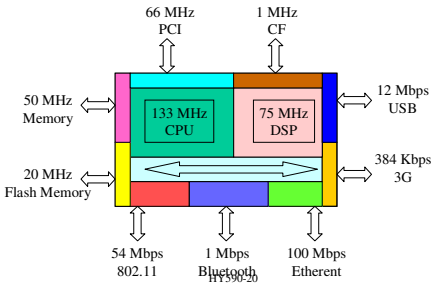
HY-590.20

Χρονισμός Ψηφιακών Συστημάτων
Χρήστος Σωτηρίου
1

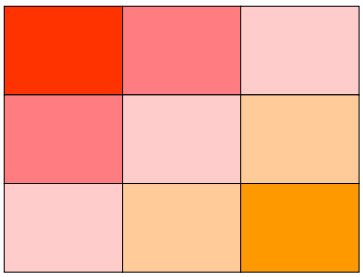
What's the problem?
An example SOC – 12 clock domains



Clock Relationship Classes, Synchronization Types

Class	Δf	Δϕ	Synchronizer
Synchronous	0	0	None
Mesochronous	0	ϕ_c	Phase compensation
Multi-synchronous	0	drifts	Adaptive phase compensation
Plesiochronous	$f_d<\varepsilon$	Varies	Adaptive phase compensation
Periodic	$f_d>\varepsilon$		Predictive
Asynchronous			Two-FFs, 2P-FIFO

Another, related challenge: Mesochronous SoC



Why Asynchronous Circuits?

- We are used to sync design
  - Logic and timing aspects are simpler
- Common arguments:
  - Low power (works)
  - High speed (very hard)
  - Low emission (works)
  - Low sensitivity to PVT variations (works)
    - Process, Voltage, Temperature
  - High modularity (SoC)
  - No clock distribution and timing problems (works)
  - Secure chips (kind of)

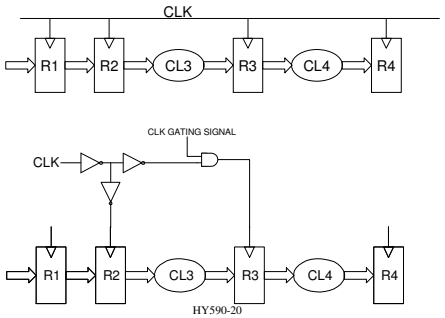
Why Not to Go Async

- Overhead (area, speed, power)
- Hard to design
  - Non-decomposable to small combinational logic blocks
  - Converting sync to async typically fails
- Few CAD tools

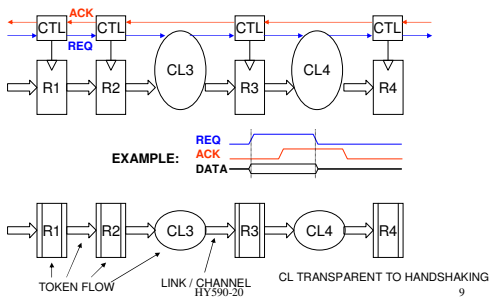
Why do we care about Async?

- We have to.
- Sync is only a nice model for small worlds.
- Async realities:
  - On-chip clock domain interfaces
  - Off-chip communication timing
  - Sync techniques get ridiculously complex due to ignorance of Async methods
  - Modular SoC

Clocking replaced by Handshaking



Clocking replaced by Handshaking

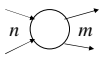


Data Token Flow (PTnets/FSMs)

- Transfer of one token ≡ one handshake cycle
- Register *k* is FULL when it has data
- When register *k+1* gets the data from *k*,
  - Register *k+1* becomes FULL
  - Register *k* now has BUBBLE (≡ data that has already been copied)
- FULL register cannot receive data.
  Only BUBBLE register may receive data.

Token Preservation

- Tokens do not disappear
- Tokens do not appear (from nowhere)
- One token does not overtake another
- A block (register or CL) with *n* inputs and *m* outputs:
  - (when it has a BUBBLE) waits for *n* tokens on inputs
  - Generates *m* tokens on outputs

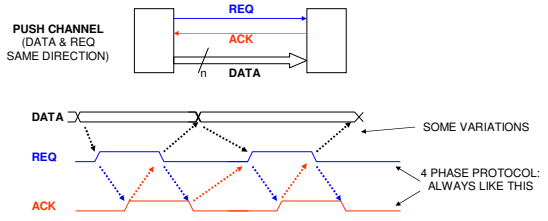


Comments on the Tokens Game

- Abstract all communications (handshake) and computations
  - Hide implementation details
- CL is transparent
  - Special type of CL required: "Function Blocks"
- Local "clocks" spread over time
  - Lower power
  - Lower emissions
  - No need to synchronize events
- Before playing more token games, let's consider some implementation details

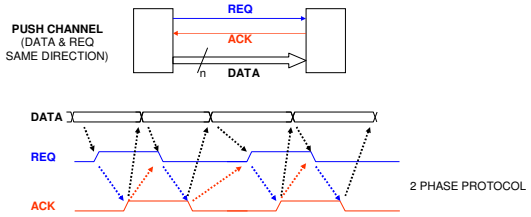
Handshake Protocols

- Bundled data (aka “single rail”)



Handshake Protocols

- Bundled data (aka “single rail”)



Bundling Assumption

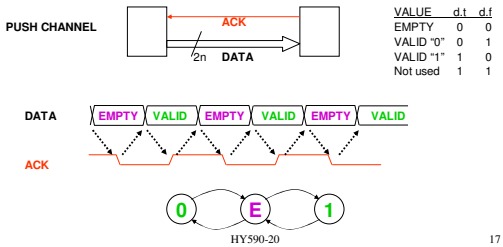
- Each data line is a single wire
  - “Bundled data” aka “single rail”
- On sending end, time(REQ) > time(DATA)
- This order is preserved on receiving end: *Valid(DATA) @ REQ* [≡ data valid precedes REQ=1]
- Non-trivial: inter-line skew must be taken care of and hidden
  - Placement and routing
  - Safety margins at sending end
  - Buffer insertion

4-phase vs 2-phase

- “return to zero” (RZ) is overhead (time and power)
- “level signaling”
- “non-return to zero” (NRZ) seems to have lower overhead
- “transition signaling”
- But implementation is more complex

4-phase dual rail protocol

- Each data bit encoded into 2 wires



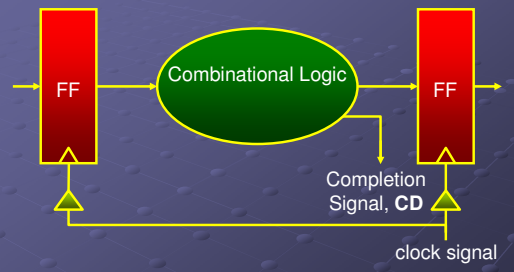
4-phase dual rail protocol

- *Delay Insensitive (DI)*
  - Each bit can propagate at own speed
- 4 phase at higher level (than signals):
  1. Sender sends valid word (V)
  2. Receiver sets ACK↑
  3. Sender sends empty word (E) (removes the data)
  4. Receiver sets ACK↓
- Each change is acknowledged / indicated
- Problems: Glitches, hazards

Classification of Protocols

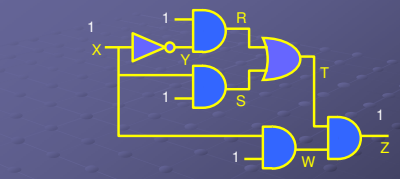
- Handshake: 2-phase or 4-phase
- Direction: Push or pull
- Encoding: Bundled data (single rail), or dual rail, or 1-of-n, or m-of-n, …

Completion Detection



The idea is known → - Added value, - Acceptable cost } Novelty

Hazards in Combinational Logic



- single input change at X.
- non-monotonic transitions.
- static-1 hazard at T.
- 1-0 hazard at Z.

Two-phase Combinational Logic

Two-phase Combinational Logic

Two-phase Combinational Logic

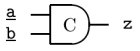
Transitions at the outputs are **monotonic**: from the spacer to the code and back

Acknowledgement/Indication

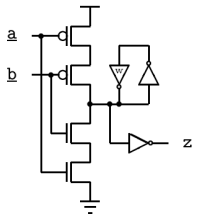
- A gate/circuit acknowledges its input if, for every input change, there is an output change.
- Example: Wire
- Non-indicating example: AND gate
  - Acknowledges all ones: {01,10}→11
  - Does not acknowledge 00→{01,10}

Muller C-Element

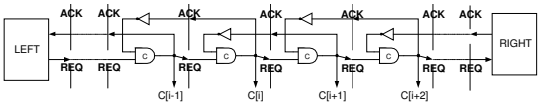
A	b	z
0	0	0
0	1	no change
1	0	no change
1	1	1



- Alternative specs:
  - If a=b then z:=a
  - a=b ∛ y:=a
  - y:=ab+y(a+b)



Muller Pipeline

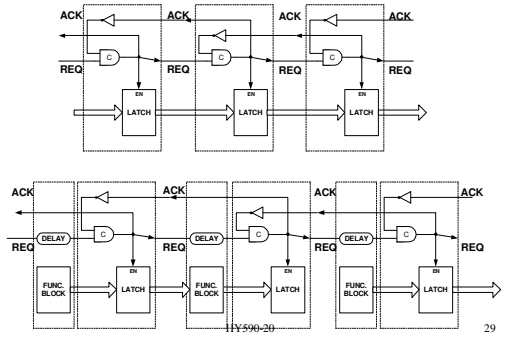


- "The" delay-insensitive handshake machine
- C[i] accepts 1/0 from C[i-1] only if C[i+1]=0/1
- Think of 1010101.. as waves: $1_0\,1_0\,1_0\,1$..
- The C-elements propagate waves precisely
- Timing depends on local delays, may vary along the pipe
- If RIGHT is quiet, pipe will fill and stall
- Same for 4-phase, 2-phase
- Symmetric – same right-to-left (like electrons and holes)

Pipeline Styles

- All based on Muller Pipeline
- 4-phase bundled data:
  - similar to sync pipes
  - based on timing assumptions
- 2-phase bundled data:
  - aka *micropipeline*
- 4-phase dual rail:
  - "the original" Muller pipe

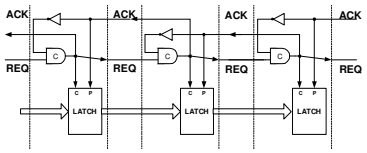
4-phase bundled data circuits



4-phase bundled data circuits

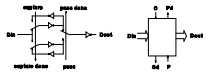
- Looks like a sync pipe, with local clocks
- When full, the C-elements are 1010101…, → only half the latches store data
- Similar to master-slave flip-flops
- Speed limited by handshake (2-way comm)
- We will study better implementations

2-phase bundled data (micropipelines)



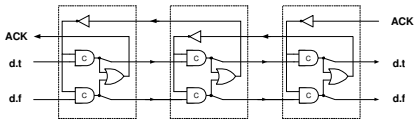
- Transition signaling
- Special "capture-pass" latches alternate between capture and pass

Capture-Pass transition-controlled latch



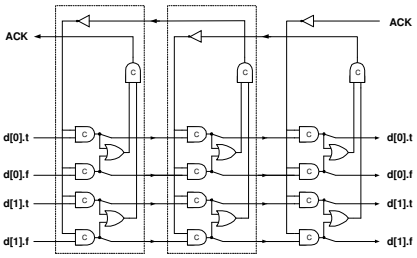
- Transitions on C and P alternate
- Micropipelines "Elegant", no RZ overhead
- But implementation (latches and other control circuits) is complex

4-phase dual rail circuits

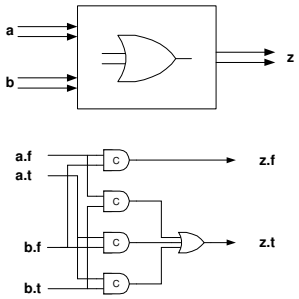


- Muller pipeline (again) with *Completion Detection*
- No REQ – embedded in the data

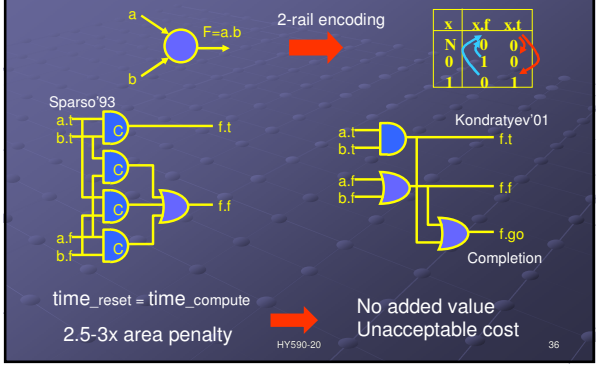
4-phase dual rail – many bits



4-phase dual rail – function blocks

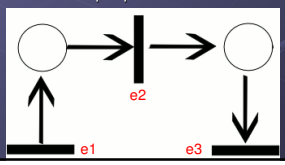


Dual-Rail Implementations



Higher-order Sequential Models

- Petri-Net
  - graph-based mathematical model
  - place and transition nodes
  - directed arcs connecting places to transitions and transitions to places
  - places hold tokens
  - transitions "fire" tokens and move them from place to place
  - transitions may be enabled or not (e1, e2, e3)
- Petri-Net Properties
  - **Liveness** : every transition will fire infinitely
  - **Boundedness**: tokens per place



Higher-order Sequential Models

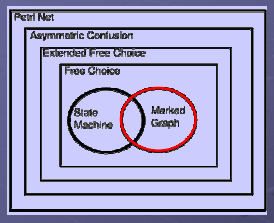
- Petri-net model can be reduced to:
  - **State Machine**
    - **exactly 1 incoming and 1 outgoing arc per transition**
    - **NO CONCURRENCY (never 2 tokens active)**
    - **CONFLICT!**
  - **Marked Graph**
    - **exactly 1 incoming and 1 outgoing arc per place**
    - **NO CONFLICT (never 2 choices possible)**
    - **CONCURRENCY!**

State Machine constrained Petri-net

Marked-Graph constrained Petri-net

Higher-order Sequential Models

- Petri-net Taxonomy



Marked Graph

Shorthand

- Signal Transition Graph (STG)
  - Free-choice Marked Graph
  - Places hidden for Visibility, only transitions shown
  - Transitions correspond to signal changes, x+ or x-