



UNIVERSITY OF
NEWCASTLE UPON TYNE



Ten years of petrifying: where are we now?

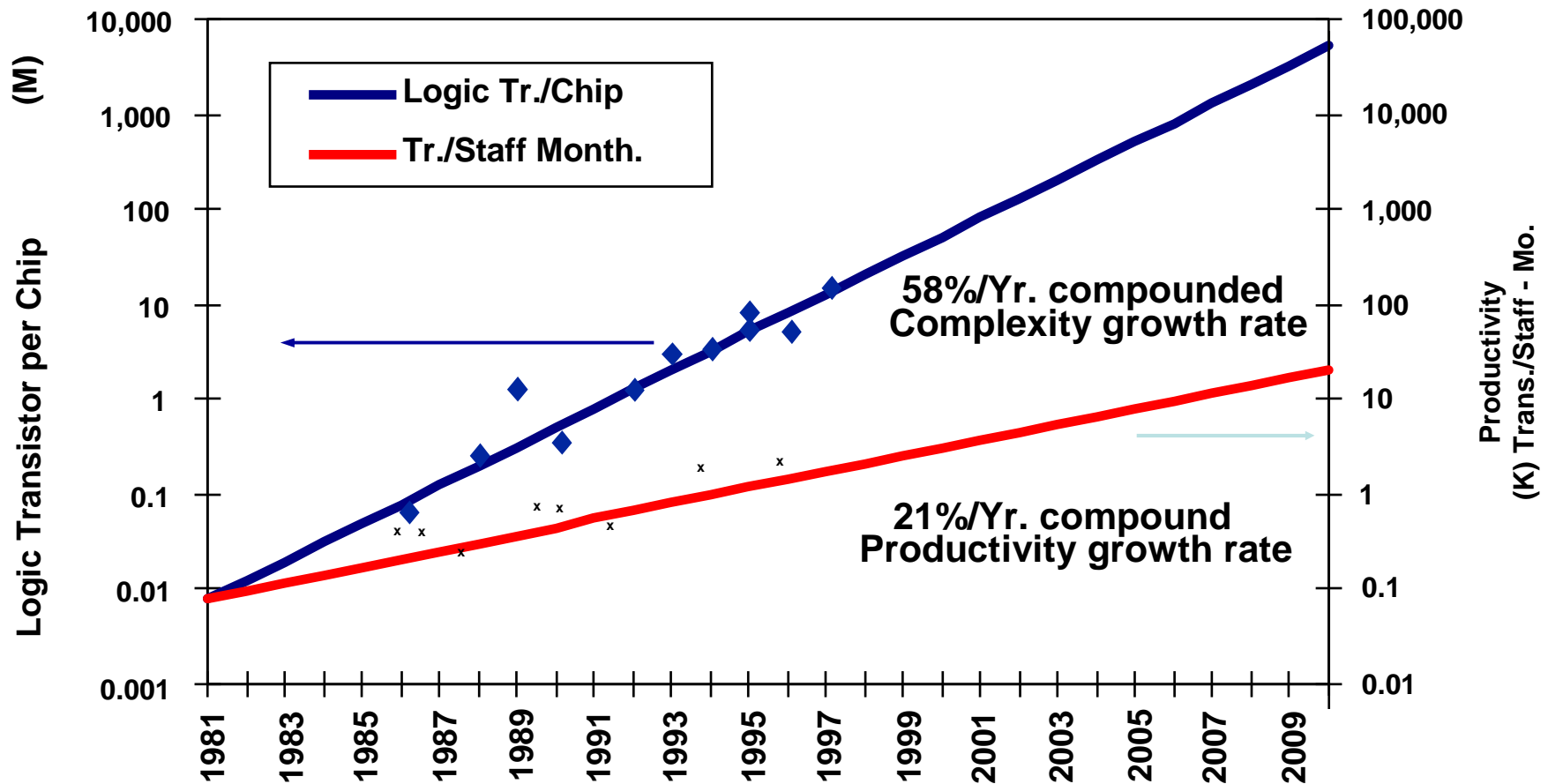
Jordi Cortadella, Univ. Politècnica Catalunya, Barcelona, Spain

Alex Yakovlev, Univ. of Newcastle upon Tyne, UK

With thanks M. Kishinevsky, A. Kondratyev and L. Lavagno

Topics

- What does Petrify do?
- Where has it been used?
- How does it compare with other tools?
- What is the future of async logic synthesis?



Source: Sematech

**Design Methodology and CAD tools
are essential to cope with complexity**

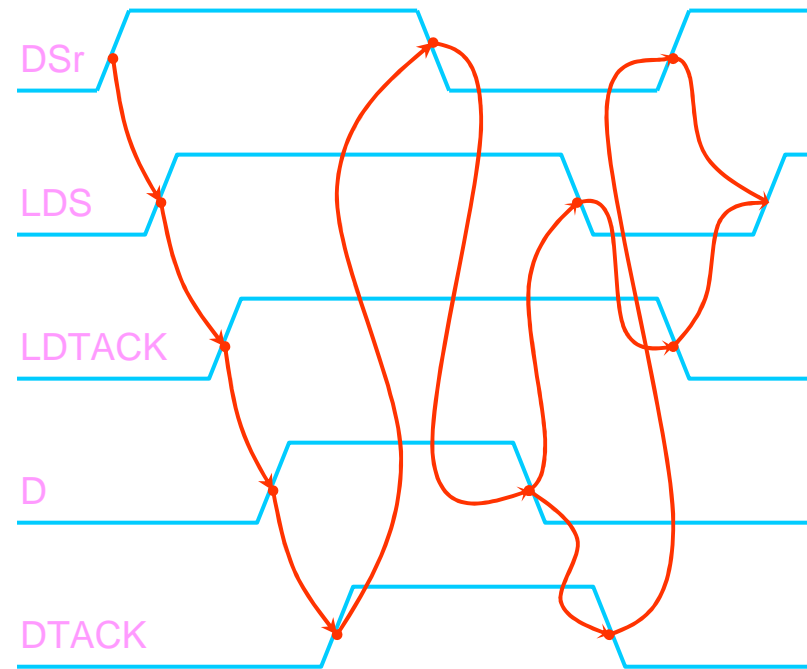
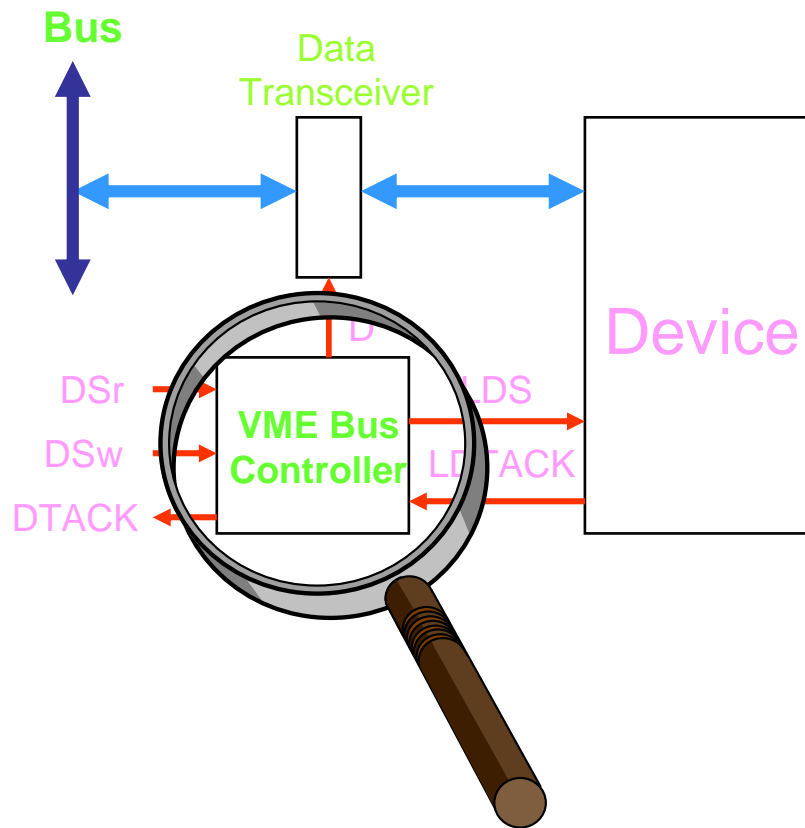
Role of logic synthesis

- Automatic logic synthesis has been a major success story in (synchronous) EDA in the 80s and 90s
- The key model behind it is FSM
- Why auto-synthesis for asynchronous circuits
 - to help widespread use of async circuits
 - async circuits are highly concurrent – even harder to design manually
 - The key model is Petri nets (captures concurrency in natural form)

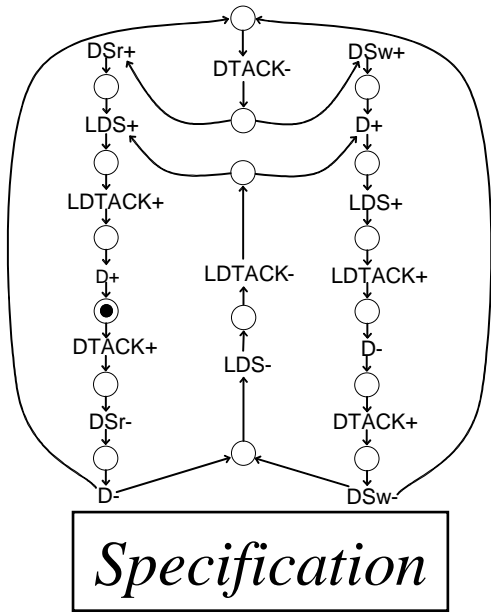


Applied research
(tools and experiments)

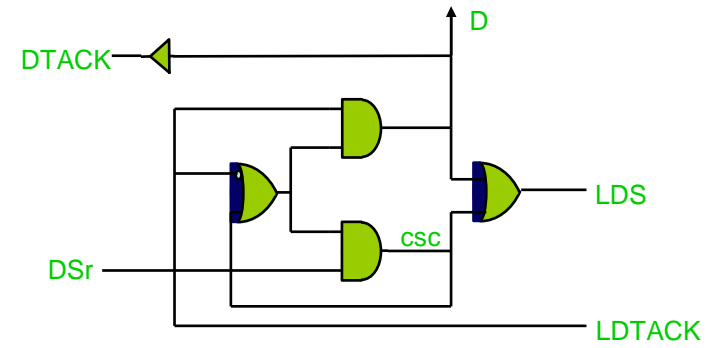
Basic research
(well-founded theory)



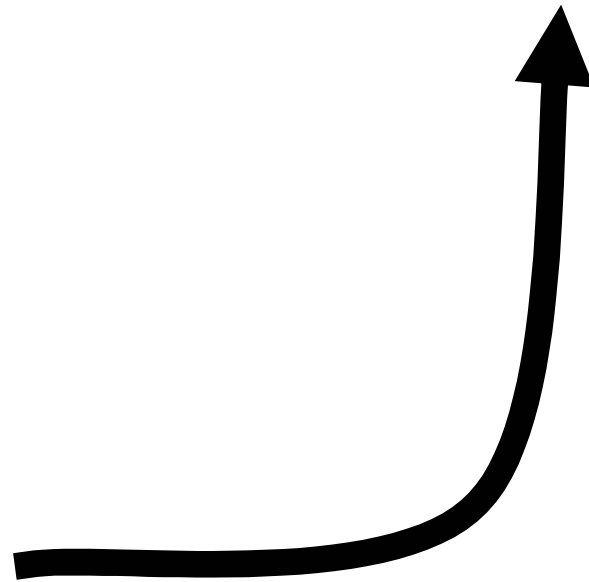
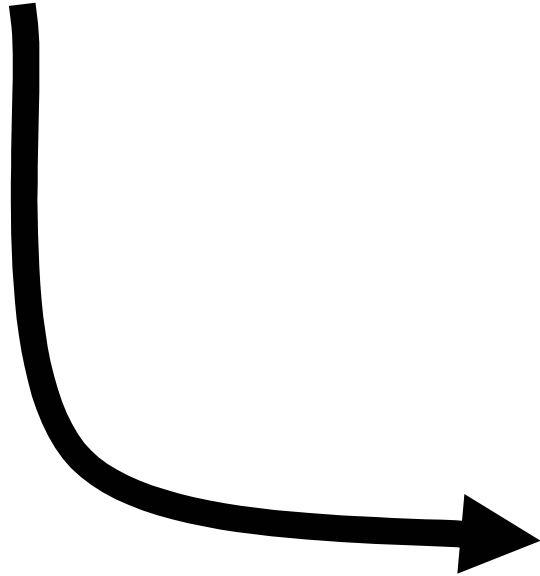
Read Cycle

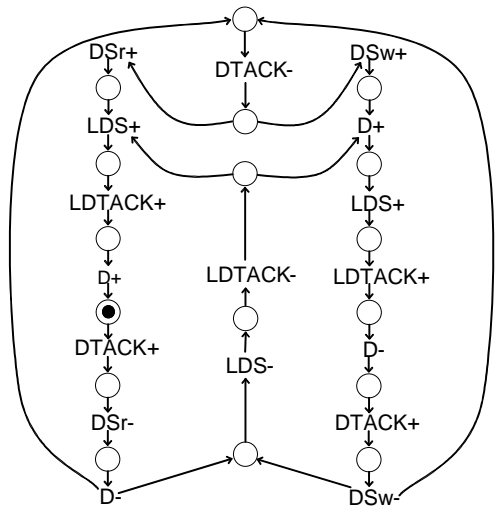


synthesis



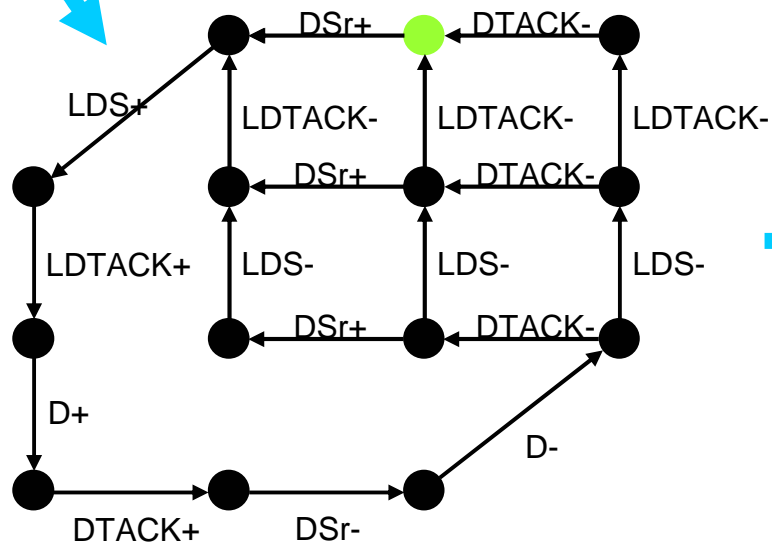
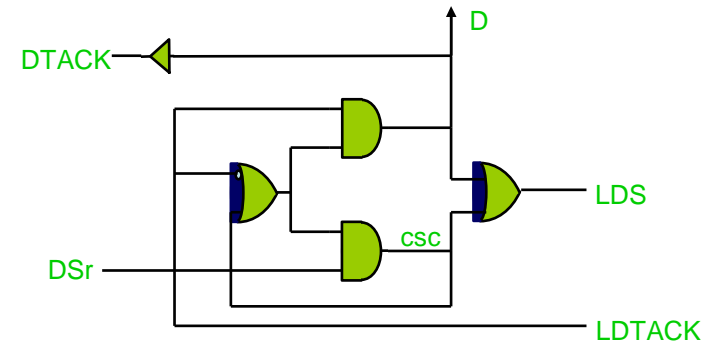
Implementation



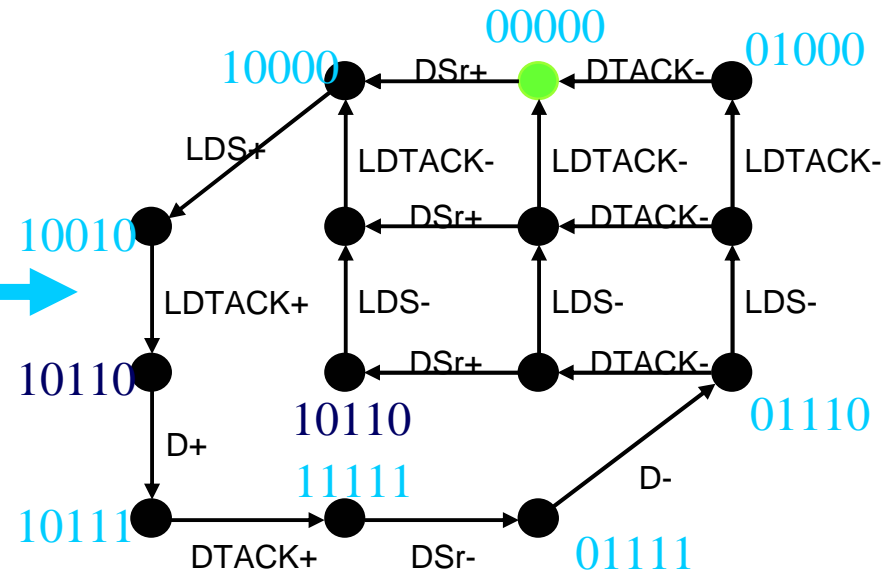


Signal Transition Graph
(Petri net)

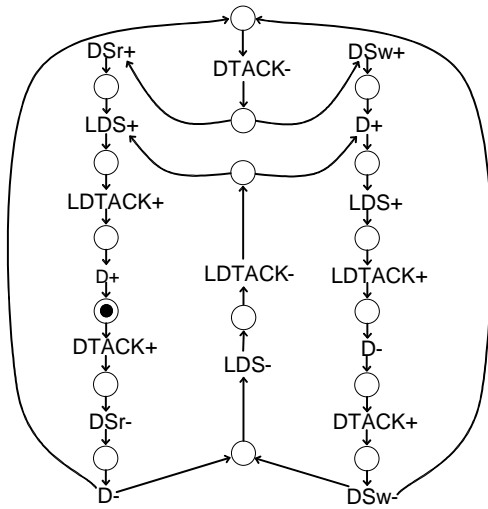
synthesis



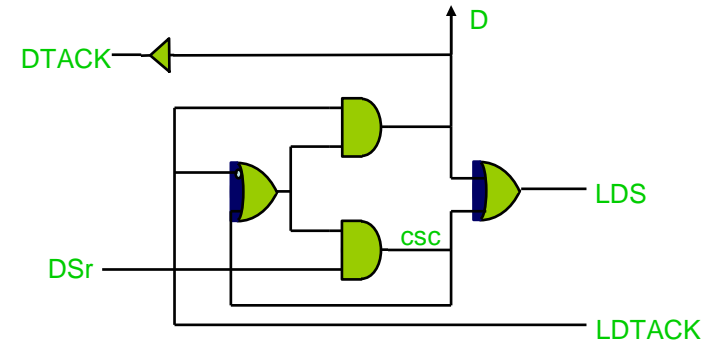
State Graph



Encoded State Graph



synthesis

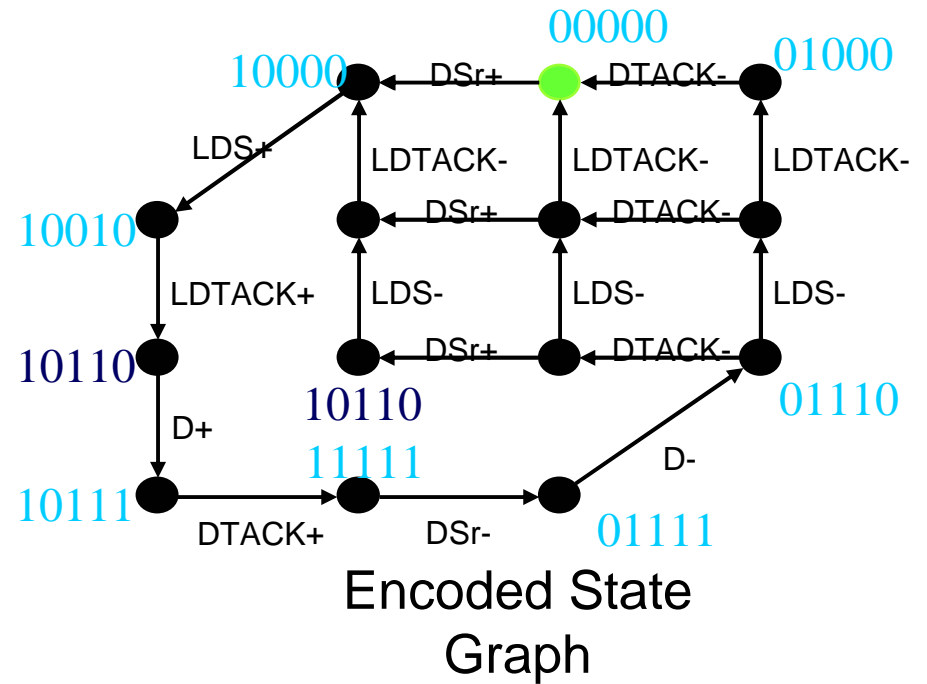


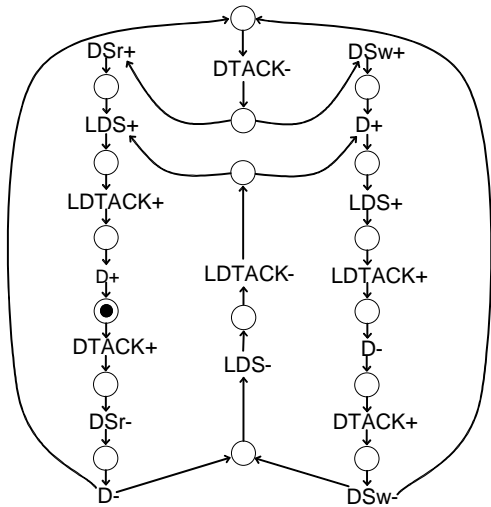
D	DTACK	DSr	00	01	11	10
LDTACK	00	0	0	-	1	
01	-	-	-	-		
11	-	-	-	-		
10	0	0	-	0		

LDS = 0

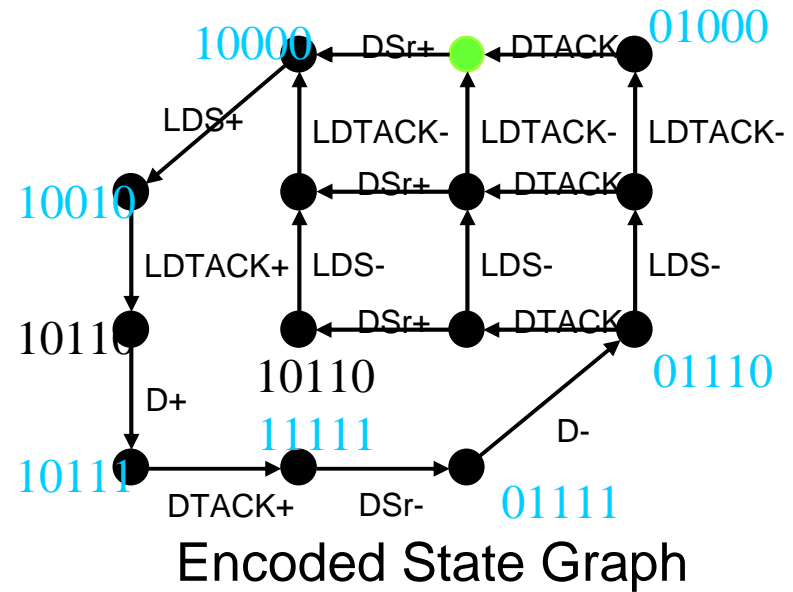
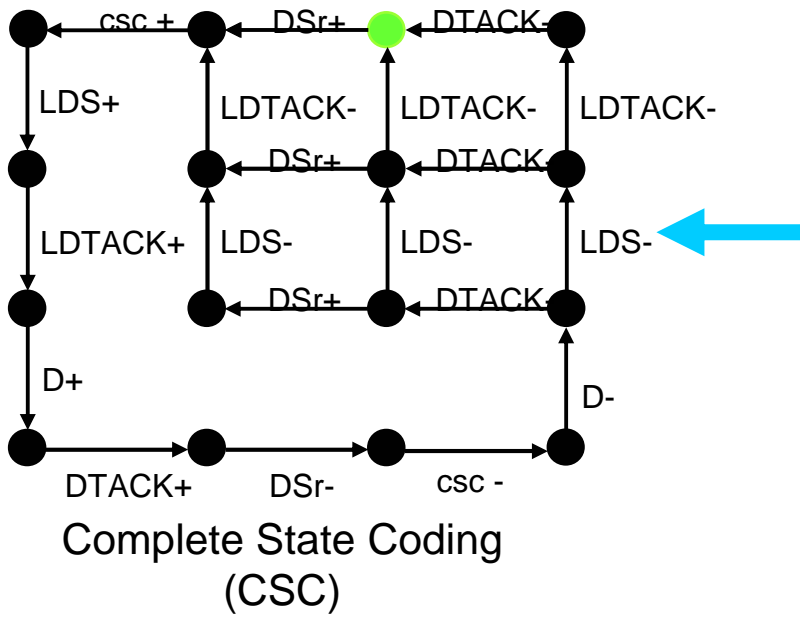
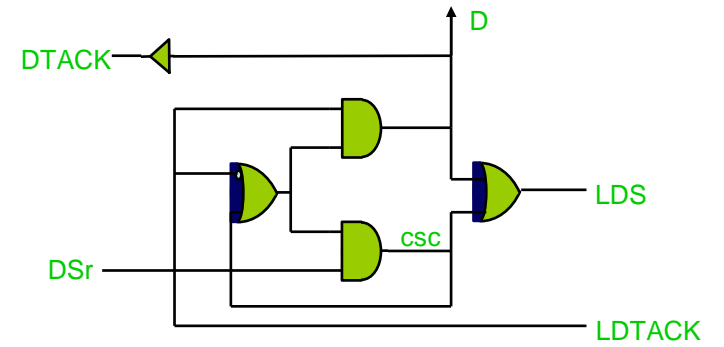
D	DTACK	DSr	00	01	11	10
LDTACK	00	-	-	-	1	
01	-	-	-	-		
11	-	1	1	1		
10	0	0	-	0/1?		

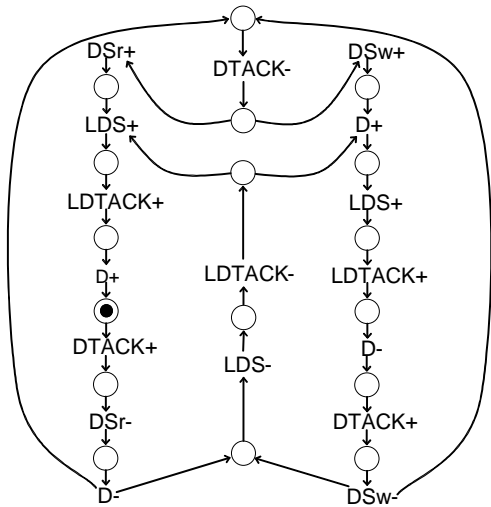
LDS = 1



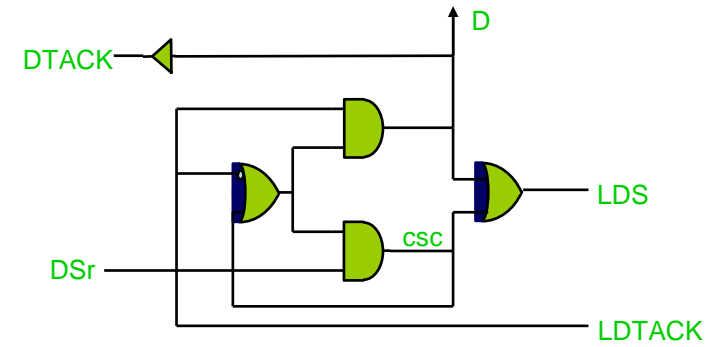


synthesis





synthesis



Logic asynchronous circuit



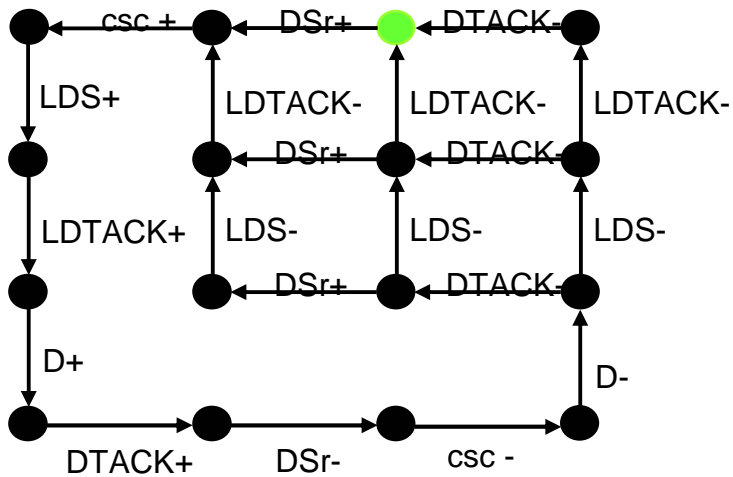
Boolean equations:

$$LDS = D \vee csc$$

$$DTACK = D$$

$$D = LDTACK$$

$$csc = DSr$$



Complete State Coding (CSC)

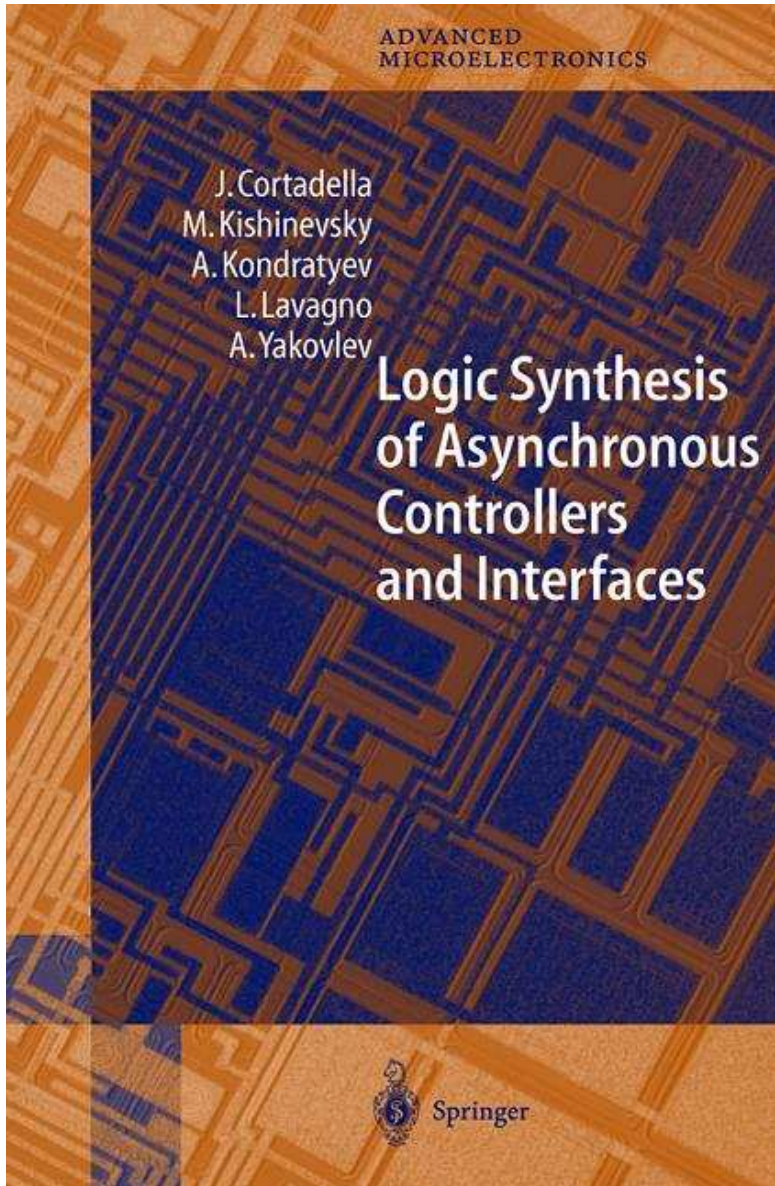
Main Features

- Synthesis of circuits from STGs
 - State encoding and decomposition
 - In complex gates, and in given libraries
 - Under timing assumptions
- Synthesis of Petri nets from Transition Systems
 - Extraction of concurrency (using theory of regions)
 - Visualisation and backannotation

Dissemination of results

*Books, tutorials, journals,
conferences, ...*

Book:

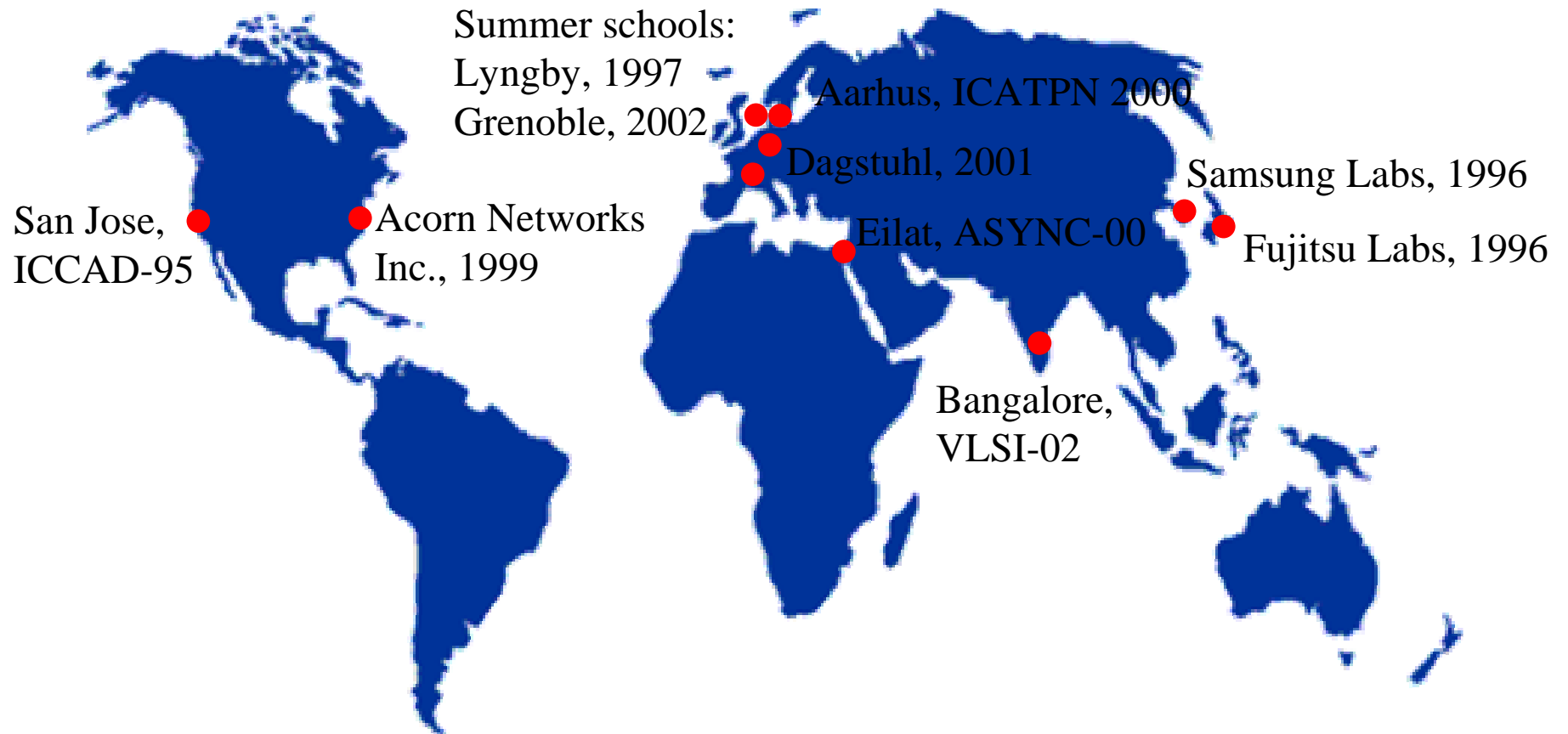


Previous books by the team:

M. Kishinevsky, A. Kondratyev, A. Taubin
and V. Varshavsky,
*Concurrent Hardware: The theory and
practice of self-timed design,*
John Wiley & Sons, 1994

L. Lavagno and A. Sangiovanni-Vincentelli,
*Algorithms for Synthesis and Testing of
Asynchronous Circuits,*
Kluwer, 1993.

Tutorials



Publications and Ph.D. thesis

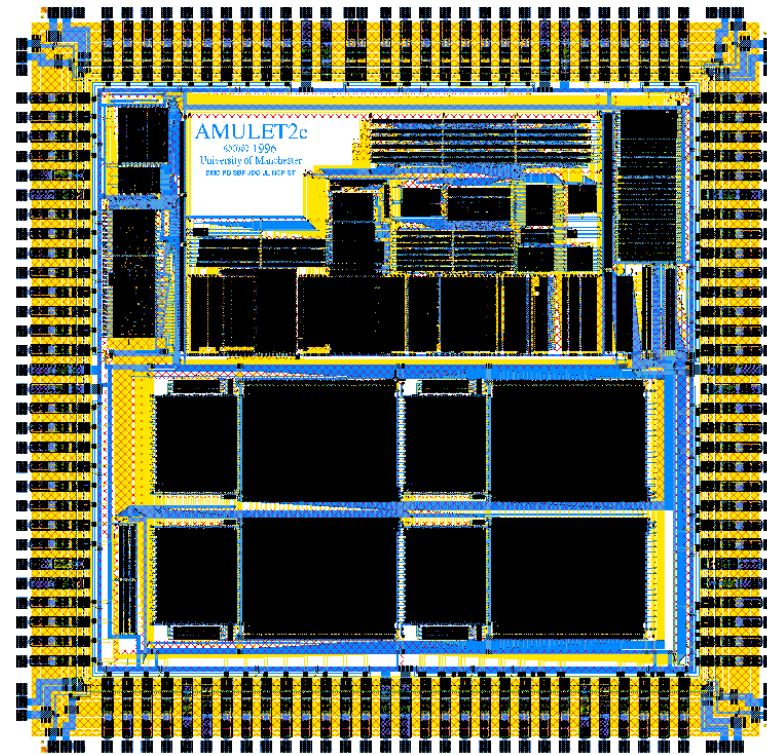
- 13 journal papers
 - *Proceedings of the IEEE, IEEE Trans. on Computers, IEEE Trans. on CAD, J. of Circuit Systems and Computers, IEICE Trans. on Information Systems, Formal Methods in Systems Design, IEEE Trans. on VLSI, IEEE Design & Test, Integration: The VLSI Journal*
- 23 conference papers
 - *ICCAD, DAC, ASYNC, ICATPN, EDTC, ...*
- Invited lectures
 - *DAC, ICATPN, CSD, HDPN, ACiD-WG*
- 4 Ph.D. thesis and 8 in progress

Practical impact of the research

Prototypes,
Teaching and
Research

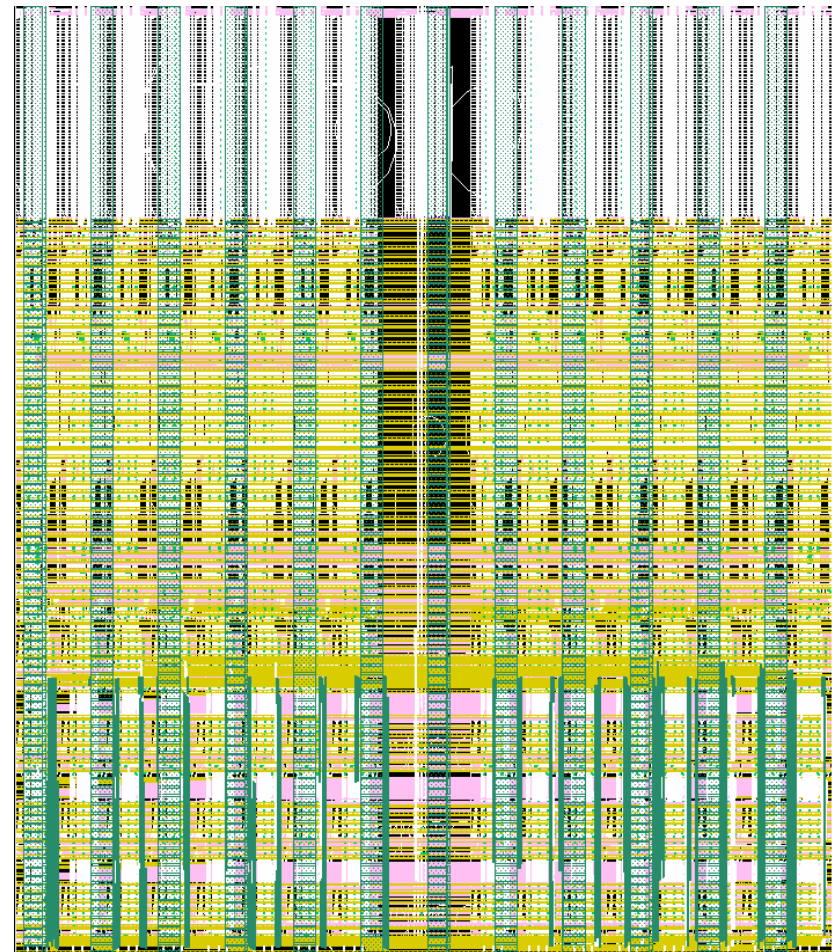
Used in several prototypes

- AMULET microprocessors (Manchester University)



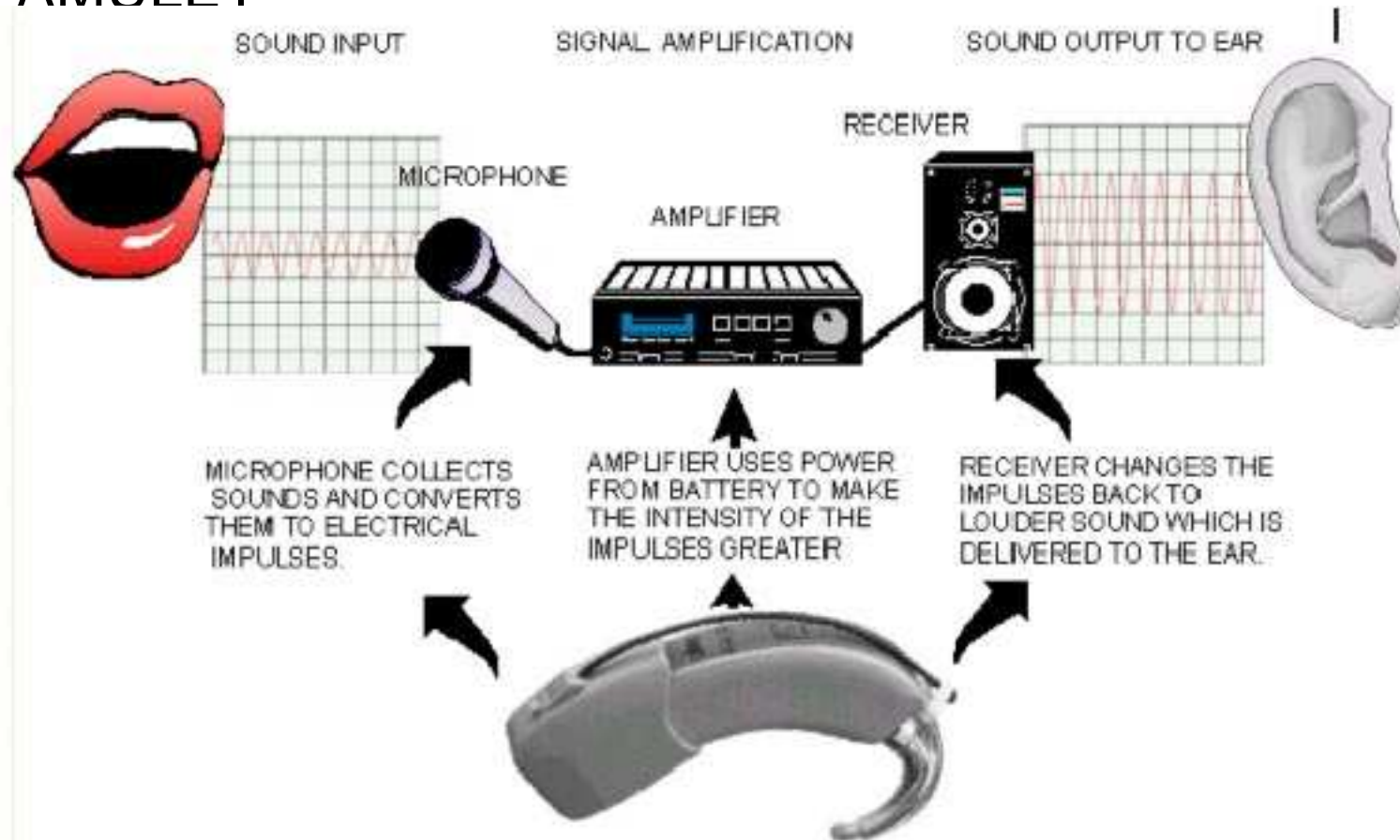
Used in several prototypes

- AMULET microprocessors (Manchester University)
- Instruction-length decoder (Intel Corporation)



Used in several prototypes

- AMULET



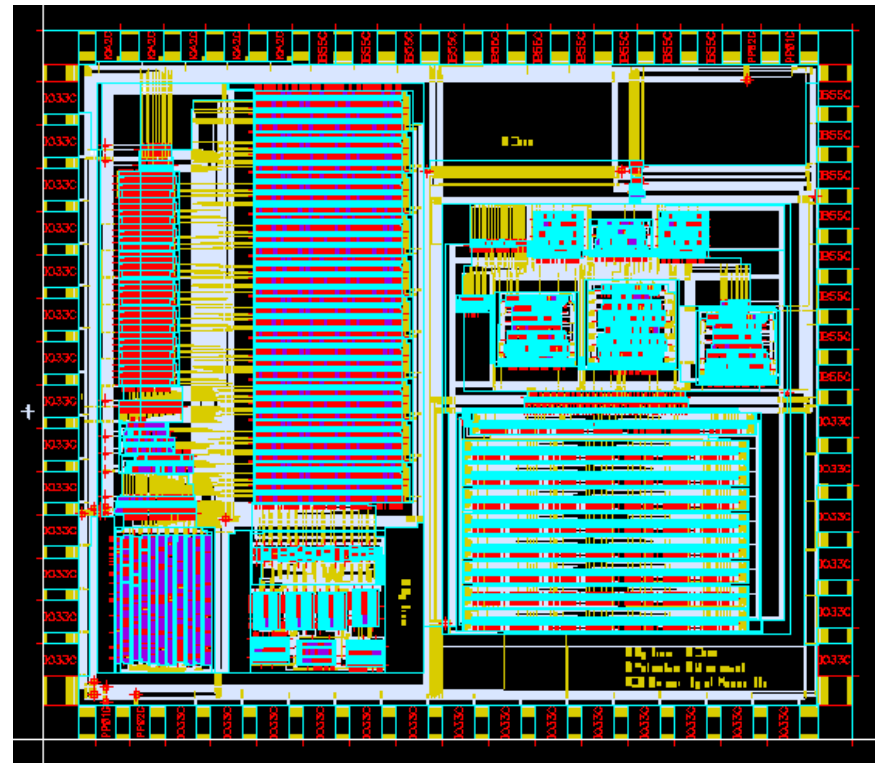
Used in several prototypes

- AMULET
microprocessors
(Manchester University)
- Instruction-length
decoder
(Intel Corporation)
- DSP for hearing aids
(Tech. Univ. of Denmark)
- Smart cards
(Cambridge University)



Used in several prototypes

- AMULET microprocessors (Manchester University)
- Instruction-length decoder (Intel Corporation)
- DSP for hearing aids (Tech. Univ. of Denmark)
- Smart cards (Cambridge University)
- Arbiters and GALS circuits (McGill University)
- and others
 - Theseus Logic
 - AT&T Cambridge
 - ...



- HADIC
(chip for async communication)
Univ. of Newcastle upon Tyne

Teaching (besides our groups)

- *One chapter of Furber & Sparso's book*
- *Technical University of Denmark*
- *Technion (Israel)*
- *University of Kaiserslautern (Germany)*
- *Boston University (USA)*
- *Indian Institute of Technology (Mumbai, India)*
- *University of Aizu (Japan)*
- *McGill University (Canada)*
- *Oregon State University (USA)*
- ...

Used in research

- *University of Manchester (UK)*
- *South Bank University (UK)*
- *IIT Mumbai (India)*
- *CSEM (Switzerland)*
- *University of North Carolina (USA)*
- *KAIST (Korea)*
- *University of Augsburg (Germany)*
- *Universidad de Zaragoza (Spain)*
- *... and many individuals using the tool by their own*

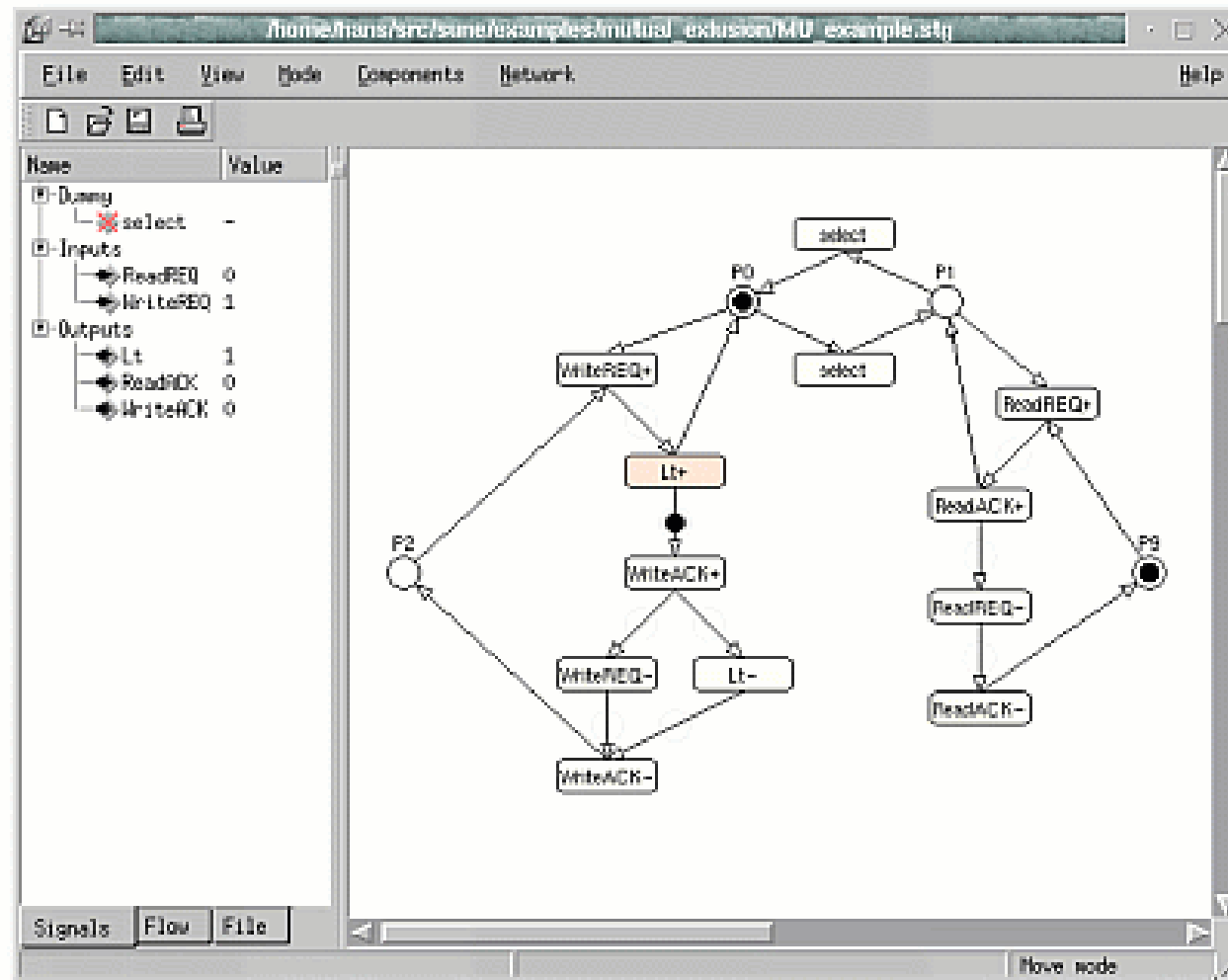
Comparison with other tools doing control synthesis

- Minimalist: based on an FSM model, uses fundamental mode (Petrify uses input-output mode), less power in terms of handling concurrency
- ATACS: uses event-based notation, but does not seem to have the power of solving state encoding problems; allows relative timing but without backannotation and visualisation

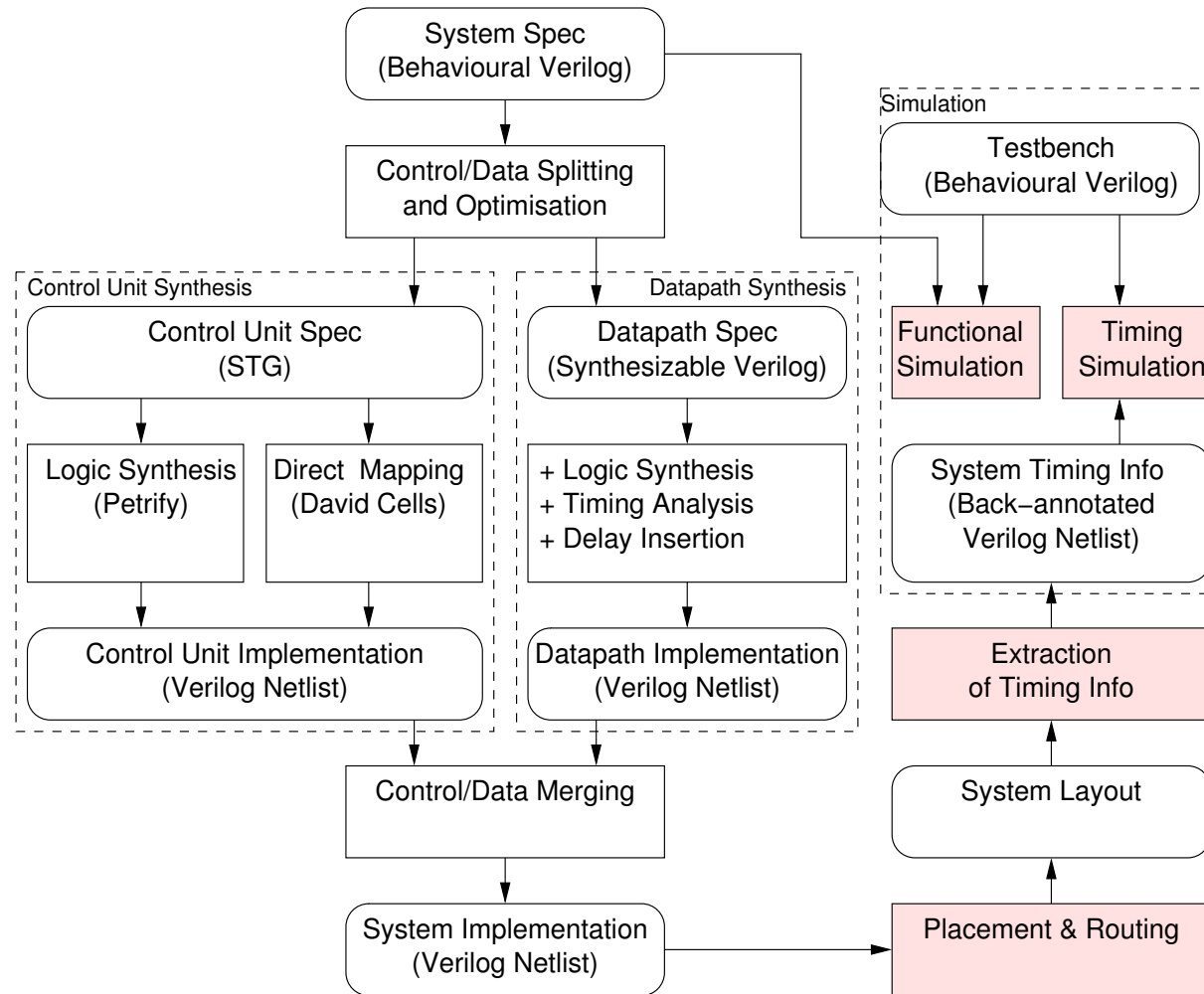
Other tools complementing Petrify design flow

- Graphical STG editor: VSTGL
- HDL front end: Pipefitter
- CSC visualization tool: ConfRes
- Verification with timing assumptions: Transyt
- Large STG analysis using unfoldings: Punf/CLP
- Direct mapping of STG: OptiMist
- DI process algebra front-end: di2pn
- STG Decomposition by contraction: DESI
- STG+XBM co-synthesis: CASCADE

Input specification: VSTGL (Technical University of Denmark)



Possible future design flow (based on pipefitter)

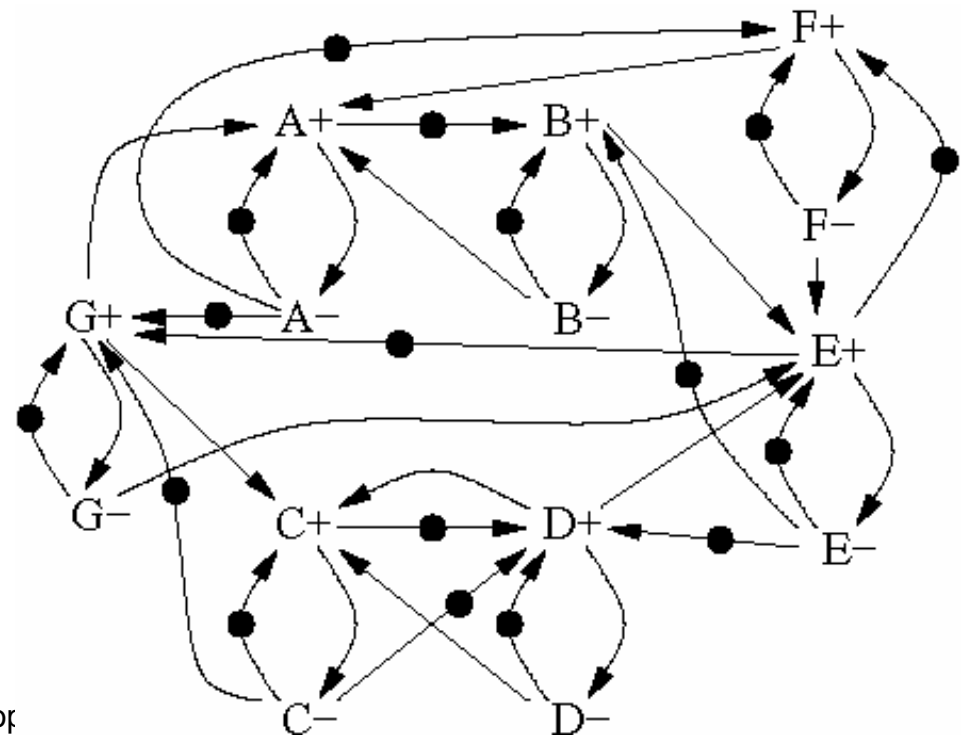
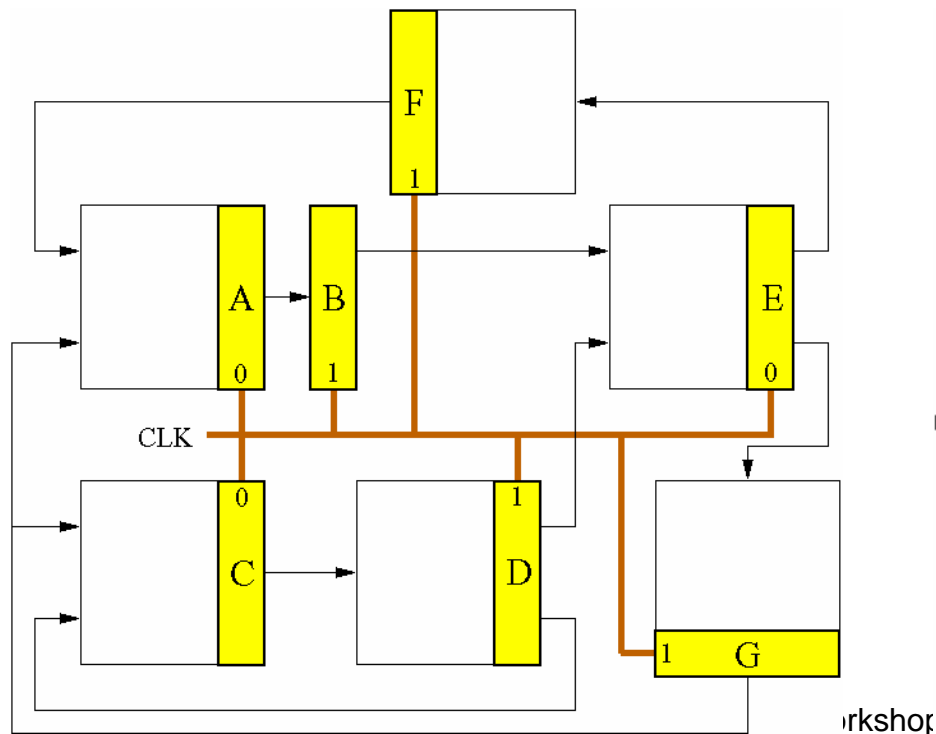


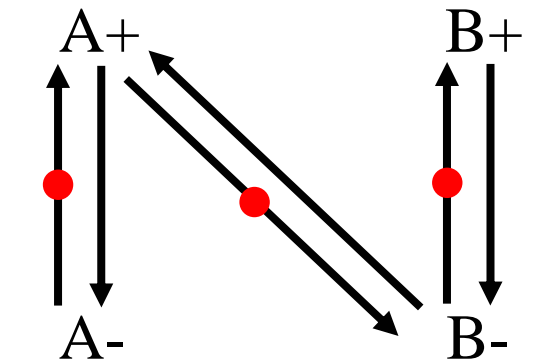
Future for Petrify

- Better front-end (CSP-like and HDLs)
- Better state space management (unfoldings, structural methods)
- Better use of direct mapping (at appropriate levels)
- Decomposition at various levels (CSP, labelled PN, STG, logic)
- Applications: desynchronisation and GALs, interfaces, control logic

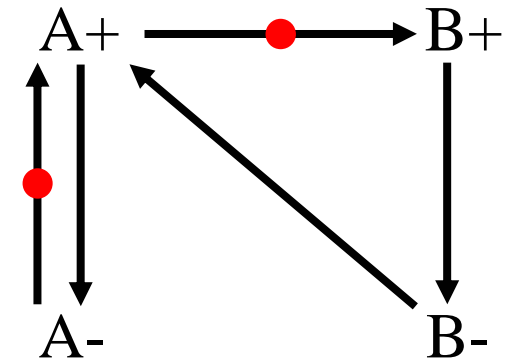
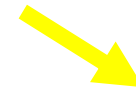
Desynchronisation (Async 2004)

- The de-synchronization model provides an abstraction of the timing behavior

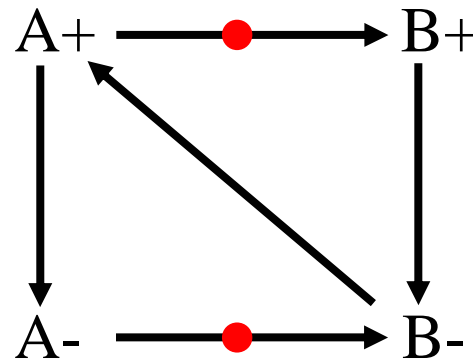
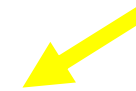




de-synchronization model



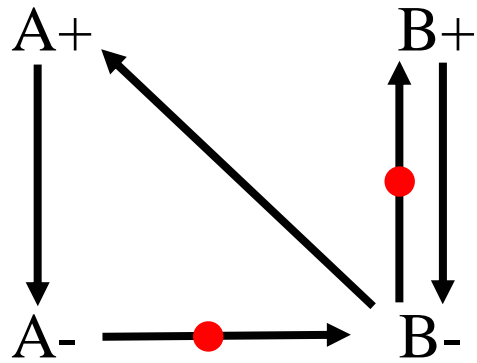
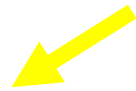
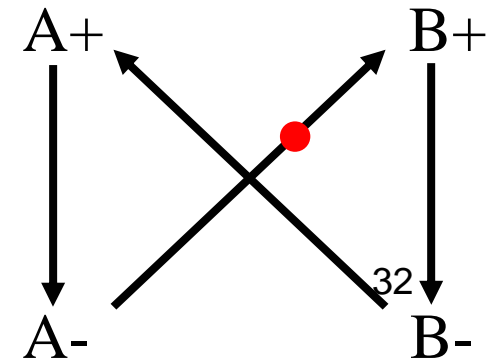
GasP, IPCMOS



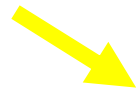
semi-decoupled (Furber & Day)



non-overlapping



fully decoupled (Furber & Day)



simple 4-phase

