Modelling, Synthesis and Verification of Hardware

Alex Yakovlev, Victor Khomenko,
Andrey Mokhov and Danil Sokolov
Newcastle University, UK

async.org.uk

workcraft.org

PN/ACSD'15: Advanced Tutorial, Brussels, June 2015

Contents of lectures

Before lunch:

- Introduction: Petri nets and Hardware Design
- Hardware modelling with Petri nets
- Petri nets and Circuit Synthesis: Basics

- Asynchronous Circuit Verification
- Asynchronous Circuit Synthesis
- Design examples

After lunch:

- Practical exercises with tools Petrify and Workcraft

Bib references

- A.V. Yakovlev, A.M.Koelmans. Petri nets and digital hardware design, Lectures on Petri nets II: Applications, Advances in Petri Nets, LNCS vol. 1492, Springer 1998, pp. 154-236
- J. Cortadella, M. Kishinevsky, L. Lavagno and A. Yakovlev, Deriving Petri Nets from Finite Transition Systems, IEEE Transactions on Computers, Vol. 47, Number 8, pages 859-882, Aug. 1998.
- J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Logic Synthesis of Asynchronous Controllers and Interfaces, Springer, March 2002, ISBN3-540-43152-7
- Concurrency and Hardware Design , Advances in Petri nets, LNCS vol. 2549, Springer, 2002, ISBN 3-540-00199-9, 345pp.
- F. Xia, A. Mokhov, Y. Zhou, Y. Chen, I. Mitrani, D. Shang, D. Sokolov, and A. Yakovlev. Towards power-elastic systems through concurrency management, IET Computers and Digital Techniques (CDT), vol. 6, no. 1, pp. 33-42, January 2012.
- A. Mokhov, A. Yakovlev, Conditional Partial Order Graphs: Model, Synthesis, and Application, IEEE Transactions on Computers, vol. 59, no. 11, pp. 1480-1493, November 2010.
- D. Sokolov, I. Poliakov and A. Yakovlev, Analysis of Static Dataflow Structures, Fundamenta Informaticae, Vol. 88, No.4, pp. 581-610, IOS Press, 2008.
- I. Poliakov, A. Mokhov, A. Rafiev, D. Sokolov and A. Yakovlev. Automated Verification of Asynchronous Circuits Using Circuit Petri Nets, Proceedings of the 14th IEEE International Symposium on Asynchronous Circuits and Systems, Newcastle upon Tyne, UK, April 2008, pp. 161-170.
- V. Khomenko, M. Koutny and A. Yakovlev. Logic Synthesis for Asynchronous Circuits Based on STG Unfoldings and Incremental SAT, Fundamenta Informaticae, Volume 70, Issue 1-2, pp 49-73, IOS Press, 2006.
- D. Shang, F.Burns, A.Koelmans, A.Yakovlev, F. Xia. Asynchronous system synthesis based on direct mapping using VHDL and Petri nets, IEE Proceedings, Computers and Digital Techniques, Vol. 151, No.3, May 2004, pp. 209-220

Hardware and Petri Nets: Introduction

Alex Yakovlev, Victor Khomenko,
Andrey Mokhov and Danil Sokolov
Newcastle University, UK
async.org.uk
workcraft.org

PN/ACSD'15: Advanced Tutorial, Brussels, June 2015

Introduction. Outline

- **Role of Hardware in modern systems**
- **Role of Hardware design tools**
- **Role of a modeling language**
- **Why Petri nets are good for Hardware Design**
- **History of relationship between Hardware Design and Petri nets**
- **Asynchronous Circuit Design**

Role of Hardware in modern systems

- Technology allows putting up to several billion transistors on a chip; Multi-core Systems on Chip are now a reality – reaching up to 100 CPU cores
- E.g. IBM's z13 CPU chip (2015) – 22nm CMOS SOI, 4B transistors, 678 mm2, 5GHz, 17 metal layers; IBM's storage controller up to 7B
- E.g. Intel's Xeon Phi (2012) – 22nm, 5B transistors, 60 cores
- Hardware and software designs are no longer separate
- Hardware becomes distributed, asynchronous and concurrent
- Hardware requires power, consumes energy, radiates heat, electro-magnetic emission
- Hardware causes, and partly recovers from, faults and failures

Dr. Gordon E. Moore's Law

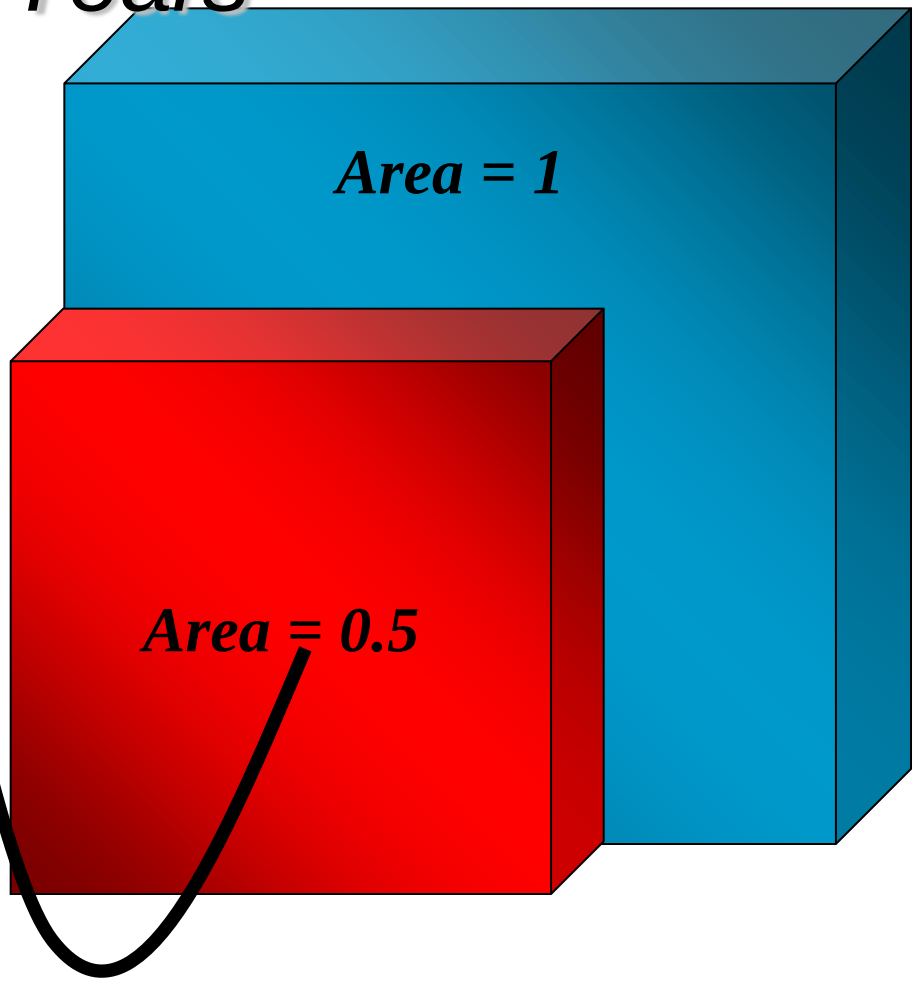
Integration's Capacity Doubles Every Two Years

Source:
Tom Williams,
Synopsys

$\sqrt{0.5} = \sim 0.7$

The Scaling Factor

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years." Gordon E. Moore, Electronic Magazine, April 19th, 1965



Cost of Scaling

Source: ITRS 2005/2006	90nm	65nm	45nm
Device Length (nm) ⇩	1X	0.7X	0.5X
Delay (ps) ⇩	1X	0.7X	0.5X
Frequency (GHz) ⇧	1X	**1.2X**	**1.45X**
Integration Capacity (BTx) ⇧	1X	2X	4X
Capacitance (fF) ⇩	1X	0.7X	0.5X
Die Size (mm^2) ⇨	1X	**1X**	**1X**
Voltage (V) ⇘	1X	**0.85X**	**0.75X**
Dynamic Power (W) ⇘	1X	**> 0.7X**	**> 0.5X**
Manufacturing (microcents/Tx) ⇩	1X	0.35X	0.12X

Source: Tom Williams, Synopsys

Costs of Scaling

Source: ITRS 2005	90nm	65nm	45nm
V_{TH} (V) ⇘	1X	**0.85X**	**0.75X**
I_{OFF} (nA/um) ⇧⇧	1X	**~3X**	**~9X**
Dynamic Power Density (W/cm²) ⇧	1X	**1.43X**	**2X**
Leakage Power Density (W/cm²) ⇧⇧	1X	**~2.5X**	**~6.5X**
Power Density (W/cm²) ⇧	1X	**~2X**	**~4X**
Cu Resistance (Ω) ⇧	1X	**2X**	**4X**
Interconnect RC Delay (ps) ⇧	1X	**~2X**	**~5X**
Packaging (cents/pin) ⇘	1X	**0.86X**	**0.73X**
Test (nanocents/Tx) ⇨	1X	**1X**	**1X**

To lower leakage by process one needs to give up on performance

Source: Tom Williams, Synopsys

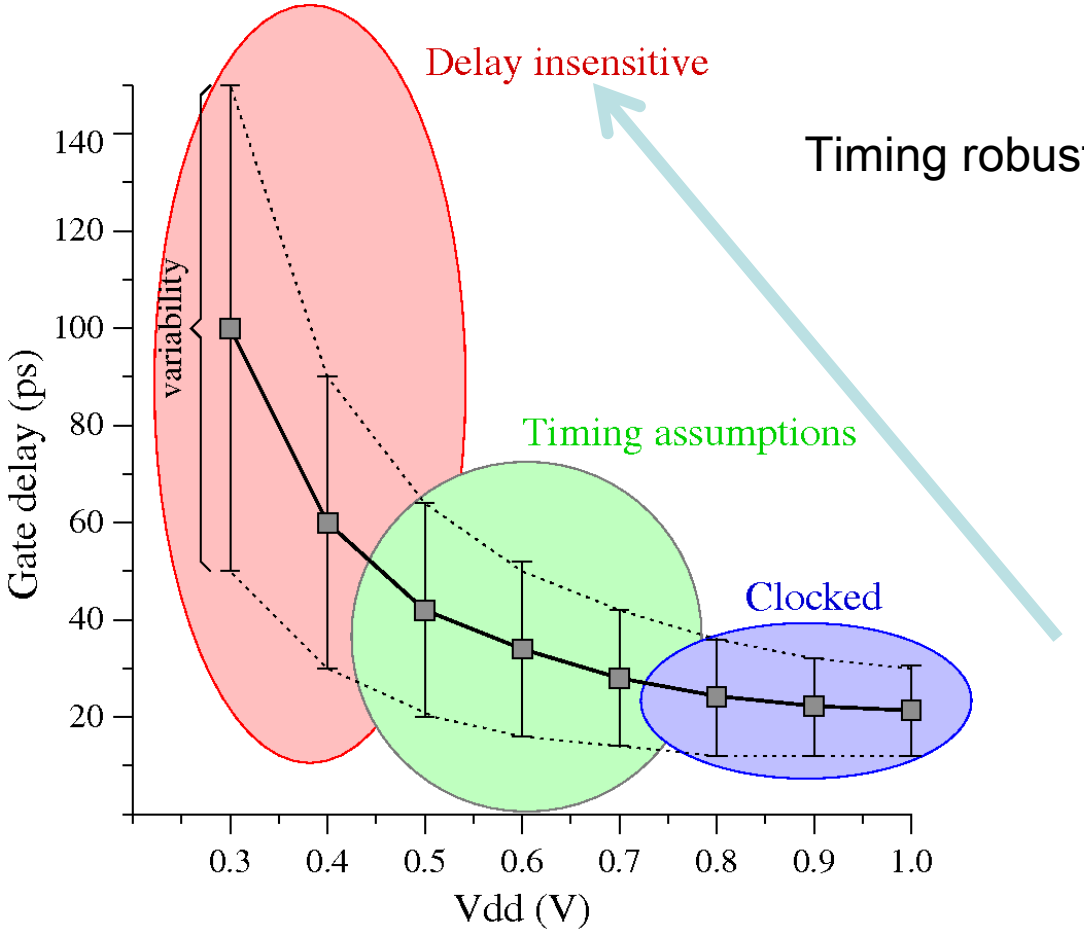
Growing Silicon Complexities

- Non-ideal device and supply/threshold voltage scaling:
  - Leakage,
  - Power management and delivery
- Non-ideal wire scaling:
  - Communication,
  - Synchronization
- High frequency coupling:
  - Noise,
  - Signal integrity,
  - Delay variation
- Process variation:
  - Characterization,
  - Error tolerance
- Lower reliability:
  - Insulator breakdown,
  - Electro-migration,
  - Single event upsets
- Manufacture handoff:
  - Time and money

Implications of Complexities

- Super-exponential increase in the complexity of the design process
- No chip-wide synchronization
  - Asynchronous design suggested as “challenge”
  - Using Globally Async and Locally Sync (GALS) is a way
- Statistical behavior of transistor / gate / cell (due to process variability)
- Some signals lost sometimes
  - Error-tolerant design

Relationship with uncertainty (e.g. timing variability)



Technology node:
90nm

Source of variability analysis:
Yu Cao, Clark, L.T., 2007

Role of Hardware design tools

- Design productivity is a problem due to chip complexity and time to market demands
- Need for well-integrated CAD with simulation, synthesis, verification and testing tools
- Modelling of system behaviour at all levels of abstraction with feedback to the designer
- Design re-use is a must but with max technology independence

Role of Modelling Language

- Design methods and tools require good modelling and specification techniques
- Those must be formal and rigorous and easy to comprehend (cf. timing diagrams, waveforms, traditionally used by logic designers)
- Today's hardware description languages allow high level of abstraction, but they are often not capable to expose the low level (behavioural) details in an adequate form
- Models must allow for equivalence-preserving refinements, decomposition, analysis and synthesis
- They must allow for non-functional qualities such as speed, size and power

Why Petri nets are good for hardware design

- simple and easy to understand graphical capture
- modelling power adjustable to various types of behaviour at different abstraction levels
- formal operational semantics and verification of correctness (safety and liveness) properties
- possibility of mechanical synthesis of circuits from various behavioural models, such as nets, transition systems, trace characterisations
- possibility of synthesis of specifications and visualisation of circuit behaviour using THEORY OF REGIONS
- Introducing extra aspects such as step semantics and policies into synthesis helps to address aspects of Globally Asynchronous Locally Synchronous (GALS) and power management

We see Petri nets more and more as a unifying modelling language for reasoning about the behaviour of digital circuits and systems, where various application-specific and engineering-specific modelling notations can be used as front-end notations.

A bit of history of their relationship

- 1950's and 60's: Foundations (Muller & Bartky, Petri, Karp & Miller, …)
- 1970's: Toward Parellel Computations (MIT, Toulouse, St. Petersburg, Manchester …)
- 1980's: First progress in VLSI and CAD, Concurrency theory, Signal Transition Graphs (STGs)
- 1990's: First asynchronous design (verification and synthesis) tools: SIS, Forcage, Petrify
- 2000's: Powerful asynchronous design flow (incl. hardware-software co-design and system-on-chip design): Balsa, Haste, Elastic Clocks
- 2010's: Design flows for hybrid sync-async systems with Petri nets as internal representation: Workcraft, Tools for GALS …

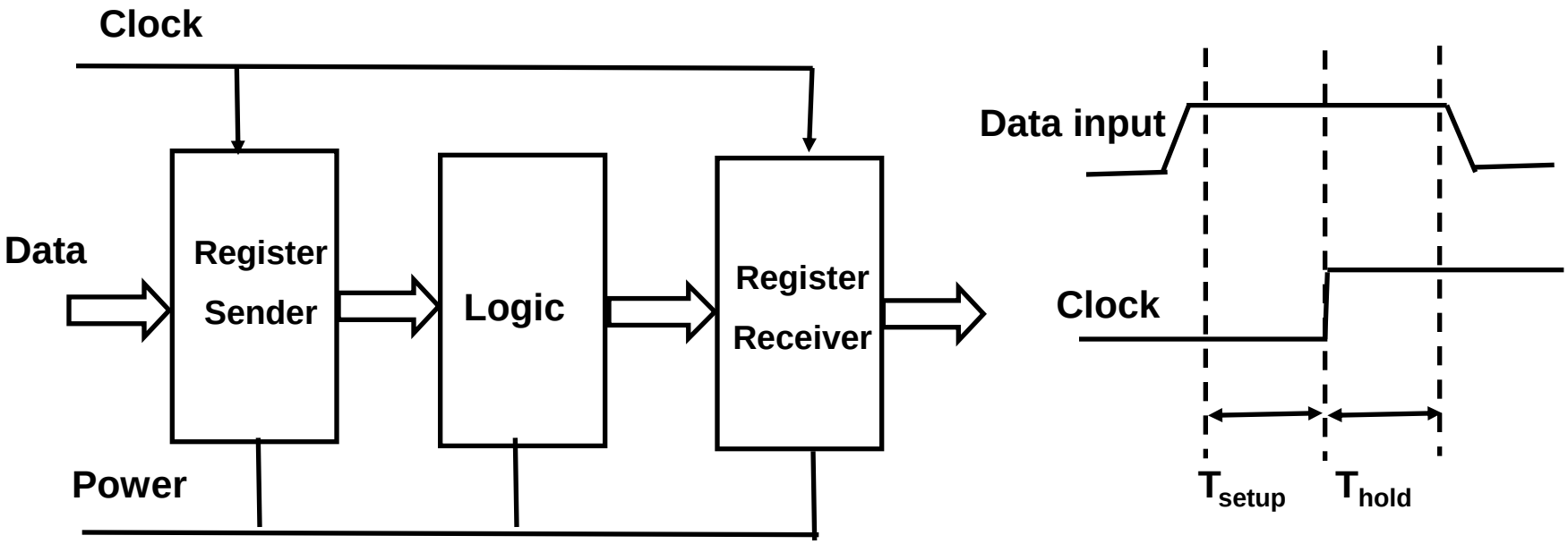
Introduction to Asynchronous Circuits

- What is an asynchronous circuit?
  - Physical (analogue) level
  - Logical level
  - Speed-independent and delay-insensitive circuits
- Why go asynchronous?
- Why control logic?
- Role of Petri nets
- Asynchronous circuit design based on Petri nets

What is an asynchronous circuit

- No global clock; circuits are self-timed or self-clocked
- Can be viewed as hardwired versions of parallel and distributed programs – statements are activated when their guards are true
- No special run-time mechanism – the “program statements” are physical components: logic gates, memory latches, or hierarchical modules
- Interconnections are also physical components: wires, busses

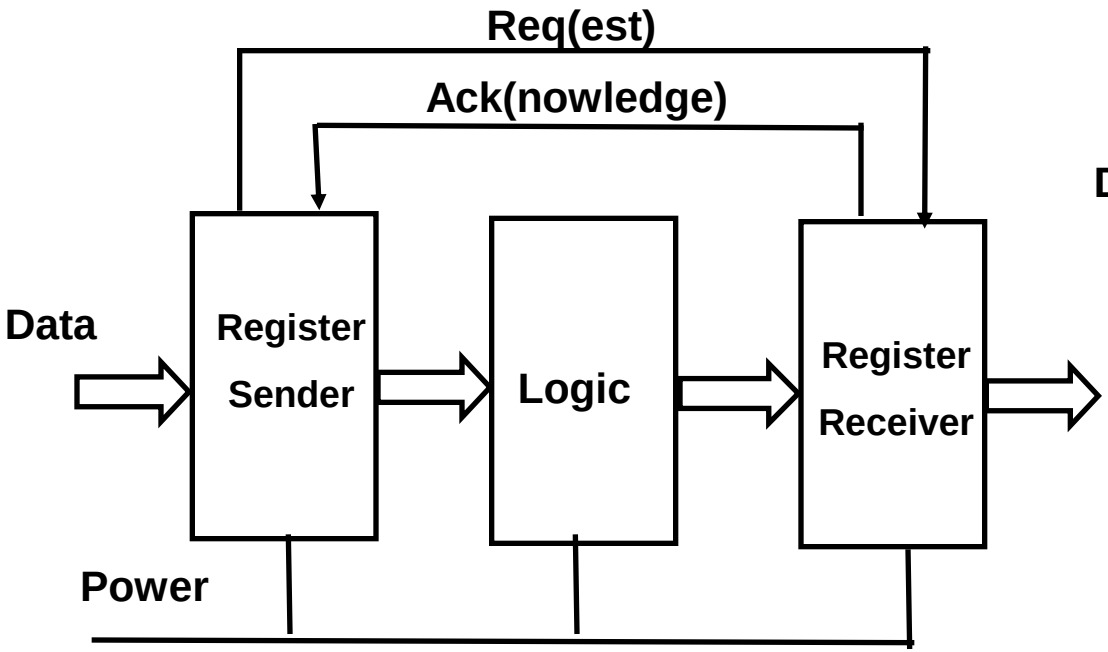
Synchronous Design

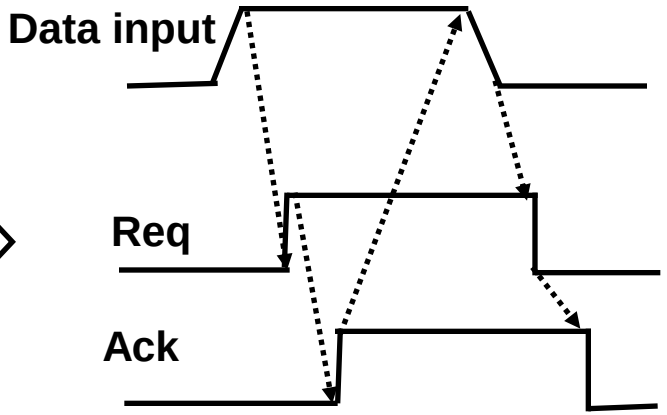


Timing constraint: input data must stay unchanged within a setup/hold window around clock event. Otherwise, the latch may fail (e.g. metastability)

Power must be stable, say at 1Volt to avoid variation of delays

Asynchronous Design





Req/Ack (local) signal handshake protocol instead of global clock

Causal relationship

Handshake signals implemented with completion detection in data path

Power may fluctuate, e.g. between 0.5V and 1.5V

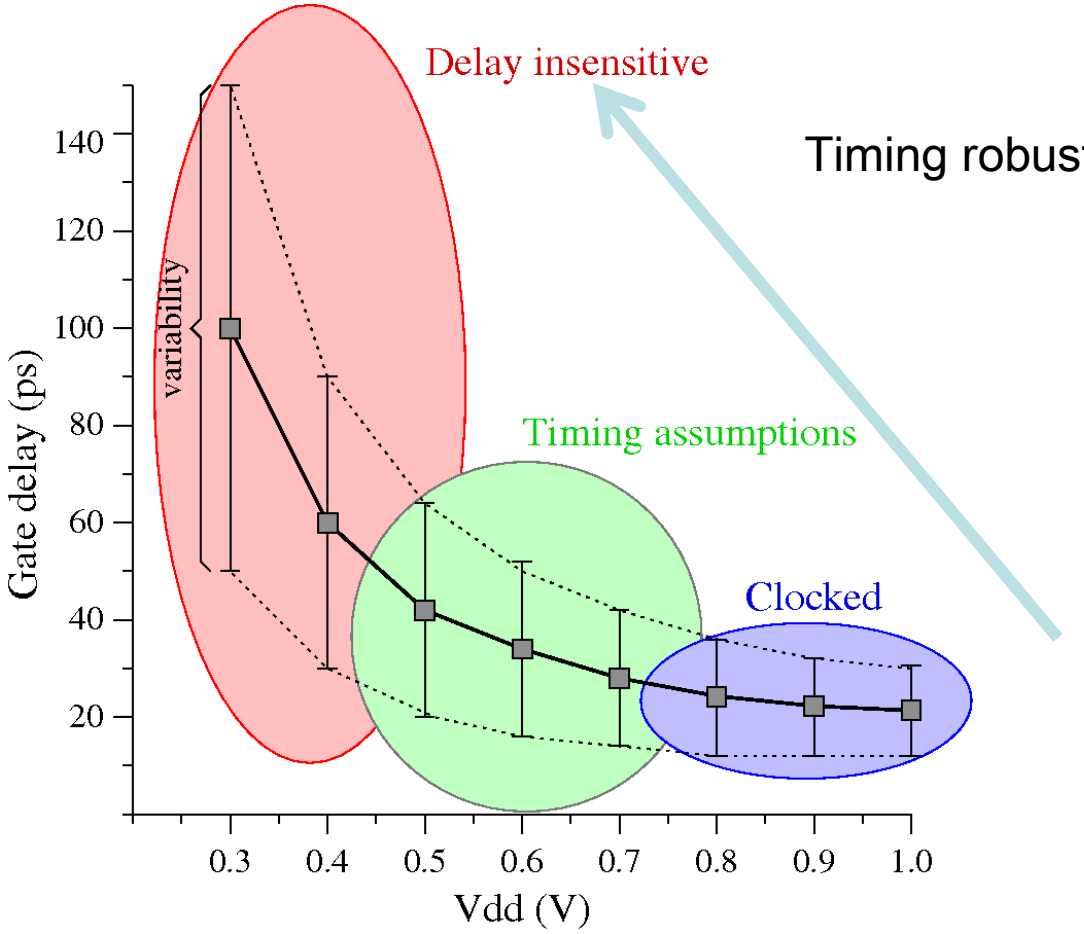
Physical (Analogue) level

- Strict view: an asynchronous circuit is a (analogue) dynamical system – e.g. to be described by differential equations
- In most cases can be safely approximated by logic level (0-to-1 and 1-to-0 transitions) abstraction; even hazards can be captured
- For some anomalous effects, such as metastability and oscillations, absolute need for analogue models
- Analogue aspects are not considered in this tutorial

Logical Level

- Circuit behaviour is described by sequences of up (0-to-1) and down (1-to-0) transitions on inputs and outputs
- The order of transitions is defined by causal relationship, not by clock (*a causes b,* directly or transitively)
- The order is partial if concurrency is present
- Two prominent classes of async circuits: speed-independent (work for any gate delay variations) and delay-insensitive (for both gate and wire delays)
- A class of async timed (not clocked!) circuits allows special timing order relations (*a occurs before b,* due to delay assumptions*)*

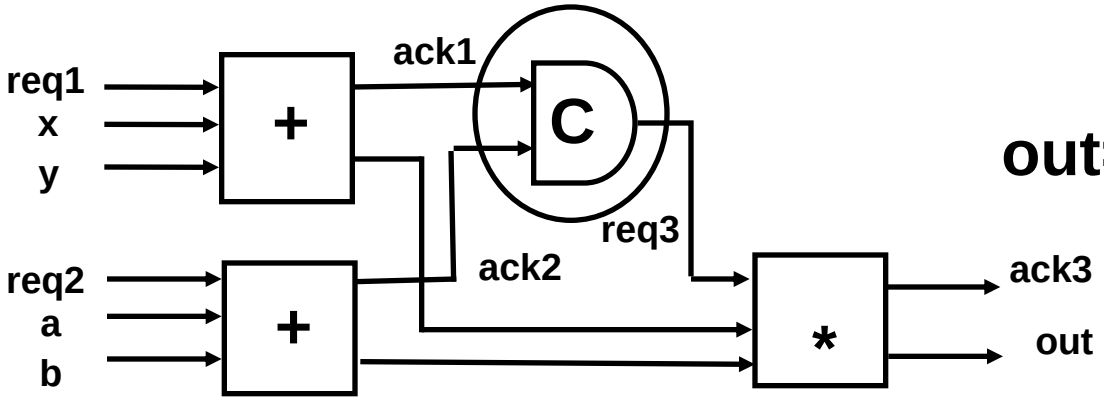
Relationship with uncertainty (e.g. timing variability)



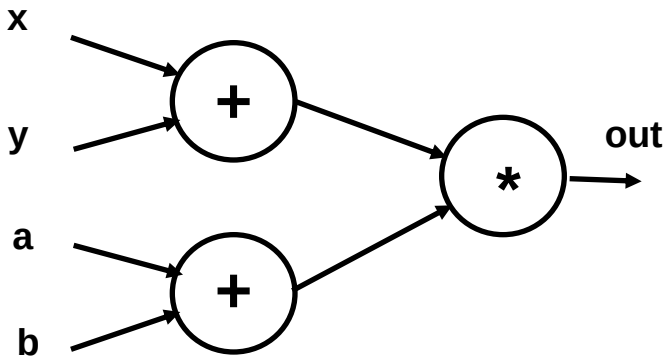
Technology node:
90nm

Source of variability analysis:
Yu Cao, Clark, L.T., 2007

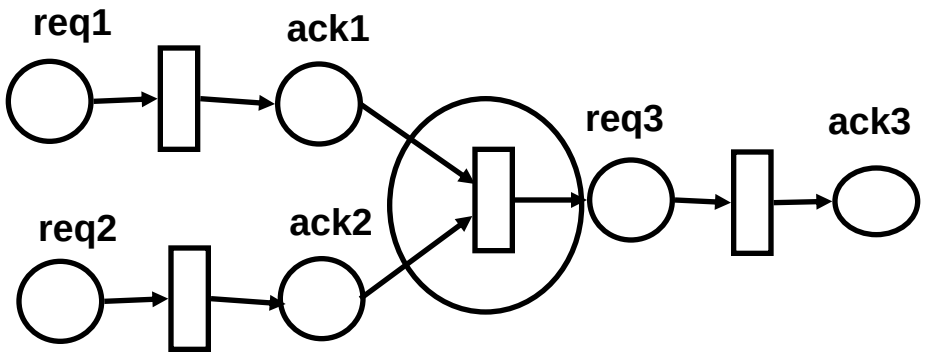
Simple circuit example



out=(x+y)*(a+b)



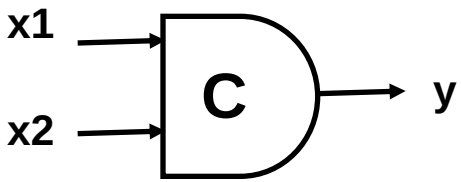
Data flow graph



Control flow graph – Petri net

Muller C-element

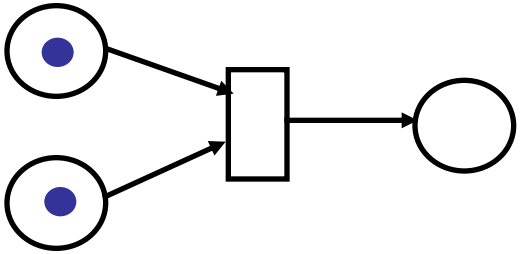
Key component in asynchronous circuit design – like a Petri net transition



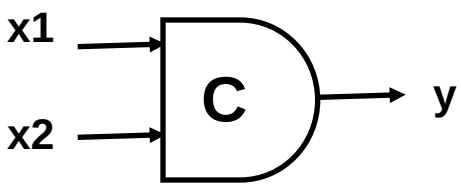
$$y = x1*x2 + (x1+x2)y$$

Set-part Reset-part

It acts symmetrically for pairs of 0-1 and 1-0 transitions – waits for both input events to occur

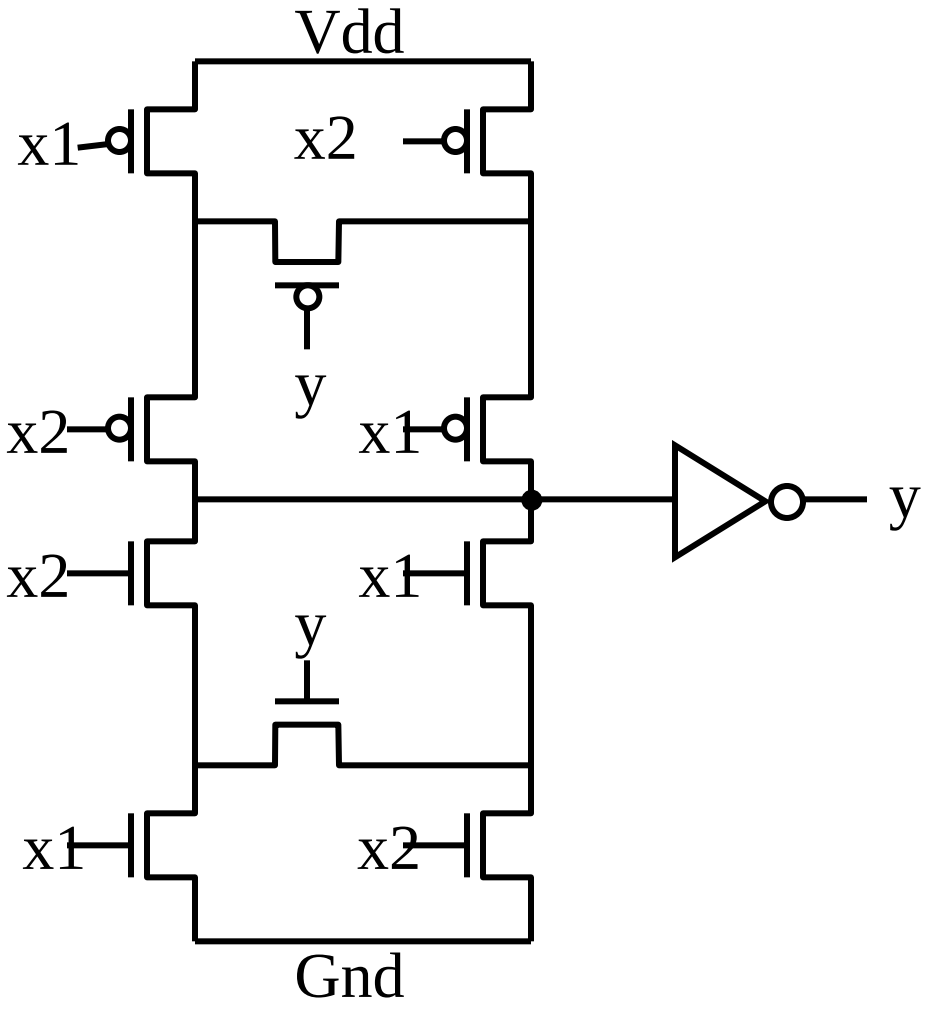


Muller C-element (in CMOS)



y=x1*x2+(x1+x2)y

Set-part **Reset-part**

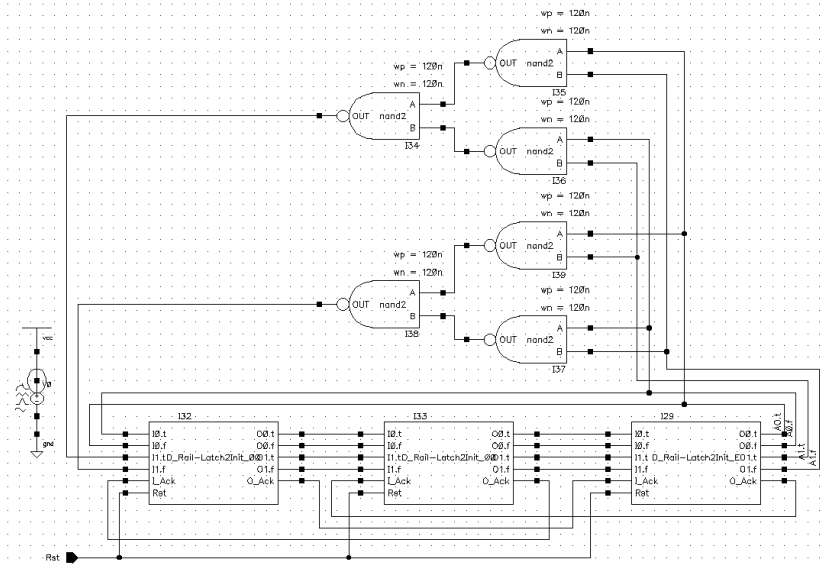


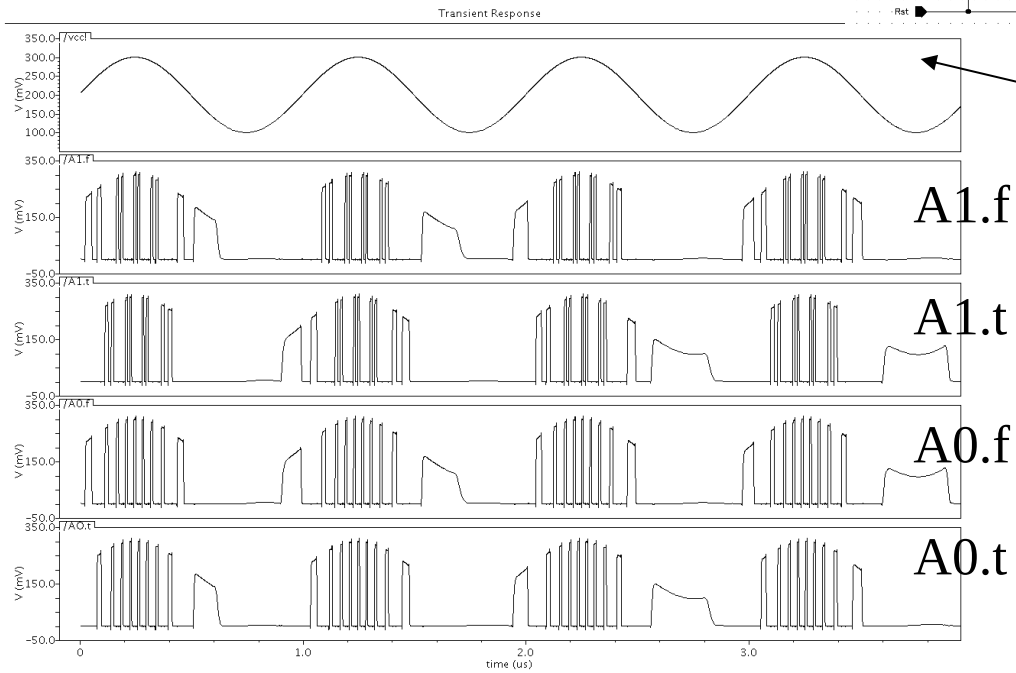
Why asynchronous is good

- Performance (work on actual, not max delays)
- Robustness (operationally scalable; no clock distribution; important when gate-to-wire delay ratio changes)
- Low Power (‘change-based’ computing – fewer signal transitions) and Power-Proportional (activity level is as much/little as power invested), opportunities for energy-harvesting systems such as wireless sensor nodes
- Low Electromagnetic Emission (more even power/frequency spectrum)
- Modularity and re-use (parts designed independently; well-defined interfaces)
- Testability (inherent self-checking via ack signals)

Async logic can work from AC supply!

2-bit Sequential Dual-rail Asynchronous Counter





A1.f

A1.t

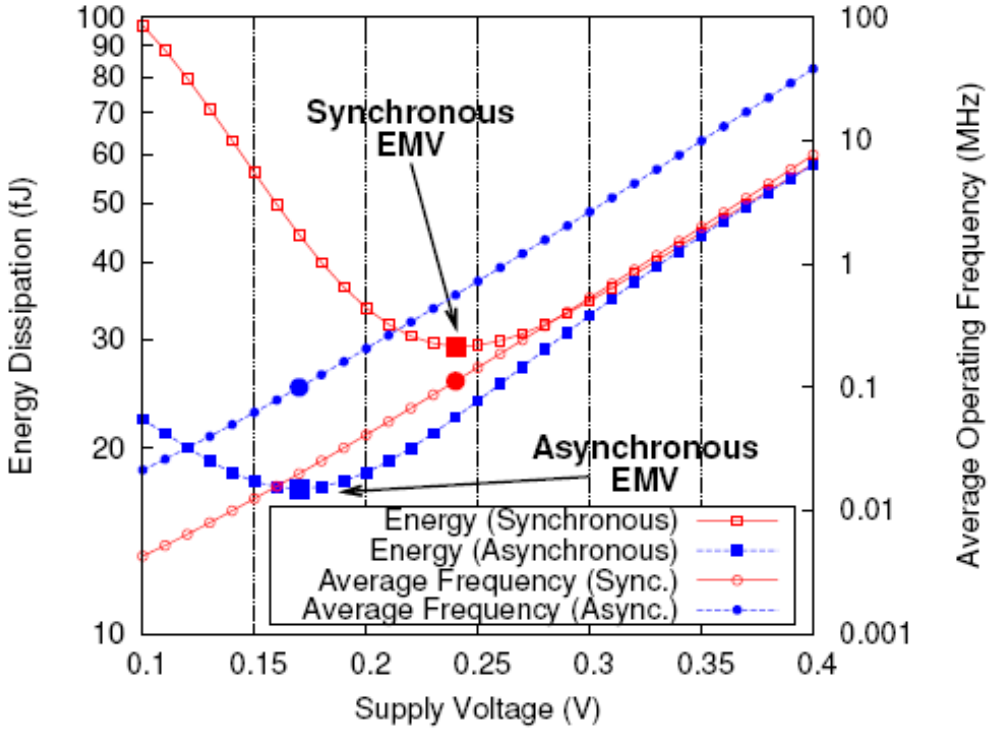
A0.f

A0.t

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

Self-timed logic with completion detection is robust to power supply variations

Sync vs Async Design (in terms of energy efficiency)



Asynchronous (self-timed) logic can provide completion detection and thus reduce the interval of leakage to minimum, thereby doing nothing well!

Source: Akgun et al, ASYNC'10

Role of Petri Nets

- We concentrate here on control logic
- Control logic is behaviourally more diverse than data path
- Petri nets capture causality and concurrency between signalling events, deterministic and non-deterministic choice in the circuit and its environment
- They allow:
  - composition of labelled PNs (transition or place sync/tion)
  - refinement of event annotation (from abstract operations down to signal transitions)
  - use of observational equivalence (lambda-events)
  - clear link with state-transition models in both directions

Hardware and Petri Nets: Modelling

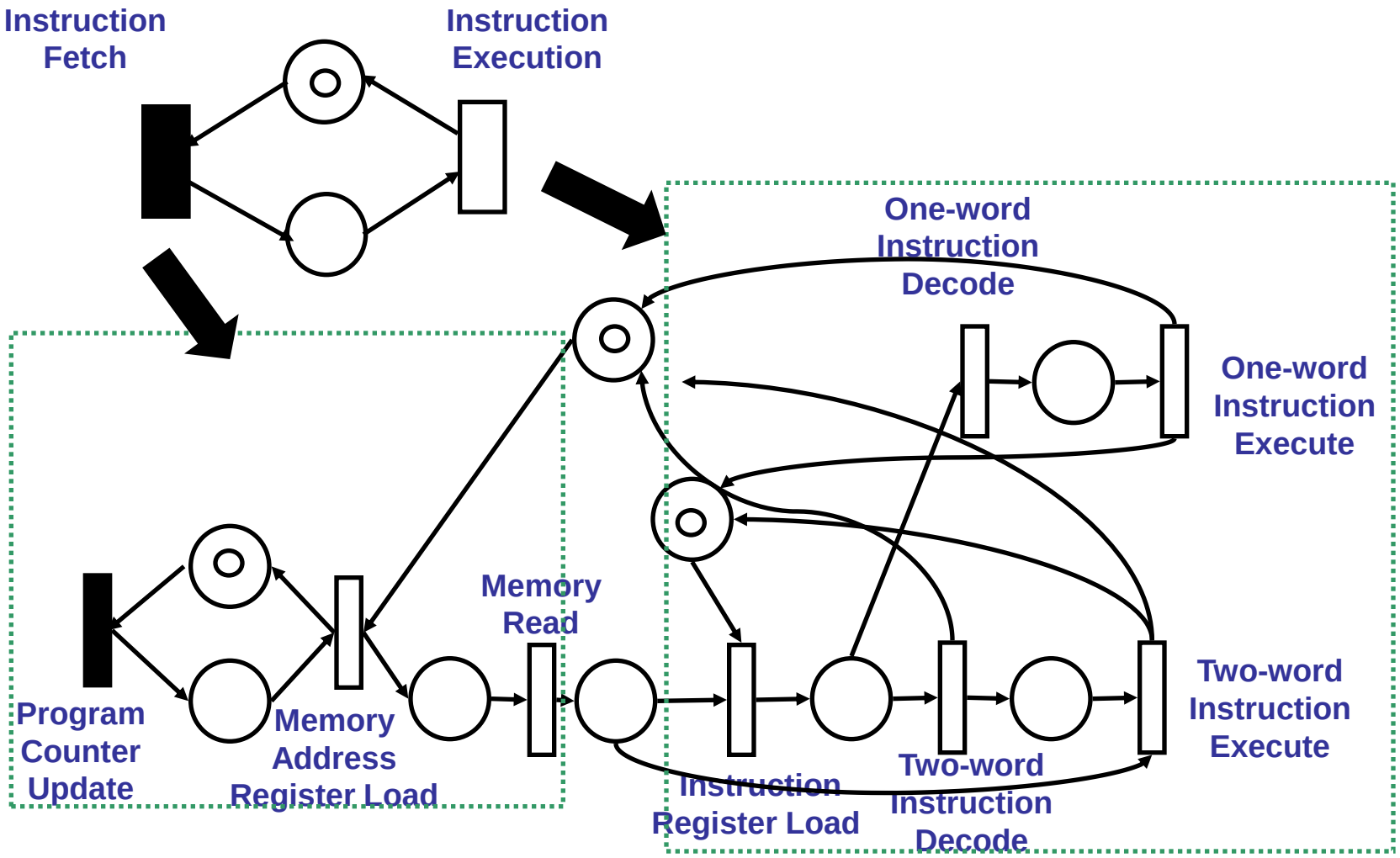
Alex Yakovlev, Victor Khomenko,
Andrey Mokhov and Danil Sokolov
Newcastle University, UK
async.org.uk
workcraft.org

PN/ACSD'15: Advanced Tutorial, Brussels, June 2015

Modelling.Outline

- **High level modelling and abstract refinement; processor example**
- **Low level modelling and logic synthesis; interface controller example**
- **Modelling of logic circuits: event-driven and level-driven parts**
- **Properties analysed**

High-level modelling:Processor Example



High-level modelling:Processor Example

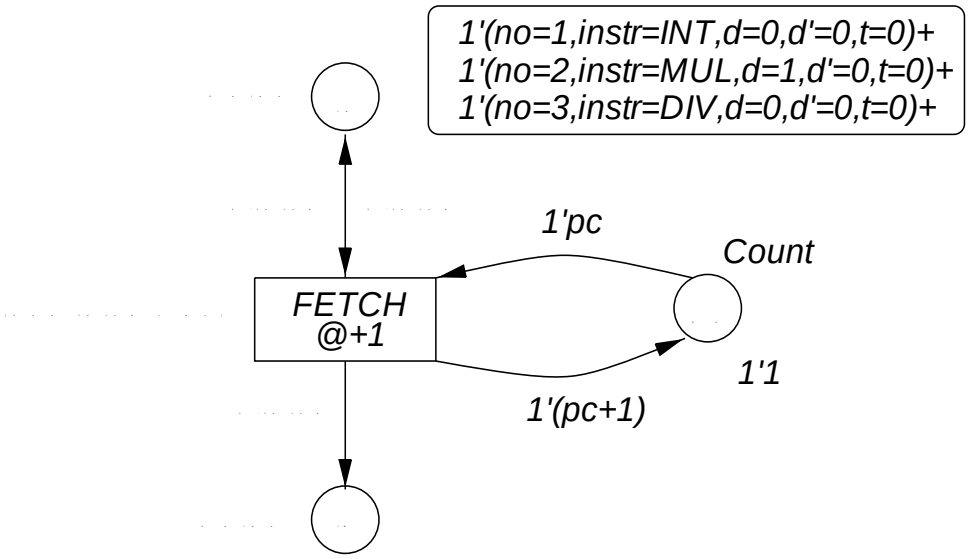
- Details of further refinement, circuit implementation (by direct translation) and performance estimation (using UltraSan) are in:

  A. Semenov, A.M. Koelmans, L.Lloyd and A. Yakovlev. Designing an asynchronous processor using Petri Nets, IEEE Micro, 17(2):54-64, March 1997

- For use of Coloured Petri net models and use of Design/CPN in processor modelling:

  F.Burns, A.M. Koelmans and A. Yakovlev. Analysing superscalar processor architectures with coloured Petri nets, Int. Journal on Software Tools for Technology Transfer, vol.2, no.2, Dec. 1998, pp. 182-191.

Using Coloured Petri nets



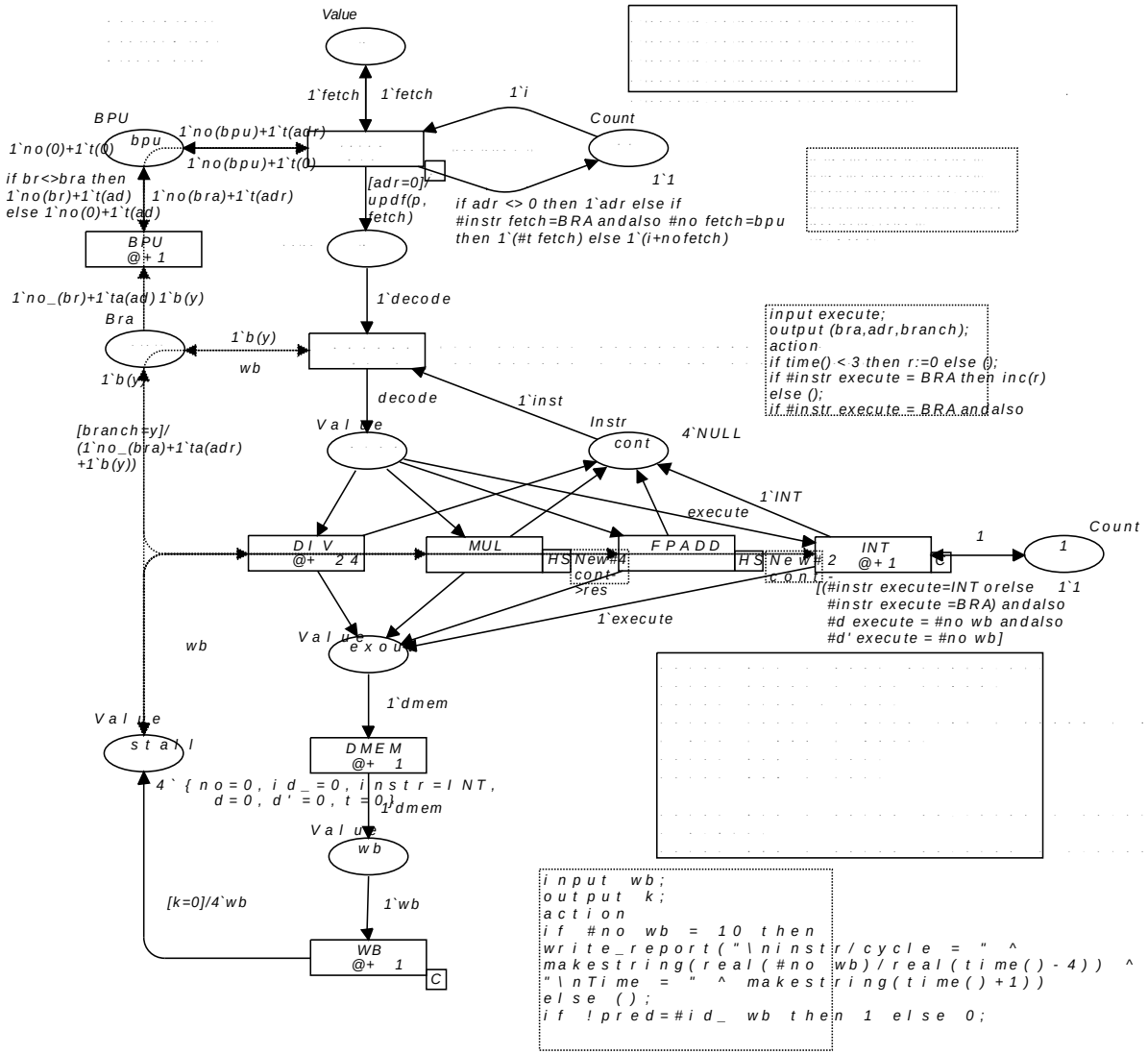
Color Set:

color Instr = with INT | FPADD | MUL | DIV | BRA | NULL;
color Line = int; color Dep = int;color Target = int; color Count = int timed;
*colour Value = record no:Line*instr:Instr*d:Dep*d':Dep*t:Target timed;*

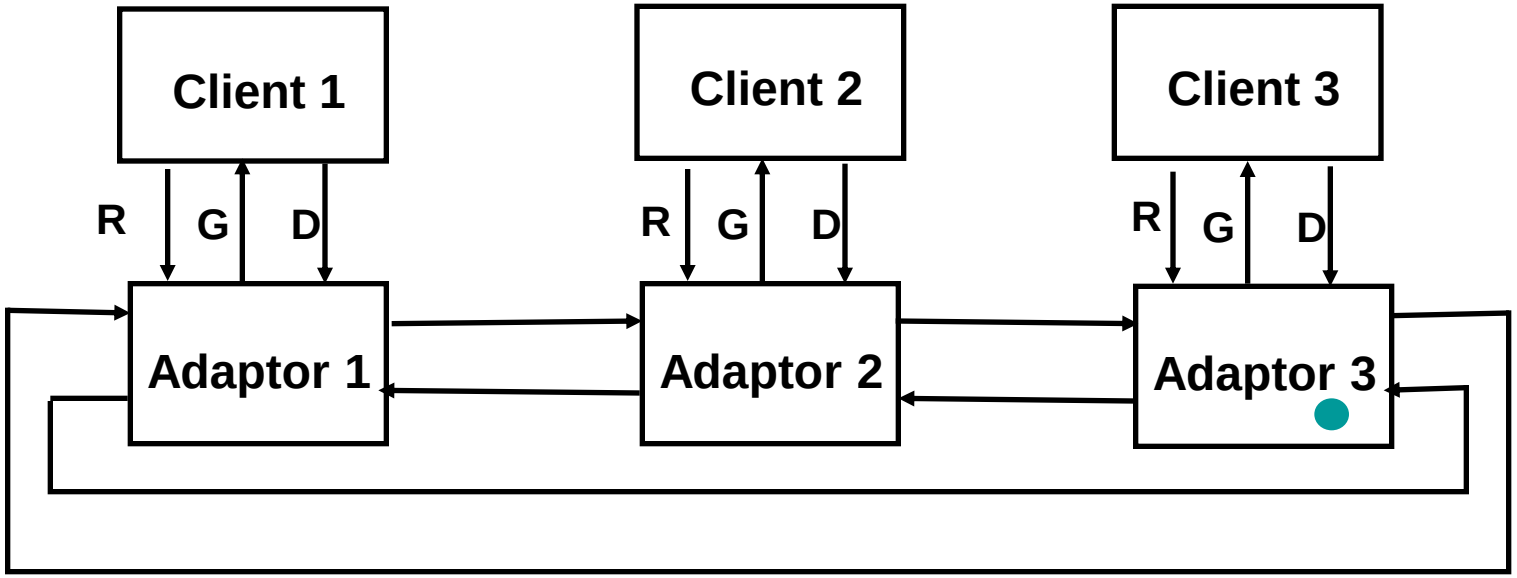
Var Set:

var fetch : Value;
var pc : Count

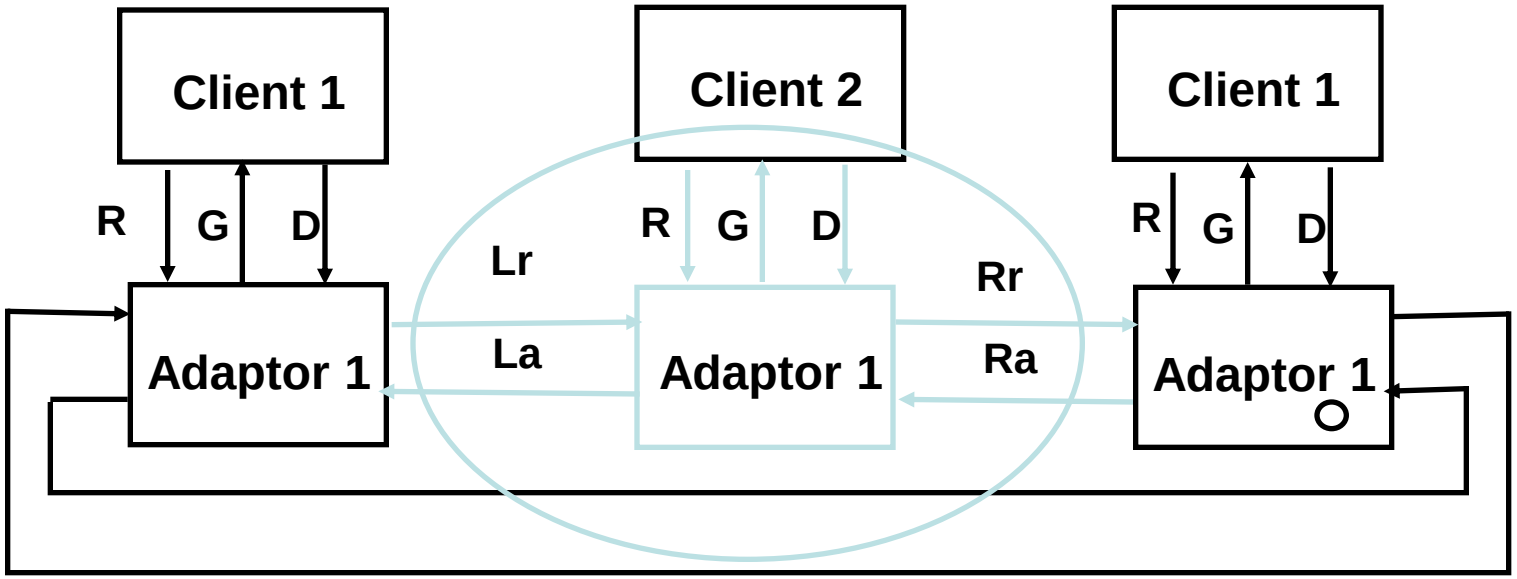
Using Coloured Petri nets



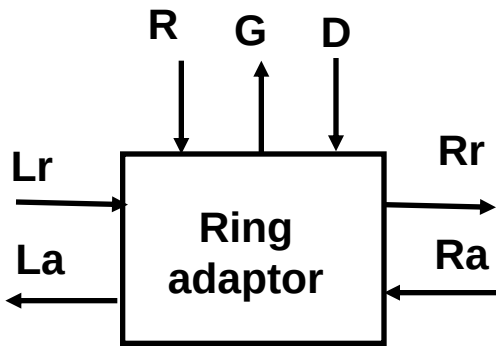
Low-level modelling: “lazy token” ring adaptor



Low-level modelling: “lazy token” ring adaptor

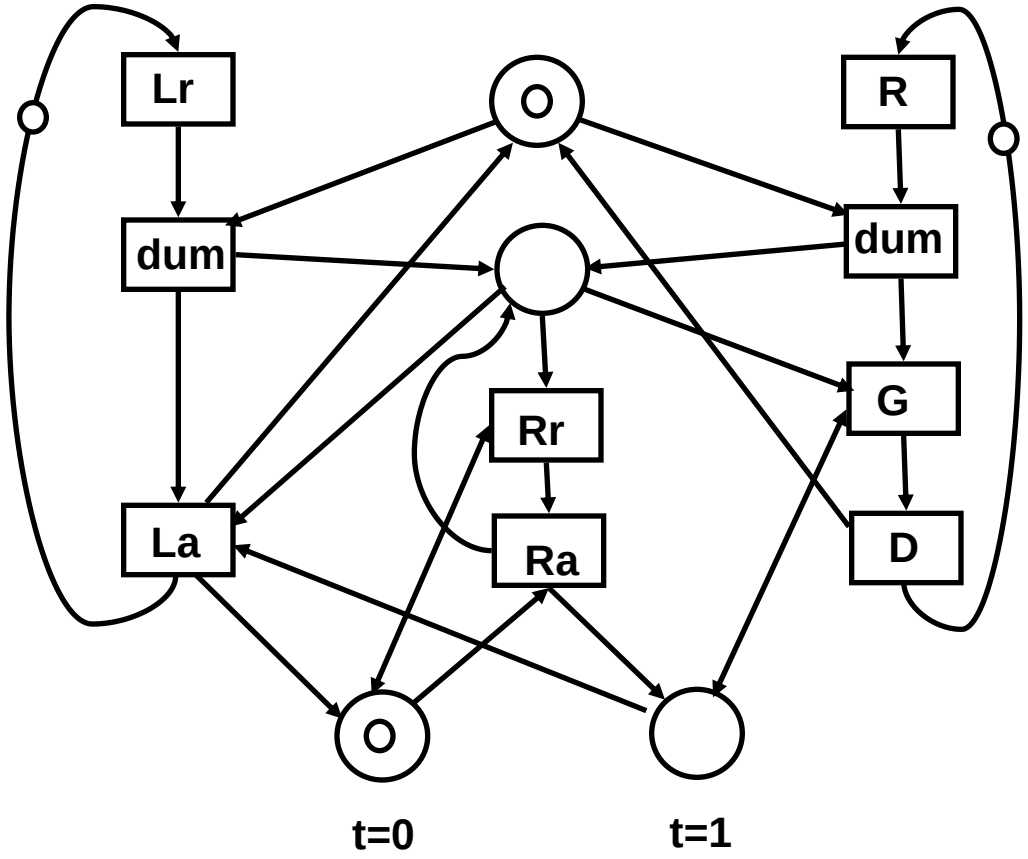


Lazy ring adaptor

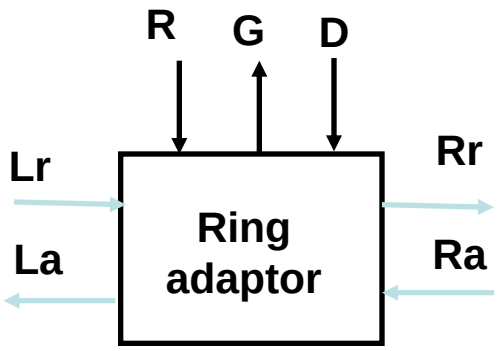


t=0

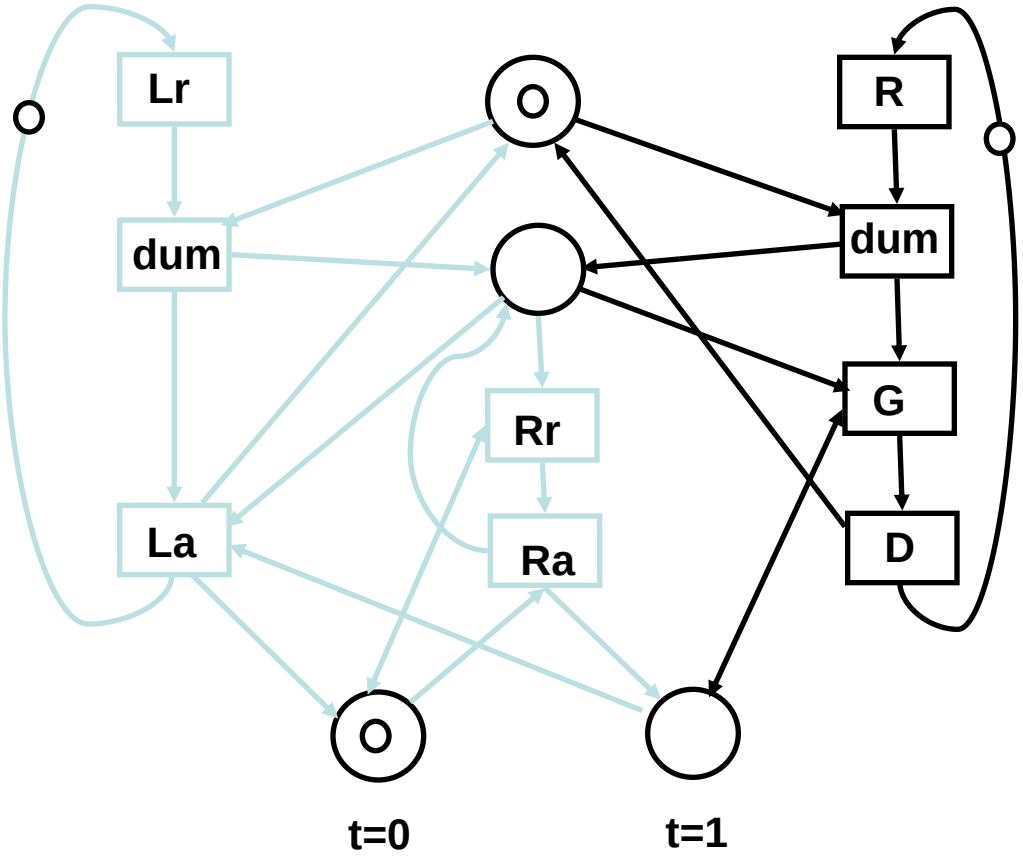
(token isn't initially here)



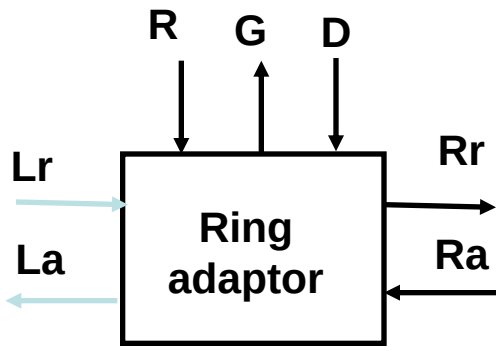
Lazy ring adaptor



t=0->1->0 (token must be taken from the right and past to the left

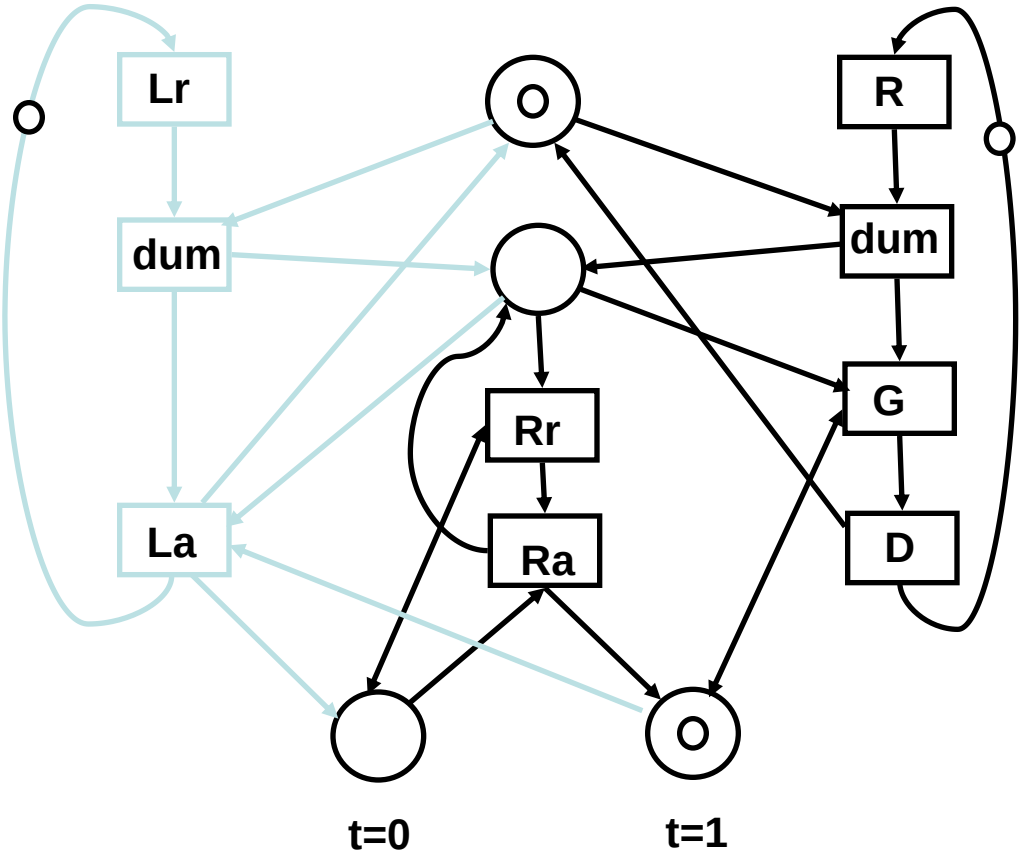


Lazy ring adaptor

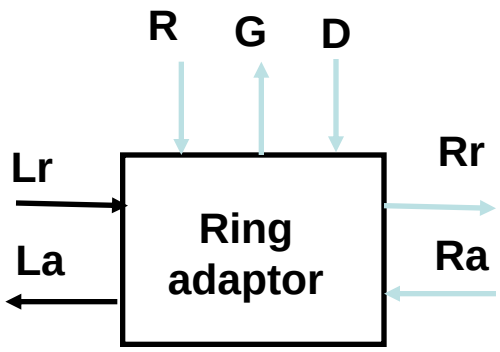


t=1

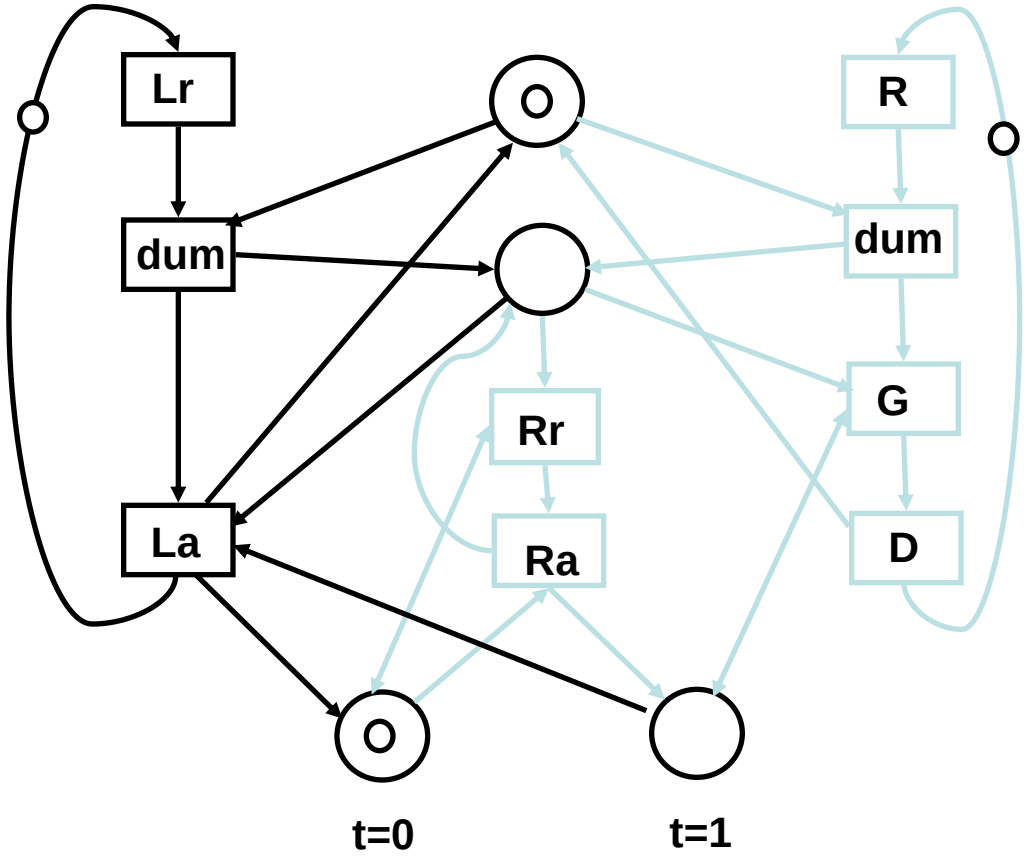
(token is already here)



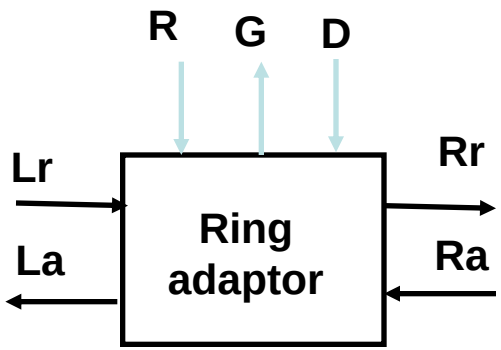
Lazy ring adaptor



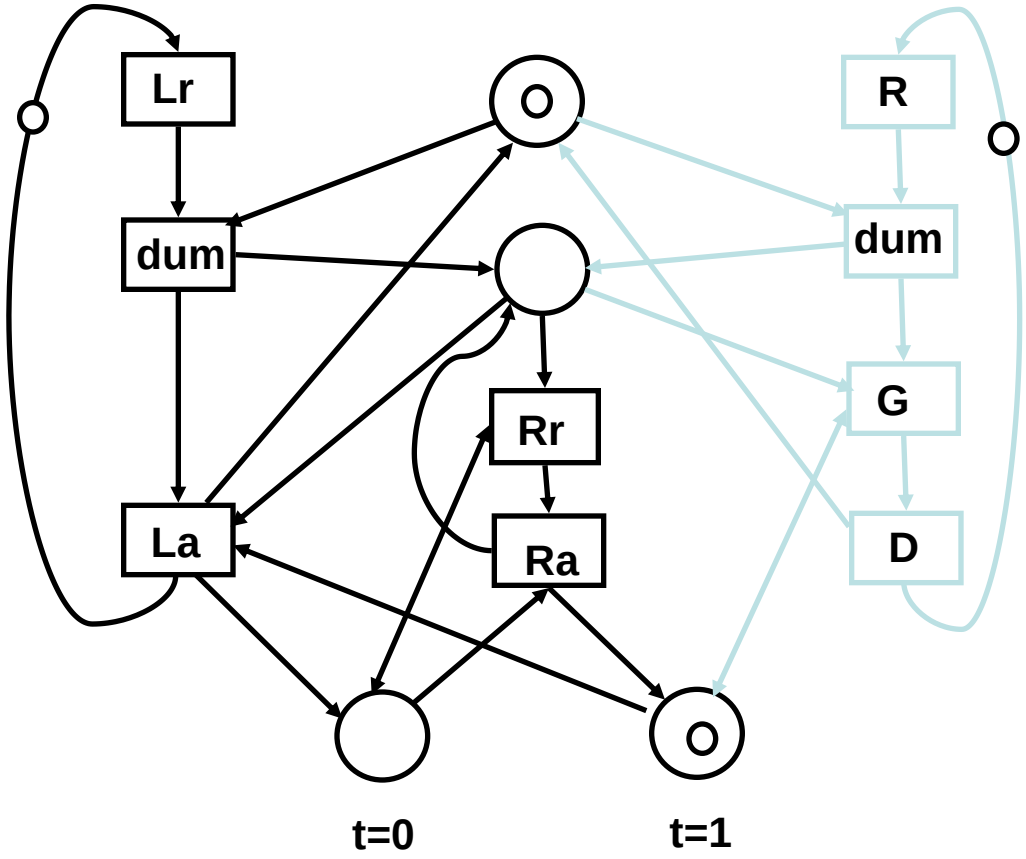
t=0->1
(token must be taken from the right)



Lazy ring adaptor

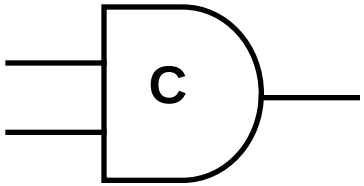


t=1
(token is here)

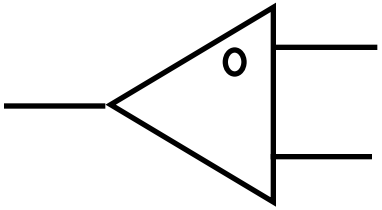


Logic Circuit Modelling

Event-driven elements

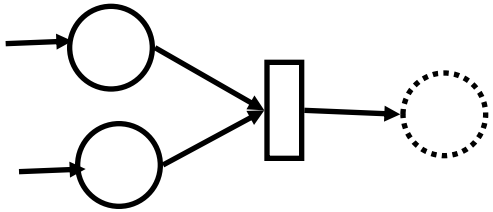


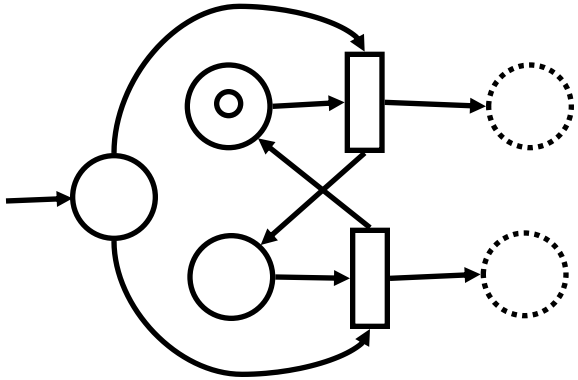
Muller C-element



Toggle

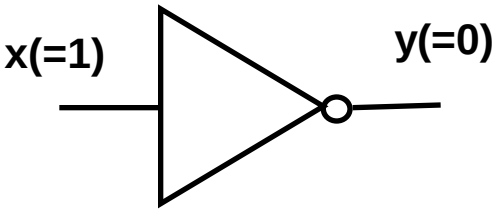
Petri net equivalents



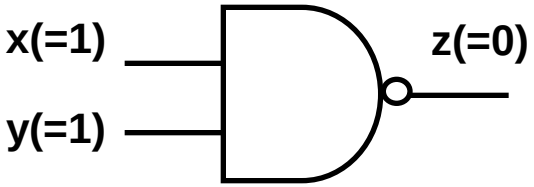


Logic Circuit Modelling

Level-driven elements

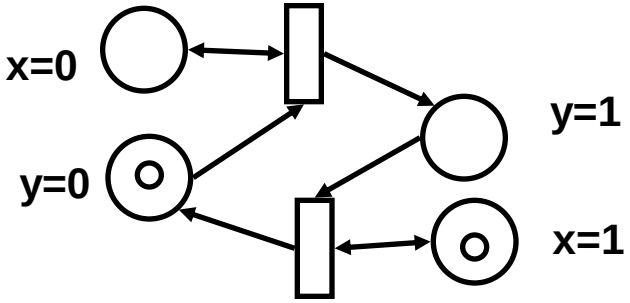


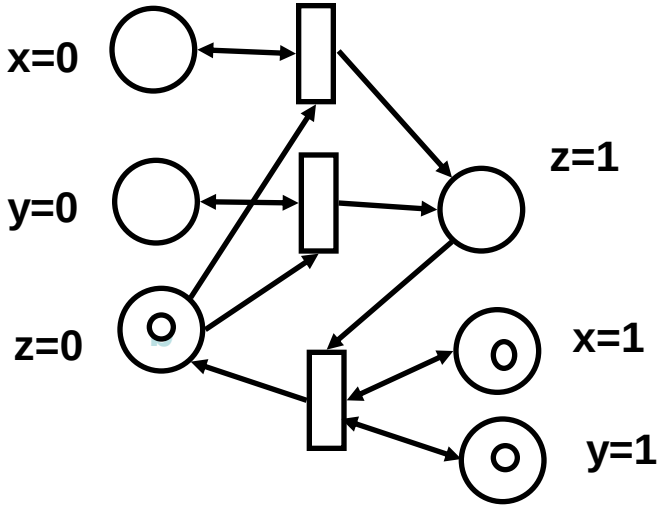
NOT gate



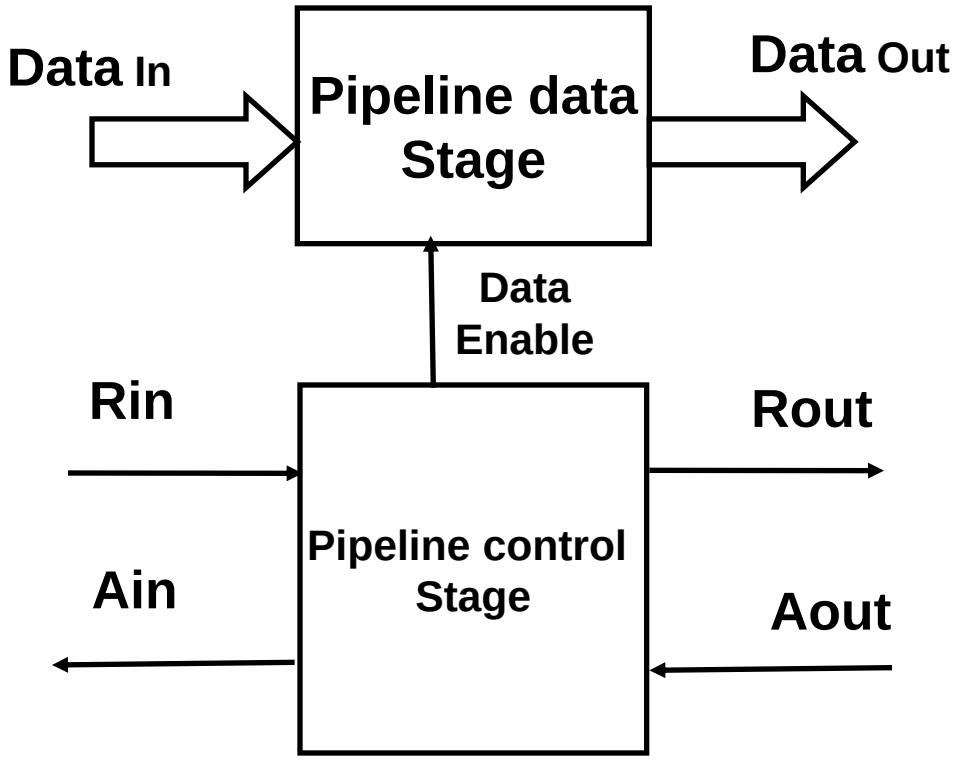
NAND gate

Petri net equivalents





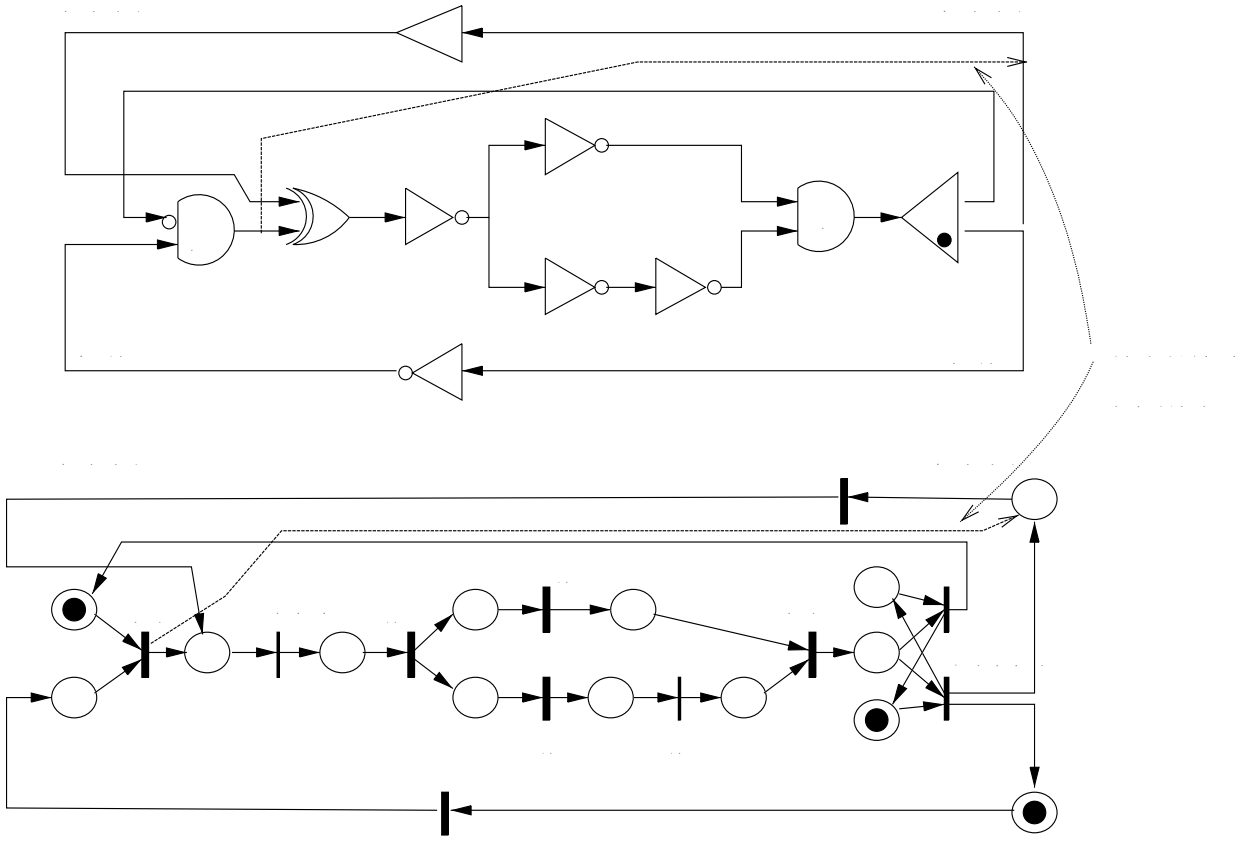
Logic Circuit Modelling: examples



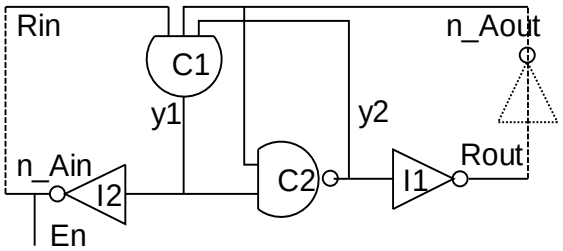
Pipeline control must guarantee:

- **Handshake protocols between the stages**
- **Safe propagation of the previous datum before the next one**

Event-driven circuit



Level-driven circuit

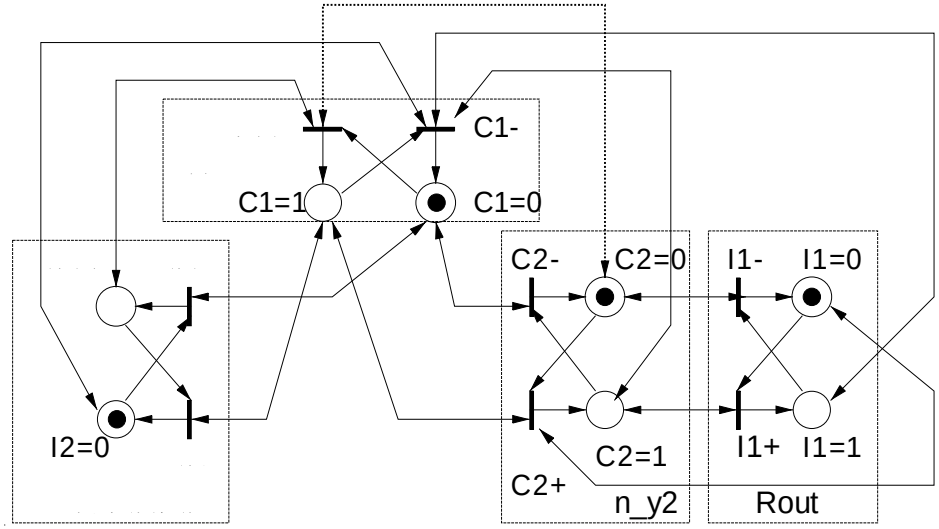


C1: y1 = Rin {y2} + y1(Rin + n_Aout + y2)

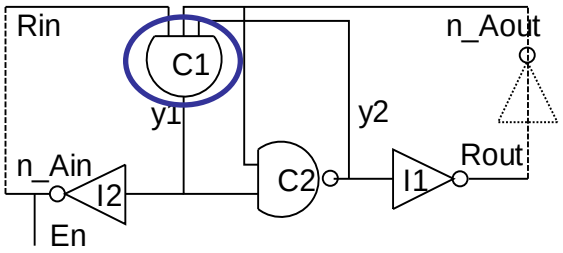
C2: n_y2 = y1 (n_Aout + n_y2)

I1: Rout = y2' or Rout = delay (n_y2)

I1: n_Ain = y1'



Level-driven circuit



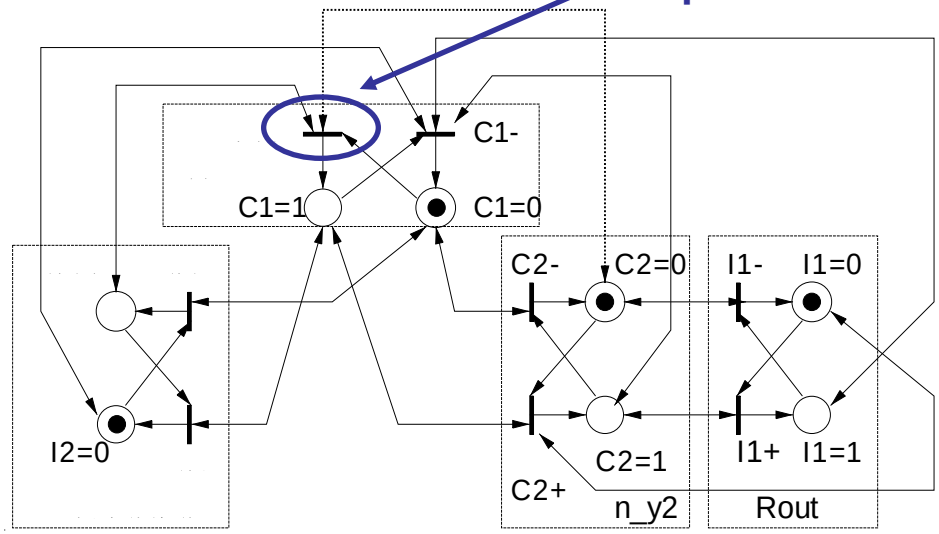
C1: y1 = Rin {y2} + y1(Rin + n_Aout + y2)

C2: n_y2 = y1 (n_Aout + n_y2)

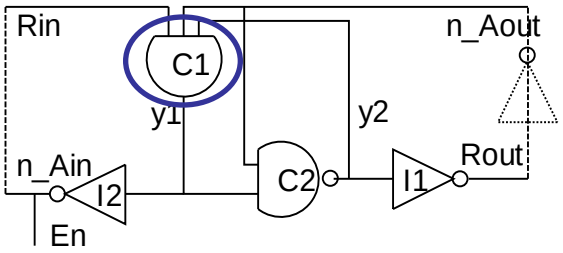
I1: Rout = y2' or Rout = delay (n_y2)

I1: n_Ain = y1'

Set-part



Level-driven circuit



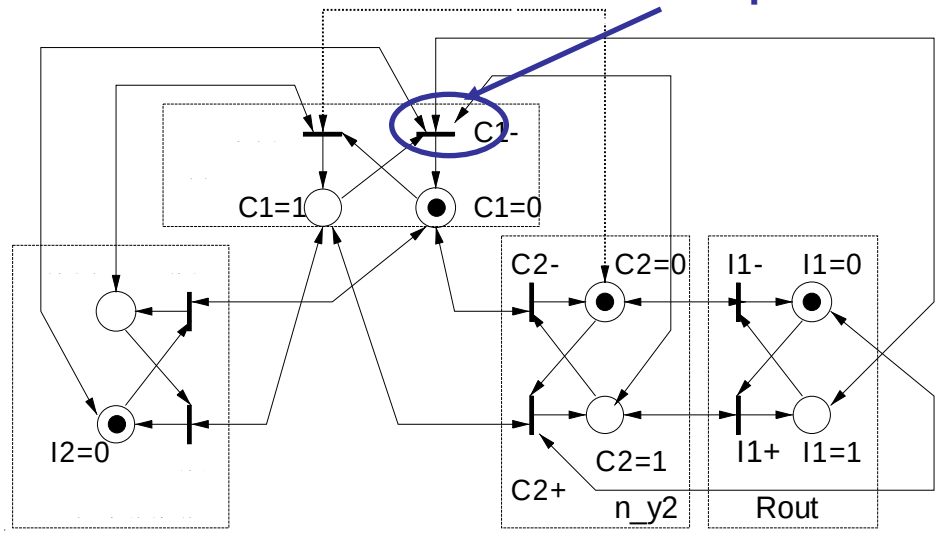
C1: y1 = Rin {y2} + y1(Rin + n_Aout + y2)

C2: n_y2 = y1 (n_Aout + n_y2)

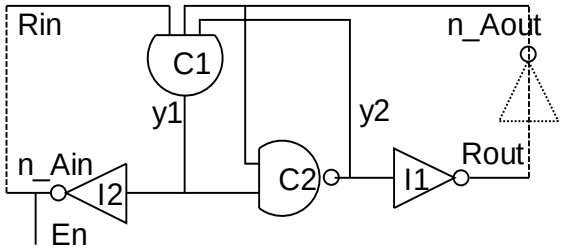
I1: Rout = y2' or Rout = delay (n_y2)

I1: n_Ain = y1

Reset-part



Level-driven circuit

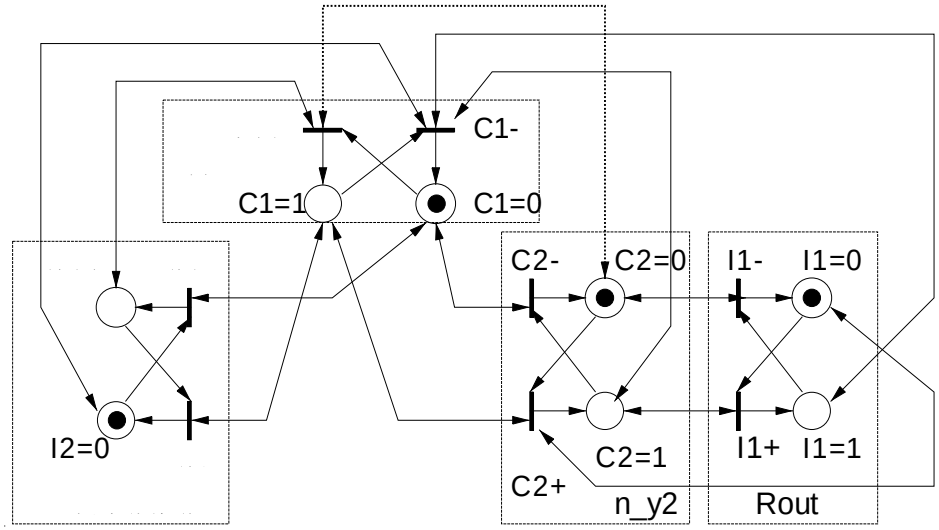


C1: y1 = Rin {y2} + y1(Rin + n_Aout + y2)

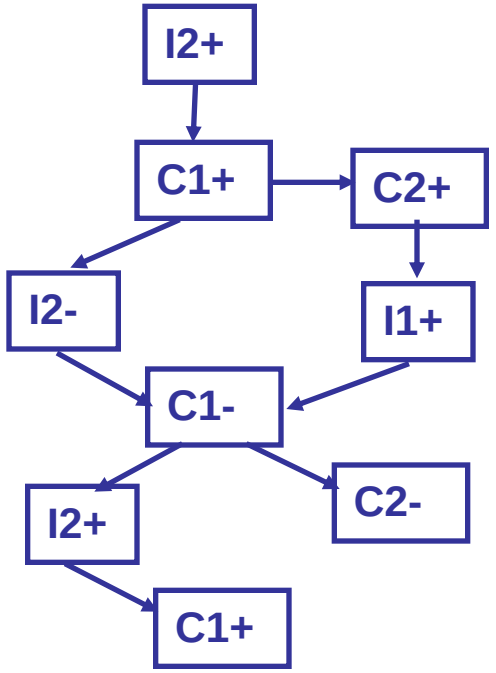
C2: n_y2 = y1 (n_Aout + n_y2)

I1: Rout = y2' or Rout = delay (n_y2)

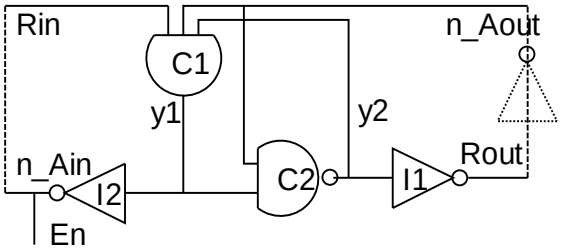
I1: n_Ain = y1'



Without y2 in Set part of y1 this trace can happen:



Level-driven circuit

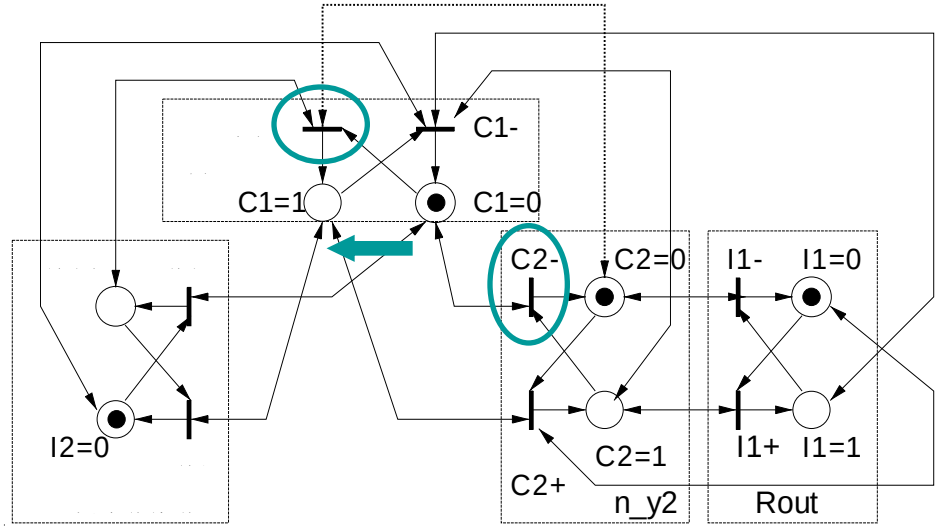


C1: y1 = Rin {y2} + y1(Rin + n_Aout + y2)

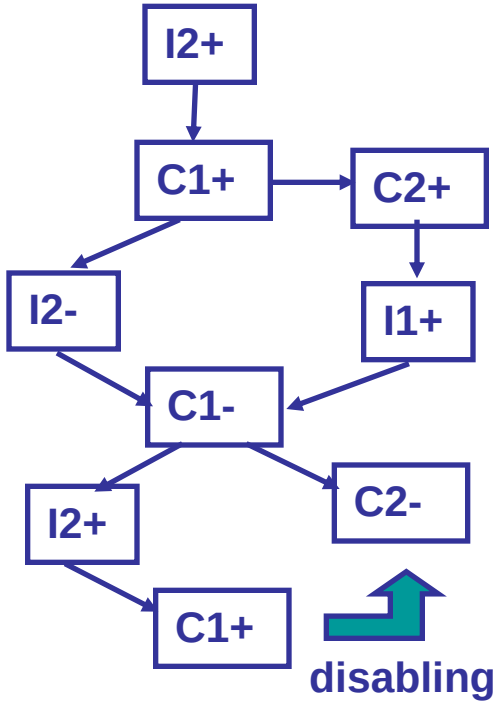
C2: n_y2 = y1 (n_Aout + n_y2)

I1: Rout = y2' or Rout = delay (n_y2)

I1: n_Ain = y1'



Without y2 in Set part of y1 this trace can happen:

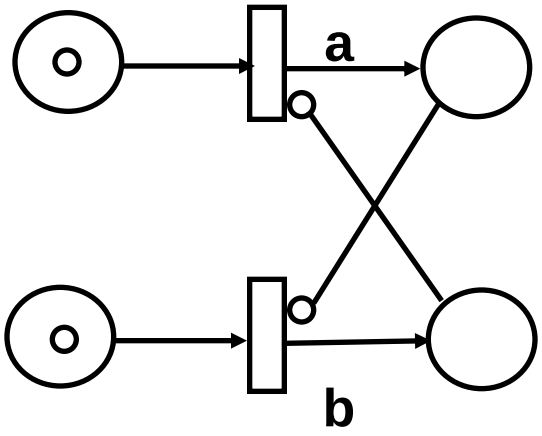


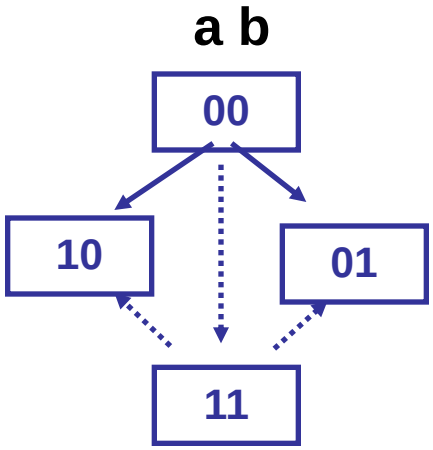
Properties analysed

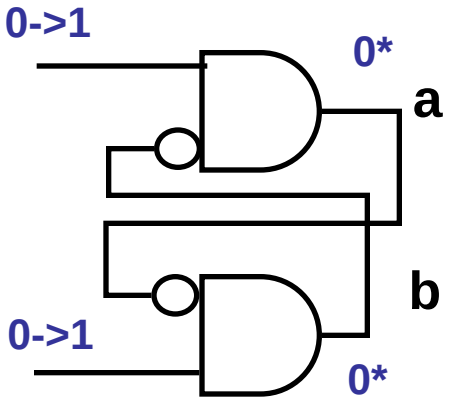
- Functional correctness (need to model environment)
- Deadlocks
- Hazards:
  - *non-1-safeness for event-based*
  - *non-persistency for level-based*
- Timing constraints
  - Absolute (need Time(d) Petri nets)
  - Relative (compose with a PN model of order conditions)

How adequate is PN model?

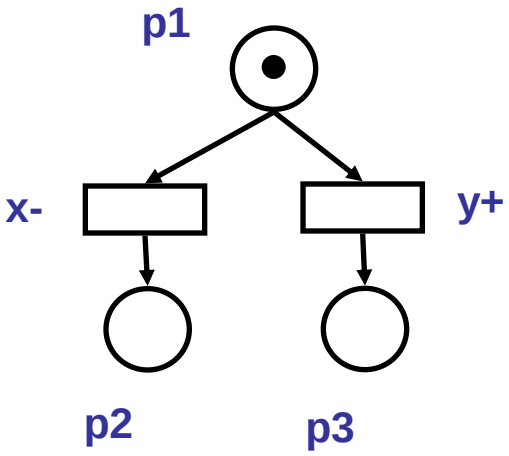
- Petri nets have events with *atomic action* semantics
- Asynchronous circuits may exhibit behaviour that does not fit within this domain – due to *inertia*

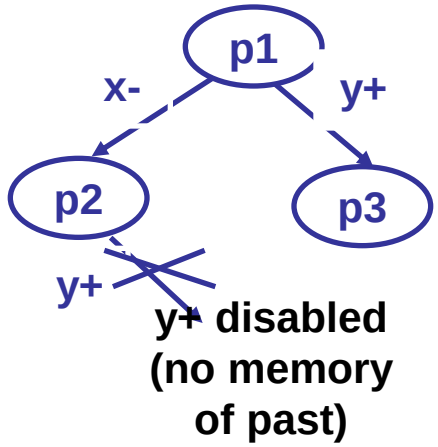




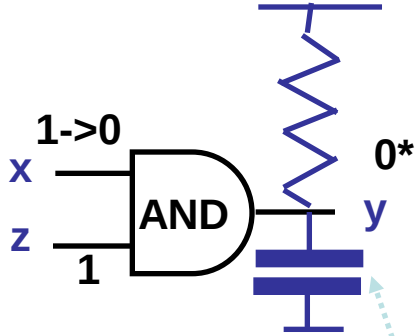


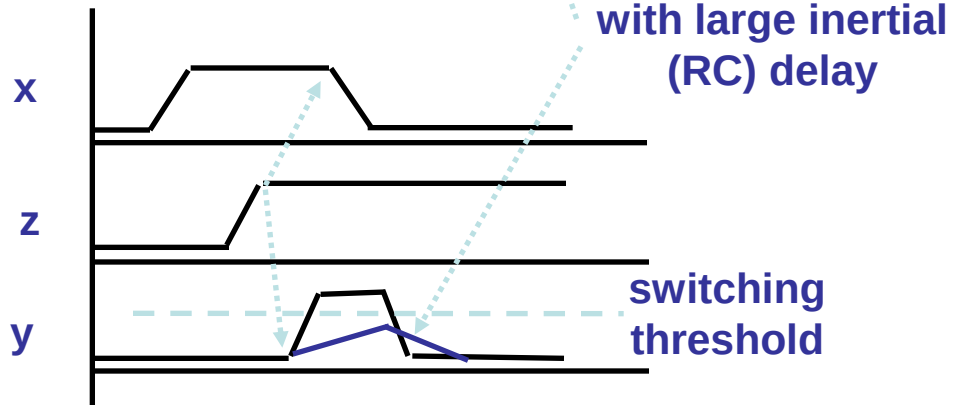
Petri Nets versus Circuits





y+ disabled (no memory of past)





Race between x- and y+ causes nondeterministic behaviour on y:

(1) Either there is a 0-1-0 pulse

(2) Or nothing

Modelling. Conclusions

- Choosing the right level of modelling is crucial
- Refinement of Petri net models and interpretation can be used in hardware design
- Petri nets are too abstract to capture analogue phenomena in circuits
- However, non-persistence or non-safeness can (conservatively) approximate the possibility of hazards

Hardware Synthesis with Petri Nets: Basics

Alex Yakovlev, Victor Khomenko,
Andrey Mokhov and Danil Sokolov
Newcastle University, UK
async.org.uk
workcraft.org

PN/ACSD'15: Advanced Tutorial, Brussels, June 2015

Asynchronous Circuit Synthesis

- Synthesis of asynchronous circuits can proceed from various behavioural descriptions
- For example, timing diagrams are a notation that is often used by engineers to specify circuit behaviour
- The model which is close to TDs is Signal Transition Graphs
- STGs are interpreted Petri nets
- Typically it is sufficient to use limited classes of Petri nets, such as Marked Graphs, Free-Choice PNs, or so-called Free and Controlled Choice PNs
- Usually 1-safe PNs are enough, but to capture effects like OR causality we may need bounded PNs

Asynchronous Circuit Synthesis (cont.)

- Synthesis of asynchronous circuits may also involve abstract synthesis from causality constraints, from traces or sequential models
- Petri nets can be used as an intermediate form in which concurrency is represented efficiently
- Circuits can be synthesized from Petri nets by their direct (structural) mapping (see Appendix)

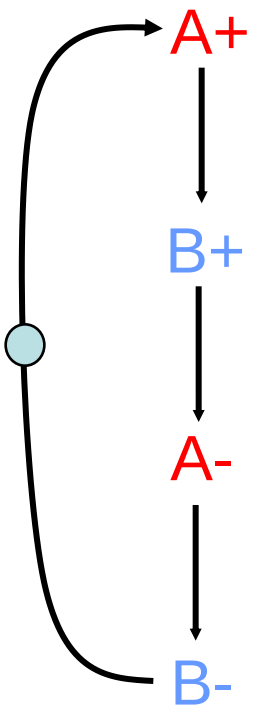
- **More details on Circuit Verification and Logic Synthesis from STGs will follow**
- **Modern tools … after lunch!**

Synthesis Issues in Appendix

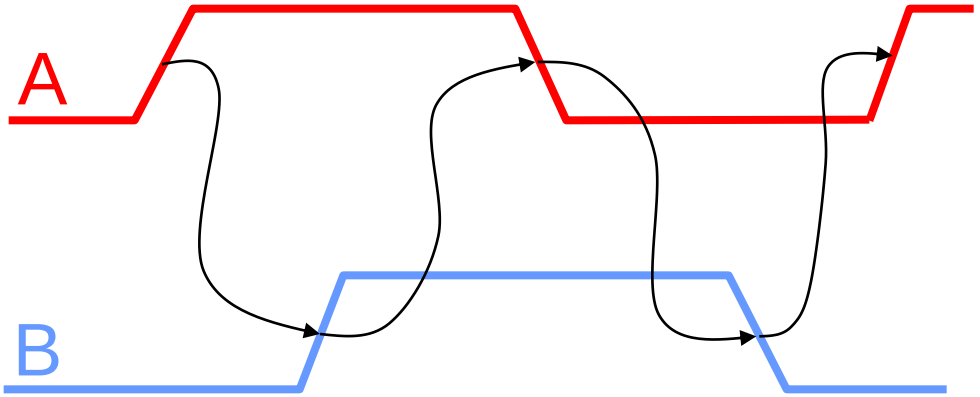
- **Synthesis of Labelled PNs (LPNs) from transition systems**
- **Handshake and signal refinement (LPN-to-STG)**
- **Direct synthesis from LPNs**

Control specification

Signal Transition Graph (STG)



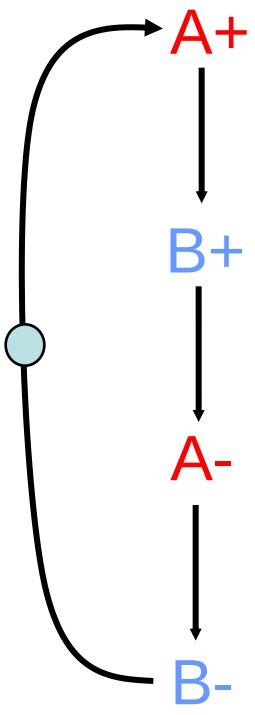
Timing Diagram

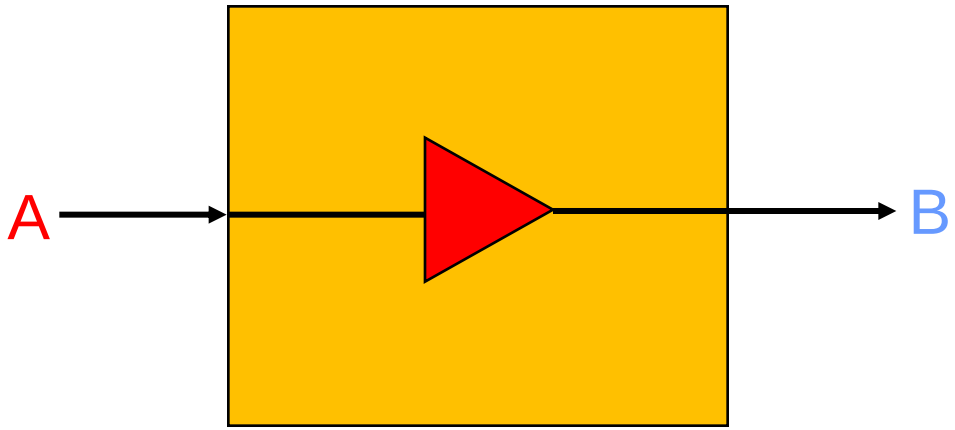


A input

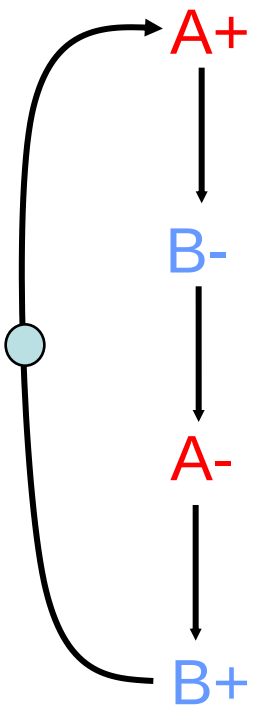
B output

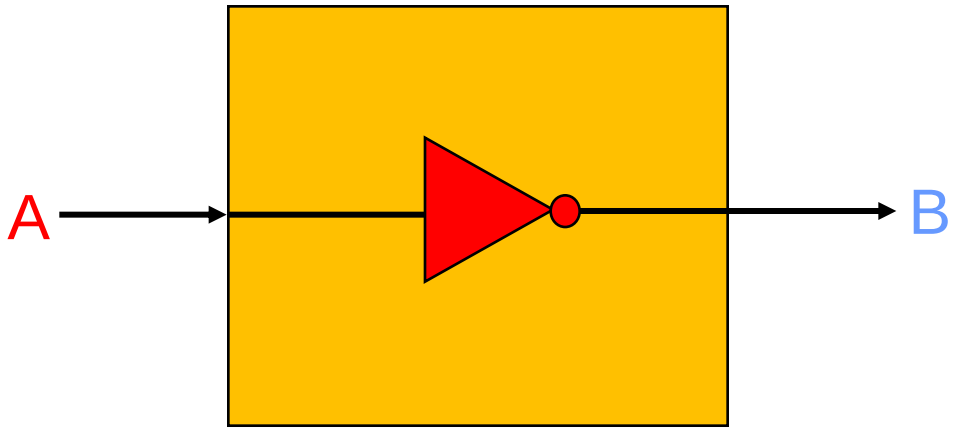
Control specification



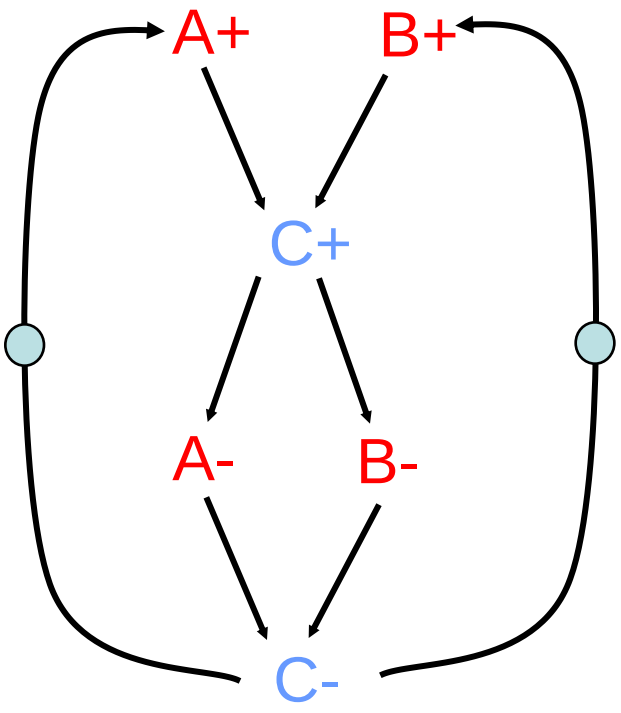


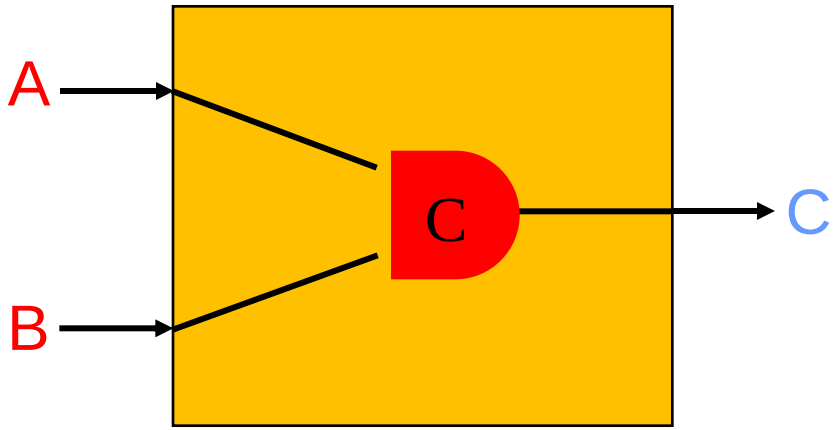
Control specification



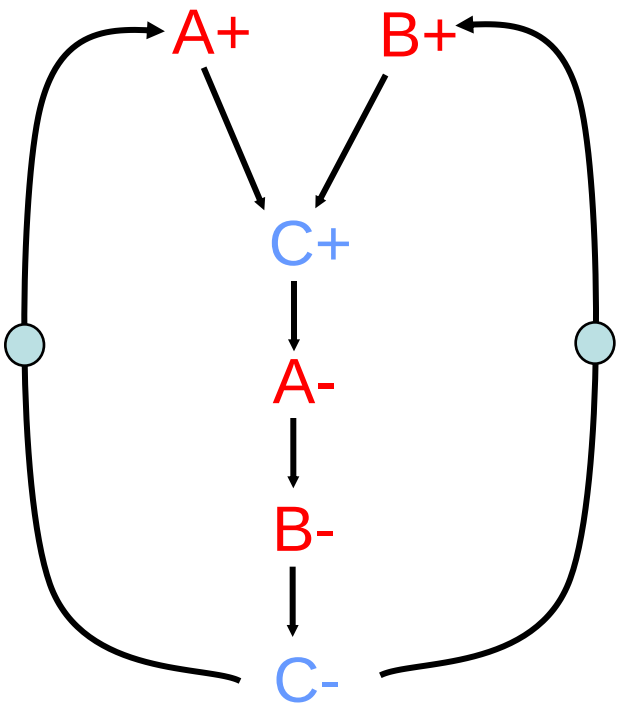


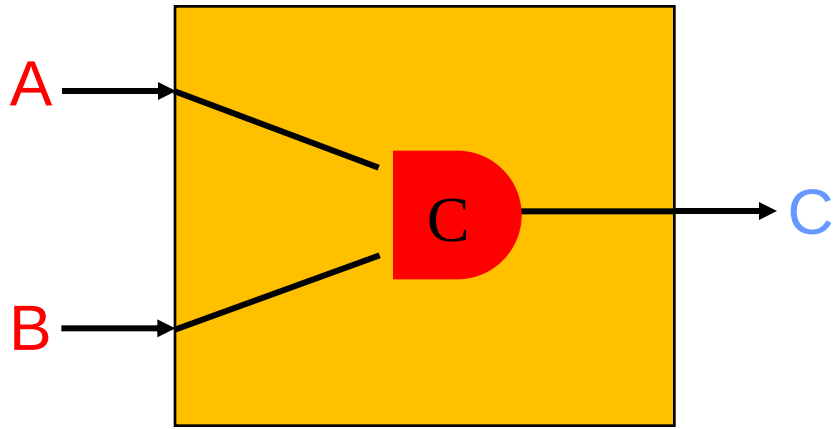
Control specification



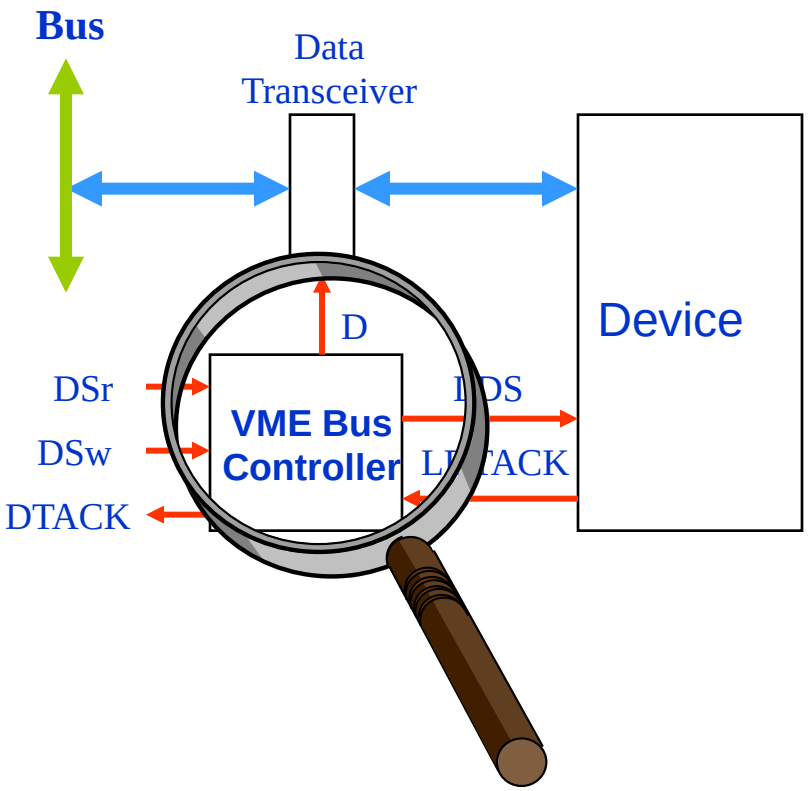


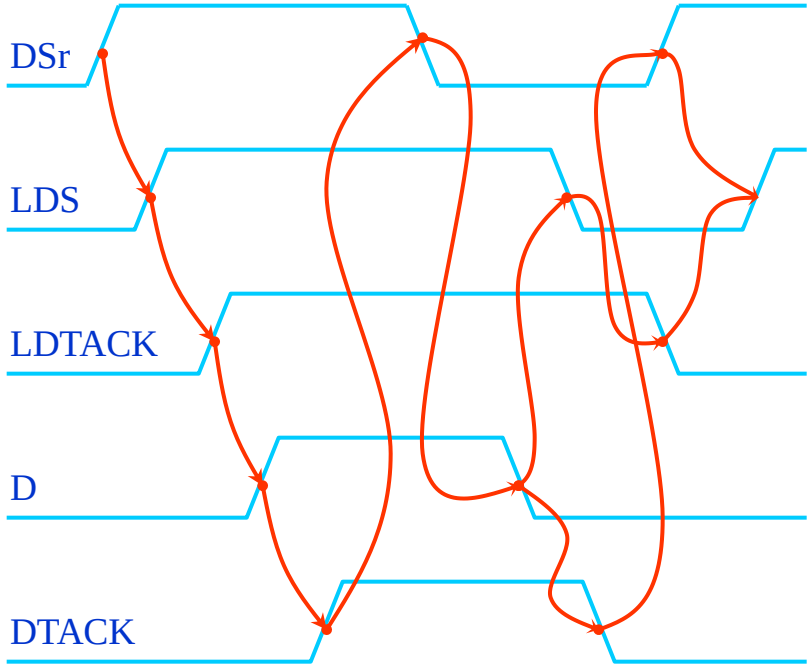
Control specification





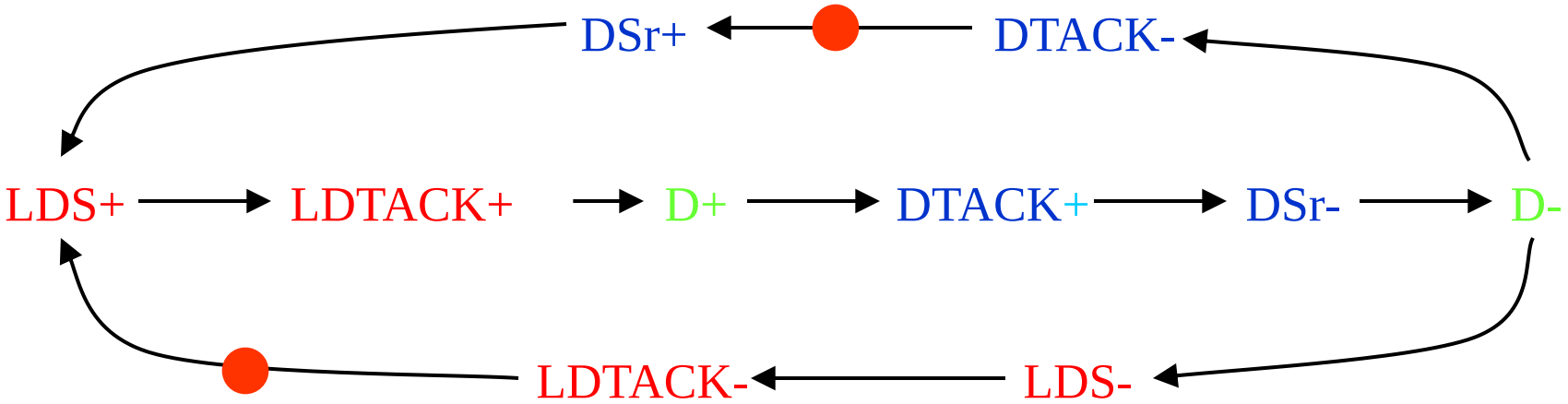
VME bus example using Petri nets

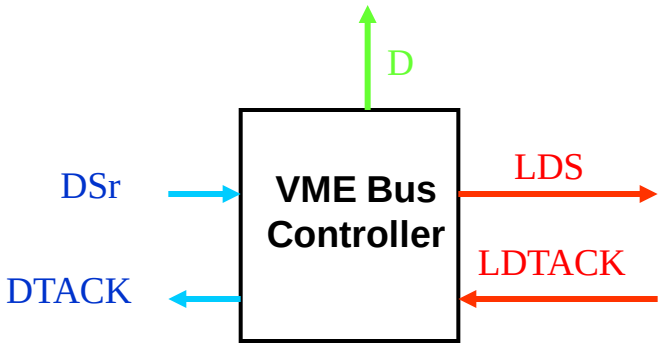




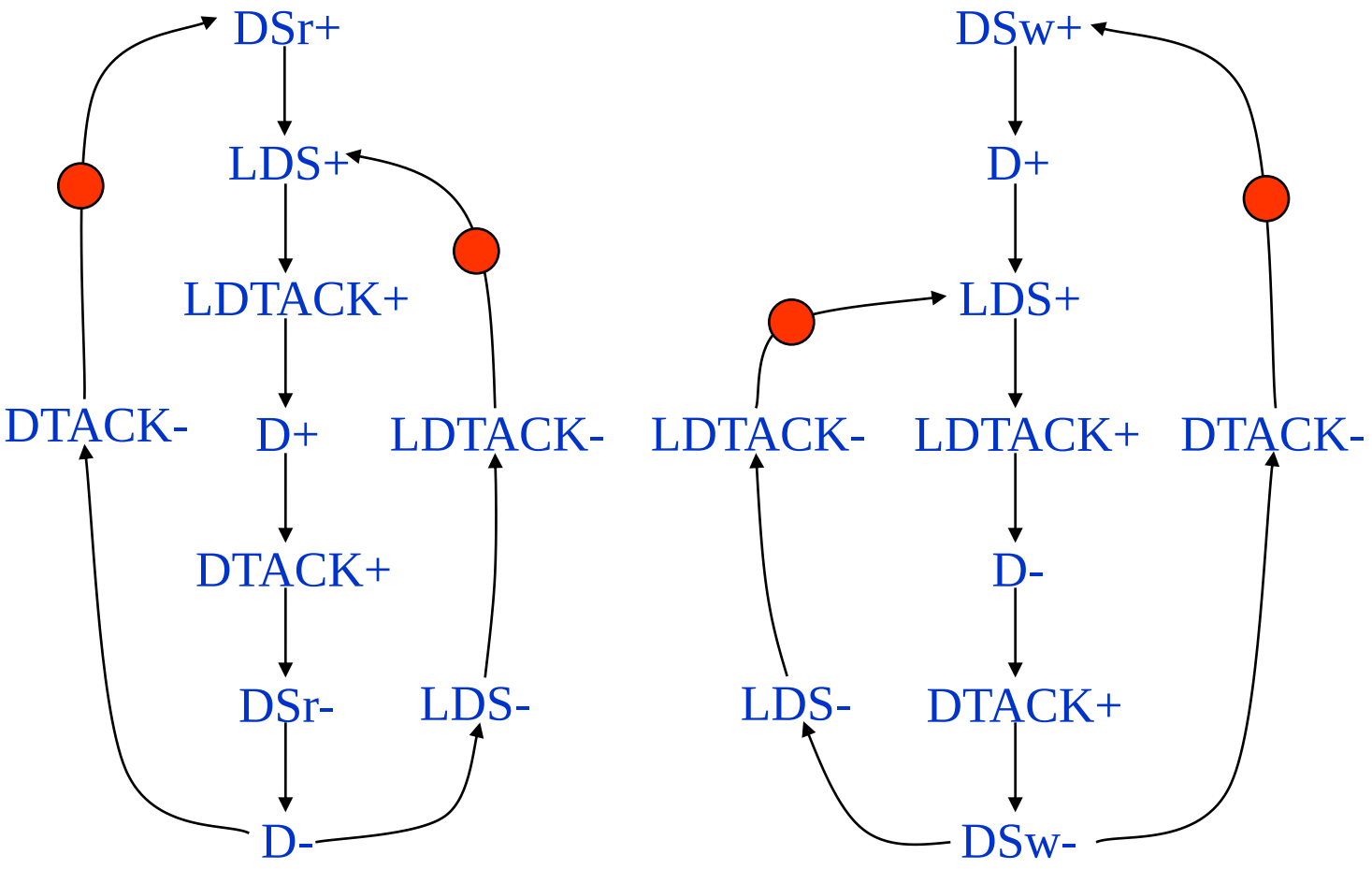
Read Cycle

STG for the READ cycle

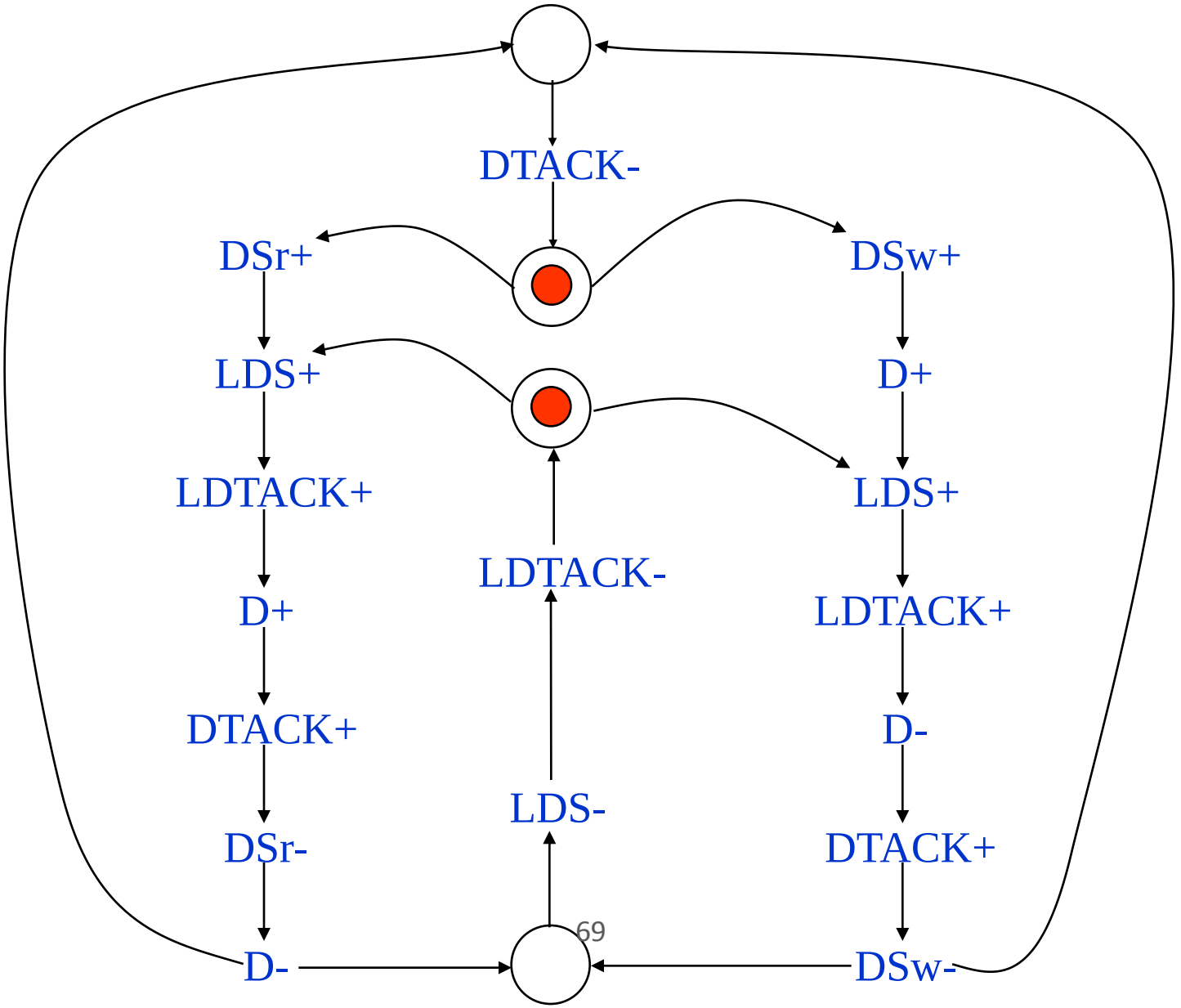




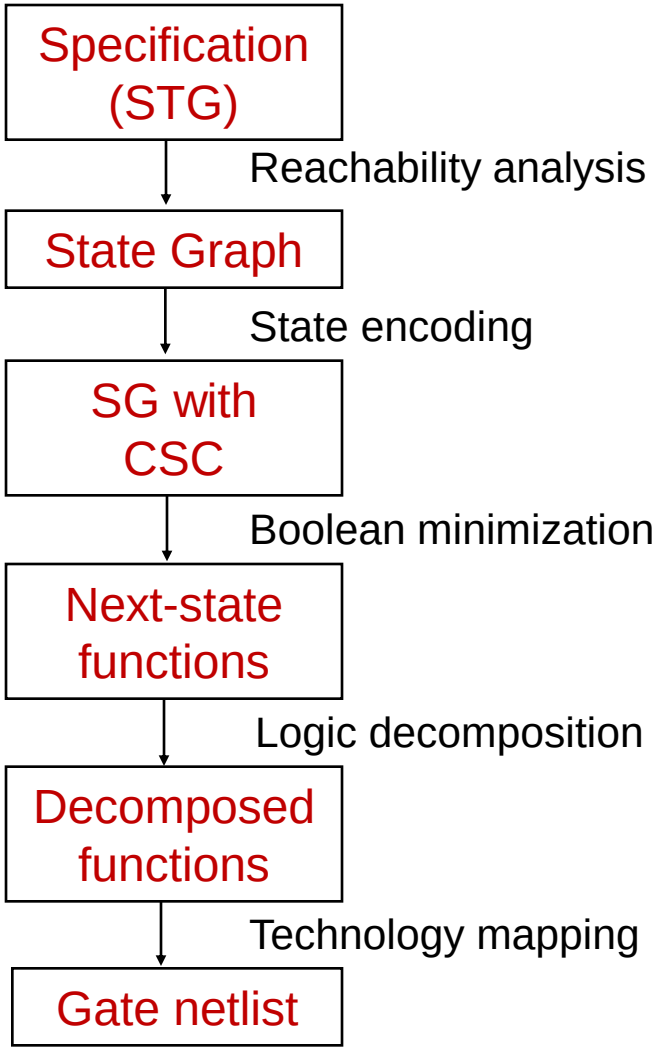
Choice: Read and Write cycles



Choice: Read and Write cycles

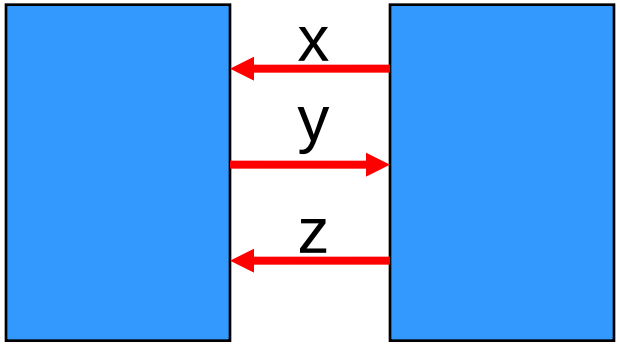


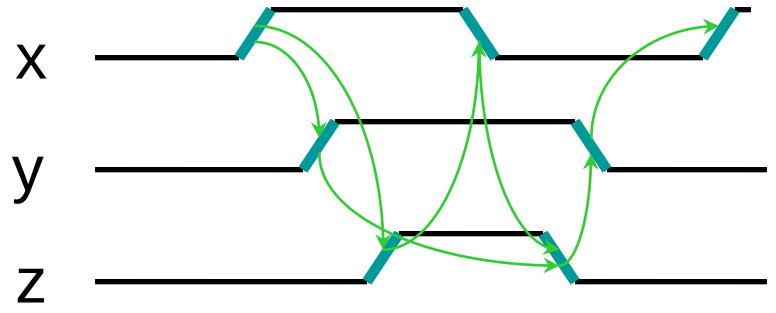
Typical STG synthesis flow

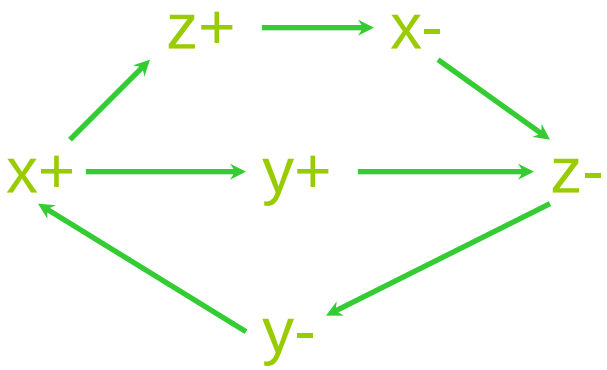


Let us consider some elements of this flow on a simple example

Specification

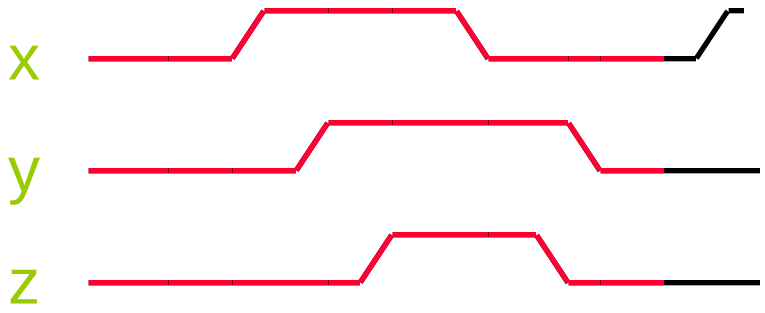


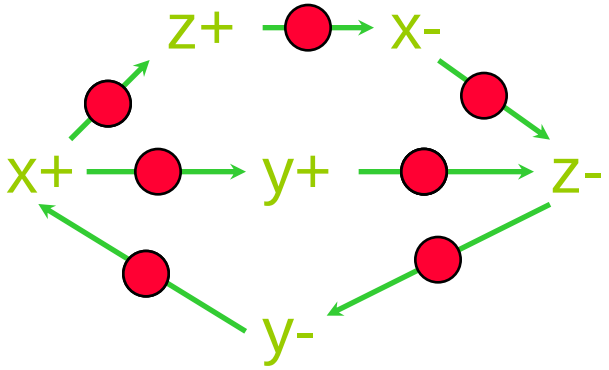




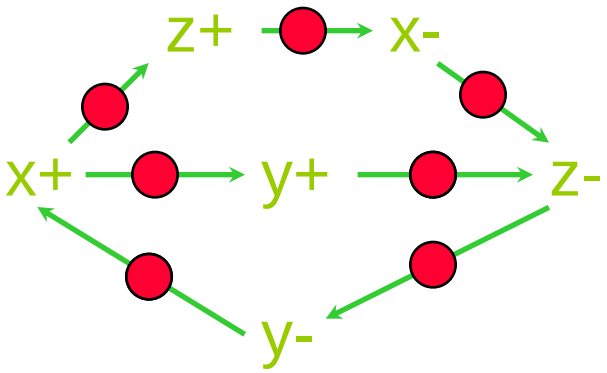
Signal Transition Graph (STG)

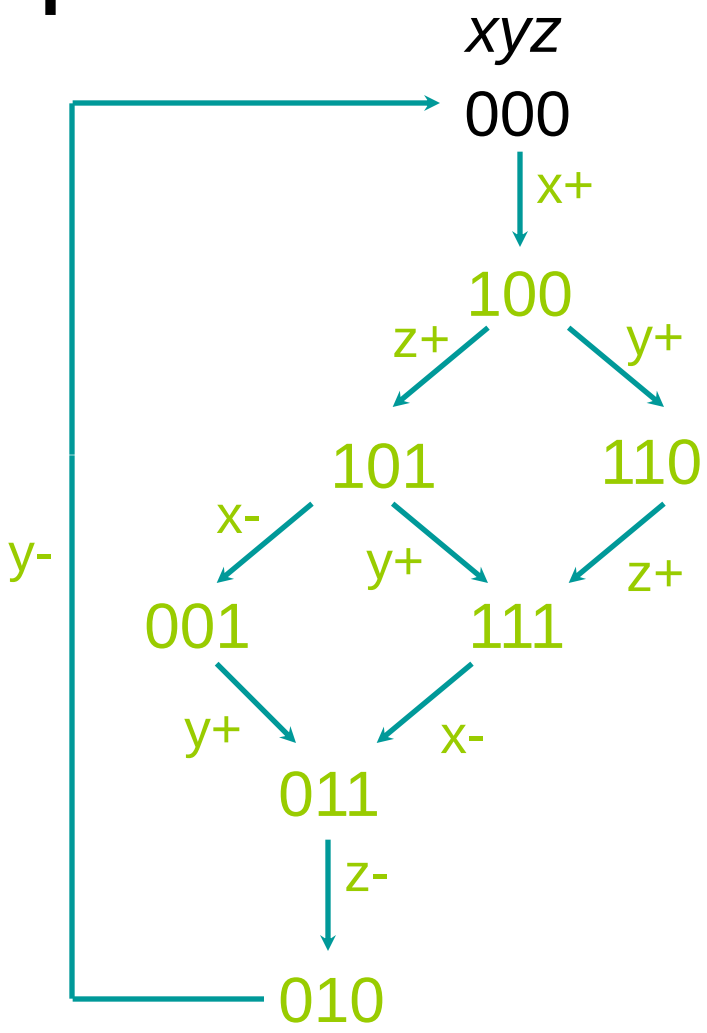
Token flow





State graph



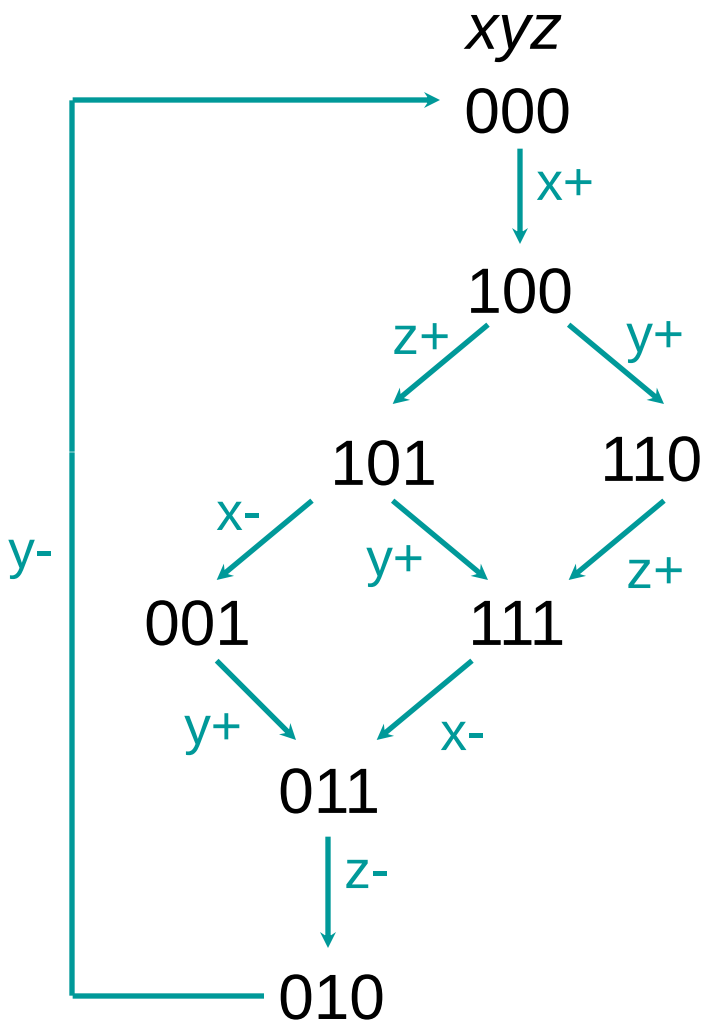


Next-state functions

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

$$y = z + x$$

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

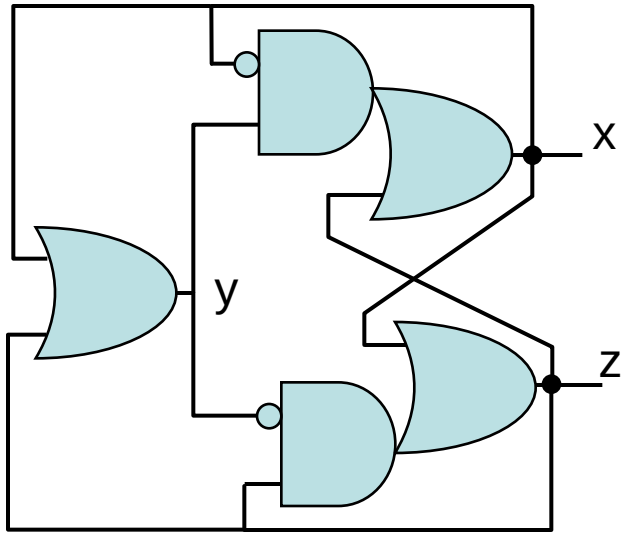


Gate netlist

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

$$y = z + x$$

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



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

Conclusions

- Synthesis of asynchronous circuits can proceed from various descriptions, e.g. from timing diagrams (often used by engineers)
- It may also involve abstract synthesis from causality constrains, from traces or sequential models
- Petri nets can be used as an intermediate form in which concurrency is represented efficiently
- Circuits can be synthesized from Petri nets by their direct (structural) mapping (see Appendix), or by using logic synthesis from STGs
- More details on Circuit Verification and Logic Synthesis from STGs will follow
- Modern tools … after lunch!