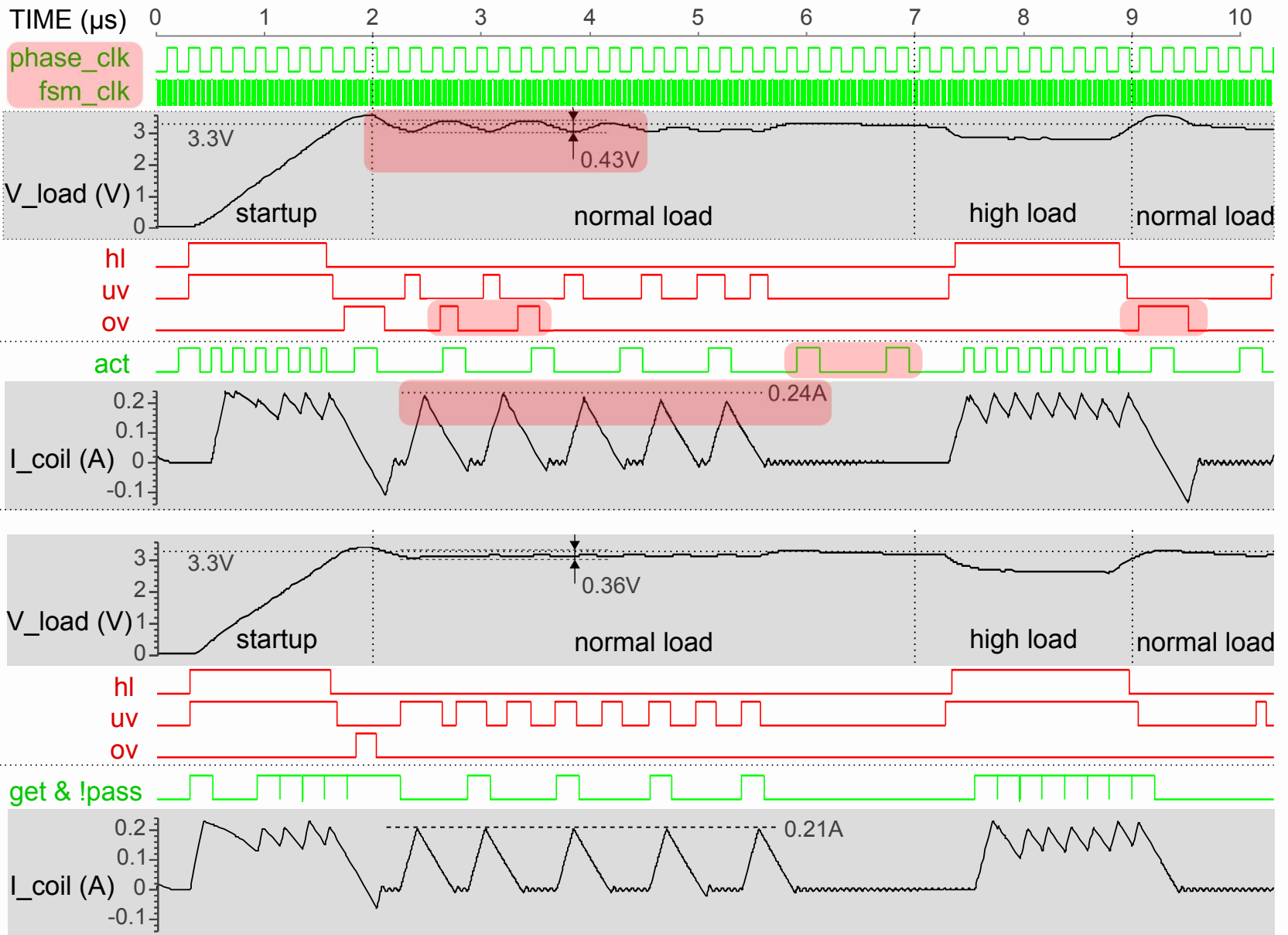
- Speed-independent implementation



# Simulation setup

- Verilog-A model of the 4-phase buck
- Control implemented in TSMC 90nm
- AMS simulation in CADENCE NC-VERILOG
- Synchronous design
  - Phase activation clock – 5MHz
  - Clocked FSM-based control – 100MHz, 333MHz, 666MHz, 1GHz
  - Sampling and synchronisation
- Asynchronous design
  - Phase activation – token ring with 200ns timer (= 5MHz)
  - Event-driven control (input-output mode)
  - Waiting rather than sampling (A2A components)

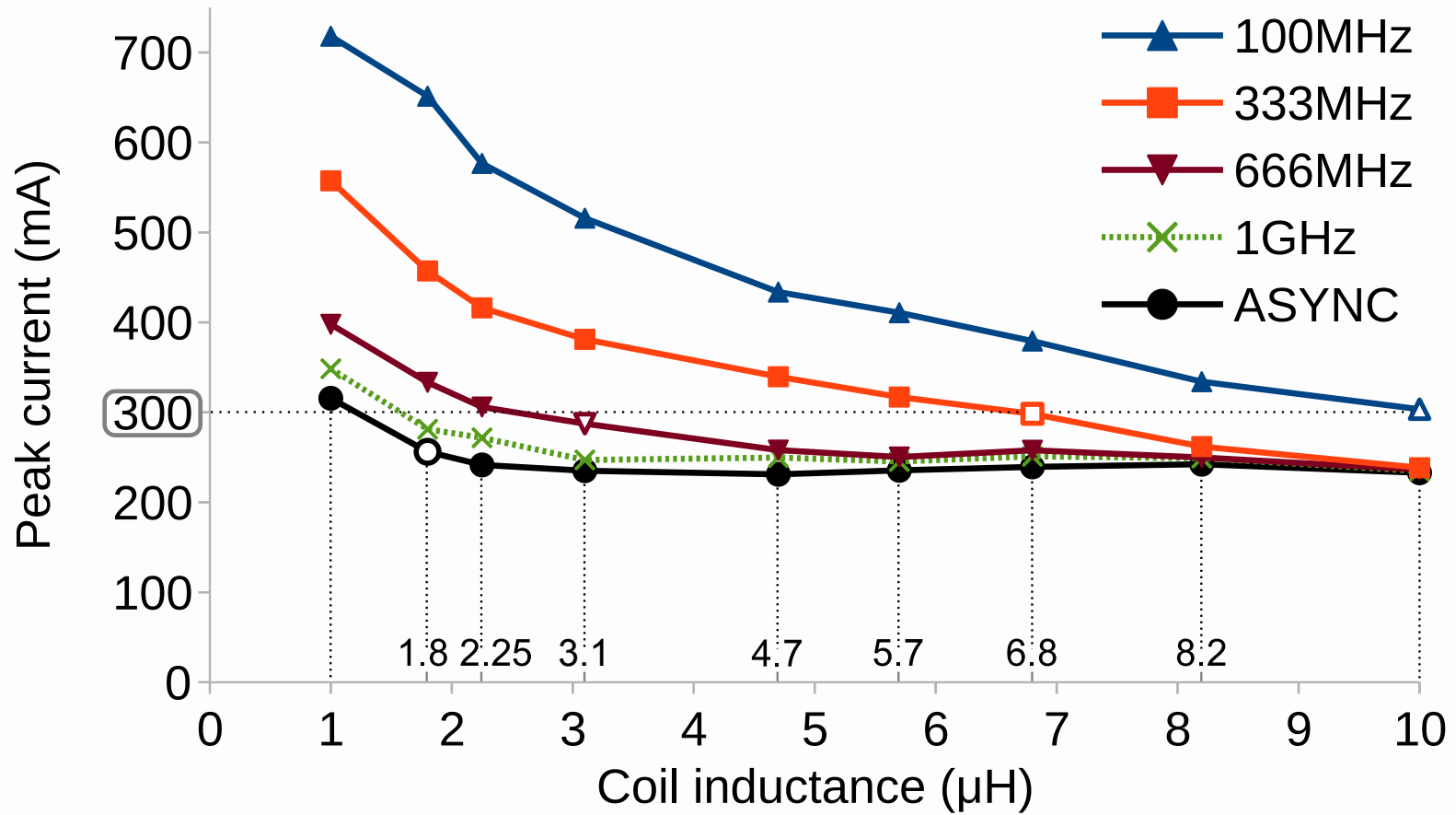
# Simulation waveforms



# Reaction time

Controller	HL (ns)	UV (ns)	OV (ns)	OC (ns)	ZC (ns)
100MHz	25.00	25.00	25.00	25.00	25.00
333MHz	7.50	7.50	7.50	7.50	7.50
666MHz	3.75	3.75	3.75	3.75	3.75
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

# Peak current



# Conclusions

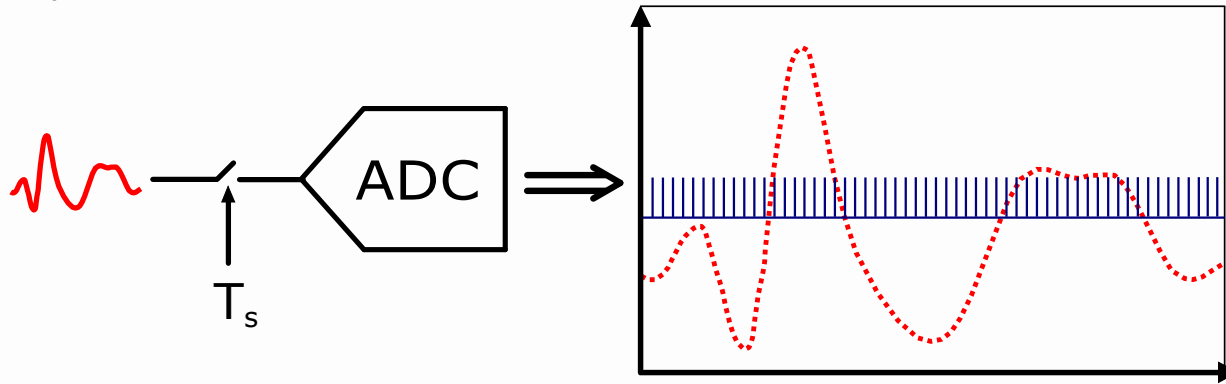
- Design flow is automated to large extent
  - Library of A2A components
  - Automatic logic synthesis
  - Formal verification at the STG and circuit levels
- Benefits of asynchronous multiphase buck controller
  - Reliable, no synchronisation failures
  - Quick response time (few gate delays)
  - Reaction time can be traded off for smaller coils
  - Lower voltage ripple and peak current



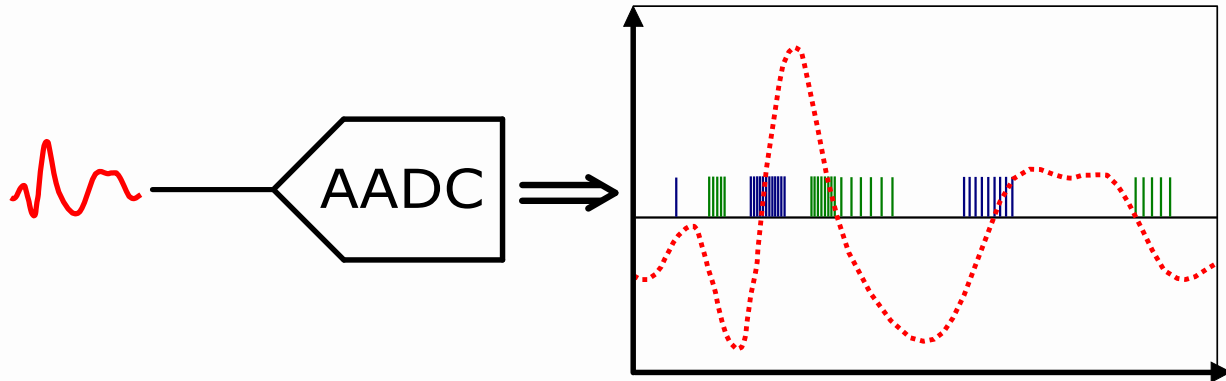
# Asynchronous ADC

# Sampling schemes

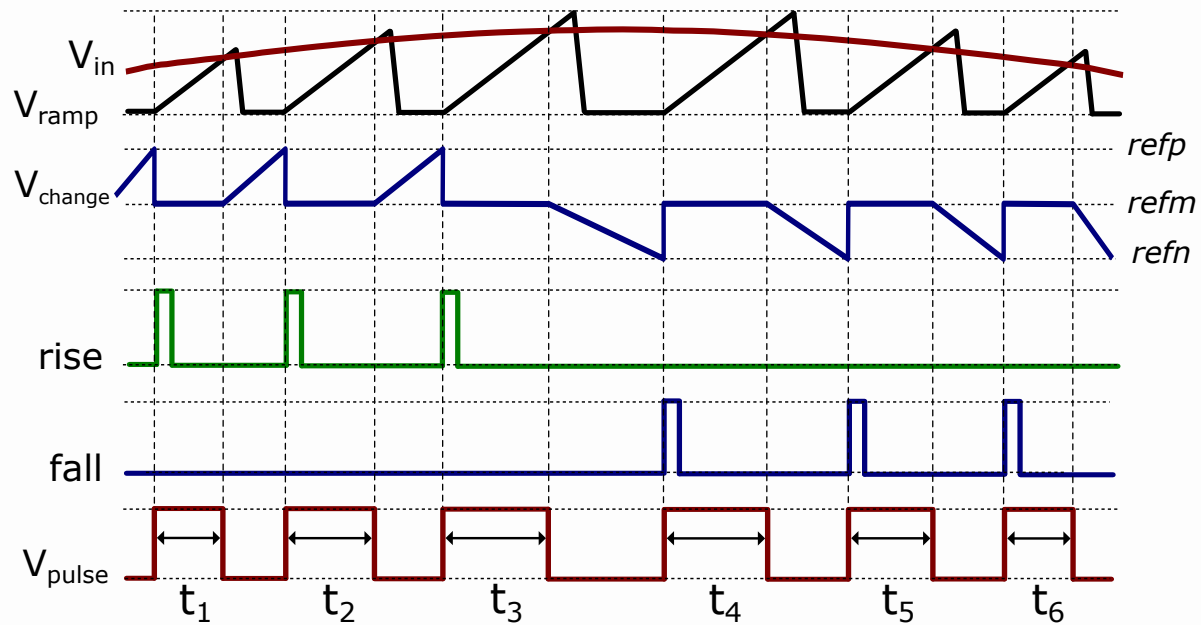
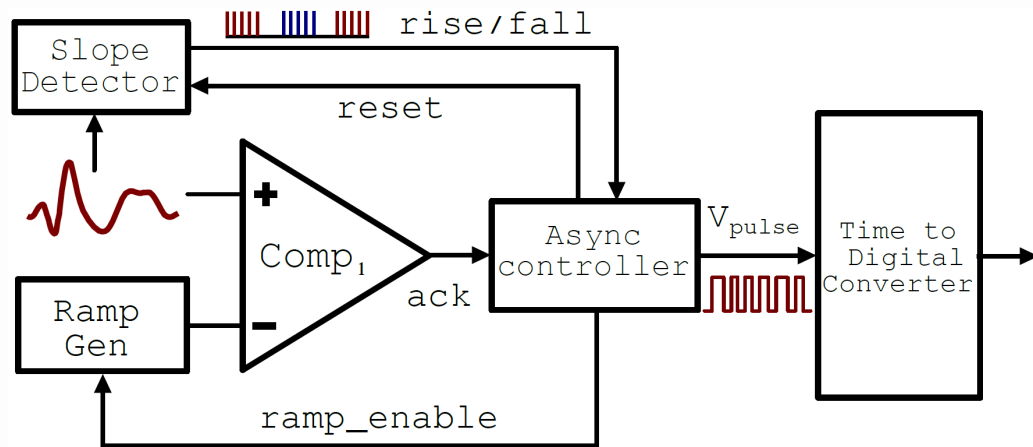
- Synchronous



- Asynchronous



# ADC design





Workcraft

<http://workcraft.org/>

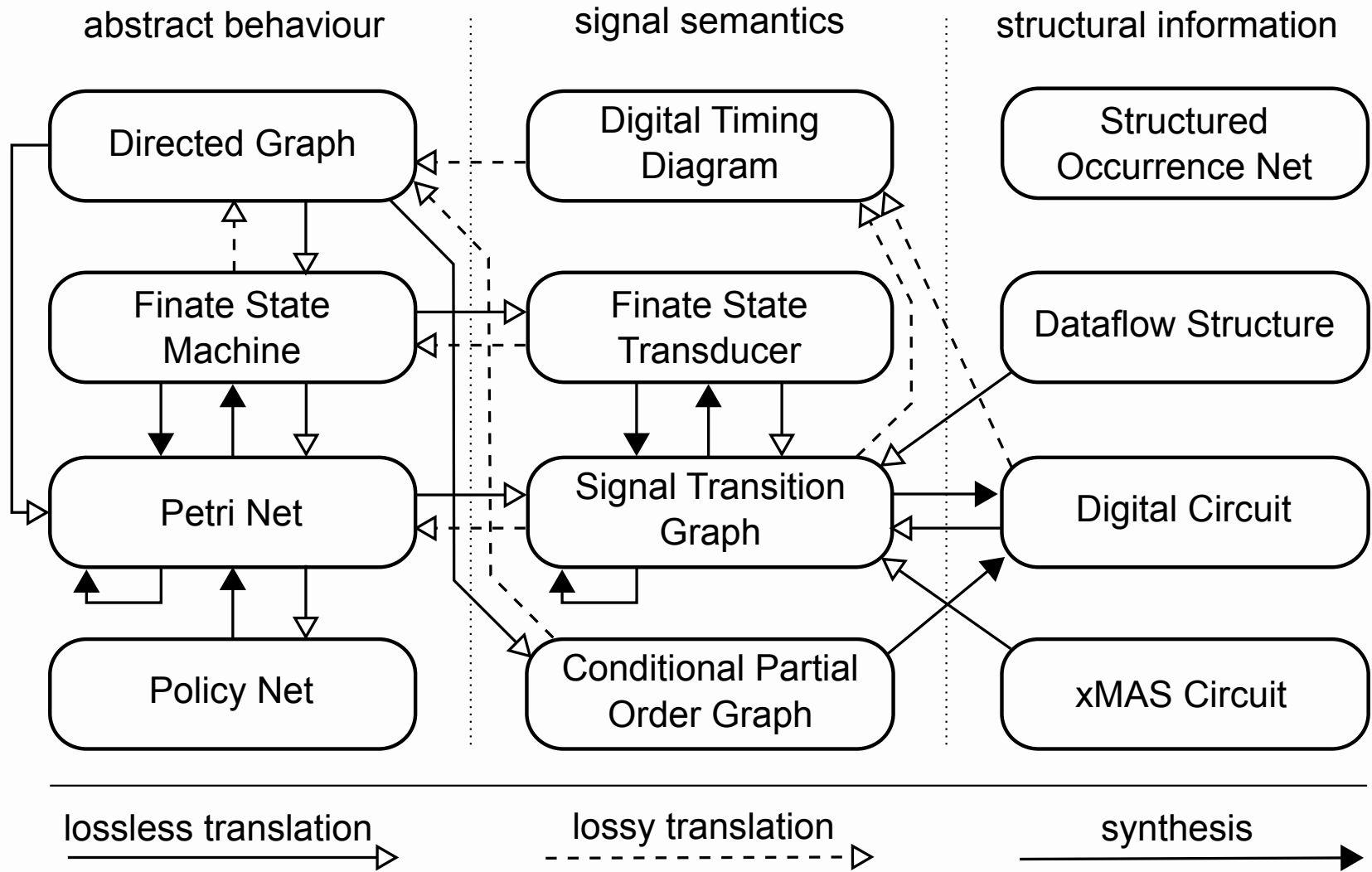
# What is WORKCRAFT?

- Framework for interpreted graph models
  - Interoperability between different abstraction levels
  - Consistency for users; convenience for developers
- Elaborate graphical user interface
  - Visual editing, analysis, and simulation
  - Easy access to common operations
  - Possibility to script specialised actions
- Interface to back-end tools for synthesis and verification
  - Reuse of established theory and tools (PETRIFY, MPSAT, PUNF)
  - Command log for debugging and scripting

# Why to use WORKCRAFT?

- Availability
  - Open-source front-end and plugins
  - Permissive freeware licenses for back-end tools
  - Frequent releases (4-6 per year)
  - Specialised tutorials and online training materials
- Extensibility
  - Plugins for new formalisms
  - Import, export and converter plugins
  - Interface to back-end tools
- Usability
  - Elaborated GUI developed with much user feedback
- Portability
  - Distributions for Windows, Linux, and OS X

# Supported graph models





# Supported features

Model	Supported features			
	Editing	Simulation	Verification	Synthesis
<b>abstract behaviour</b>				
Directed Graph	Yes	Yes	Yes	n/a
Finite State Machine	Yes	Yes	Yes	Yes <sup>1)</sup>
Petri Net	Yes	Yes	Yes	Yes <sup>2)</sup>
Policy Net	Yes	Yes	Yes	n/a
<b>signal semantics</b>				
Digital Timing Diagram	Yes	No	n/a	n/a
Finite State Transducer	Yes	Yes	Yes	Yes <sup>3)</sup>
Signal Transition Graph	Yes	Yes	Yes	Yes <sup>4)</sup>
Conditional Partial Order Graph	Yes	Some	No	Yes
<b>structural information</b>				
Structured Occurrence Net	Yes	Yes	Yes	n/a
Dataflow Structure	Yes	Yes	Yes	No
Digital Circuit	Yes	Yes	Yes	n/a
xMAS Circuit	Yes	Yes	Some	No

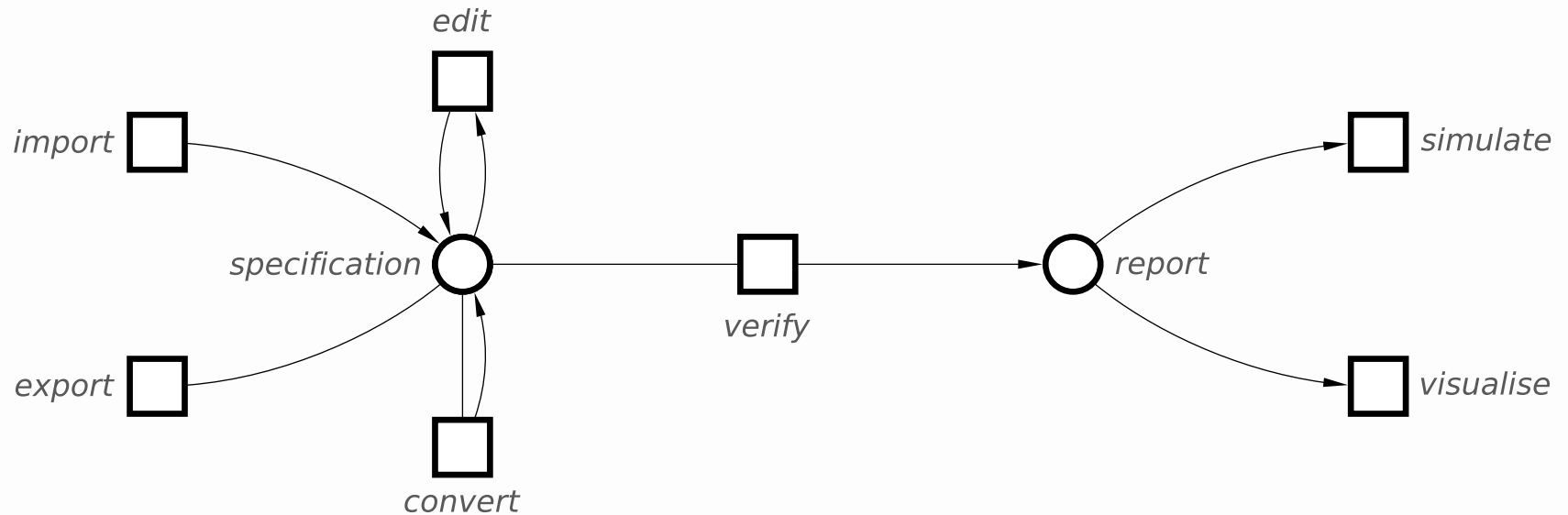
1) synthesis into Petri Net

2) re-synthesis into simpler Petri Net

3) synthesis into Signal Transition Graph

4) synthesis into Digital Circuit and re-synthesis into simpler Petri Net

# Design flow



- Import: ASTG, Verilog
- Export: ASTG, Verilog, SVG/Dot/PDF/EPS
- Convert: synthesis or translation
- Verify: reachability analysis (REACH predicates, SVA-like invariants)
- Visualise: CSC conflict cores, circuit initialisation, bottleneck

# Design flow: Asynchronous circuits

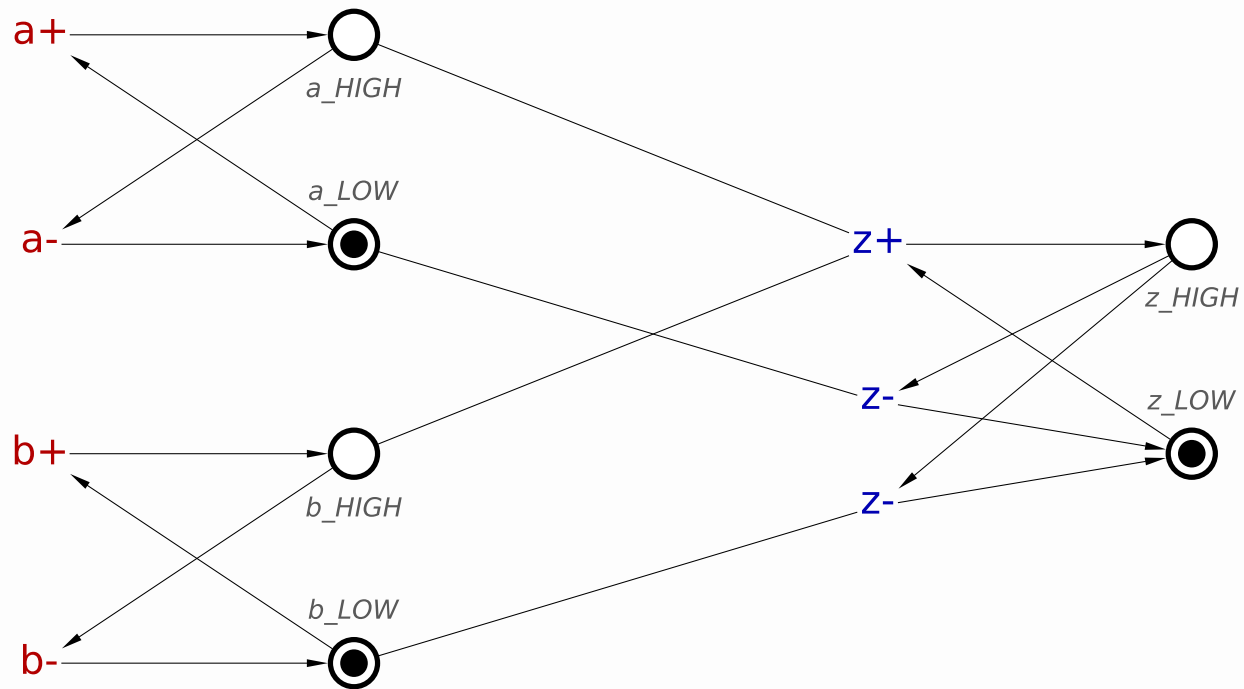
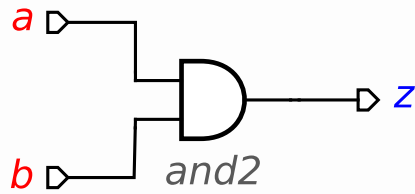
1. Specification of desired circuit behaviour with an STG model
2. Verification of the STG model
  - (a) Standard implementability properties:  
consistency, deadlock freeness, output persistency
  - (b) Design-specific custom properties
3. Resolution of complete state coding (CSC) conflicts
4. Circuit synthesis in one of the supported design styles
5. Manual tweaking and optimisation of the circuit
6. Verification of circuit against the initial specification
  - (a) Synthesis tools are complicated and may have bugs
  - (b) Manual editing is error-prone
7. Exporting the circuit as a Verilog netlist for conventional EDA backend

# What is hidden from the user?

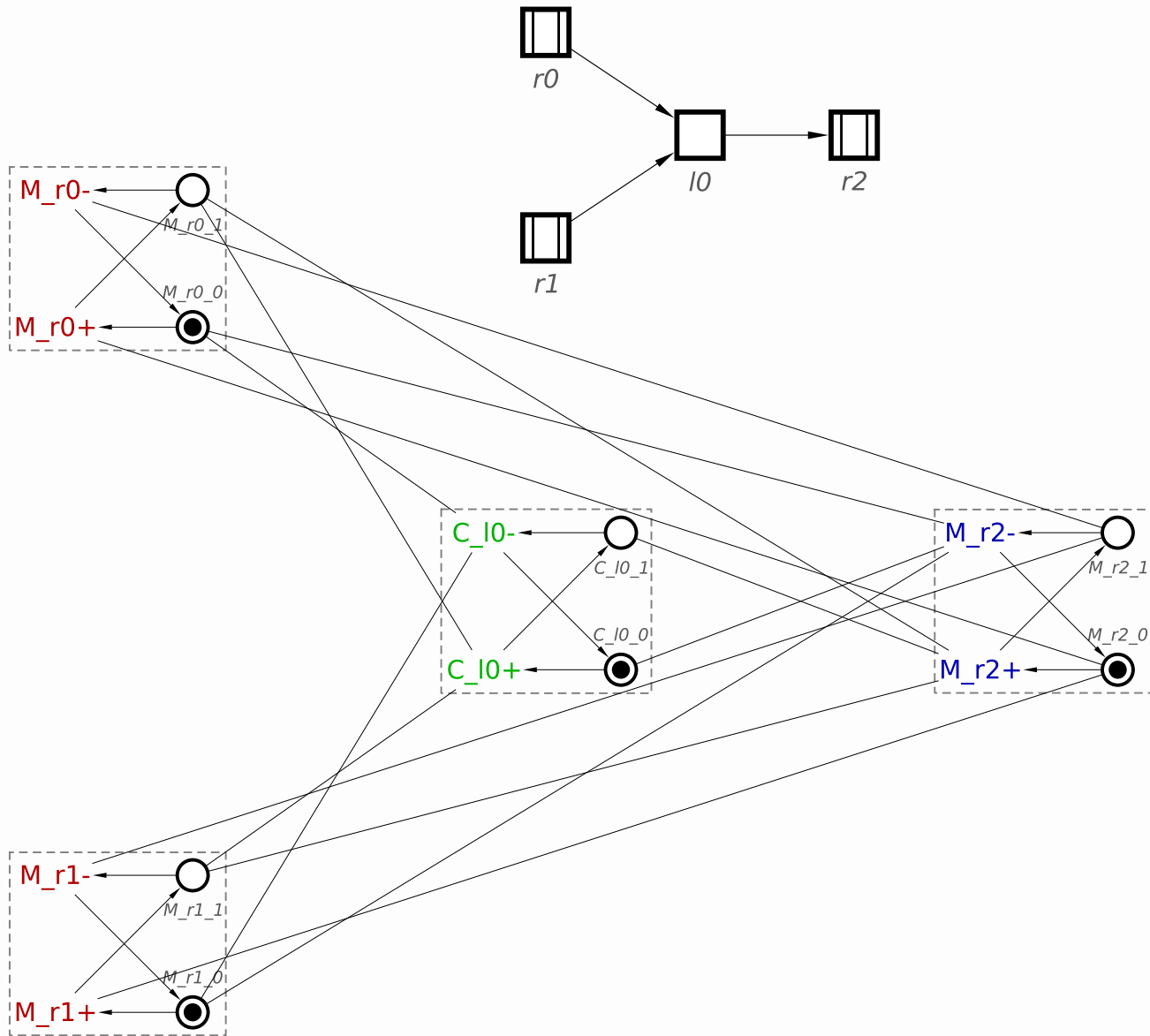
Verification that the circuit conforms to its specification

1. Circuit is converted to an equivalent STG – circuit STG
2. Internal signal transitions in the environment STG (contract between the circuit and its environment) are replaced by dummies
3. Circuit STG and environment STG are composed by PCOMP back-end
4. Conformation property is expressed in REACH language
5. Composed STG is unfolded by calling PUNF back-end
6. Unfolding prefix and REACH expression are passed to MPSAT back-end
7. Verification results are parsed by the front-end
8. Violation trace is projected to the circuit for simulation and debugging

# Circuit Petri nets as assembly language



# Circuit Petri nets: Dataflow pipelines

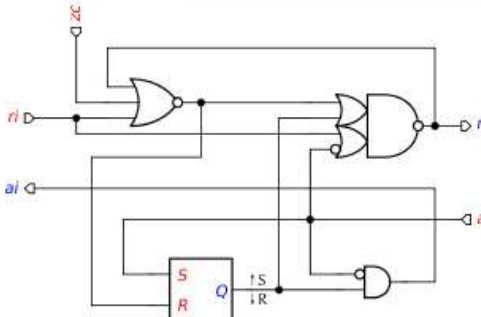


# WORKCRAFT live demo

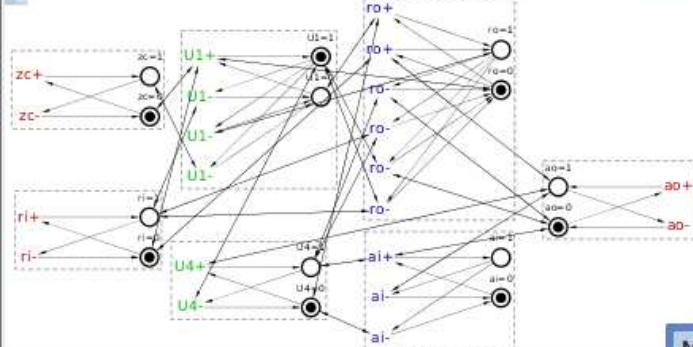
Workcraft

File Edit View Tools Help

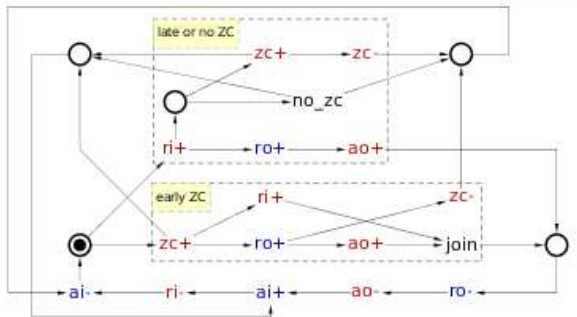
\*circuit-ZCH-map [circuit]



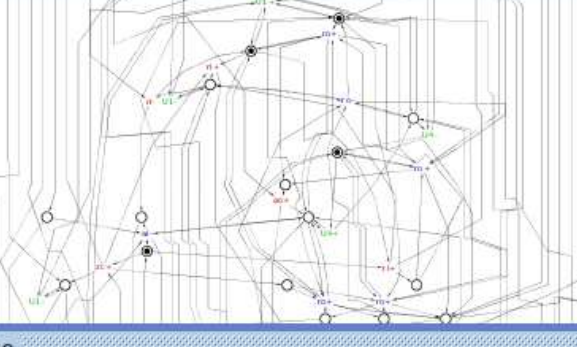
\*circuit-ZCH-map 1 [STG]



stg-ZCH [STG]



pcompressult6863112684546701504 [STG]



Property editor [model]

Environment... /.../stg-ZC... x

Tool controls

Editor tools

workspace

workspace

External

- circuit-ZCH-map 1.work
- circuit-ZCH-map.work \*
- stg-ZCH.work
- pcompressult68631126845

Output Problems Javascript Tasks

```
INORDER = ao r1 zc a1 ro csc0;  
OUTORDER = [a1] [ro] [csc0];  
[a1] = csc0 ao'; # gate and2_1:combinational  
[1] = r1' zc' ro'; # gate nor3:combinational  
#PRAGMA: zero delay  
[2] = ao'; # gate inv:combinational  
[ro] = [1]' csc0' + [2]' r1'; # gate oai22:combinational  
[csc0] = csc0 [1]' + ao; # gate sr_nor:async
```

# Set/reset pins: reset(ro)  
Exporting model "Untitled" to file "/tmp/workcraft-circuit-ZCH-map-8245652389417126911/dev.g".

Message

Under the given environment (stg-ZCH.work) the circuit is:

- \* conformant
- \* deadlock-free
- \* hazard-free

OK

