Asynchronous Logic and GALS Design : Principles and State-of-the-Art

Alex Yakovlev, Newcastle University, UK
Jens Sparsø, Technical University of Denmark, Denmark
Yvain Thonnart, Minatec - CEA/LETI, France
Pascal Vivet, Minatec - CEA/LETI, France

Asynchronous Logic and GALS Design: Principles and State of the Art

Outline :

- **Introduction to Asynchronous Logic (*A. Yakovlev*)**
- **Practical Asynchronous Design Automation (*J. Sparsø*)**

break

- **GALS, an intermediate design style (*Y. Thonnart*)**
- **State-of-the-art of asynchronous logic in the industry (*P. Vivet)***

2

Part I: Introduction to Asynchronous Logic

Alex Yakovlev
School of EECE, Newcastle University, UK
alex.yakovlev@ncl.ac.uk
http:async.org.uk

Part 1: Outline

- Introduction
  - Basic principles of Asynchronous Behaviour
  - Motivation: advantages and problems
- Principles of Asynchronous Logic Design
  - Basics of design: signalling and encoding schemes, data and control path blocks
  - Classes of Asynchronous Circuits
  - Models of Asynchronous Control
- System level Design Issues
  - Arbitration, Synchronisation, Metastability
  - Asynchronous Communications

Part 1: Outline

- Introduction
  - Basic principles of Asynchronous Behaviour
  - Motivation: advantages and problems
- Principles of Asynchronous Logic Design
  - Basics of design: signalling and encoding schemes, data and control path blocks
  - Classes of Asynchronous Circuits
  - Models of Asynchronous Control
- System level Design Issues
  - Arbitration, Synchronisation, Metastability
  - Asynchronous Communications

Asynchronous Behaviour

- Synchronous vs Asynchronous behaviour in general terms, examples:
  - Orchestra playing with vs without a conductor
  - Party of people having a set menu vs a la carte
- Synchronous means all parts of the system acting globally in tact, even if some or all part ‘do nothing’
- Asynchronous means parts of the system act on demand rather than on global clock tick
- Acting in computation and communication is, generally, changing the system state
- Synchrony and Asynchrony can be in found in CPUs, Memory, Communications, SoCs, NoCs etc.

Clocked design

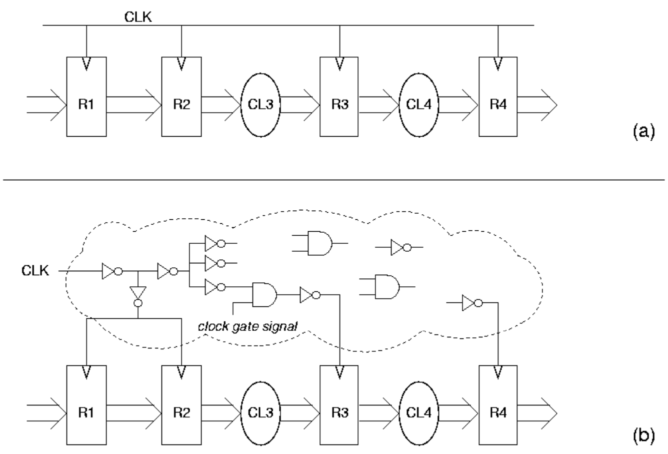
- The dominating design style
- Two underlying assumptions:
  - Binary signals: {0; 1}
  - Discrete time: ”clock ticks” (data validity implicitly assumed at clock events)
- It’s an abstraction (the world is analogue, i.e., continuous time varying signals).
- Design is simple provided assumptions hold.
  - What assumptions?

Alex Yakovlev
Newcastle University

Synchronous clocking

How we think

What we design



Alex Yakovlev
Newcastle University

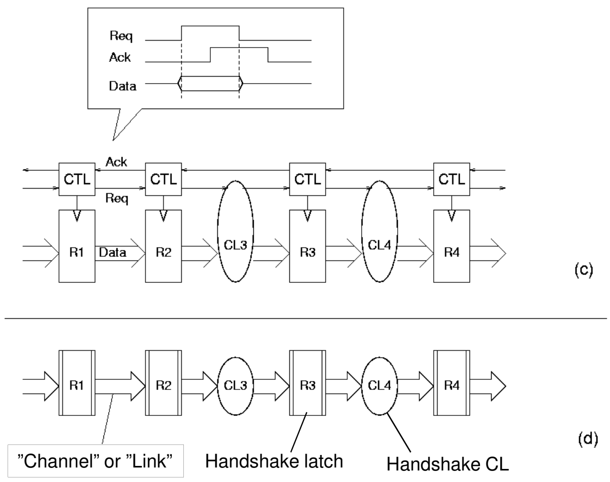
Async or Clockless design

- Underlying assumption:
  - Binary signals: {0; 1}
  - Continuous time: (data validity is explicitly indicated along with the data) ... i.e. we give up the discrete-time assumption.
- Handshaking between registers/latches
- ”Aperiodic local clocks” derived from handshake signals

Asynchronous handshaking

What we design

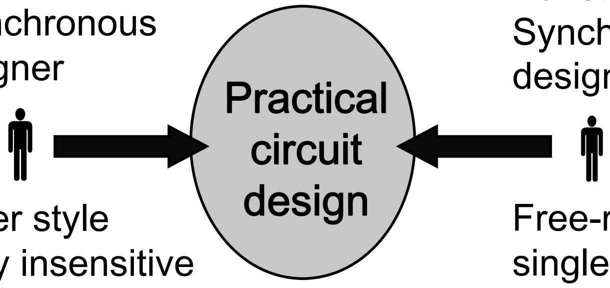
How we think



Synchronous / Asynchronous?

Purist Asynchronous designer

Muller style delay insensitive 4-phase dual-rail



Purist Synchronous designer

Free-running single-phase global clock

Alex Yakovlev
Newcastle University

Clocking: problems / challenges

- **Clock skew and clock distribution**
  - **becoming increasingly difficult to handle**
- **Robustness to PVT variations**
  - **can't easily adjust to Vdd fluctuations, Vth variability etc.**
- **Timing closure**
  - **becoming increasingly difficult to obtain**
- **The clock wastes power**
  - **it causes considerable unnecessary activity**
- **The clock forces all parts of the system to operate at the same speed**
  - **parts have different natural speeds**
- **The clock generates EMC problems**
  - **It produces high radio powers at harmonics of clock frequency**

Alex Yakovlev
Newcastle University

Advantages of Asynchronous

- **Low power consumption due to:**
  - **Automatic fine-grain clock gating and variable length clocks.**
  - **Automatic instantaneous stand-by (leakage only) at arbitrary granularity in time and function.**
  - **Distributed localized control.**
  - **More architectural options/freedom.**
  - **More freedom to scale the supply voltage.**

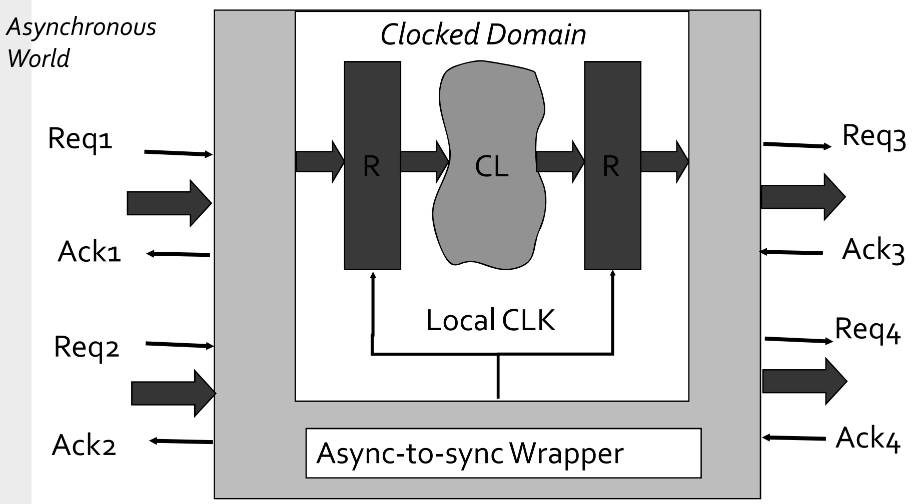
Advantages of Asynchronous

- **Modularity**
  … timing is explicit at interfaces
- **Higher operating speed**
  … speed is determined by actual case latencies
- **Less EMI and smoother Idd**
  … the local ”clocks” tend to tick at random points in time
- **Low sensitivity to PVT variations**
  … timing based on matched delays (or even *delay insensitive*)
- **Secure chips**
  ... white noise current spectrum

Why Asynchronous is hard?

- Overhead (area, speed, power)
  - Control and handshaking
- Hard to design
  - yes and no, ... It's different – there are very many styles and variants to go and one can easily get confused which is better
- Few CAD tools
  - the situation is improving
  - not as high level as sync
- Test
  - Possible, but not as mature as sync

Globally Async Locally Sync (GALS)



Part 1: Outline

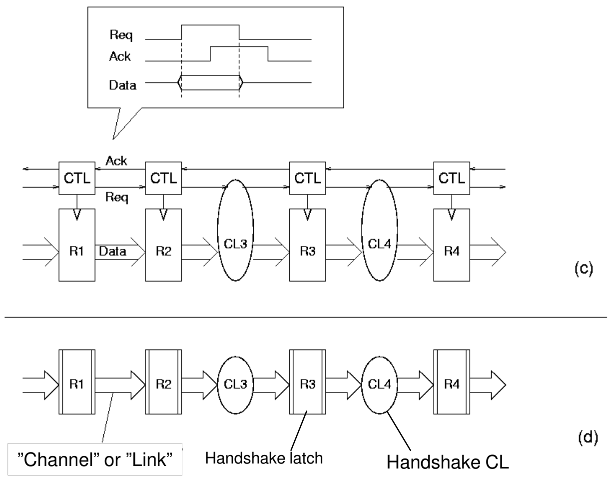
- Introduction
  - Basic principles of Asynchronous Behaviour
  - Motivation: advantages and problems
- Principles of Asynchronous Logic Design
  - Basics of design: signalling and encoding schemes, data and control path blocks
  - Classes of Asynchronous Circuits
  - Models of Asynchronous Control
- System level Design Issues
  - Arbitration, Synchronisation, Metastability
  - Asynchronous Communications

Alex Yakovlev
Newcastle University

Asynchronous handshaking

What we design

How we think

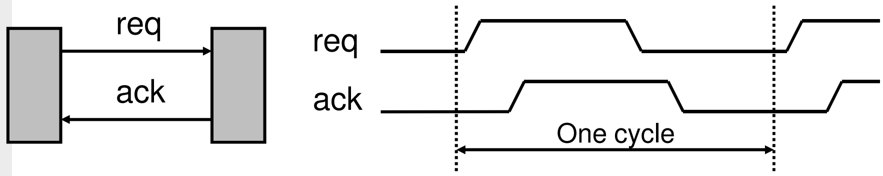


We need two things :
⇒ protocol *and* encoding.
⇒ These are orthogonal

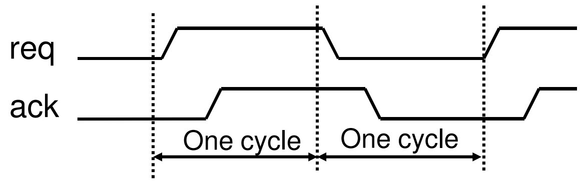
Alex Yakovlev
Newcastle University

Handshake Signalling Protocols

- **Level Signalling (RTZ or 4-phase)**



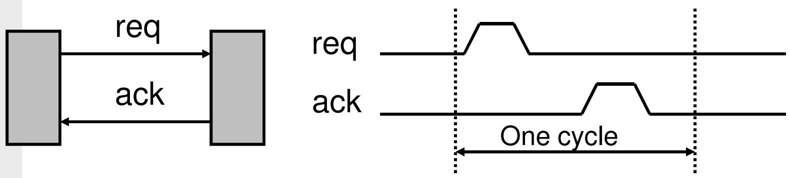
- **Transition Signalling (NRZ or 2-phase)**



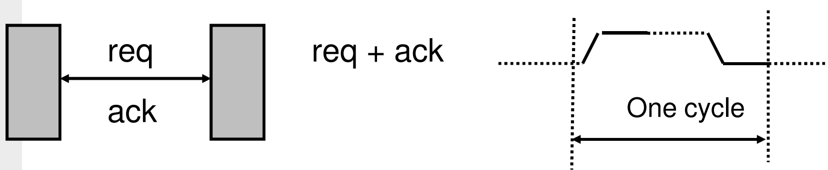
Alex Yakovlev
Newcastle University

Handshake Signalling Protocols

- **Pulse Signalling**



- **Single-track Signalling (GasP)**

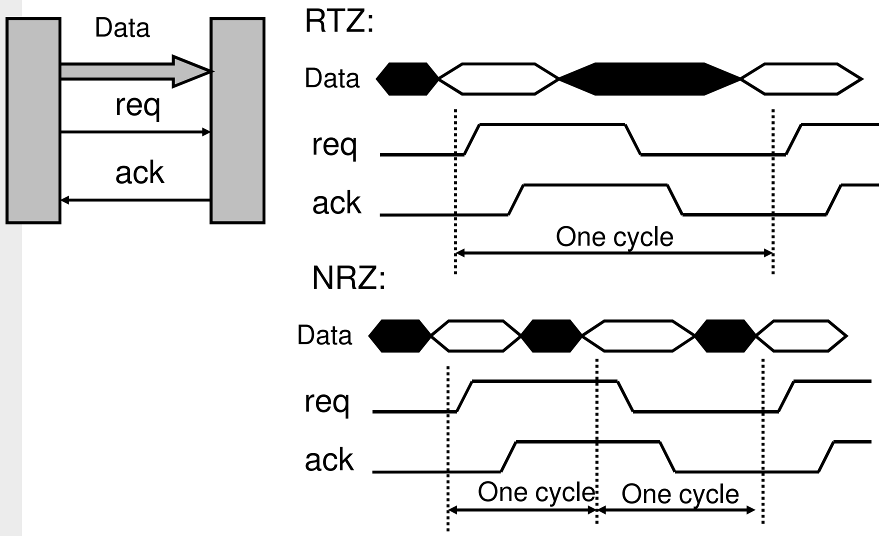


Alex Yakovlev
Newcastle University

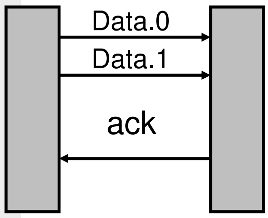
Data encoding

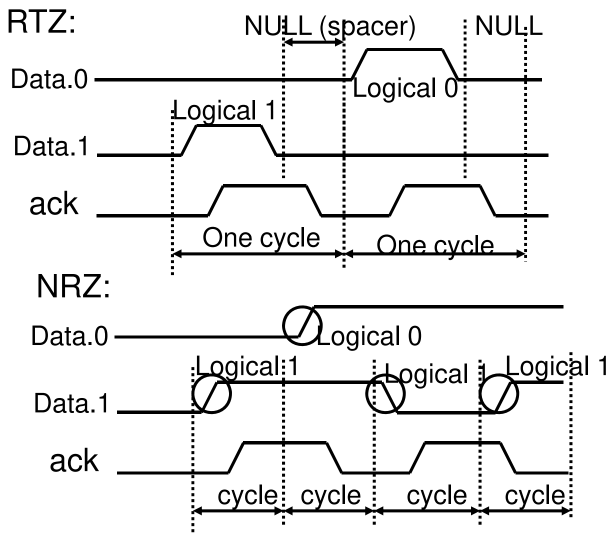
- **Bundled data**
  - **Code is positional binary, token is determined by Req+ signal; Req+ arrives with a safe set-up delay from data**
- **Delay-insensitive codes (tokens determined by the codeword values, require a spacer, or NULL, state if RTZ)**
  - **1-of-2 (Dual-rail per bit) – systematic code, encoding, decoding straightforward**
  - **m-of-n (n>2) – not systematic, i.e. incur encoding and decoding costs, optimal when m=n/2**
  - **One-hot ,1-of-n (n>2), completion detection is easy, not practical beyond n>4**
  - **Systematic, such as Berger, incur complex completion detection**

Bundled Data



DI encoded data (Dual-Rail)



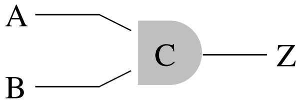


NRZ coding leads to complex logic implementation; special ways to track odd and even phases and logic values are needed, such as LEDR

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

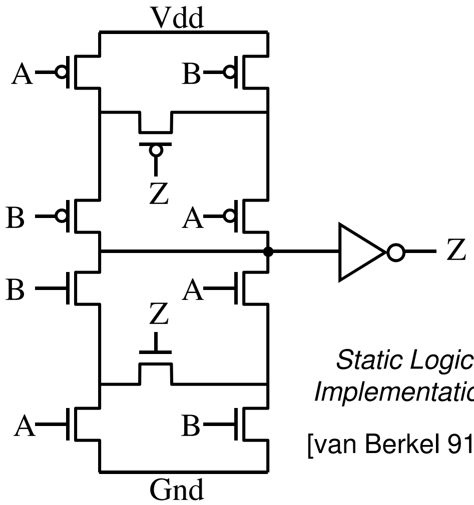
- **1-of-4:**
  - **0001=> 00, 0010=>01, 0100=>10, 1000=>11**
- **2-of-4:**
  - **1100, 1010, 1001, 0110, 0101, 0011 – total 6 combinations (cf. 2-bit dual-rail – 4 comb.)**
- **3-of-6:**
  - **111000, 110100, …, 000111 – total 20 combinations (can encode 4 bits + 4 control tokens)**
- **2-of-7:**
 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

The Muller C element



C

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

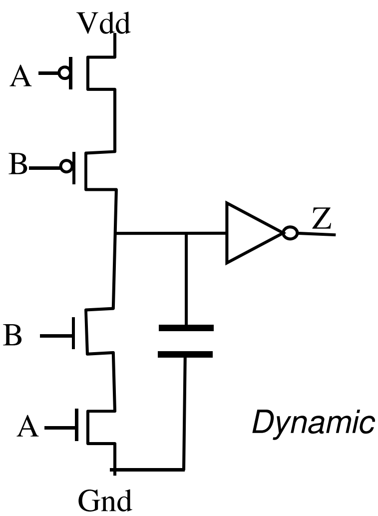


Static Logic Implementation

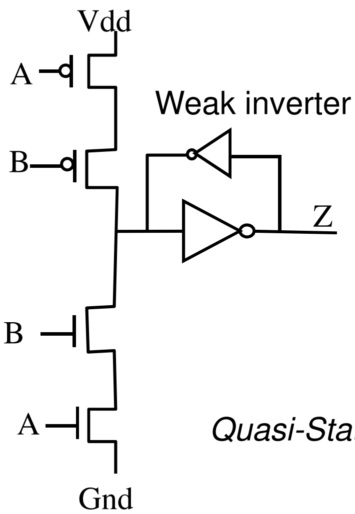
[van Berkel 91]

Alex Yakovlev
Newcastle University

C-element: Other implementations



Dynamic

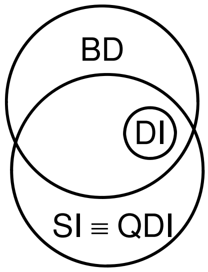


Quasi-Static

Alex Yakovlev
Newcastle University

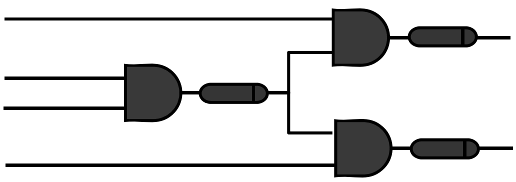
Delay models for async. circuits

- **Bounded delays (BD): realistic for gates and wires.**
  - **Technology mapping is easy, verification is difficult**
- **Speed independent (SI): Unbounded (pessimistic) delays for gates and "negligible" (optimistic) delays for wires.**
  - **Technology mapping is more difficult, verification is easy**
- **Delay insensitive (DI): Unbounded (pessimistic) delays for gates and wires.**
  - **DI class (built out of basic gates) is almost empty**
- **Quasi-delay insensitive (QDI): Delay insensitive except for critical wire forks (*isochronic forks*).**
  - **In practice it is the same as speed independent**

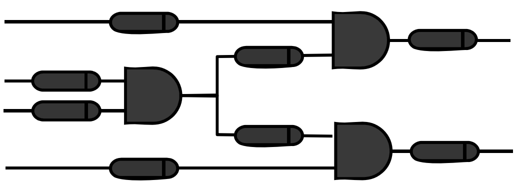


Gate vs wire delay models

- **Gate delay model: delays in gates, no delays in wires**



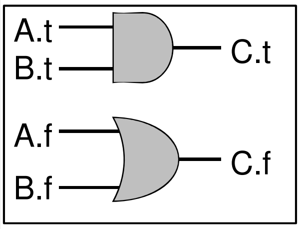
- **Wire delay model: delays in gates and wires**



Data Path Logic

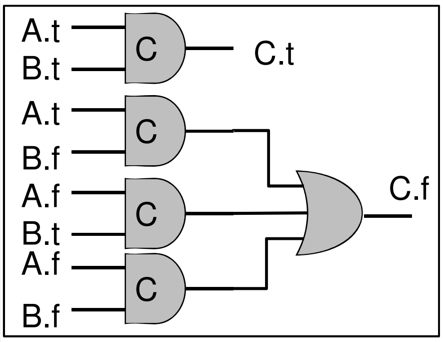
- **Dual-Rail type logic**

Dual-rail static logic



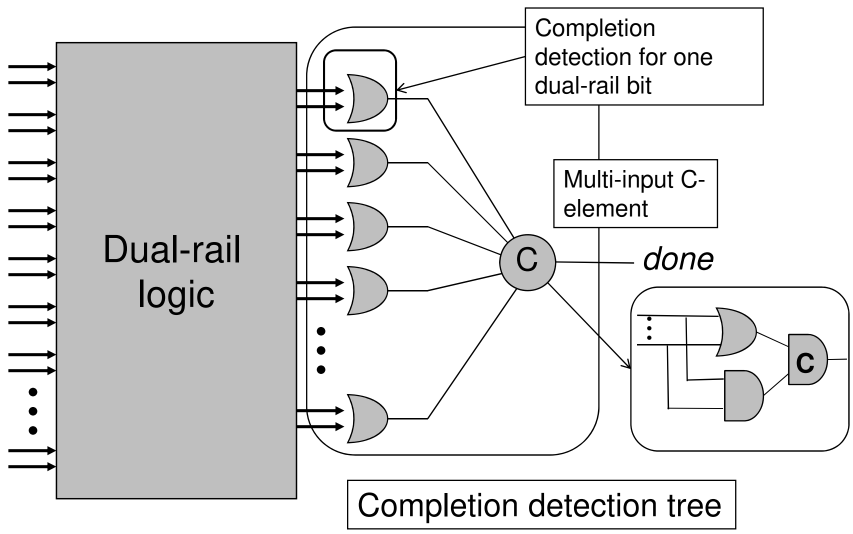
Dual-rail AND gate
with "early propagation"

Requires the environment to wait until inputs have transitioned from Null to Codeword and from Codeword to NULL



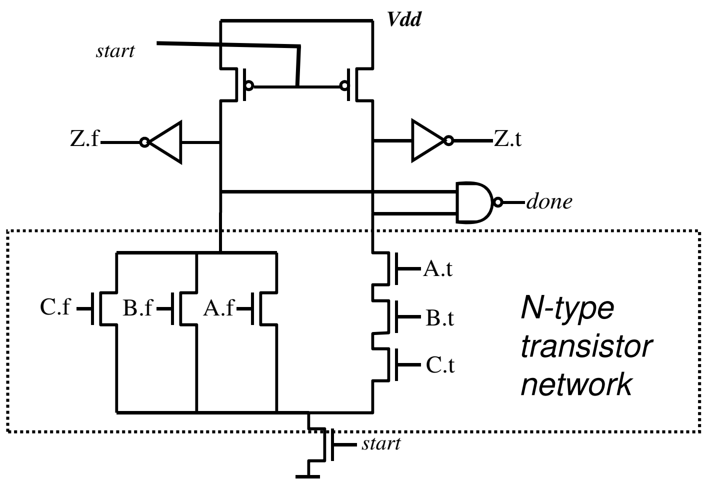
Dual-rail AND gate
with full input
acknowledgement

Completion detection



Alex Yakovlev
Newcastle University
29

Differential cascode voltage switch logic



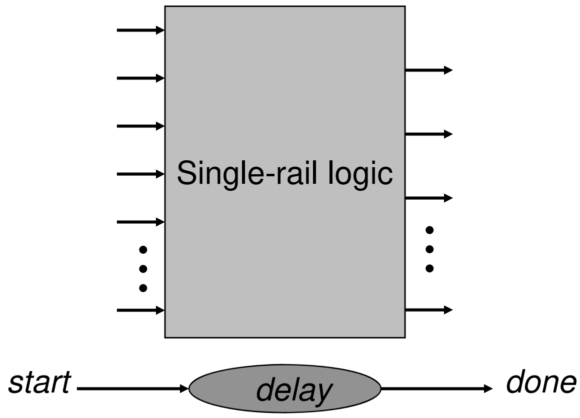
3-input AND/NAND gate

Alex Yakovlev
Newcastle University
30

Data Path Logic

- **Bundled-Data type logic**

Bundled-data logic blocks

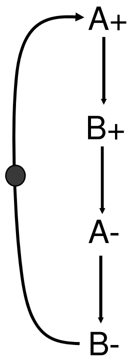


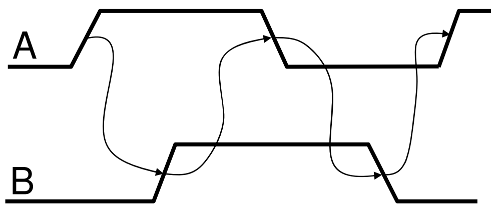
Conventional logic + matched delay

Control specification

Signal Transition Graph (STG)

Timing Diagram



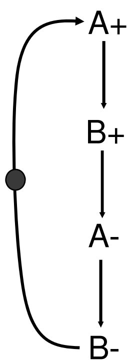


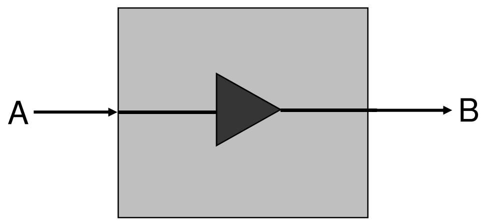
A input
B output

Alex Yakovlev
Newcastle University

33

Control specification

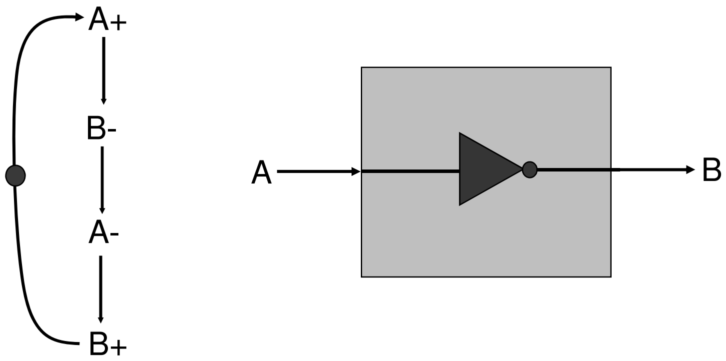




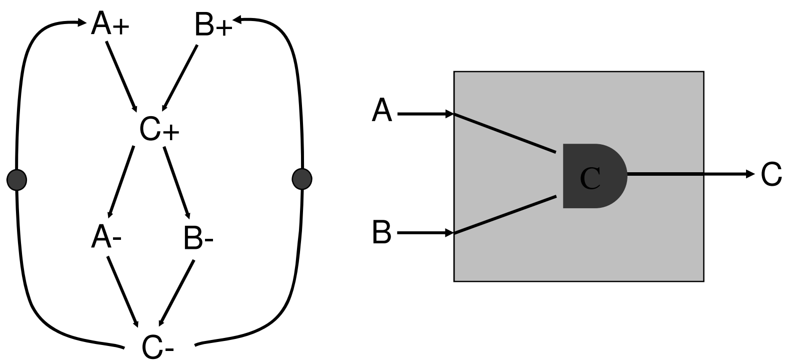
Alex Yakovlev
Newcastle University

34

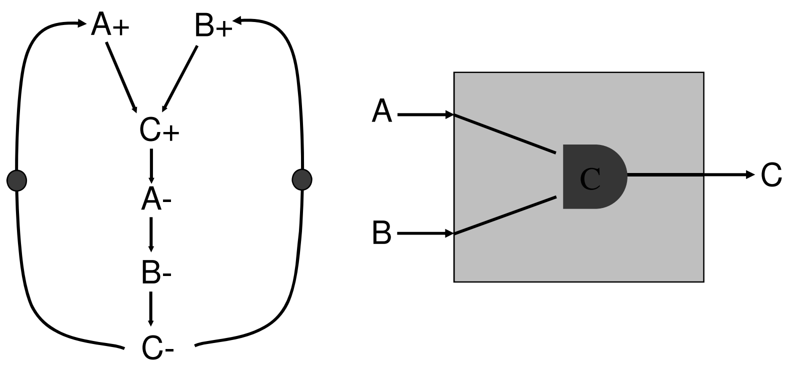
Control specification



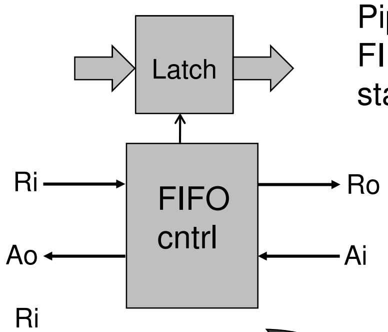
Control specification



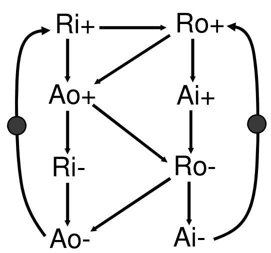
Control specification

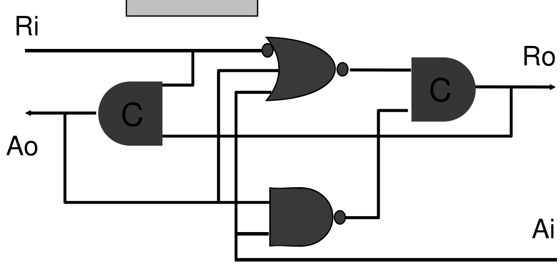


Control synthesis



Pipeline FIFO stage





Synthesis method and Petrify tool

J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno and A. Yakovlev, Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers, IEICE Trans. Inf. & Syst., Vol. E80-D, No.3, March 1997, pp. 315-325.
http://www.lsi.upc.edu/~jordicf/petrify/

Performance Analysis ?

- Performance in Synchronous Logic ?
  - RTL level ?
    - Efficient Timing Analysis tool computes the worst case delay and determines the clock period
  - Architecture level ?
    - Overall throughput and latency is in the number of clock cycles

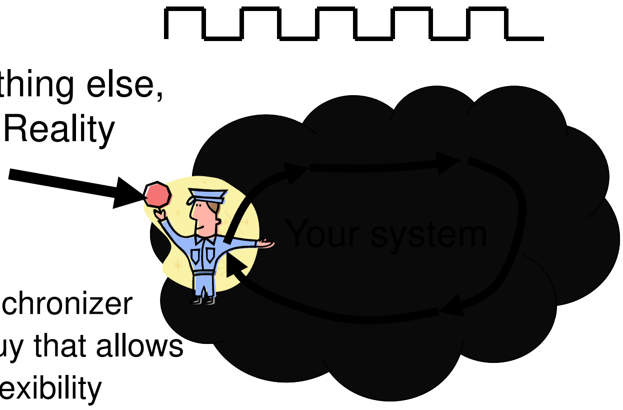
- Performance in Asynchronous logic ?
  - Local timing of cells can be easily computed
  - At handshake/component level, all timing parameters are variable
    - *Forward latency, Reverse Latency, Local Cycle Time, …*
    - Mean time computing everywhere, how to formalize ?
  - Architecture level ?
    - Dynamic «Elastic» pipeline
    - The overall picture depends of number of tokens, number of places, computation values, gate/delay values
  - In simple case (Token Ring)
    - The optimal throughput value can be computed
  - Some optimization and heuristic theory
    - *« Time separation of events »*
    - *« slack matching » theory*

Part 1: Outline

- Introduction
  - Basic principles of Asynchronous Behaviour
  - Motivation: advantages and problems
- Principles of Asynchronous Logic Design
  - Basics of design: signalling and encoding schemes, data and control path blocks
  - Classes of Asynchronous Circuits
  - Models of Asynchronous Control
- System level Design Issues
  - Arbitration, Synchronisation, Metastability
  - Asynchronous Communications

The digital IP world and the rest of the world

Everything else, or Reality

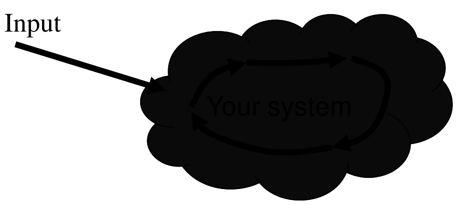


The synchronizer is the guy that allows timing flexibility

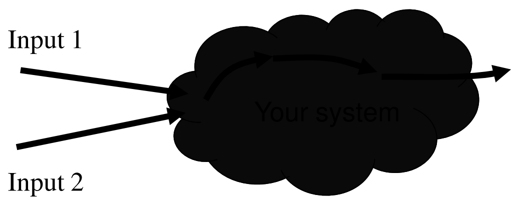
Alex Yakovlev
Newcastle University

Synchronizers and arbiters

- Synchronizer
  Decides which clock cycle to use for the input data



- Asynchronous arbiter
  Decides the order of inputs



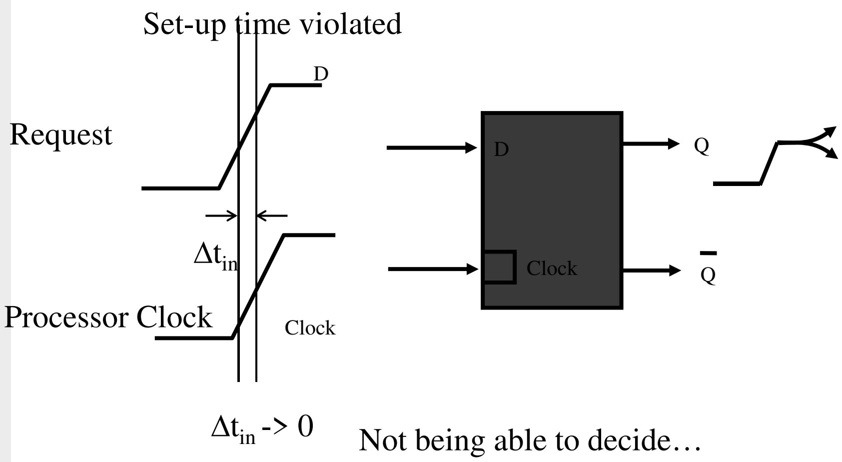
Alex Yakovlev
Newcastle University

Time Comparison Hardware

- Digital comparison hardware (which compares integers) is easy
  - Fast
  - Bounded time
- Analog comparison hardware (which compares reals like time) is hard
  - Normally fast, but takes longer as the difference becomes smaller
  - Can take forever, (Buridan's Ass ~1340)
- Synchronization and arbitration involve comparison of time
- Known to early computer designers:
  - Lubkin 1952, Catt 1966
  - Chaney and Littlefield 1966/72

Alex Yakovlev
Newcastle University

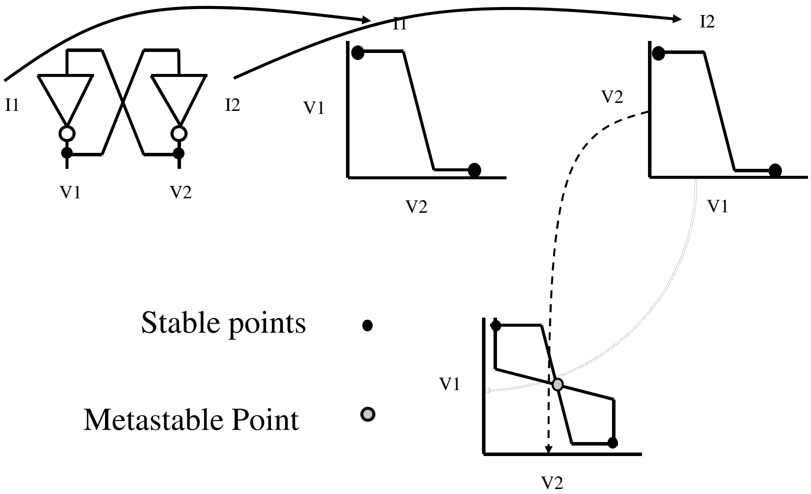
Metastability is....



Not being able to decide…

Alex Yakovlev
Newcastle University

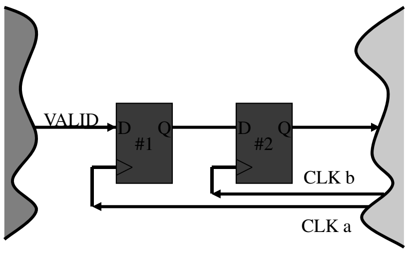
Metastability in a Latch



Alex Yakovlev
Newcastle University

Synchronizer

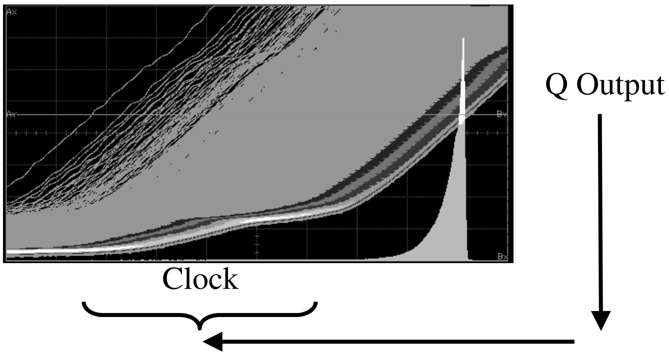
- ***t*** **is time allowed for the Q to change between CLK a and CLK b**
- **τ is the recovery time constant, usually the gain-bandwidth of the circuit**
- **T_w is the "metastability window"**
- **τ and T_w depend on the circuit**
- **We assume that all values of Δt_{in} are equally probable**



$$MTBF = \frac{e^{t/\tau}}{T_w \cdot f_c \cdot f_d}$$

Alex Yakovlev
Newcastle University

Typical responses

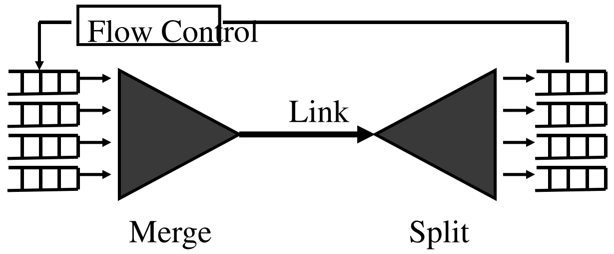


- All starting points are equally probable
- Most are a long way from the "balance point"
- A few are very close and take a long time to resolve

Synchronizer state of the art

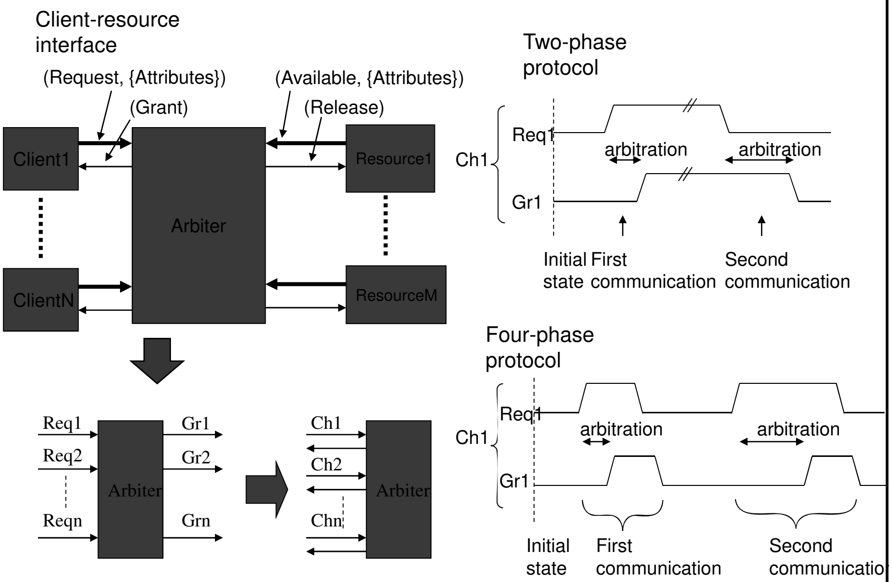
- You require about 35τ in order to get the MTBF out to about 1 century. (That's for 1 synchronizer)
- There is nothing else you can do while synchronizing
- Each typical static gate delay is equivalent to about 5τ. Synchronizers are analog devices, so worse affected by scaling
- Bigger SoCs, in future systems so more synchronizers, worse reliability
- Inputs can be 'malicious' i.e. always causing metastability.

Arbitration : Router priority example

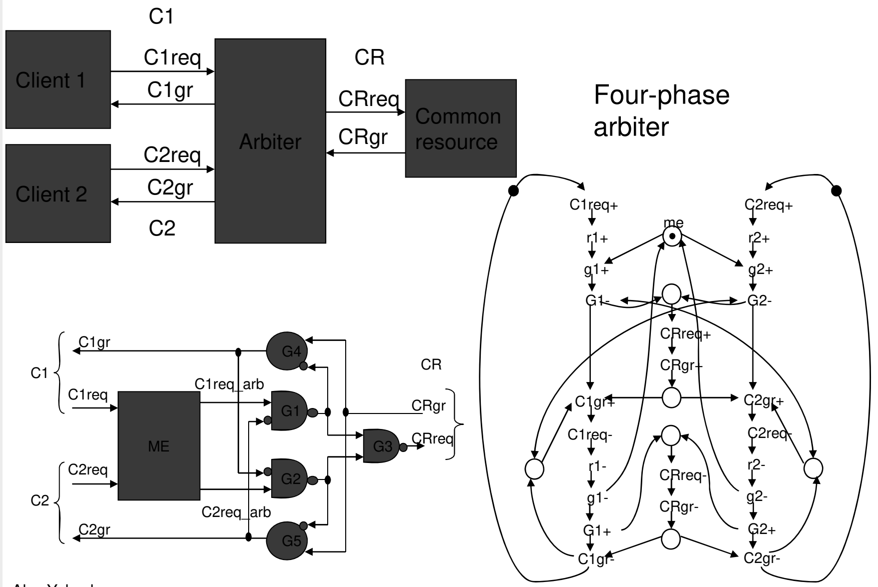


- **Virtual channels implement scheduling algorithm**
- **Contention for link resolved by priority circuits**

Arbiters



Arbiters

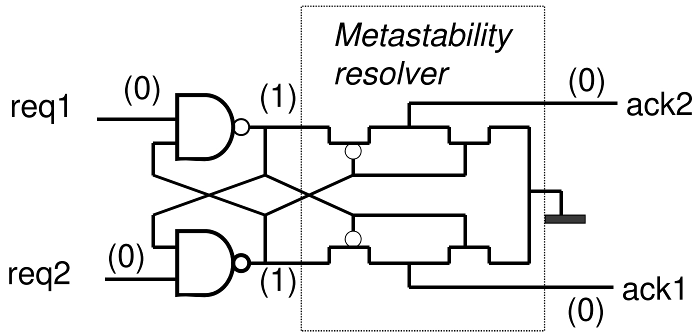


Alex Yakovlev
Newcastle University

51

Mutual exclusion element

Basic arbitration element: Mutex



An asynchronous data latch with MS resolver can be built similarly

Alex Yakovlev
Newcastle University

52

Some references

- **General Async Design:** J. Sparsø and S.B. Furber, editors. *Principles of Asynchronous Circuit Design*, Kluwer Academic Publishers, 2001. (electronic version of a tutorial based on this book can be found on: http://www2.imm.dtu.dk/pubdb/views/edoc_download.php/855/pdf/imm855.pdf
- **Async Control Synthesis:** J. Cortadella, M. Kishinevsky, A. Kondratyev, L. Lavagno, and A. Yakovlev. *Logic Synthesis of Asynchronous Controllers and Interfaces*. Springer-Verlag, 2002. (Petrify software can be downloaded from: http://www.lsi.upc.edu/~jordicf/petrify/)
- **Arbiters and Synchronizers:** D.J. Kinniment, Synchronization and Arbitration in Digital Systems, Wiley and Sons, 2007 (a tutorial on arbitration and synchronization from ASYNC/NOCS 2008 can be found: http://async.org.uk/async2008/async-nocs-slides/Tutorial-Monday/Kinniment-ASYNC-2008-Tutorial.pdf)
- **Asynchronous on-chip interconnect:** John Bainbridge, Asynchronous System-on-Chip Interconnect, BCS Distinguished Dissertations, Springer-Verlag, 2002 (electronic version of the PhD thesis can be found on: http://intranet.cs.man.ac.uk/apt/publications/thesis/bainbridge00_phd.php)

Conclusion

- **Asynchronous means that at least some parts in the designed system have no global clocking**
- **Asynchronous design is inevitable in the future to deal with complex systems on silicon**
- **Asynchronous design can be hard – many types of protocols, encoding schemes, concurrency issues, delay models and assumptions**
- **Asynchronous design means thinking in terms of handshakes, causality, relative timing – hence different specification models**
- **Performance models and analysis are different – not in terms of clock cycles**
- **Many new tools are needed and they are on the way!**
- **Synchronisation and arbitration require careful treatment of metastability**

Part II: Practical Asynchronous Design Automation

Jens Sparsø
DTU Informatics
Department of Informatics and Mathematical Modelling
Technical University of Denmark
jsp@imm.dtu.dk

DATE 10

Outline of Part II

1. **Elastic circuits and de-synchronization**
   [Using your existing synchronous CAD tools]
   Synthesize netlist, keep data-path and replace clock network by asynchronous control structure
   - ***… a tour: SGT →SLT→ALT***

2. **Syntax-directed translation**
   [State-of-the-art asynchronous CAD-tools]
   Used by Philips/Handshake Solutions and many others)
   - **Basic principles**
   - **Some recent developments**
     - **Control-flow vs. data-flow**
     - **High Level Synthesis**

Classification of digital circuits

… based on their notion of time 1)

	Discrete time ⟷ Contineous time	
Globally timed	**SGT** Synchronous Globally Timed	**AGT** Asynchronous Globally Timed
Locally timed	**SLT** Synchronous Locally Timed	**ALT** Asynchronous Locally Timed

Globally timed ⟷ **Locally timed**

Discrete time ⟷ **Contineous time**

[1] S.A. Ward and R.H. Halstead Jr., *Computation Structures*, (McGraw-Hill, 1990) Chapter 7

Classification

Globally timed:
- One single FSM+DP

Locally timed:
- Several interacting FSM+DP's
- Handshaking (Start / Finish, Req / Ack, Valid / Stop)

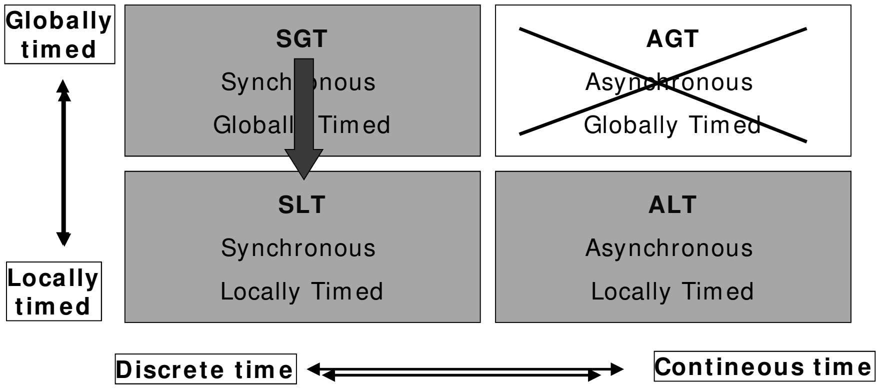
Globally timed	**SGT** Synchronous Globally Timed	~~**AGT** Asynchronous Globally Timed~~
Locally timed	**SLT** Synchronous Locally Timed	**ALT** Asynchronous Locally Timed

Globally timed ⟷ **Locally timed**

Discrete time ⟷ **Contineous time**

[1] S.A. Ward and R.H. Halstead Jr., *Computation Structures*, (McGraw-Hill, 1990) Chapter 7

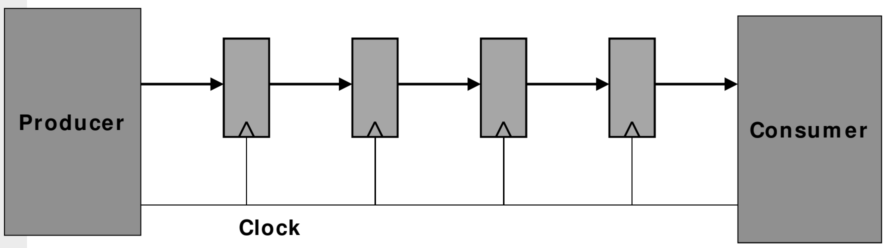
De-synchronization: SGT → SLT → ALT



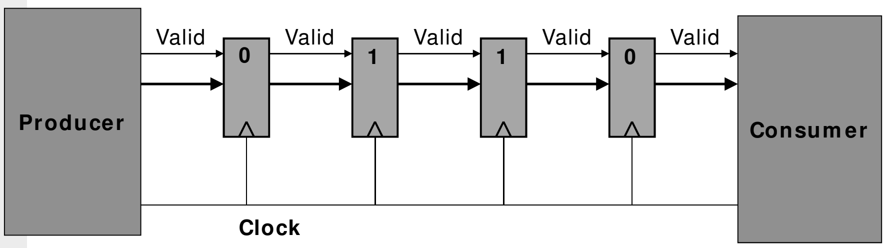
1) S.A. Ward and R.H. Halstead Jr., *Computation Structures*, (McGraw-Hill, 1990) Chapter 7

Simple clocked design (SGT)

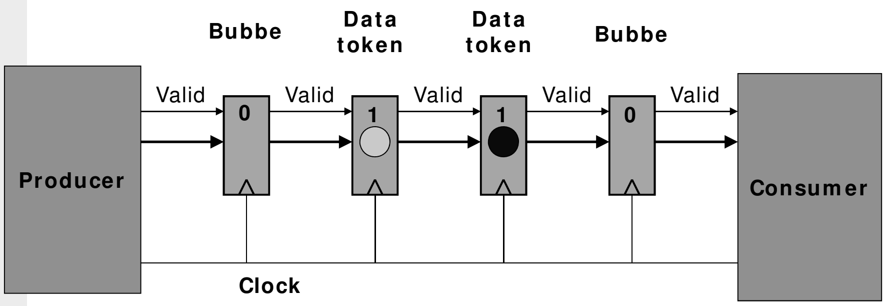
- **One data value per register per clock**



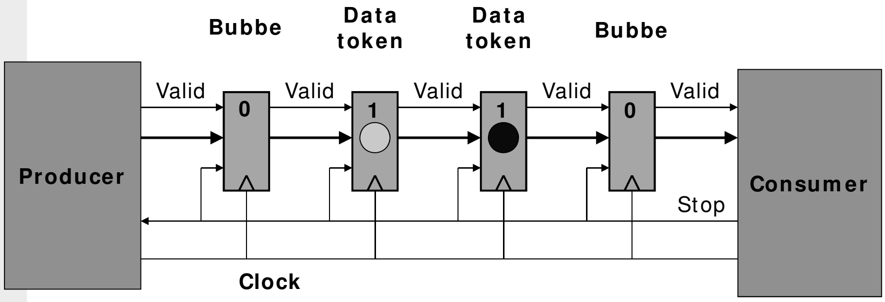
Slow producer (valid signal)



Slow producer (valid signal)



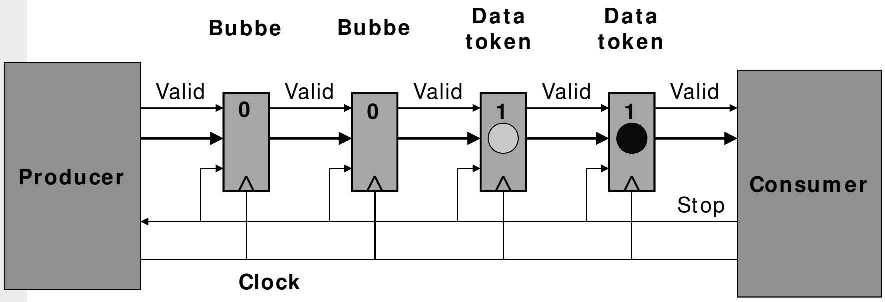
Slow consumer (stop signal)



J. Sparsø, DTU Informatics (Date'10 tutorial)

9

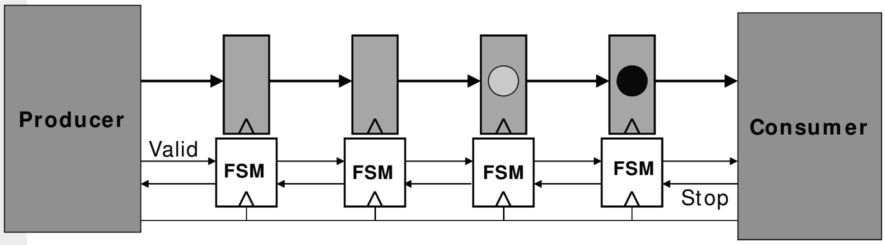
Slow consumer (stop signal)



J. Sparsø, DTU Informatics (Date'10 tutorial)

10

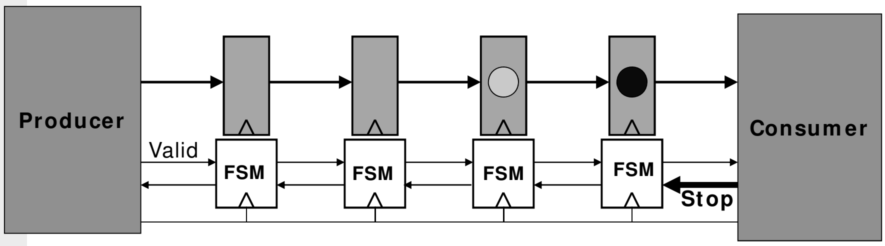
… break long stop-signal path



J. Sparsø, DTU Informatics (Date'10 tutorial)

11

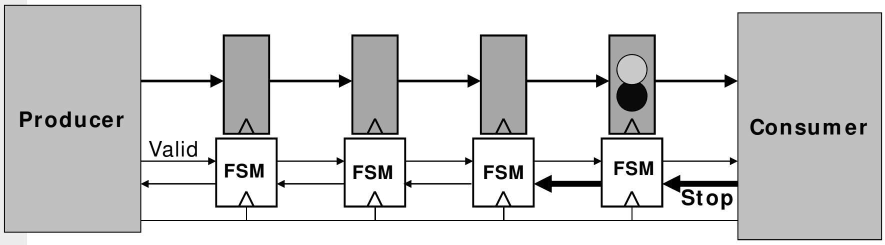
… break long stop-signal path



J. Sparsø, DTU Informatics (Date'10 tutorial)

12

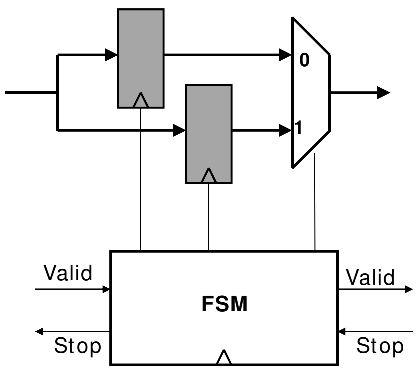
… a stage must buffer 2 tokens



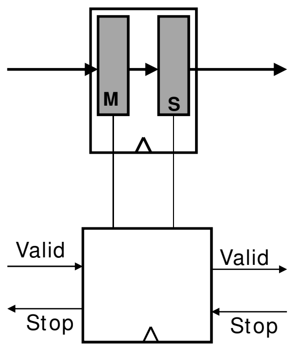
- Stop can only propagate one stage to the left in one clock cycle.
- Each stage must be able to buffer an extra data token
- A stage is: 1 FSM + 2 regs + 1 MUX / 1 FSM + 2 latches (see next slide)

A double-buffered stage

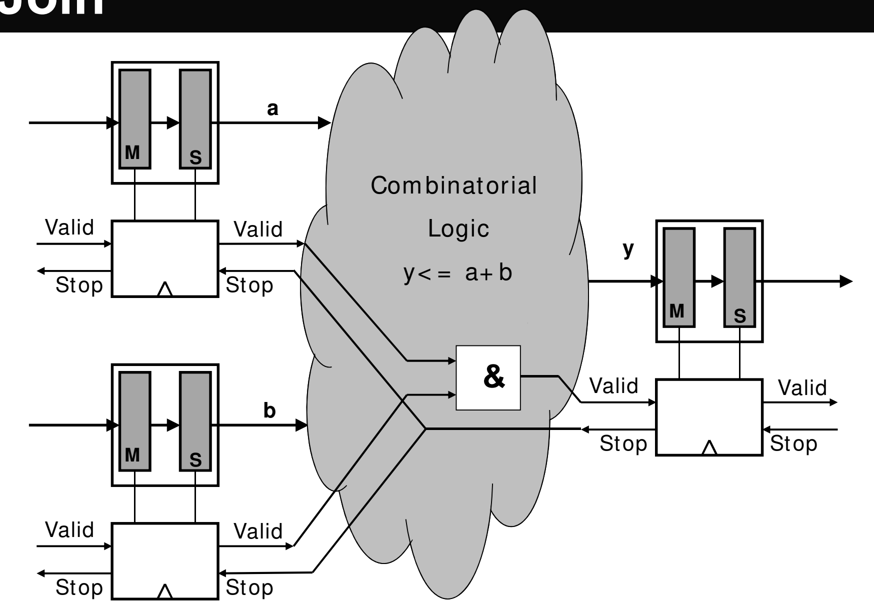
1 FSM + 2 regs + 1 MUX



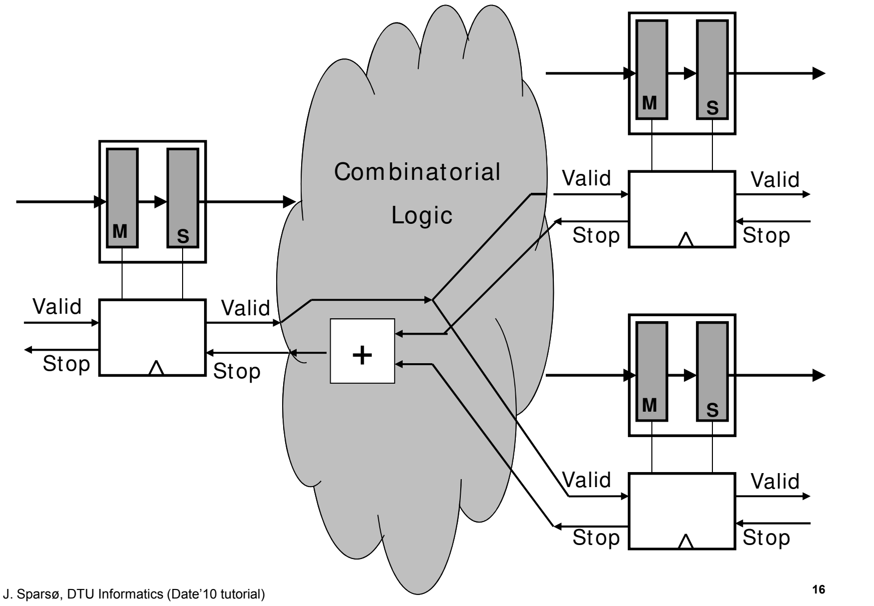
1 FSM + 2 latches



Join


a
M
S
Combinatorial
Logic
y<= a+b
Valid
Stop
Valid
Stop
y
M
S
&
Valid
Stop
Valid
Stop
b
M
S
Valid
Stop
Valid
Stop

Fork


M
S
Combinatorial
Logic
M
S
Valid
Stop
Valid
Stop
Valid
Stop
Valid
Stop
+
M
S
Valid
Stop
Valid
Stop

References

- This was SGT →SLT
  - Synchronous *locally timed*
  - Synchronous *latency insensitive*
  - Synchronous *elastic*
- Some References:

  - J. Carmona, J. Cortadella, M. Kishinevsky, and A. Taubin, "Elastic Circuits," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 28, no. 10, pp. 1437–1455, Oct. 2009
  - L. Carloni, K. McMillan, and A. Sangiovanni-Vincentelli, "Theory of latency-insensitive design," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 20, no. 9, pp. 1059–1076, Sep. 2001.
  - J. Cortadella, M. Kishinevsky, and B. Grundmann, "Synthesis of synchronous elastic architectures," in P*roc. ACM/IEEE DAC*, Jul. 2006, pp. 657–662.
  - A. Peeters and K. van Berkel, "Synchronous handshake circuits," in *Proc. Int. Symp. Adv. Res. Asynchronous Circuits Syst. (ASYNC'01)*, 2001, pp. 86–95.

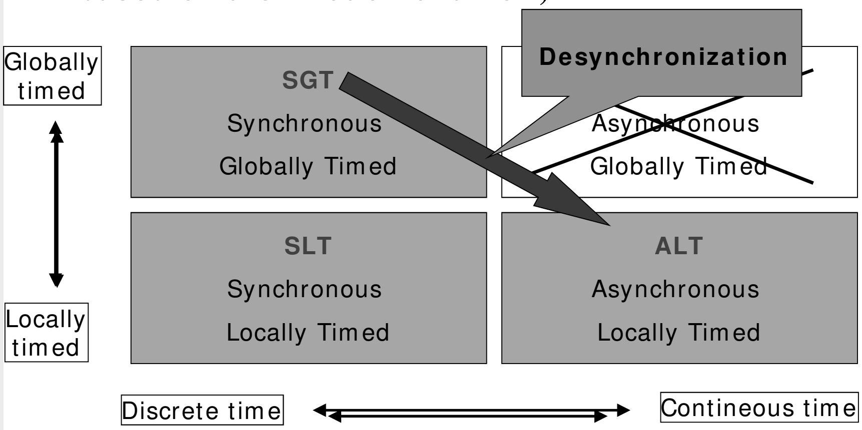
- Check also:
  - Elastix Inc., http://www.elastix-corp.com/.

Summary so far

- This was SGT →SLT:
  - Designing synchronously (SGT)
  - Adding elasticity (SGT→SLT)
    - Substitute MS flip-flops by pairs of latches
    - Add joins, forks and *clocked* latch controllers
- Carrying on: SLT →ALT or SGT→ALT
  - De-synchronization
    - Throw out clock.
    - Substitute MS flip-flops by pairs of latches
    - Add *asynchronous* latch controllers, joins, forks, and delay elements
- Fundamental issue:
  - Behaviour/function must be the same
  - Token-flow equivalence

Classification of digital circuits

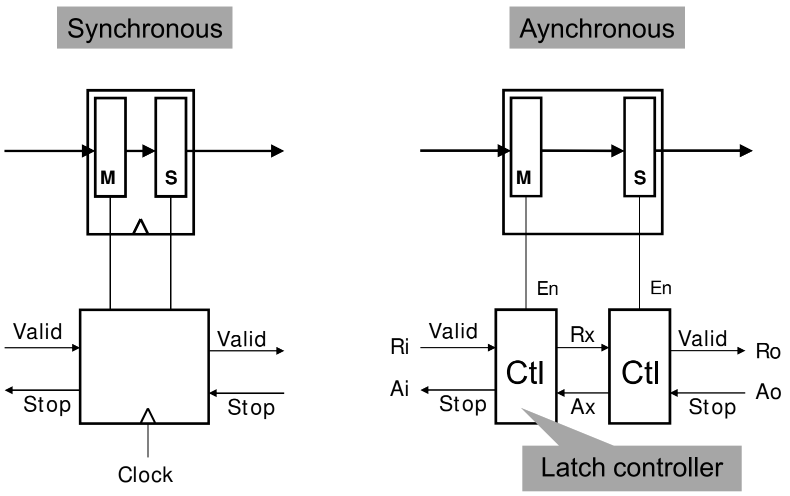
… based on their notion of time 1)



[1] S.A. Ward and R.H. Halstead Jr., *Computation Structures*, (McGraw-Hill, 1990) Chapter 7

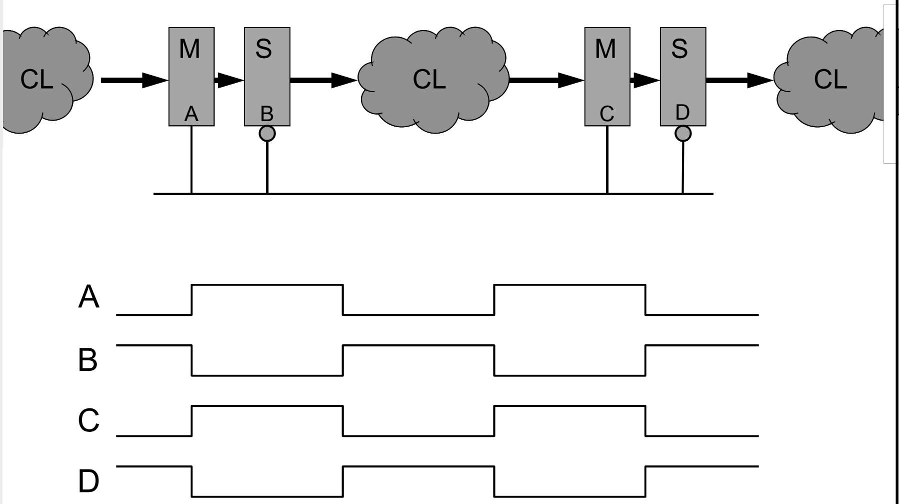
J. Sparsø, DTU Informatics (Date'10 tutorial) 19

A double-buffered stage

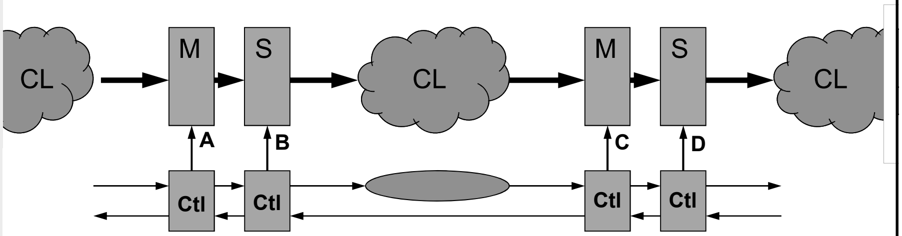


J. Sparsø, DTU Informatics (Date'10 tutorial) 20

Clocked → Asynchronous



Clocked → Asynchronous



- **Many "clock systems" (A, B, C, D, …) are possible.**
- **The clocked maste-slave operation is just one special case.**
- **What matters is the (safe) flow of tokens**

- **Let's for simplicity look at the latches only**

• Acknowledgement:

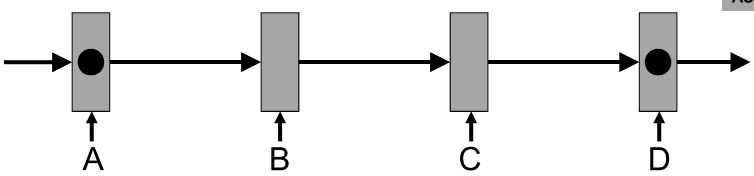
The folowing slides on desynchronization are based on material extracted from the presentation of the paper:

Blunno, J. Cortadella, A. Kondratyev, L. Lavagno K. Lwin, C. Sotiriou, ”Handshake protocols for desynchronization”
Proc. of ASYNC’04

The full presentation is found at
http://www.async04.gr/presentations/handshake_prot_for_desync.ppt

De-synchronozation model

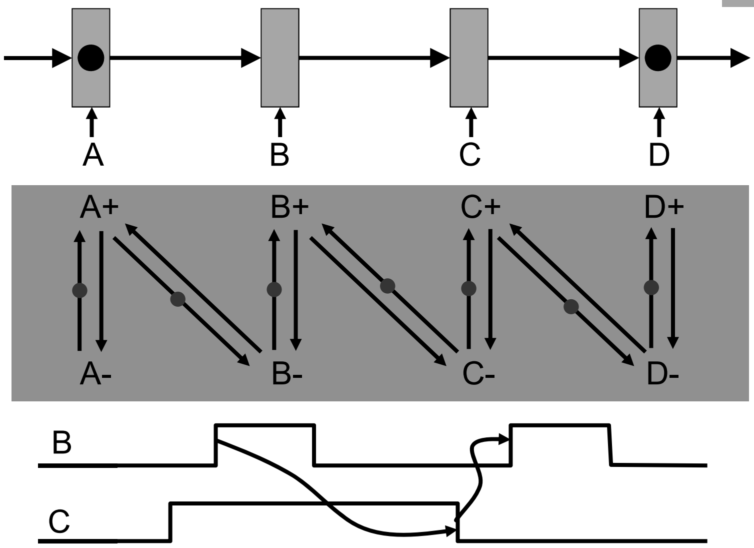
Based on: Blunno at al, ASYNC’04



- **Rules for safe operation:**
  - **A+; A-; A+; A- :latch control signals must alternate.**
  - **B+ →C– : a latch (C) cannot capture (a data item) unless a data item has been captured in (or at least passed on from) its predecessor (B).**
  - **C– →B+ : a latch (B) cannot capture a new data item unless the current data item has been captured by its successor (C).**

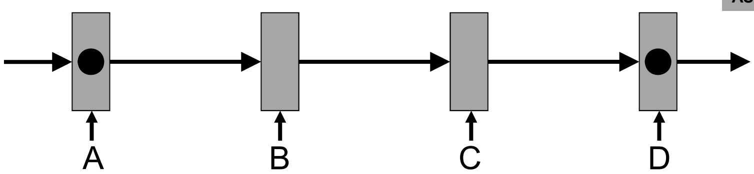
De-synchronozation model

Based on: Blunno at al, ASYNC'04



De-synchronozation model

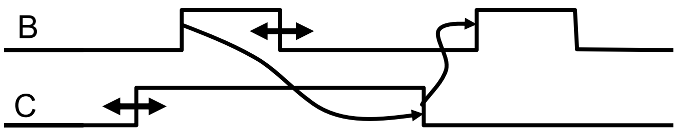
Based on: Blunno at al, ASYNC'04

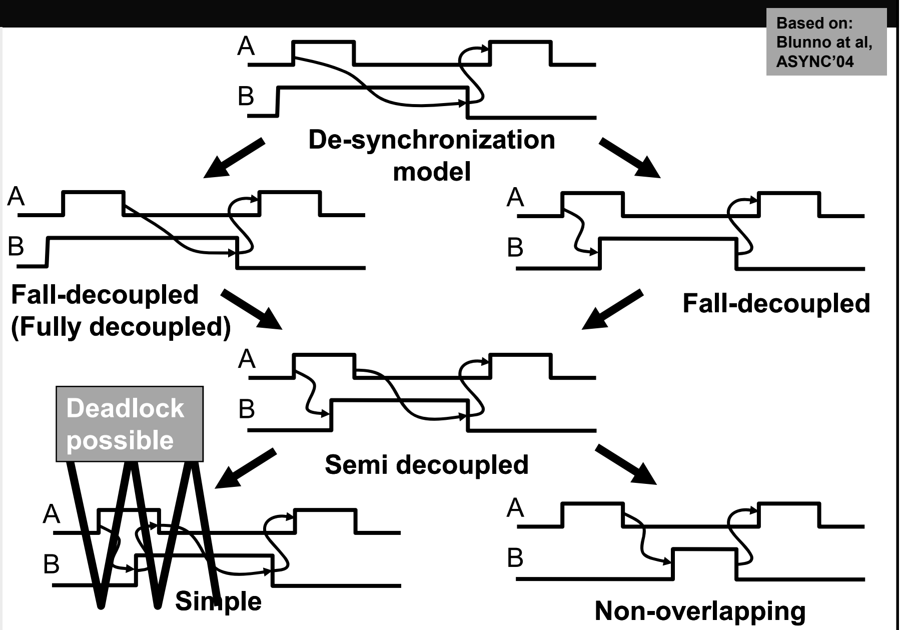


Only interested in signals A, B, C and D
(but not all the Req and Ack signals)

- No requirements for B– and C+

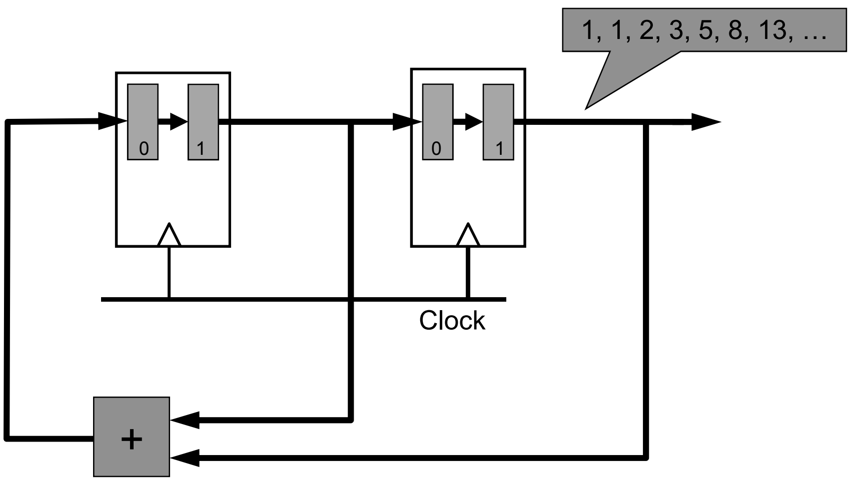
→ Many different latch controllers can be used





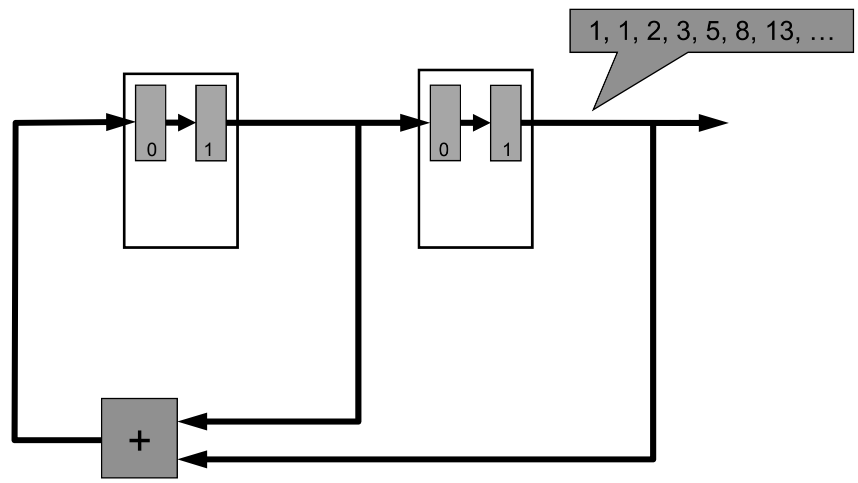
J. Sparsø, DTU Informatics (Date'10 tutorial) 27

Synchronnous FIB-circuit

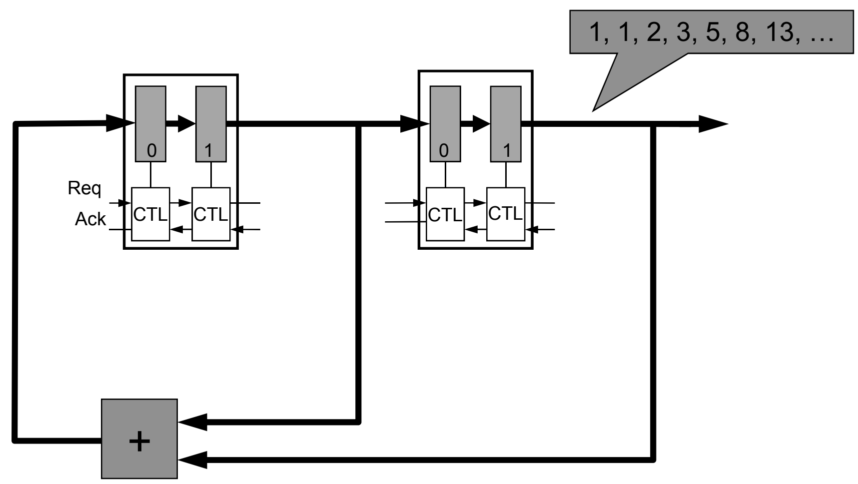


J. Sparsø, DTU Informatics (Date'10 tutorial) 28

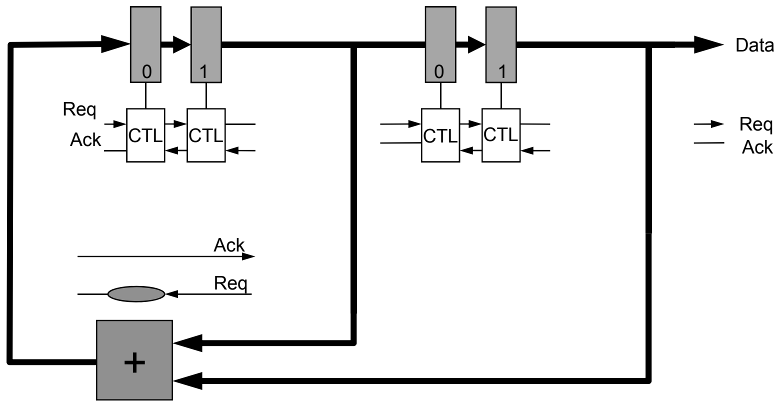
… remove clock



… add latch controllers



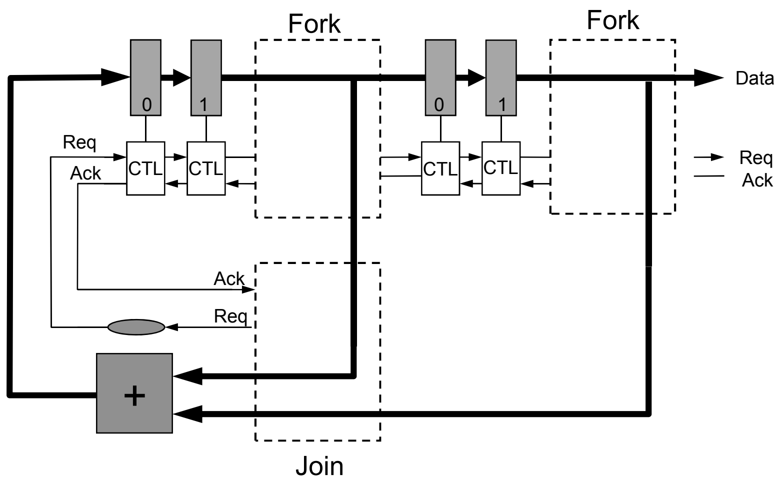
… add delay-elements



J. Sparsø, DTU Informatics (Date'10 tutorial)

31

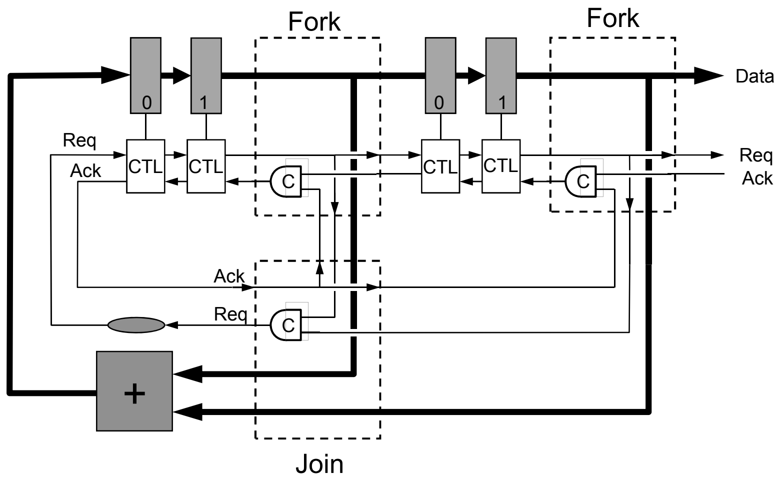
… and add Joins and Forks



J. Sparsø, DTU Informatics (Date'10 tutorial)

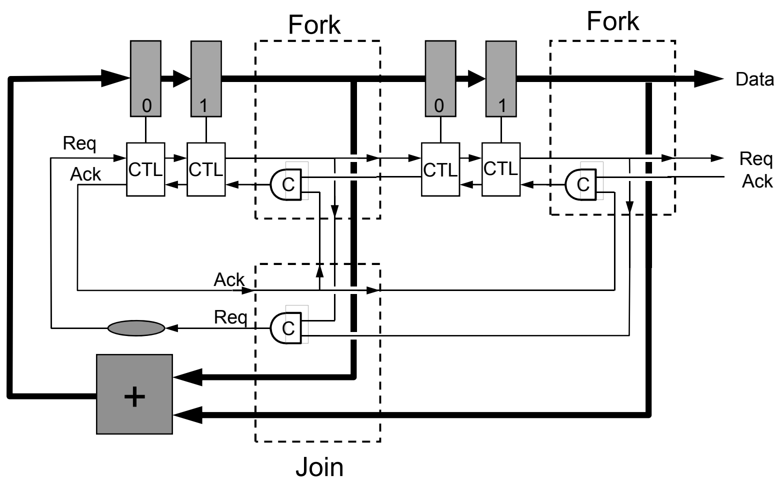
32

… and add Joins and Forks



J. Sparsø, DTU Informatics (Date'10 tutorial) 33

De-synchronized FIB-circuit



J. Sparsø, DTU Informatics (Date'10 tutorial) 34

Summary de-synchronization

- Think synchronous
- Design synchronous:
  - One clock
  - Edge-triggered flip-flops
- De-synchronize (automatically)
  - Remove clock
  - 1 edge-triggered flip-flop = 2 latches
  - Add latch controllers
    (any mix of "valid" controllers is allowed)
  - Add Joins and Forks
  - Add delay elements
- Run it asynchronously

J. Sparsø, DTU Informatics (Date'10 tutorial) 35

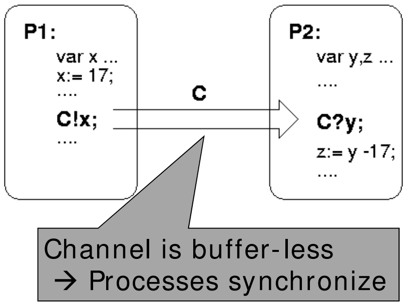
Outline of Part II

1. **Elastic circuits and de-synchronization**
   [Using your existing synchronous CAD tools]
   Synthesize netlist, keep data-path and replace clock network by asynchronous control structure
   - *... a tour: SGT →SLT→ALT*
2. **Syntax-directed translation**
   [State of the art asynchronous CAD-tools]
   Used by Philips/Handshake Solutions and many others)
   - Basic principles
   - Some recent developments
     - Control-flow vs. Data-flow
     - High Level Synthesis

J. Sparsø, DTU Informatics (Date'10 tutorial) 36

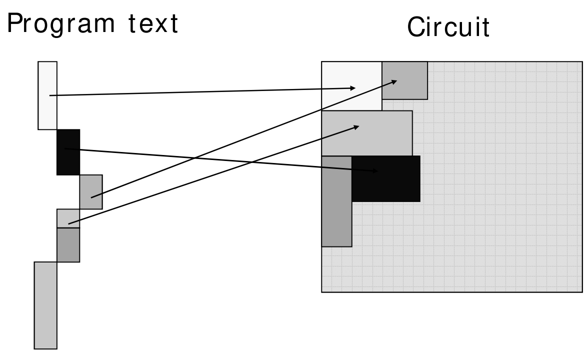
HDL's for asynchronous design

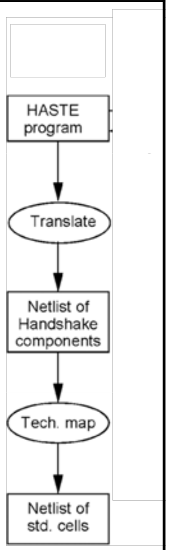
- **VHDL or Verilog:**
  - **Event driven + parallel processes. Fine, but …**
  - **… "programming" of req-ack handshake is tedious.**
- **Inspiration from parallel programming languages:**
  CSP [1] , OCCAM, …
  - **Message passing across communiction channels (Send, Receive, Probe)**
- **Asynchronous HDL's**
  - **Haste (Tangram)** Handshake Solutions
  - **Balsa** U. of Manchester
  - **CHP** Caltech
  - **…**



[1] C.A.R. Hoare, "Communicating sequential processes" *Communications of the ACM, 21(8):666-677,* 1978.

Syntax-directed translation





Asynchronous design

- VHDL, Verilog, SystemC
  - Modelling and simulation (event driven!)
- **Haste,** Balsa, CHP**,** OCCAM,
  - Modelling and simulation
  - Syntax-directed translation (synthesis)

- Data-flow structures
- Handshake components

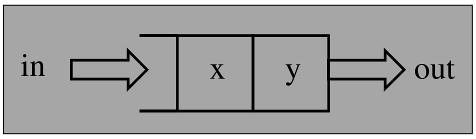
Abstraction similar to sync. RTL

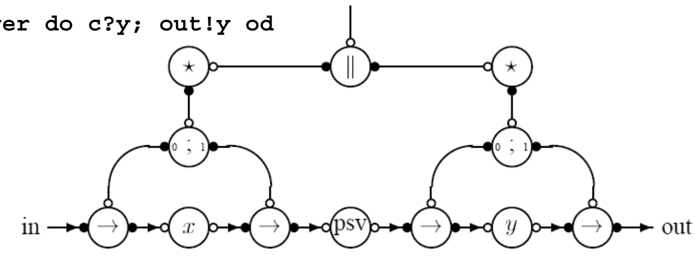
- Data path + control
- Handshake protocols
- Circuit design styles
- Timing assumptions

J. Sparsø, DTU Informatics (Date'10 tutorial) 39

Example: a 2-stage FIFO

```
  int = type [0..255]
& fifo: main proc (in?chan int & out!chan int).
  begin
  x,y:var int
  C : chan int
  |
       forever do in?x ; c!x od
    || forever do c?y; out!y od
  end
```

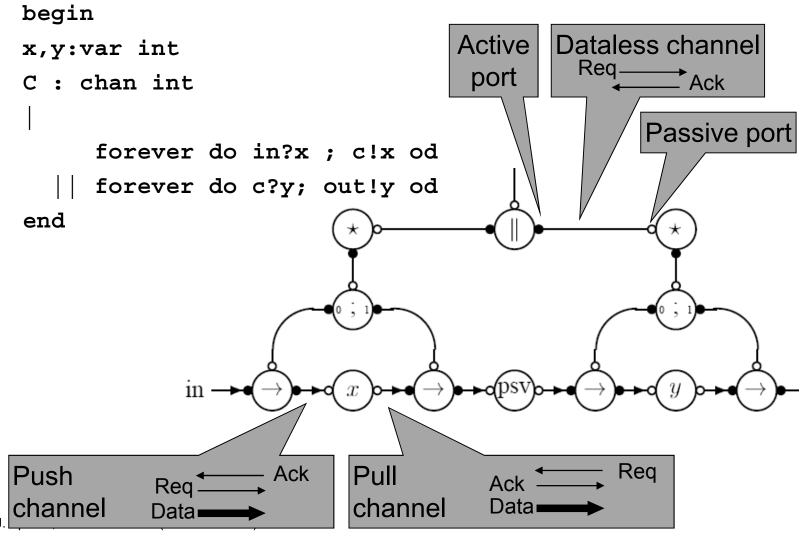




J. Sparsø, DTU Informatics (Date'10 tutorial) 40

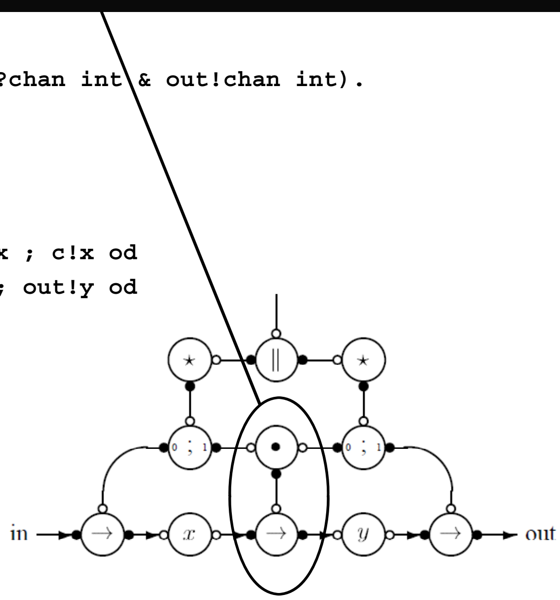
Example: a 2-stage FIFO

```
  int = type [0..255]
& fifo: main proc (in?chan int & out!chan int).
  begin
  x,y:var int
  C : chan int
  |
      forever do in?x ; c!x od
    || forever do c?y; out!y od
  end
```



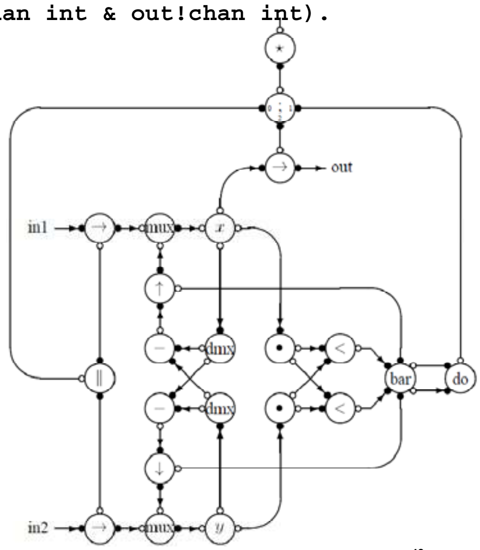
Peephole optimization

```
  int = type [0..255]
& fifo: main proc (in?chan int & out!chan int).
  begin
  x,y:var int
  C : chan int
  |
      forever do in?x ; c!x od
    || forever do c?y; out!y od
  end
```



Example: GCD

```
  int = type [0..255]
& gcd: main proc (in1,in2?chan int & out!chan int).
  begin  x,y:var int ff
  | forever do
      in1?x || in2?y
    ; do x<y then y:=y-x
      or y<x then x:=X-y
      od
    ; out!x
    od
  end
```



Lessons learned

– Generally slow circuits
+ Generally low power
? Is this efficient hardware
? How to optimize
? Programming vs. designing HW

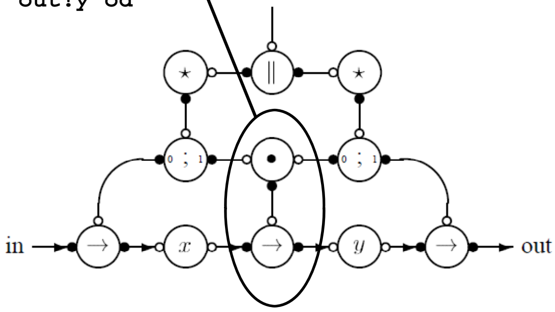
Outline of Part II

1. **Elastic circuits and de-synchronization**
   [Using your existing synchronous CAD tools]
   Synthesize netlist, keep data-path and replace clock network by asynchronous control structure
   - ***... a tour: SGT →SLT→ALT***

2. **Syntax-directed translation**
   [State of the art asynchronous CAD-tools]
   Used by Philips/Handshake Solutions and many others)
   - **Basic principles**
   - **Some recent developments**
     - **Control-flow vs. Data-flow**
     - **High Level Synthesis**

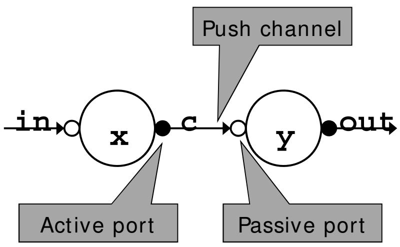
Control-driven FIFO implement.

```
  int = type [0..255]
& fifo: main proc (in?chan int & out!chan int).
  begin
  x,y:var int
  C : chan int
  |
       forever do in?x ; c!x od
    || forever do c?y; out!y od
  end
```

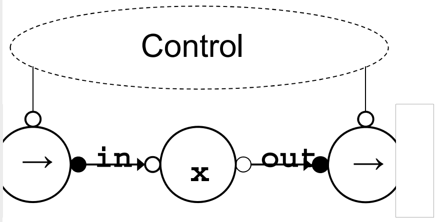


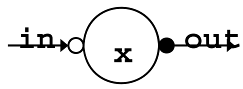
Data-driven FIFO implemt.

```
  int = type [0..255]
& fifo: main proc (in?chan int & out!chan int).
  begin
  x,y:var int
  C : chan int
  |
       forever do in?x ; c!x od
    || forever do c?y; out!y od
  end
```



Control-driven vs. Data-driven





- **Arbitrary reading and writing of variables. Only necessary actions.**
- **Mix of push and pull channels enable elegant and efficient solutions.**
- **Control overhead.**
- **Low speed, low power and energy.**
- **Suited for control dominated applications/algorithms.**

- **Data-flow single assignment i.e. repeat {write once; read once}.**
- **To maintain a variable that is used but not modified, you have to read and write back.**
- **No control logic.**
- **Fast, high(er) power.**
- **Particularly well suited for pipelined stream-processing.**

Ongoing work (Data-driven synthesis)

- **Balsa → Teak (U. of Manchester)**
  - A. Bardsley, L. Tarazona, and D. Edwards. Teak: A Token-Flow Implementation for the Balsa Language". Ninth International Conference on Application of Concurrency to System Design (ACSD 2009). p. 23-31, 2009.
  - S. Taylor, D. Edwards, L. A. Plana, and D. Tarazona. Asynchronous Data-Driven Circuit Synthesis. To be published in : I*EEE Transactions on Very Large Scale Integration (VLSI) Systems.*
  - S. Taylor, D. Edwards, L.A. Plana. Automatic Compilation of Data-Driven Circuits. In 14th IEEE International Symposium on Asynchronous Circuits and Systems (ASYNC'08), p. 3-14, 2008.
- **Haste/TiDE-AE → ??**
  - **??**

Outline of Part II

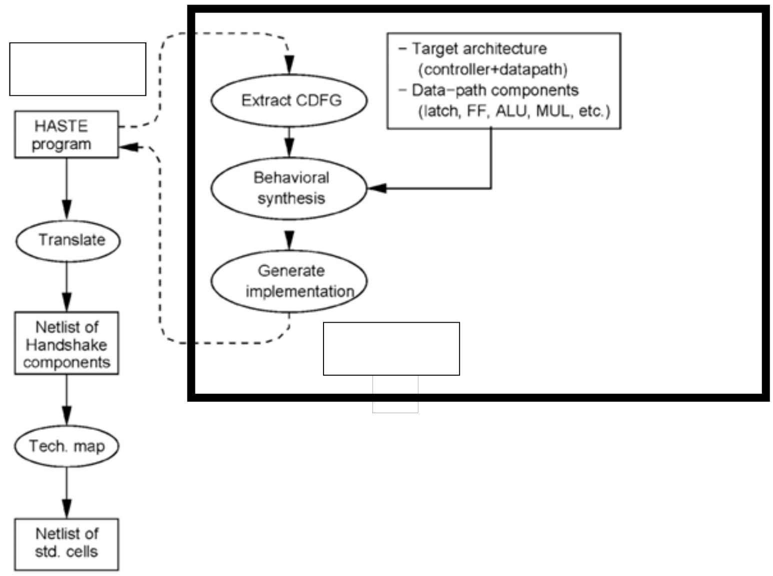
1. **Elastic circuits and de-synchronization**
   [Using your existing synchronous CAD tools]
   Synthesize netlist, keep data-path and replace clock network by asynchronous control structure
   - ***… a tour: SGT →SLT→ALT***

2. **Syntax-directed translation**
   [State of the art asynchronous CAD-tools]
   Used by Philips/Handshake Solutions and many others)
   - **Basic principles**
   - **Some recent developments**
     – **Control-flow vs. Data-flow**
     – **High Level Synthesis**

High level synthesis

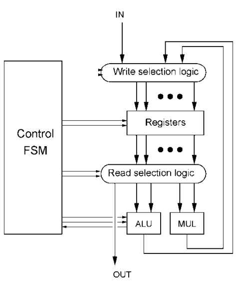
- **Syntax-directed translation:**
  - **Source program ≡ Circuit implementation**
- **Idea: Optimize at source code level**
  - **Haste → Haste**
  - **Matlab → Haste**
    – **Automatic constraint driven optimization (Area, speed, Power)**

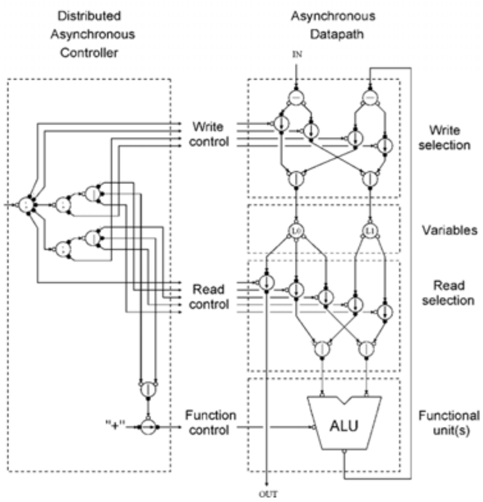
Example: Haste → Haste



J. Sparsø, DTU Informatics (Date'10 tutorial) 51

… targeting a FSMD-style impl.





J. Sparsø, DTU Informatics (Date'10 tutorial) 52

Work at DTU

- **A fully automatic Haste-in-Haste-out synthesis tool.**
- **Can handle large non-trivial subset of Haste.**
- **Results:**
  - **Area: 5-58% reduction (avg. 30%)**
  - **Speed: 0-67% reduction (avg. 40%)**
- **Source-to-source optimization (behavioural synthesis) combined with syntax-directed translation is a promising approach.**
- **Using syntax-directed translation as backend for a synthesis system from *<your favourite high level language>* seems promising as well.**

Some references

- S. F. Nielsen, J. Sparsø, J.B. Jensen, and J.B. Nielsen. A Behavioral Synthesis Frontend to the Haste/TiDE Design Flow**.** In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*, p. 185–194, 2009.
- S. F. Nielsen, J. Sparsø, and J. Madsen. High-level synthesis of asynchronous circuits using syntax directed translation as backend. *IEEE Transactions on VLSI Systems*, 17(2):248–261, Feb. 2009.
- J. Hansen and M. Singh. Concurrency-enhancing transformations for asynchronous behavioral specifications: A datadriven approach. In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*, p. 15–25. 2008.
- M. Tranchero, L. Reyneri, A. Bink and M. de Wit. An Automatic Approach to Generate Haste Code from Simulink Specifications In *Proc. IEEE International Symposium on Asynchronous Circuits and Systems*, p. 185–194, 2009.
- J. Teifel and R. Manohar. "Static Tokens: Using Dataflow to Automate Concurrent Pipeline Synthesis," *Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems*, p. 17-27**,** 2004.

Part II: Conclusion

1. **Elastic circuits and de-synchronization**
   [Using your existing synchronous CAD tools]
   Synthesize netlist, keep data-path and replace clock network by asynchronous control structure
   - ***… a tour: SGT →SLT→ALT***

2. **Syntax-directed translation**
   [State-of-the-art asynchronous CAD-tools]
   Used by Philips/Handshake Solutions and many others)
   - **Basic principles**
   - **Some recent developments**
     - **Control-flow vs. Data-flow**
     - **High Level Synthesis**

END

- **Thank you!**
- **Questions?**

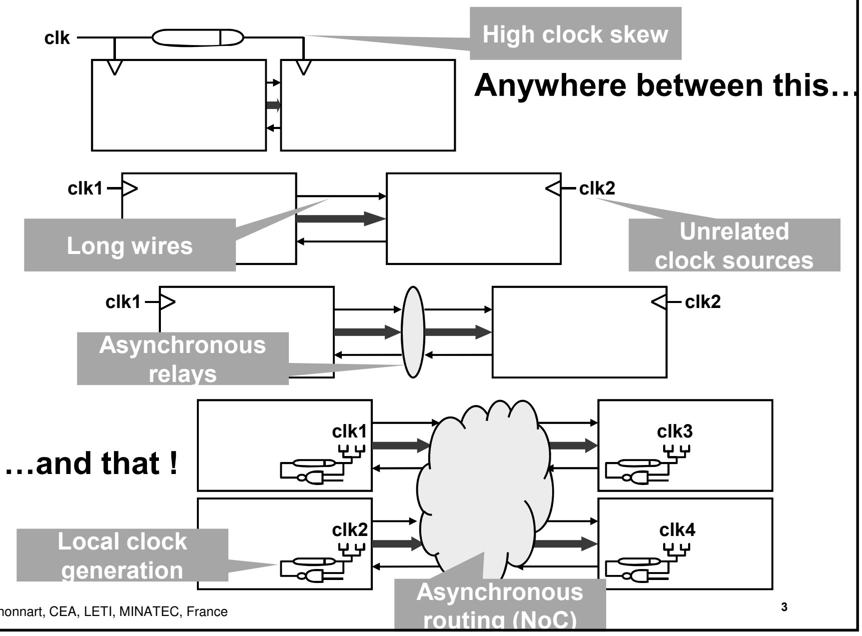
Part III - On the way to asynchronous circuits, Globally-Asynchronous Locally-Synchronous design

Yvain Thonnart
CEA, LETI, MINATEC, France
Department of Design, Architecture & Embedded Software
yvain.thonnart@cea.fr

Part III - Outline

1. GALS overview
    1. Introduction to GALS
    2. Advocating GALS circuits
    3. GALS design challenges
    4. GALS taxonomy
2. GALS intefaces design
    1. Pausable clocking
    2. Bi-synchronous FIFOs
    3. Interfaces with asynchronous logic
3. GALS physical implementation & test
    1. GALS interface implementation
    2. Test of GALS circuits

Globally Asynchronous Locally Synchronous



The GALS approach

An increasing tendency in Systems on Chip:

- A plurality of synchronous islands communicating asynchronously with each other

That is actually not so young:

- Chapiro, 1984
- A bit neglected in traditional VLSI design
  - RTL is synchronous,
  - FPGAs are (mostly) synchronous
  - Education is vastly synchronous
- Yet to “handle with care”
  - failure is quite easy !

Advocating GALS (1)

- Example of a smartphone
  - ~20 IPs are getting usual
  - Physical interface peripherials ~200MHz
  - Modem DSP processing ~400MHz
  - MAC processing ~600MHz
  - GP Host ~600MHz ➔ 1GHz
  - Graphical & Video accelerators ~500MHz
  - Interconnect ~400MHz
- Lots of different frequencies depending on needs and capabilities
  - IP reuse
  - target frequency
- Needs GALS assembling to tune each block to optimal frequency

Advocating GALS (2)

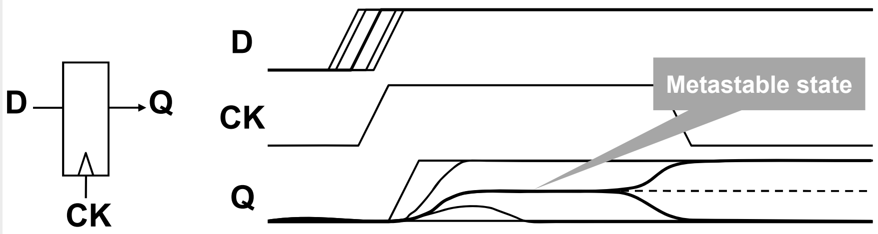
- Clock Tree Synthesis at top level
  - Wire delay / gate delay ratio is exploding with new technologies
  - Increasing functionality ➔ increasing depth
    - more latency
    - more skew
    - more jitter…
  - Heterogeneous architectures
    - Not always a neat rectangle
    - L, T, U, W… shapes
- GALS decomposition allows smaller clock trees

Advocating GALS (3)

- GALS to smoothen current peaks
  - A single clock edge on a billion transistors drives quite a lot of current
  - Off-phase / unrelated clocks reduce the currents peaks
- GALS to enable DVFS/AVFS
  - Set each part at its minimal operating frequency & voltage respecting the real-time constraints
- GALS partitioning allows for low-power design

Resynchronization & metastability (1)

- Issue with asynchronous inputs

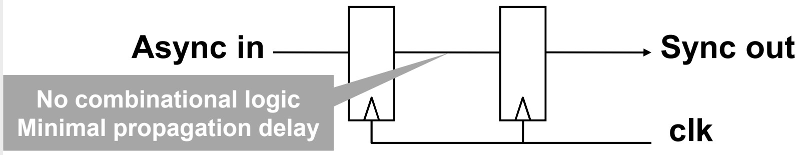


- Possibly indefinitely long undefined logic value
  - Seitz, 1980
- Mean time between failures (MTBF)

$$MTBF = \frac{e^{T/\tau}}{T_W f_c f_d} \quad \begin{cases} T & \text{recovery time} \\ f_c, f_d & \text{sampling \& data frequencies} \\ \tau, T_W & \text{technology parameters (~ 1 \& 2 FO4 delays)} \end{cases}$$

Resynchronization & metastability (2)

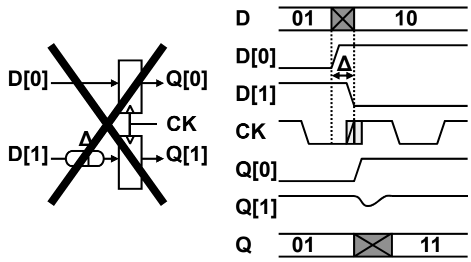
- Solution: widen the window
  1. Force a full clock cycle between two flip-flops

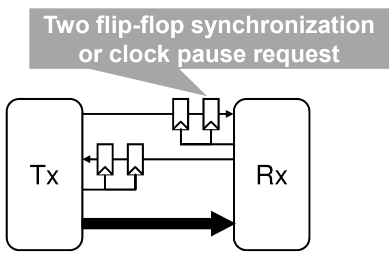


  2. Use a single flip-flop, but a delayed clock
     - Chapiro's unsynchronous machines
     - Evolved to the pausable clock concept
     - Further refinements to locally delayed latching

Data Resynchronization

- An asynchronous data bus shall never be sampled
  - Risk of inconsistent values between bits
- Only control with a single bit change is resynchronized
  - Data consumption is delayed until request is correctly resynchronized
- Backwards flow control needed not to miss data
  - acknowledge





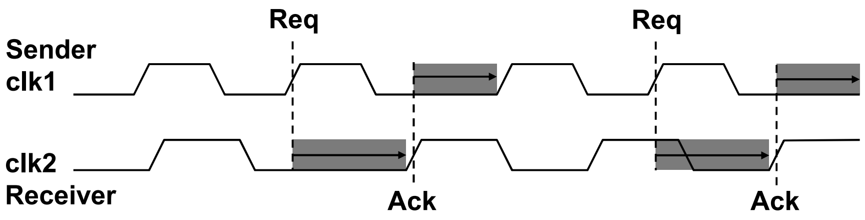
Resynchronization impact on performance

- Increases latency
  - Control is delayed by the synchronization cost
  - Delay of several clock cycles in the forward path between clock domains
- Decreases performance in bounded systems
  - E.g. a cache-miss: processor is stalled for
    - –latency of request from cache to memory
    - –latency of response from memory to cache
- Backward path impact
  - Don't forget to acknowledge !
  - Influence of the round-trip delay : acknowledge is given up to 4 cycles after request

  ➔ Risk of bursty traffic

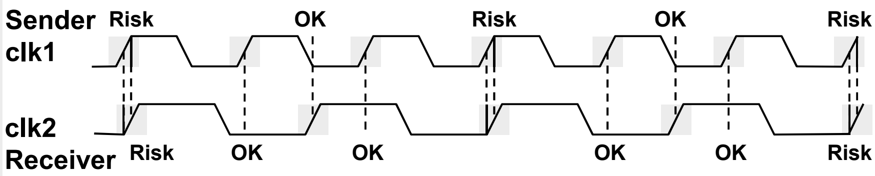
Pausable clocking

- Principle
  - Clock is delayed during transfers to prevent metastability in the data registers
  - No additional buffering is required
  - Clock period is not strictly fixed
- Chronogram



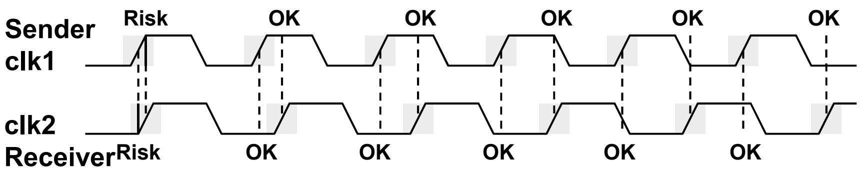
Heterochronous

- Principle
  - Two independent free running clocks from external source or locally generated
  - No hypothesis on clock frequency relation
  - Re-synchronization with potential metastability is needed on each cycle
- Chronogram



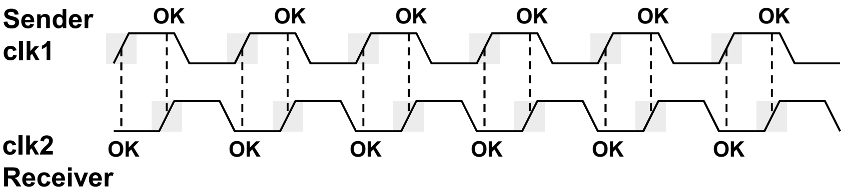
Plesiochronous

- Principle
  - Two independent clocks with similar frequencies
    - –e.g. Two PLLs
  - Clocks slowly drift out of phase
  - Re-synchronization is needed, with potential metastability once in a while
  - Acknowledge alleviated thanks to similar periods
- Chronogram



Mesochr. / Loosely synchronous

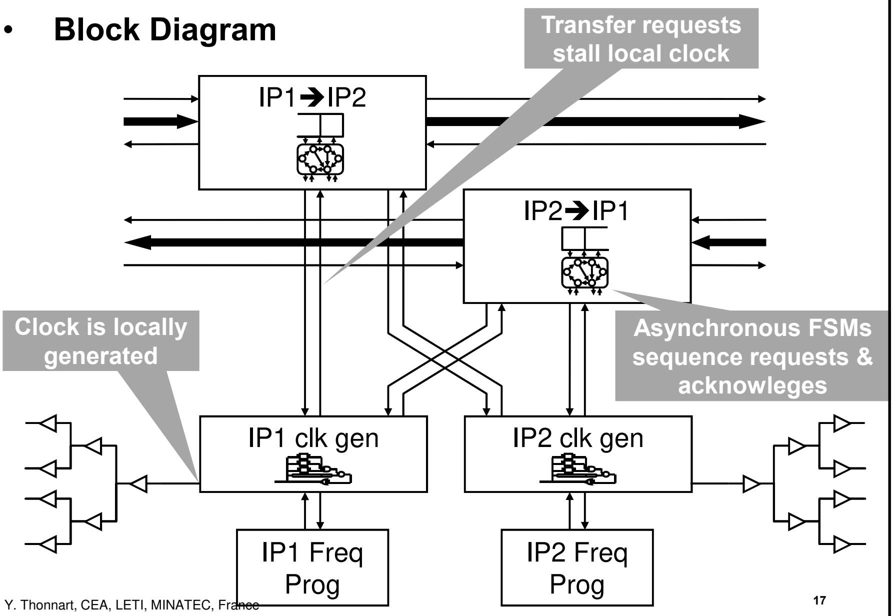
- Principle
  - A single clock source is used, but with different sub-trees for different IPs
  - No synchronizer: relies on opposite phases with reduced time margins (~1/4 cycle setup time)
  - If phases are not opposite, risk of malicious synchronization: metastable at each cycle !
- Chronogram



Part III - Outline

Pausable clocking

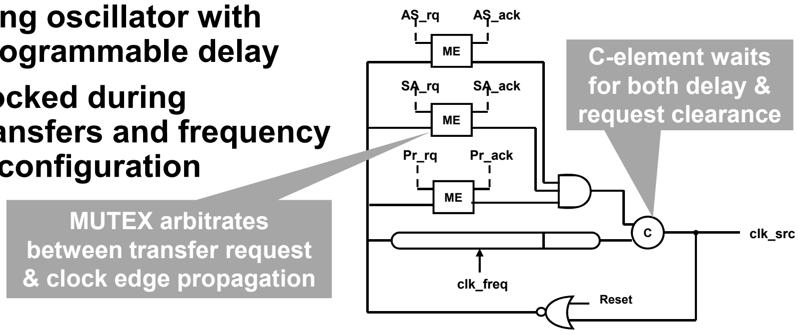
- **Block Diagram**



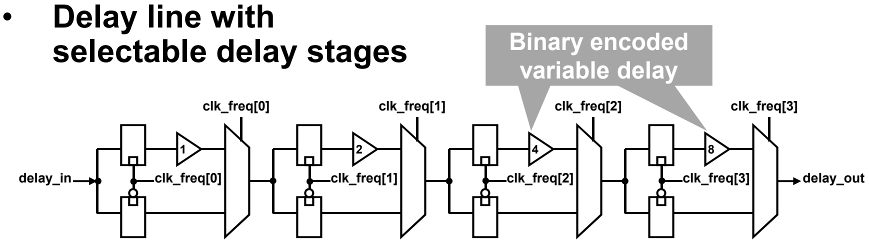
Y. Thonnart, CEA, LETI, MINATEC, France 17

Local clock generator

- **Ring oscillator with programmable delay**
- **Locked during transfers and frequency reconfiguration**



- **Delay line with selectable delay stages**



Y. Thonnart, CEA, LETI, MINATEC, France 18

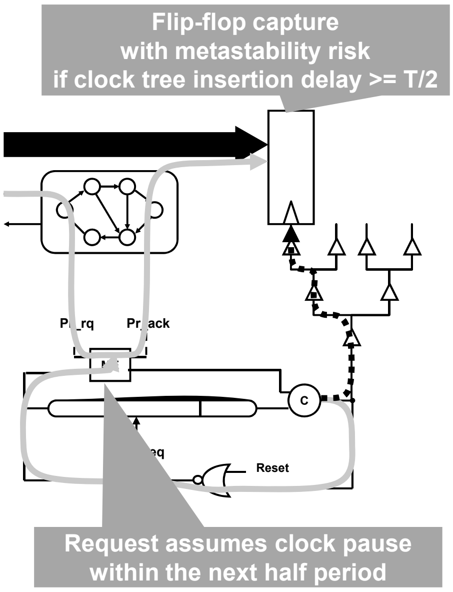
Asynchronous FSMs

- Requests from other domain should
  - Generate asynchronous requests to local clock generator
  - Wait for clock pause acknowledge
  - Wait for data payload
  - Position a valid signal to clock domain
  - Release asynchronous requests
  - ➔ Without the need of a clock
  - ➔ Needs asynchronous FSMs
- designed using Async. Synthesis tools
  - STG / Petri nets (Petrify)
  - Burst Mode AFSMs (Minimalist, 3D)
  - ➔ Sensitive to logic remapping & timing
  - ➔ Hard to optimize for performance

Y. Thonnart, CEA, LETI, MINATEC, France 19

Clock tree insertion issue

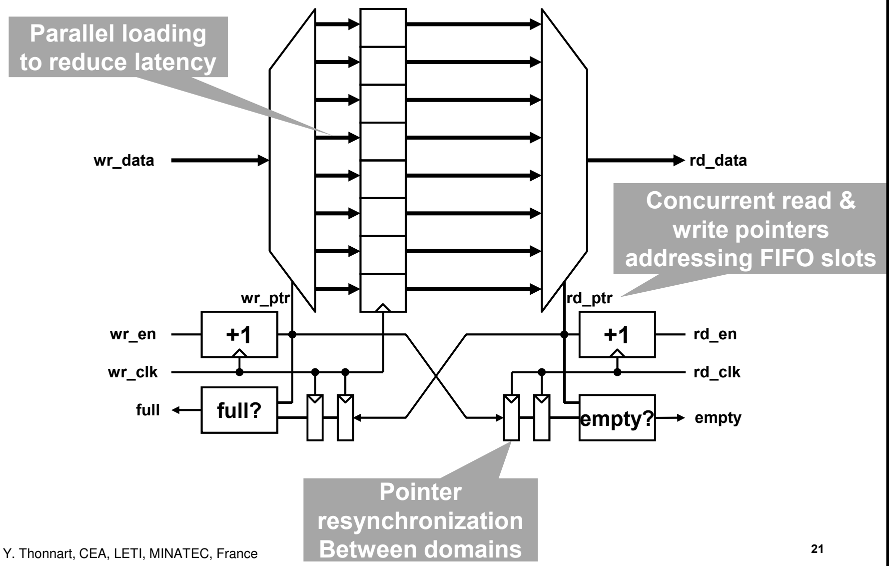
- Pausable clock assumes negligible clock tree insertion delay
  - Clock pause is valid within the clock phase
- Not compatible with big high performance design
  - Clock tree of at most half a clock period
- Still well suited to low-area / low-perf designs



Y. Thonnart, CEA, LETI, MINATEC, France 20

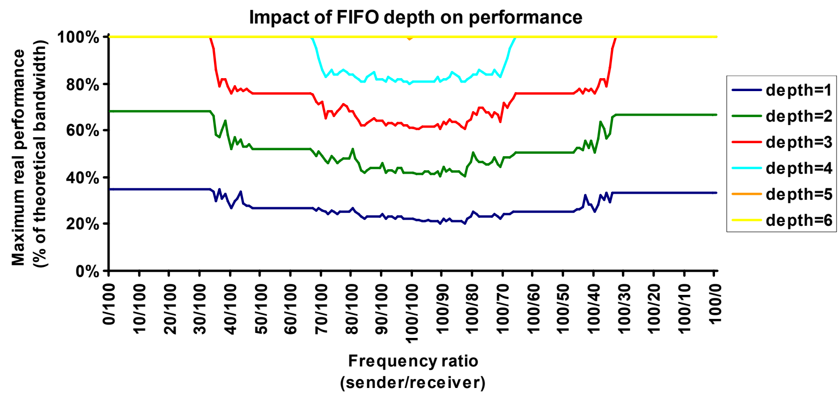
Bi-synchronous FIFO buffers

- **Block Diagram**



Influence of FIFO depth

- **FIFO depth should be at least 5 or 6 to guarantee maximum throughput in similar frequency range**
  - **Depending on datapath width, can be a big area**



Control signals

- Full/Empty detectors
  - Needed for flow control
  - Prevent read and write pointers to take over each other
    - Avoid overwriting an unread data
    - Avoid re-reading an old data
- 2 Strategies
  1. Expose write pointer & read pointer in both domains
     - Resynchronize pointers
     - Compare values in each time domain
     - Need a single bit change between increments to avoid inconsistent resynchronization
  2. Compute state asynchronously
     - Resynchronize only full/empty flags
     - Need specific logic for asynchronous comparison (Chelcea04, full-custom precharge logic)

Cross-domain state encodings (1)

- Gray encoding
  - Compact (2^N values on N bits)
  - (Almost) limited to 2^N values
  - Well suited to RAM based deep FIFOs (>8 - >16)
- Token-based
  - Less compact (~N values on N bits)
  - Adaptable to any FIFO depth
  - Well suited to small FIFOs (depth<8)
    - Bubble encoding (adjacent-2-hot)
    - Johnson encoding (twisted ring)
      - Allows crossover detection (2N values on N bits)
    - 1-hot Fully asynchronous with precharge

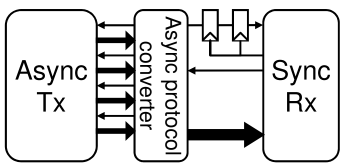
Cross-domain state encodings (2)

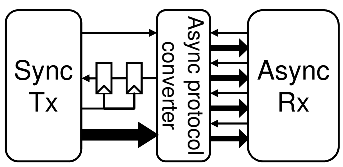
- Crossover issue
  - When write & read pointers are equal
  - Unable to distinguish between full and empty
- Lossy solution
  - Always leave an empty place (almost full)
- Optimized solution
  - ➔ Add a parity bit
  - ➔ Decode real address

Parity	Address	Gray		Johnson	
		coded	decoded	coded	decoded
0	0	0 00	00	0 000	0001
0	1	0 01	01	0 001	0010
0	2	0 11	11	0 011	0100
0	3	0 10	10	0 111	1000
1	0	1 10	00	1 111	0001
1	1	1 11	01	1 110	0010
1	2	1 01	11	1 100	0100
1	3	1 00	10	1 000	1000

Asynchronous communications

- Both pausable clock interfaces & bi-synchronous FIFOs may be adapted to present an asynchronous interface on the global side
- Conversion between synchronous logic and asynchronous logic
  - Better suited to bundled-data asynchronous design
    - – 4-phase indicating logic has a cost
  - Yet 2-phase & BD style are much more sensitive to timing than QDI
    - – Protocol conversion may be used
    - – Robustness comes with QDI



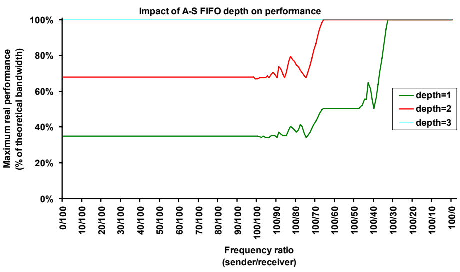


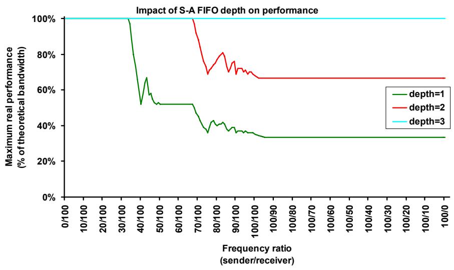
Mixing GALS with fully asynchronous logic

- As relay stations on long distances
  - Synchronous retiming stages would either:
    - Need a new timing domain
    - Extend an existing domain to a non-connex area
    - ➔Always incurs additional latency
  - Asynchronous pipeline on long wires
    - ➔No latency overhead
- With fully-asynchronous routing (NoC)
  - Routing & arbitration is performed in asynchronous logic
  - No top-level timing constraints
  - Lower latency than synchronous equivalent

Compared performance between asynchronous and bisynchronous FIFOs

- Lower latency
  - No need for resynchronization for signals towards asynchronous side
  - Forward latency is reduced in the S-A interface
  - Round-trip delay is closer to 3 cycles instead of 5
- Lower area
  - FIFO size can be reduced without performance degradation because of lower round-trip delay





Part III - Outline

1. GALS overview
   1. Introduction to GALS
   2. Advocating GALS circuits
   3. GALS design challenges
   4. GALS taxonomy
2. GALS intefaces design
   1. Pausable clocking
   2. FIFO interfaces
   3. Interfaces with asynchronous logic
3. <u>GALS physical implementation & test</u>
   1. GALS interface implementation
   2. Test of GALS circuits

Full-custom or Standard-Cell ?

- Bi-synchronous FIFOs can be synthesized from RTL using only the core standard cell library
  - Easiest integration within CAD tools
- Pausable-clock GALS interfaces require specific cells
  - C-element, MUTEX
  - AFSMs can be mapped onto core logic but are optimized with custom cells
- Depending on design style, asynchronous logic uses a few C-elements, or full-custom pre-charge logic
- For GALS design, custom cells are industrially viable
  - Few specific developments
    - no asynchronous computation, always the same needs
  - Massive re-use for all interfaces

Timing constraints

- All paths crossing time domains should be identified
  1. Avoid runaway signals
  2. Data payload is always protected
     by forward and backward control signals
  3. Every control signal needs to be synchronized
     - Single wire change on each event
- Since data is protected by control, no absolute constraint is needed (within 1 clock cycle)
  - False paths can be used
  - Max delay can enforce good buffering
- Control signals
  - On the critical path for latency ➔ Max delay preferred
  - Metastability resolution cycle should be over-constrained with a max delay as close to 0 as possible
- Asynchronous logic needs its own specific constraints
  - Min delay / Max delay / Dont touch

GALS IP reuse

- Simple bi-synchronous FIFOs can be integrated as soft IPs in a synchronous design flow
- Designs requiring unconventional constraints should be isolated into a hard macro:
  - Optimized FIFOs
  - Pausable-clock interfaces
  - Globally Asynchronous interfaces
  - Local clock generators
- The hard macros present synchronous interfaces and may be re-used in a standard design flow
- Care must be taken for:
  - re-entrant clock tree (local clock generators)
  - Asynchronous interfaces with a synchronous semantics

Standalone verification & signoff

- GALS Interfaces delivered with back-end views
  - integration (GDSII, LEF, Lib)
  - verification (verilog, SDF)
- Thorough verification performed standalone with constrained random tests
  - dedicated testbench
  - variable clock frequencies at interfaces
  - variable production/consumption rates to stress corner cases
  - With variable clock tree delay for LCG
- Simulation at gate-level with timing back-annotation
  - Using cross combination of process corners (bc/nom/wc) for SDFs (sender/IF/receiver)
- Simulation using SSTA to account for device variability in performance characterization

Test of GALS circuits - IPs

- IPs tested using conventional methods
  - Scan chains
  - RAM BIST
- 2 options
  - Dedicated test access (JTAG…)
    - Test clock multiplexed with local clock
    - Lockup latches between scan chains from different time domains
  - Functional path reuse for test access
    - Test clock generated depending on the occurrence of data
- IP without interface is about 99% of the potential defects

Test of GALS circuits - Interfaces

- Cannot be covered by the scan chains
- Remaining part tested by functional patterns
  - Good coverage achievable
    - ➔no intensive computation, only communication: independent data bits
    - –Minimal control : self checking request / acknowledge handshake on asynchronous paths, few states on synchronous logic
- 2 options:
  - external trigger of scenarios activating the interface
  - Design specific BIST forcing communication (triggered from test access port)

Part III - Conclusion

- GALS is more and more relevant in complex SoCs:
  - Feasibility
  - Performance
  - Low-Power
- GALS synchronization is a sensitive topic, but robust solutions exist
  - Pausable clock for low area / low speed
  - FIFO based for high performance
- GALS manufacturing is reaching maturity for the industry
  - Mostly bi-synchronous FIFOs
  - True global asynchronous communications are still mostly developed in academia

References

- **D. Chapiro, "Globally-asynchronous locally-synchronous systems"** *PhD dissertation*, Stanford, 1984.
- **C. Seitz, "System timing"** *Introduction to VLSI Systems*, chapter 7 Addison-Wesley, 1980.
- **D. Kinniment, K. Heron, G. Russell, "Measuring deep metastability"** *ASYNC'06* , pp. 2-11, 2006.
- **C. Dike, E. Burton, "Miller and noise effects in a synchronizing flip-flop"** *IEEE JSSC* 34(6), pp. 849-855, 1999.
- **R. Ginosar, "Fourteen ways to fool your synchronizer"** *ASYNC'03*, pp. 89–97, 2003.
- **J. Muttersbach, T. Villiger, W. Fichtner, "Practical design of GALS systems"** *ASYNC'00*, pp. 52-61, 2000.
- **F. Gurkaynak, S. Oetiker, H. Kaeslin, N. Felber, W. Fichtner, "GALS at ETH Zurich: Success or Failure?"** *ASYNC'06*, pp. 150-159, 2006.
- **R. Dobkin, R. Ginosar, C. Sotiriou, "High rate data synchronization in GALS SoCs"** *IEEE TVLSI* 14(10), pp.1063-1074, 2006.
- **A. Chakraborty, M. Greenstreet, "Efficient self-timed interfaces for crossing clock domains"** *ASYNC'03*, pp. 78-88, 2003.
- **T. Chelcea, S. Nowick, "Robust interfaces for mixed-timing systems"** *IEEE TVLSI* 12(8), pp. 857-873, 2004.
- **I. Miro-Panades, A. Greiner, "Bi-synchronous FIFO for synchronous circuit communication well suited for NoC in GALS Architectures"** *NOCS'07*, pp.83-94, 2007.
- **R. Apperson, Z. Yu, M. Meeuwsen, T. Mohsenin, B. Baas, "A scalable dual-clock FIFO for data transfers between arbitrary and haltable clock domains"** *IEEE TVLSI* 15(10), pp.1125-1134, 2007
- **Y. Thonnart, E. Beigne, P. Vivet, "Design and Implementation of a GALS Adapter for ANoC Based Architectures,"** *ASYNC'09*, pp.13-22, 2009.

Y. Thonnart, CEA, LETI, MINATEC, France 37

State-of-the-art of asynchronous logic in the industry

Pascal Vivet,
MINATEC - CEA/LETI,
Department of Design, Architecture & Embedded Software
Pascal.Vivet@cea.fr

DATE 10

March 8th 2010

Outline

- Main actors in academia and in the industry
- Presentation of some main realizations and existing industrial asynchronous circuits
- Presentation of CAD tools & IP & Circuit vendors

Main actors in async. domain : In the industry

- Europe
  - Elastix, Spain – USA/CA
  - Handshake Solutions, Nederlands
  - Tiempo, France
  - Silistix, UK
- USA
  - Achronix,
  - Fulcrum,
  - IBM
  - Intel,
  - Timeless,
  - Theseus *(does not seems alive anymore)*
  - Sun - Oracle

⇒ Mostly startup companies on CAD tools & some circuit niche

⇒ A few R&D labs within major companies (IBM, Intel, Sun)

Main actors in async. domain : In the academia

- In the USA
  - Caltech
  - Columbia Univ.
  - Cornell Univ.
  - Portland State University (ARC lab)
  - UNC Chapel Hill
  - Univ. of British Columbia (Canada)
  - Univ. of South California
  - Univ. of Utah
- In Europe
  - CEA-LETI, France
  - IHP, Germany
  - Cambridge Univ., UK
  - Newcastle Univ., UK
  - Manchester Univ., UK
  - Politechnico de Torino, Italy
  - Technical University of Denmark, Denmark
  - Technion, Israel,
  - TIMA, France
  - UPC, Spain
- In Japan
  - Himeji Institute of Technology
  - University of Tokyo

⇒ This is all the story of a few worldwide specialists …

⇒ The ASYNC IEEE conference series since 1994

Asynchronous Logic : Usefull links

- The asynchronous home page *[Manchester, UK]*
  (Publications, Research Groups, Tools, events, links, …)
  http://intranet.cs.man.ac.uk/apt/async/
- The asynchronous mailing list *[Columbia Univ., NY]*
  asynchronous@lists.cs.columbia.edu

Outline

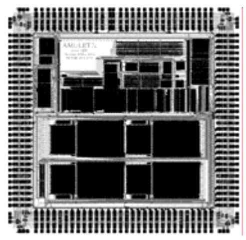
- Main actors in academia and in the industry
- Presentation of some main realizations and existing industrial asynchronous circuits
- Presentation of CAD tools & IP vendors

Main realizations in async. design

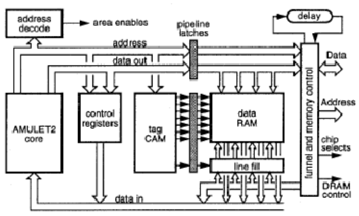
- In the old days, there used to be some asynchronous logic *[Huffman, Muller, Seitz, …]*
- 1980's : The clock became a friend, synchronous design paradigm became the common way. Thanks latter on to high level synthesis (RTL …)
- 1989 : Micropipeline *[I. Sutherland],*
- 1990's : Asynchronous microprocessors,
- 2000's : GALS design and then NoC design

Amulet Processors (Manchester Univ.)

- Amulet1 (*1995*)
  - RISC 32-bits ARM ISA
  - Micropipeline 2-phase
  - Std-cell + Full-custom design
- Amulet2e (*1996*)
  - Micropipeline 4-phase
  - Self-timed cache (4kByte)
  - Sleep mode
  - CMOS 0.5µm, core size 5mm x 5mm
  - 42 MIPS, 150mW @ 3.3V
  - *Equivalent to an ARM7*
- Amulet3i (*2000*)
  - Out of order LD/ST completion
  - Internal asynchronous bus (Marble)
  - Integrated in a complete synchronous ARM SoC
  - 8kbyte RAM, 16kB ROM
  - CMOS 0.35µm
  - 100 MIPS, 215mW @ 3.3V
  - *Comparable to an ARM9 @ 12OMHz*
- Key Advantages ?
  - Reduced power consumption
  - Low noise, low EMI



Amulet2e Layout



Amulet2e Architecture

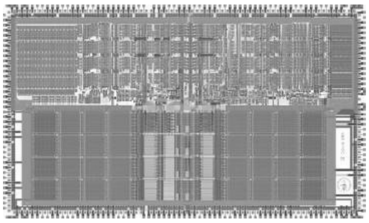
[S.B. Furber, J.D. Garside, ASYNC'97 & ASYNC'99]

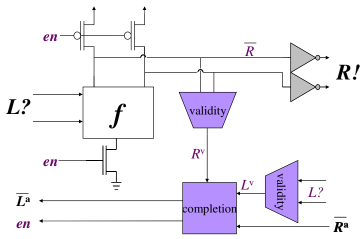
http://intranet.cs.man.ac.uk/apt/projects/processors/amulet/

MiniMIPS (Caltech, 1998)

- **RISC 32-bits MIPS R3000**
  - **Standard 32-bit RISC ISA,**
  - **Precise exceptions,**
  - **4kB ICache + 4kB DCache,**
  - **no TLB**
  - **Out of Order execution**
- **CMOS 0.6µm, 2 Mtransistors**
- **Design Style**
  - **Quasi-Delay-Insensitive**
  - **Deep "2D" Pipeline**
  - **Full custom layout**
- **Performances**
  - **190Mips, 4 Watts @ 3.3V**
  - **100Mips, 850mW @ 2.0V**
  - **60Mips, 220mW @ 1.5V**

⇒ more than x4 in frequency, compared to synchronous commercial version at that time





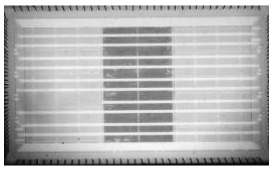
[A.J. Martin, A. Lines, Adv. Res. in VLSI 97]

Pascal Vivet, CEA-LETI, MINATEC

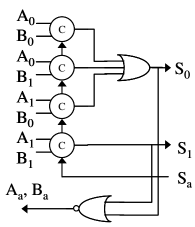
9

ASPRO (Cnet / Tima, 1998)

- **RISC 16-bits processor**
  - **Out of order completion pipeline,**
  - **12kB Program, 64kB Data,**
  - **Asynchronous serial links,**
  - **MAC unit (16x16+32),**
  - **Idle mode,**
- **Design Style**
  - **Quasi-Delay-Insensitive**
  - **Deep pipeline**
  - **Std-cell based design**
  - **CHP2VHDL translator for simulation**
- **CMOS 0.25µm**
  - **6 Mtransistors, 40mm²**
  - **140 Mips, 500mW @ 2.5V**
  - **24 Mips, 27mW @ 1.0V**

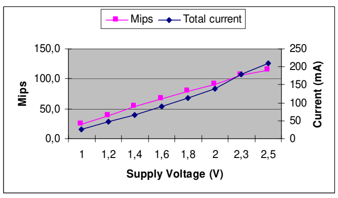


ASPRO layout



Dual-rail Pipelined AND gate exemple

[M. Renaudin, P. Vivet, ASYNC'98, ESSCIRC'99]



ASPRO results

Pascal Vivet, CEA-LETI, MINATEC

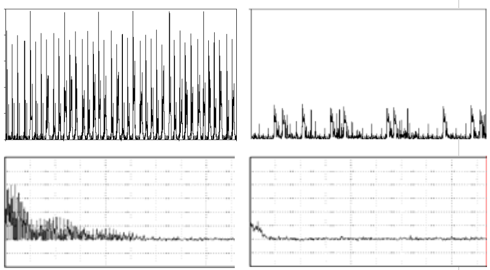
10

80c51 (Philips, 1998)

- Fully compliant 80c51 microcontroller core
  - CISC 8bits, includes low power modes
  - 20kB ROM, 1kB RAM
  - I2C, UART, Timers, DC/DC converter,

[H. Gageldonk, A. Peeters, ASYNC'98]

- Design style
  - Handshake bundled-data circuits
  - Fully designed using Tangram
    - Language and CAD Tools
- CMOS 0.5µm
  - 4 Mips, 9mW @ 3.3V
  - Very low power
    - 4x less wrt sync. version
  - Very low noise
- Integrated in Pagers circuits
  - Mono-chip Pagers system thanks to low-noise digital
  - Do not need to stop clock during TX/RX



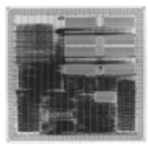
Current in Time and Freq domain
Synchronous and Asynchronous versions

See also a more recent 80c51 version (Lutonium) [Caltech]

Other processors

- TITAC1 & TITAC2 (Tokyo, 1997)
  - MIPS R2000 32-bits
  - Scalable Delay Insensitive
  - CMOS 0.35µm
  - 54 Mips, 2.11 Watts @ 3.3V

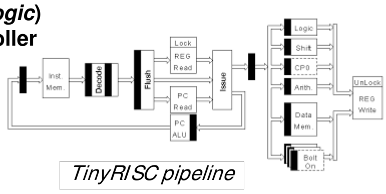
[Taka, ICCD'97]



TITAC2 layout

- Tiny RISC (DTU, 1998)
  - Async version of TR1401 RISC core (***LSI logic***)
  - 4-phase with normally opaque latch controller
  - CMOS 0.35µm std-cell
  - 74-123 Mips, 116mW @ 3.3V

[K. T. Christensen, J. Sparso, Async'98]



TinyRISC pipeline

- The RAPPID prototype (INTEL, 1999)
  - The x86 Instruction Length Decoder
  - 0.25µm, Full Custom
  - Aggressive Timing Hypothesis & Verifications
  - Key advantage : mean time computing : from 2.5 to 4.5 instructions every ns
  - Compared to synchronous version @400MHz :
    - 3x throughput, ½ latency, ½ power

[K. Stevens, M. Roncken, Async'99]

Asynchronous CPU cores (1990's)

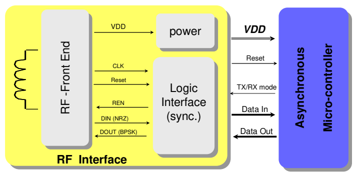
- Many asynchronous CPUs have been built successfully.
- And then, what about async. / sync. comparison of CPU cores ?
  - Can achieve better performance (2x) (*QDI deep pipeline*)
  - Can achieve better power (2x to 4x) (*Bundle Data*)
  - Always at the cost of area (from 1.5x to 2x)
  - At that time, design is complex : no synthesis tools widely available
    - QDI : derivation rules from CHP (channel description)
    - Bundle Data : latch controller from STG (Petrify, …)
    - Sometimes complex timing hypothesis to enforce (RAPPID)
- The main advantage of asynchronous logic is then somewhere else :
  - Mixed mode circuits & low-noise
  - With a tool with Tangram in Philips, which allow industrial products (*Pagers*)
- Other asynchronous experiments not only with CPU Cores ?
  - RFID
  - Crypto-engines

<= Use the low-noise / low-power properties of async logic

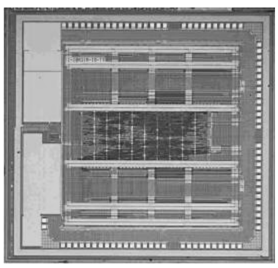
RFID System (Cnet / Tima, 2000)

A Smart Card RFID System, based on an Asynchronous Core

- A Contactless smart card
  - RFID TX/RX module (14443-B)
  - On-chip coil
  - Power reception system
- An asynchronous 8-bit QDI micro-controller
  - not sensitive to supply-voltage variations
  - low power / low voltage / low noise
  - Dedicated Instruction Set (8bit)
  - Low Power architecture
  - QDI 4-phase / 1-of-4 for power reduction
  - Synchronous LD/ST interface to the RFID
- CMOS 0.25µm, Std-cell
  - 1 Mtransistor, 13 mm²
  - 24 Mips / 28 mW @ 2.5V
  - 4,5 Mips / 0.8 mW @ 1.0V



MICABI RFID System



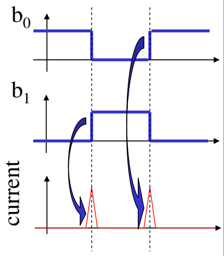
MICABI layout

[Abri, Senn, Renaudin, Vivet, JSSC'2001]

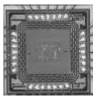
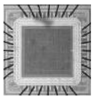
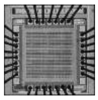
A similar experiment was carried on by Philips in 2000 based on the 80c51

Crypto Engine DES/AES (Tima, Leti, 2003)

- **Avoid attacks through DPA (*Differential Power Analysis*)**
- **Benefits of asynchronous logic:**
  - **Low-noise / low-power**
  - **Symmetry of the dual-rail encoded logic**



[F. Bouesse, B. Robisson, DCIS'2004]

	Synchronous DES	Asynchronous DES	Asynchronous DES
Design Flow & DPA design	Standard tools No specific care	TAST tools Logical Balancing	Manual Design Manual Logical Balancing
Techno & Core Size	180nm, 370µ * 370µ	180nm, 840µ * 840µ	130nm, 975µ * 975µ
Compute Time	-	200ns	200ns
Security Level*	1	60	100

similar works by S. Moore, F. Gurkaynak, …

Pascal Vivet, CEA-LETI, MINATEC

** : measured time to attack the circuit using DPA*

15

GALS design (2000's)

- **An intermediate design style**
  - ***Let's partition the architecture in independent synchronous islands and have asynchronous communications***
- **Benefits ?**
  - ***Use Standard tool for the synchronous design***
  - ***Modular and Scalable Design,***
  - ***Natural enabler for Low-Power (DVFS), Low-Noise***
- **Comparison of the main GALS design styles:**

	Pausable Clocking	FIFO-based	Boundary Synchronization
Area overhead	Low	Medium – High	Low
Latency	Low	High	Medium
Throughput	Lowered wrt. Clock Pause rate	High	Medium
Power Consumption	Low	High	Medium
Additional Cells	Mutex, Delayline, Muller-C	Empty/Full flag	Muller-C, Mutex
Advantages	No Metastability	Simple Solution, Throughput	Low overhead
Disadvantages	Local Clock generators, Throughput	Area Overhead, Latency	Requires verification, Throughput

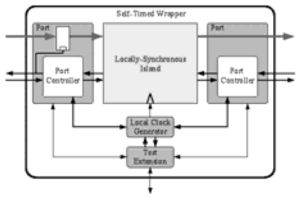
GALS Circuits: Overview and Outlook,
Krstic, M.; Grass, E.; Gurkaynak, F.K.; Vivet, P.
Design & Test of Computers, 2007

Pascal Vivet, CEA-LETI, MINATEC

16

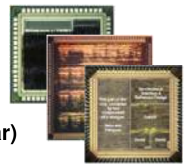
GALS at ETH labs (Zurich)

- Developped a complete GALS methodology
  - Use of Pausable Clocks + LCG
  - Various input / output port controllers
    - – AFSM's synthesized with 3D tool
  - Include a test methodology
  - Can use standard CAD tools with extra scripts



ETH self-timed wrapper

- Designed various prototype chips
  - Marilyn (2000)
    - – Safer SK 128 algorithm. 9 GALS modules.
    - – 0.25µm, 320 MHz max
  - Shir Khan (2002)
    - – Various interconnect topologies (bus, ring, Xbar)
    - – 0.25µm, 220 MHz max
  - Acacia (2005)
    - – AES
    - – 0.25µm, 177,7Mb/s, Energy 1.2mJ/Mb

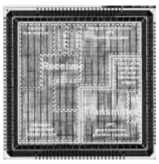


	Merlin	Marilyn	Δ(%)
Area [mm^2]	1.232	1.560	+21
Throughput [Mb/s]	303	232	-30
Energy [mJ/Mb]	737	555	-32
Max. Clock [MHz]	300	240	-25

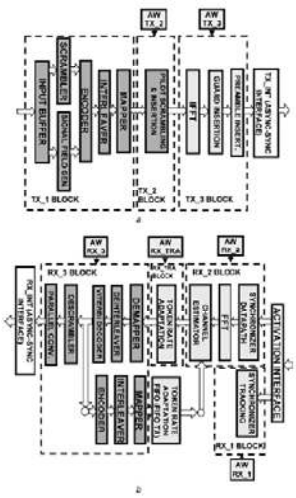
[F. Gürkaynak, GALS at ETH : a success or a failure ?, ASYNC'2006]

GALS at IHP labs (Leibniz)

- IEEE 802.11a GALS baseband processor
  - Includes various IP cores
    - – Viterbi decoder,
    - – FFT, IFFT,
    - – CORDIC processor
- Design style
  - Request driven clock generation scheme
  - Use std-tools & std-cell
  - Asynchronous GALS wrapper
    - ~ 3.5% area
- Results
  - CMOS 0.25µm
  - Freq(IPs) in [20-80 MHz]
  - Power Consumption: 330mW
    - – *same as a gated-clock synchronous version*
  - Noise reduction
    - – 5dB,
    - – 30% max power peak



IHP 802.11a GALS baseband processor



[M. Krstic, E. Grass, ASYNC'2005, CDT'2006]

GALS designs : (intermediate) conclusion

- GALS design overview
  - Not so many working silicon's
  - Recent overview paper [S. Moore, ASYNC'2007]
  - Pausable Clock have some limitations [Ginosar, 2004]
    - *Throughput is intrinsically limited by Clock Tree delay*

- GALS : the Pros ?
  - GALS brings modularity, low-power, and low-noise
  - GALS is feasible, using available CAD tools
    - Library, methodology, test
  - Good partitioning is key !
    - « GALSisfication » to reduce communication cost

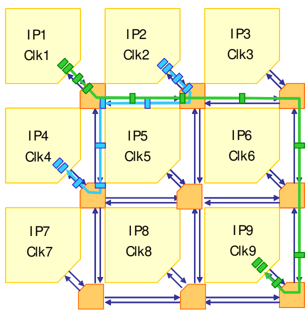
- GALS : the Cons ?
  - With LCG, limitation of clock generator resolution
  - It requires lots of expertise : no real CAD tool & no abstraction yet
  - No real clear advantages compared to synchronous design
    - *On classical System-on-Chip*

GALS and Network-on-Chip

- GALS, a natural enabler for Network-on-Chip
  - NoC brings higher communication semantic to GALS signaling
    - From asynchronous word Xfer to NoC packet Xfer
  - Full benefit of GALS modularity & scalability
    - Fully independent clock domains at large scale
  - Easy partitioning for power control (DVFS)
  - Can cope with on-chip variability

- Many GALS NoC solutions :
  - From multi-synchronous NoC
    - Mesochronous, …
  - To fully asynchronous NoC
    - Asynchronous routers and links,
    - "GALS" interface between NoC & IP

- Recent Fully asynchronous NoCs
  - Mango (DTU)
  - ASPIN (LIP6)
  - QNOC (Technion)
  - ANOC (CEA-LETI)
  - Chain (Silistix)

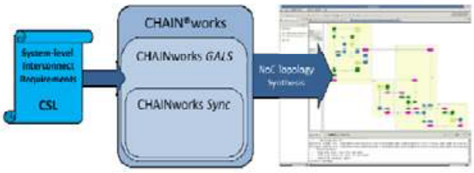


A GALS NoC Template

Silistix

http://www.silistix.com

- Provide a fully asynchronous GALS interconnect
  - Asynchronous QDI design style
  - Use of M-of-N encoding (2-of-7 ?)
  - Building block to build a GALS NoC interconnect.



- CHAIN®works tool chain :
  - Use of CSL (Connection Specific Language)
  - To explore & generate an interconnect architecture
  - Full support of standard CAD tools and languages

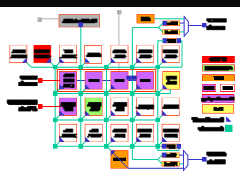
http://www.silistix.com

- High level integration
  - Support connection of various bus protocol (AHB, AXI, OCP)
  - Include packetisation, serialisation, variable link width, …
  - Support for clock gating, DVFS, multi-clock, …
  - Support for synchronous only interconnect (for small designs)

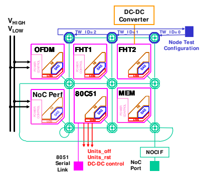
Further references can be found in [J. Bainbridge, S. Furber, IEEEMicro'2002]

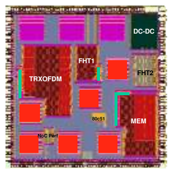
ANOC-based prototypes (Cea-Leti)

- **FAUST (2005)**
  CMOS 130 nm – 20 routers
  SISO OFDM and CDMA
  Reconfigurable chip
  GALS if : Gray FIFO, 170 MHz
  ANoC ~ [150-200] MHz
  [ISSCC'07, JSSC'08]

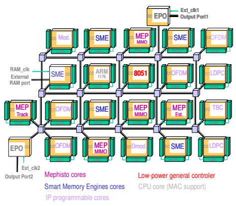


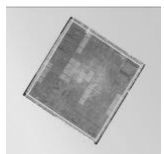
- **ALPIN (2007)**
  CMOS 65 nm – 9 routers
  Low-power demonstrator
  DVFS, Router power-down
  GALS if : Pausable Clock, 200 MHz
  ANoC : 500 MHz
  [ASYNC'08, NOCS'08, VLSI'08, JSSC'09]





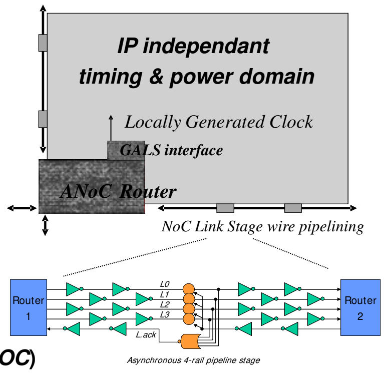
- **MAGALI (2008)**
  CMOS 65 nm – 15 routers
  MIMO 4G Telecom
  Multitask / Reconfigurable cores
  GALS if : Johnson FIFOs, 500 MHz
  ANoC : 550 MHz
  [NOCS'09, ISSCC'10, DATE'10]





ANOC : Design and Results

- Design and Back-End Methodology
  - QDI 4-phase ; 1-of-4 ; std-cell
  - Use Hard Macros
    - ANoC Router
    - ANoC GALS interfaces
  - Use asynchronous pipelined Links
    - long wire optimization
  - Place & Route timing modeling ?
    - Use a dummy synchronous model …



- Performances (*CMOS 65nm*)
  - Comparison with synchronous design (*SNOC*)

	ANOC	SNOC
Area	0.17 mm2	0.094 mm2
Frequency	550 MFlit/s	480 MHz
Latency	2.3 ns	4.2 ns
Average Power	11.2 mW	82 mW

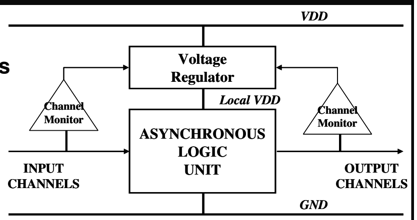
Pascal Vivet, CEA-LETI, MINATEC

[Y. Thonnart, P. Vivet, DATE'2010]

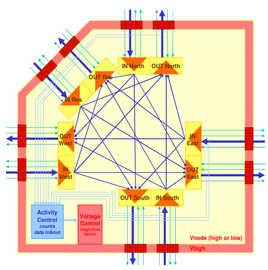
23

ANOC : Low-Power control

- Low-Leakage ANoC Router : Objective
  - Detect inactivity on asynchronous channels
  - Place design in idle/standby mode



- Define on/off message in NoC protocol
  - Count number of active NoC messages
  - When counter=0, switch down the router
  - Fast weak up time with new incoming message
- Thanks to Asynchronous logic robustness
  - Logic is full functional between 0.7V to 1.2 V



	Original version	Power-On	Power-down
Supply voltage	1.2V	1.2V	0.6-0.8V
Flit Throughput	1.8 ns (550Mflit/s)	1.8 ns (550Mflit/s)	7.2 ns (140Mflit/s)
Flit Latency	2.3 ns	2.5 ns	5.8 ns
Leakage	200 µA (240µW)	210 µA (250µW)	80 µA (100µW)
Energy	30 pJ/flit	30 pJ/flit	14 pJ/flit

Pascal Vivet, CEA-LETI, MINATEC

[Y. Thonnart, P. Vivet, ASYNC'08]

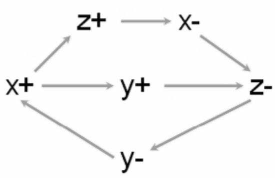
24

Outline

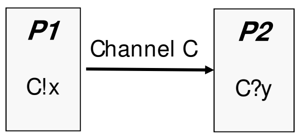
- Main actors in academia and in the industry
- Presentation of some main realizations and existing industrial asynchronous circuits
- Presentation of CAD tools & IP vendors

Asynchronous Logic Tools

- From « Asynchronous FSM » synthesis tools
  - Tool examples :
    - 3D, Minimalist, Petrify, …
  - Target synthesis of asynchronous controllers
    - Complexity < 100 gates
    - Ex : bundle-data micro-pipeline controllers



- To “language level” synthesis tools
  - Handshake Channel communication semantic is mandatory,
    - CSP-like languages
    - A blocking communication models an asynchronous handshake channel
  - Benefits
    - Designer focus on behavior,
    - not on control and implementation !
  - Process decomposition
    - preserves behavior, add pipeline



P1 and P2 synchronize, as a result y:=x

Asynchronous Languages and Tools

- **Introduction of some new languages**
  - **CHP [A.J. Martin, Caltech, 1989]**
  - **BALSA [Univ. Manchester, 1995]**
  - **HASTE (Handshake solutions) - (*prev. TANGRAM in Philips*)**
- **Add asynchronous channel semantic to existing HDL languages**
  - **Enhanced SystemC [C. Koch-Hofer, TIMA]**
  - **Enhanced VHDL (Theseus Logic)**
  - **Verilog (TIMELESS)**
  - **System Verilog (TIEMPO)**

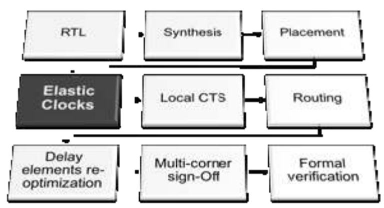
Verilog Fork-Join used to model intra-process concurrency

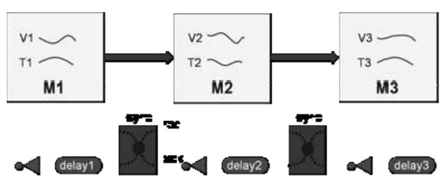
- **Use standard RTL language**
  - **And transform it into asynchronous design …**
    - *Topic Widely covered by J. Sparso before*

ELASTIX

http://www.elastix-corp.com/

- **Elastix Startup Company**
  - **Co-funded in 2007 by J. Cortadella**
  - **Located in Barcelona and US**
- **Complete CAD tool flow**
  - **Start from an existing RTL,**
  - **Transform it to asynchronous pipeline,**
  - **Tools to support B&E implementation**
    - ***Using classical P&R tools***
- **What benefits ?**
  - **A slight increase in area,**
  - **Some gains in power,**
    - **Mainly limit the peak power**
  - **Performances will benefit from :**
    - **Mean time computing,**
    - **Variability margins,**
    - **GALS and DVFS**





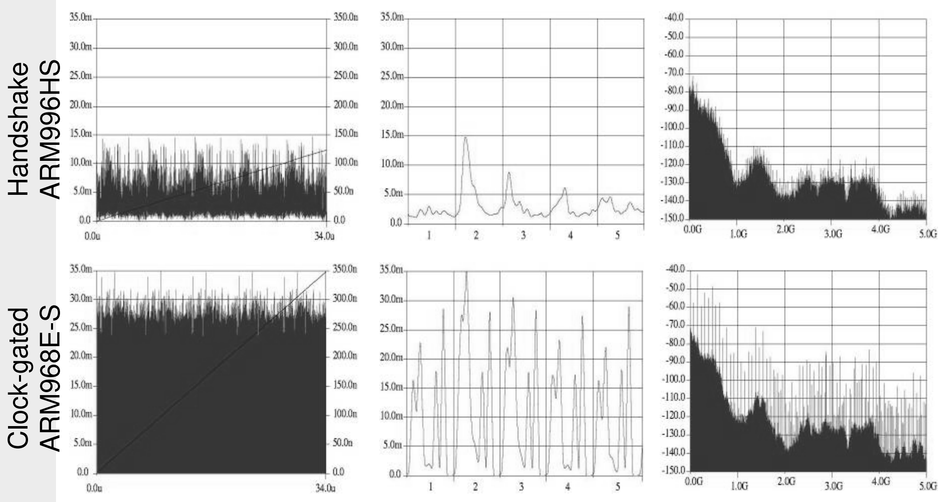
J. Cortadella, A. Kondratyev, L. Lavagno, and C. Sotiriou. « Desynchronization: Synthesis of asynchronous circuits from synchronous specifications ». IEEE Transactions on Computer-Aided Design, pp. 1904–1921, October 2006.

- **HANDSHAKE SOLUTIONS**

http://www.handshakesolutions.com/

- **Started as a Philips Research Project in 1986**
- **Went through an incubator in Philips and in NXP**
- **Eindhoven, NL**

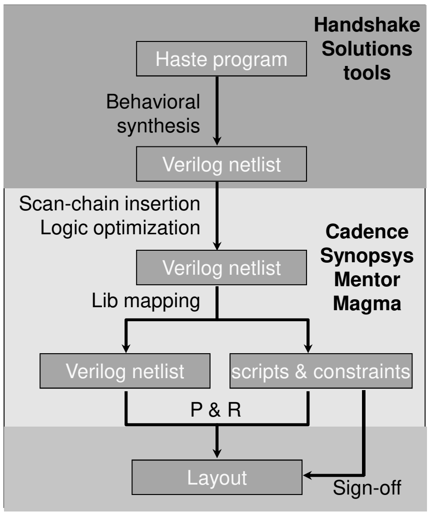
Technology benefits



Power Current peaks Emission

Solution ingredients: TiDE
Timeless Design Environment

- TiDE is a frontend to your existing EDA flow
- TiDE is *complementary to* and *compatible with* third-party EDA tools
- High-level design entry (Haste)
- Standard-cell hand-over
- Scan-test-based Design-for-Test
- FPGA prototyping through synchronous preview of design
- Integrated support for placement and routing, logic optimization and timing sign-off



Pascal Vivet, CEA-LETI, MINATEC

31

Market success

- **TiDE has been used for a range of geometries:**
  - **0.8µ, 0.6µ, 0.35µ, 0.25µ, 0.18µ, 0.14µ, 0.13µ**
  - **90nm, 65nm, 45nm**
- **More than 750 million ICs with Handshake Technology sold**
- **Applications in:**
  - Smartcards
  - Automotive
  - Wireless connectivity

32

Smart card controllers
Products and derivatives

Energy efficiency enables high performance in contactless operation *and* extra non-volatile memory

- More than 80% of the world's smart passports
- Access control at NASA
- Nokia's 6131 NFC phone

Pascal Vivet, CEA-LETI, MINATEC

- **TIEMPO**

www.tiempo-ic.com

- **Created in 2007**
- **Result of research within INPG/TIMA laboratory**
- **Grenoble, France.**

Pascal Vivet, CEA-LETI, MINATEC 34

TIEMPO overview

- Tiempo offers powerful asynchronous core IPs supported by an innovative design and synthesis flow for low power embedded electronics and secured devices
- Tiempo asynchronous design technology:
  - Is fully clockless (i.e. no local clocks)
  - Is delay insensitive = functionally correct regardless of any delay in gates and wires (no delay assumption)
  - Allow designs with both ultra-low power and high performances
  - Can be described with high-level models, in standard language

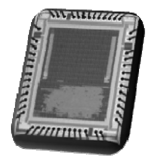
Tiempo

About Tiempo
- Created in 2007
- 21 people
- Located Near Grenoble, France

TIEMPO offer

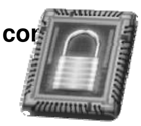
www.tiempo-ic.com

- Tiempo IP portofolio:

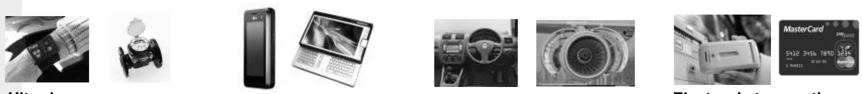


TAM16: ultra-low power 16-bit microcontroller
- < 50µA/MIPS
- fast wake-up
- Silicon Proved

TAK5: ultra-low power crypto-processor family
- TDES: DES/3DES IP cor
- TAES: AES IP core
- TPKA: RSA/ECC co-processor IP



- Target Applications:



Ultra-low power embedded electronics · Mobile consumer electronics · Automotive · Aerospace · Electronic transactions

- **People using asynchronous logic … but not (*really*) saying it …**

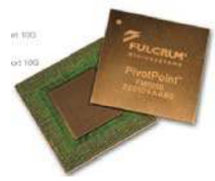
- **FULCRUM**

http://www.fulcrummicro.com/

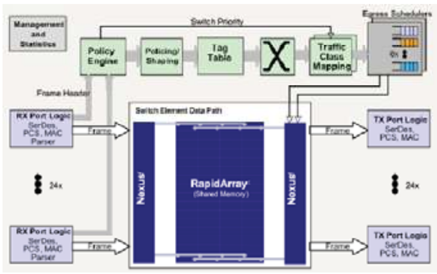
- **Funded in 2000 by A. Lines and U. Cummings**
- **Commercialize Caltech asynchronous technology**
- **70 people (Los Angeles area)**

Fulcrum Microsystem

- Ethernet Switch products
  - PivotPoint FM1XXX (2003)
    - 6 port SPI-4 14.4G switch
  - FocalPoint FM2XXX "Tahoe" (2005)
    - 24 port 10G Ethernet L2 switch
  - FocalPoint FM4XXXX "Bali" (2007)
    - 24 port 10G Ethernet L3/L4 switch/router



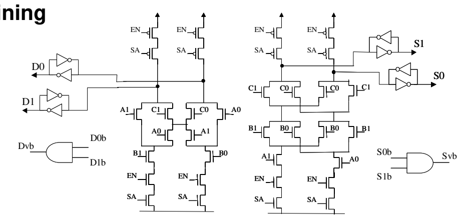
- Key advantages
  - Lowest switch latency (200ns)
  - Better performances / power
  - Typical performances (130nm TSMC)
    - Freq ~ 750 MHz
    - Power ~ 1.5W per Ethernet port
- Competitors
  - BroadCom, Marvell



Exemple of Switch diagram

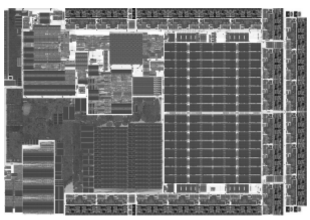
Fulcrum : Design Style

- Quasi-Delay-Insensitive (QDI) timing model
  - 4-phase ; 1of N channels
  - Domino Logic with Integrated Pipelining
    - Precharge 1of N domino logic
    - Combines latching with logic
    - WCHB, PCHB, PCFB templates
  - Typically 18 transitions per cycle
  - Full-custom layout



Exemple of Ful-Adder PCHB cell

- Some notable circuits ?
  - Single & Dual Ported SRAM up to many MB's
    - 1K by 16-bit banks
    - Pipelined interconnect scalable
    - Comparable area
  - Clock-domain-conversion
    - with synchronous ASIC cores
  - Crossbars
    - Very small & low latency crossbars



BALI layout

Fulcrum : Pro's & Con's

- Avantages :
  - High speed operation
    - Products run at 750MHz in TSMC 130nm LVOD
    - Very low latency
  - Implicit power gating
    - Inactive circuits consume no active power
    - No glitching, less EMI, smaller dI/dt
  - Top-down design by "decomposition"
  - Modularity and Reuse
    - Specifications are simpler without timing information
    - Mixed frequencies are easy to integrate

- Disadvantages:
  - Larger area
    - Remove clocks, add handshake circuitry
    - Logic often 4X larger than synchronous ASIC
    - Perhaps closer to full-custom synchronous overhead?
    - However, SRAM's are similar area at higher speed
  - Custom CAD tools
    - Commercial tools don't support most of the flow
    - Many tools at all stages of design had to be developed
  - Lack of experienced designers

- **ACHRONIX**

achronix SEMICONDUCTOR CORPORATION

http://www.achronix.com

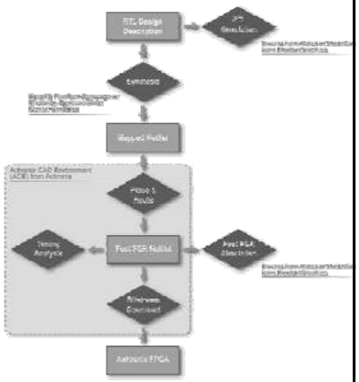
- **Startup created in 2004, in San Jose, USA**
- **Founder and CTO, R. Manohar, partly in Cornell Univ.**

ACHRONIX

- World's fastest FPGA products
  - Internal frequency 1.5 GHz
  - IO's & Interfaces compatible with classical FPGA requirements
    - DDR3 1066 Mbps
    - LVDS 1000 Mbps
    - Ethernet, Sata, PCI-express, …

- FPGA architecture
  - Fully asynchronous and pipelined
  - New template of FPGA array
  - Complete Tool Chain to implement classical RTL

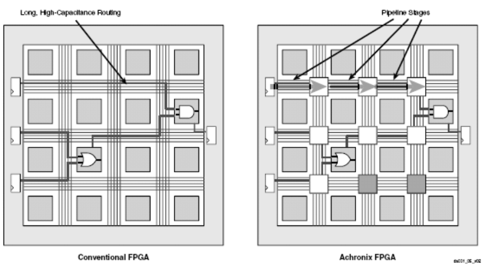
=> With 1.5 GHz, get faster RTL design that a std-cell ASIC !



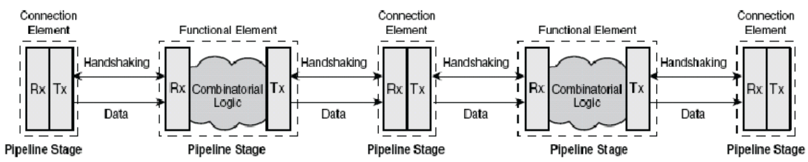
- Application Domains ?
  - Networking
  - Telecommunications
  - Encryption
  - High-performance computing
  - Video and Imaging
  - Digital signal processing
  - Industrial
  - Test and measurement
  - Military and aerospace

ACHRONIX

- FPGA architecture
  - No global clock
  - Fully asynchronous and pipelined
    - *Including the FPGA cell interconnections*



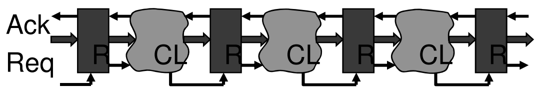
- Asynchronous Technology
  - QDI logic (dual-rail / 4-phase)
  - picoPIPE™ technology
    - Domino Logic with pipeline
    - « Synchronous RTL » of the design is embedded within async handshaking



Outline

- Main actors in academia and in the industry
- Presentation of some main realizations and existing industrial asynchronous circuits
- Presentation of CAD tools & IP vendors
- Conclusion

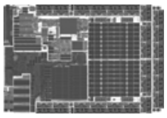
Conclusion (1)

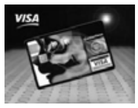


- **Asynchronous logic ?**
  - **No clock !**
    - **Large design space to play with timing & synchronization schemes**
  - **Lots of asynchronous design styles**
    - **With more or less timing hypothesis**
    - **Actually not opposed to each others : a continuum from pure synchronous to strictly delay insensitive**
  - **Still some on-going R&D on the "basis"**
    - **New asynchronous protocols, …**
    - **Arbiters & Metastability,**
- **Asynchronous logic well known advantages ?**
  - **Modularity, Mean Time computing, Low-Power, Low-Noise, Robustness, …**
  - **Fast asynchronous circuits (QDI style) can bring great performances,**
    - ***for easier full custom layout***
  - **Slower asynchronous circuits (Bundle Data) can bring low power / low noise**
    - ***for embedded applications***

Conclusion (2)

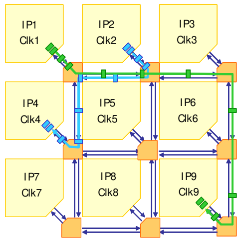
- Several R&D demonstrators and some real industrial products :
  - CPU cores,
  - Crypto engines,
  - Pagers, SmartCards, Passports,
  - Async FPGA, Ethernet Switches,





- GALS has an intermediate solution
  - Natural enabler for DVFS, variability control,
  - No real clear advantage when doing GALS only,
  - But GALS NoC has a compelling advantage
    - for building efficient MPSoC architectures
  - A startup :

Silistix Interconnect Driven Design



Pascal Vivet, CEA-LETI, MINATEC 47

Conclusion (3)

- Testability is an issue
  - There exists some solutions,
  - Not always compliant with industry standards
- Asynchronous logic is a difficult technology
  - Still reserved to a few specialists
    - Weight of education & not so many large companies in the game
  - Need CAD tools at high level entry !
    - Handshake Solutions is *(was ?)* providing a complete CAD solution
- High Level synthesis recent works
  - High level optimization with code-2-code optimization
  - Transforming synchronous design to asynchronous design
    - ELASTIX company
- Some new CAD technologies are appearing :
  - TIEMPO
  - TIMELESS

ELASTIX

Tiempo

Pascal Vivet, CEA-LETI, MINATEC 48

- *All my thanks to people who provided me information*
- *All my apologizes to work I did not mentioned*

Any questions ?

Come to :

ASYNC'2010
3-6 May 2010,
Grenoble, France.

http://asyncsymposium.org/async2010/Home.html