Asynchronous Communication and Self-timed Systems: Can they help to simulate Brain?

Alex Yakovlev, Fei Xia
Microelectronic Design Group
Elec.Elec&Comp Eng, Newcastle

Acknowledgement to Ran Ginosar (Technion) for providing slides about ITRS

Agenda

- What seems difficult … but still possible in Silicon (messages from ITRS)
- The problem of System Timing
- Why we want to design asynchronous systems
- What should be communication/timing mechanisms in brain-like machines?

Int. Technology Roadmap for Semiconductors

- Published every two years (last 2003, http://public.itrs.net/)
- Industry driven: fabs, equipment, EDA, design, testing, integrated companies (Intel,…)
- 15 years outlook (6 short-term, 9 long-term)

It says for example that:

- 2010 will bring a system-on-a-chip with:
  - 4 billion 50-nanometer transistors, run at 10GHz
  - Moore's law: steady growth at 60% in the number of transistors per chip per year as the functionality of a chip doubles every 1.5-2 years.
- Process parameter variation, power dissipation, clock distribution, design productivity (validation and test) present new challenges for Design and Test

ITRS: Four main product areas

- High speed microprocessors
  - 300 mm^2 area, highest density, highest clock rates
- DRAM: Highest density, special niche
- Analog / Mixed signal: LNA, PA, VCO, ADC
  - Challenges: Automated design (lack of designers), low Vdd, high device variation, high noise, high leakage, SOC integration
- System on Chip (used to be ASIC)
  - Smaller dies (5-50Mtx/2001), clock 10% of max, low power

Growing Silicon Complexities

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

Growing System Complexities

- Reuse:
  - Heterogeneous SOC,
  - AMS
- Verification:
  - Spec design,
  - DfV,
  - hSOC,
  - AMS
- Test:
  - Noise/delay testing,
  - Test reuse,
  - Tester timing
- Implementation Portability:
  - Fab-independent designs
- Embedded Software Design:
  - Co-design,
  - Co-verification,
  - Platform-based design
- Design Management:
  - Team size,
  - Geographic distribution,
  - Data management,
  - Metrics,
  - Supply chain management

Implications of Complexities

- No chip-wide synchronization
  - Asynchronous design suggested as "challenge"
- Statistical behavior of transistor / gate / cell
- Some signals lost sometimes
  - Error-tolerant design

Network on Chip (NoC)

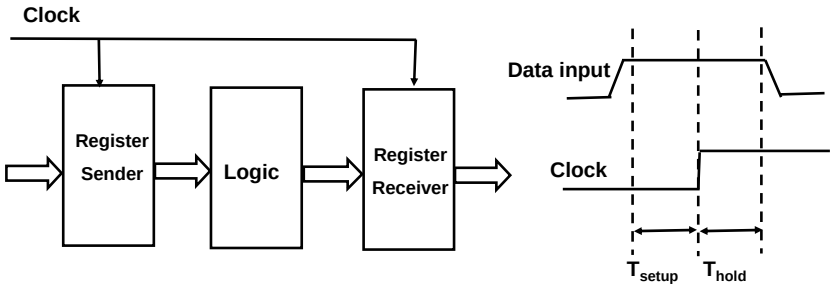
- Single bus does not meet needs, even with:
  - Spilt transactions
  - Pre-emptive scheduling
  - Dynamic priorities
- Complex design
- Load limits speed
- Many off-bus special cases
- New approaches:
  - Multi-buses – keep the bus semantic / software
  - Nets hidden behind bus semantic
  - On-chip "nets" – just send a packet…

New Types of SoC

- Platform-based SoC
  - Many IPs and NoC pre-packaged
  - Add your own logic and submit to fab
- SOPC: System on Programmable Chip
  - Many IPs and NoC pre-packaged
  - A gate-array for your own logic:
    - Customizable:
      - Chips was fabbed most of the way
      - Only last few metal layers waiting for your custom part
    - Field-Programmable:
      - Chip bought as is, customized by customer like FPGA

The Problem of System Timing

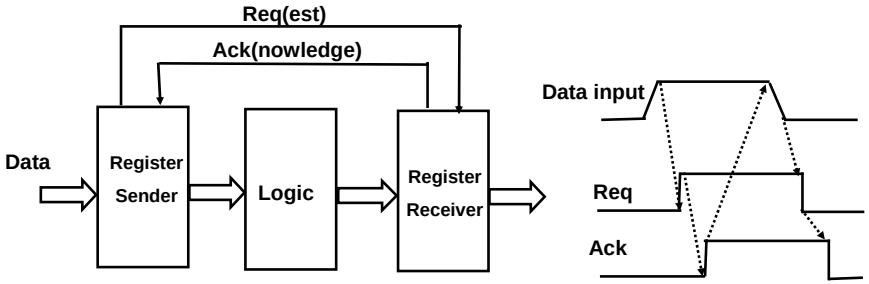
Synchronous (globally clocked) design



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

The Problem of System Timing

Asynchronous (unclocked) design



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

Causal relationship. Handshake signals implemented with completion detection in data path

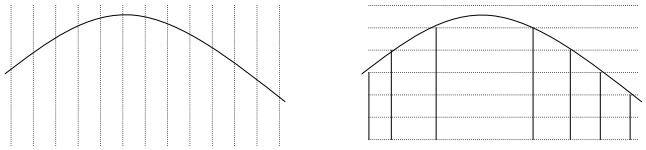
The Problem of System Timing

- Globally Clocked Design reigns in Industry. It is conceptually simple and many D&T tools support it.
- It however faces problems: clock distribution, timing closure, excessive power consumption, EMI, … and it is NOT natural to many applications (perhaps it is just good for pure computations) – timing relations are not causal!
- Asynchronous Design has a lot of design methods developed (some at Newcastle), and some use in Industry (e.g. Philips) but it is not well supported by tools and the majority of design community … they just don't see where it is natural

The Problem of System Timing

- Moreover, the common understanding of Asynchrony is quite limited to the idea of handshakes, which actually makes systems look as Synchronous! … because handshakes act as interlocks.
- This leads to systems that are not running freely according to their “natural motive powers”
- We need truly Asynchronous, Self-timed Systems!

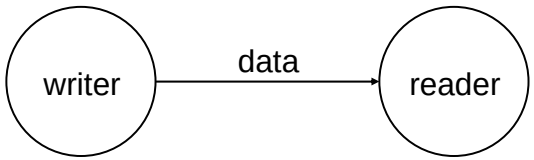
Example – A to D conversion based on level-crossing rather than on periodic sampling



- Asynchronous processing.
- Improved EMI - dependent on data being processed.
- Lower power - energy only used when work is done.

Data communication between processes

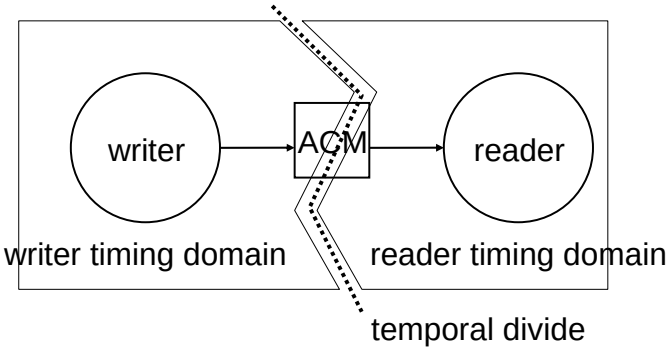
- Point to point: connecting two processes



Data is a stream of items of a set type.
Writer and reader are cyclic processes.
Writer provides one item of data per cycle.
Reader uses one item per cycle.

ACMs

- An ACM is implemented with shared memory

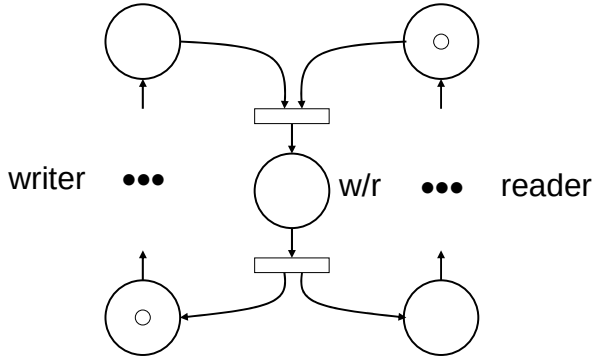


ACMs

- An ACM can specify the following protocols
  - Writer may be required to wait for reader
  - Writer may not be required to wait for reader
  - Reader may be required to wait for writer
  - Reader may not be required to wait for writer
- Qualitative asynchrony specifications naturally divides all ACMs into four types

Data communication

- Traditional approach



Data communication

- Traditional approach is synchronized
  - Either reader or writer must wait for the other side during the transfer of **every** data item
  - Not ideal for many concurrent systems (esp. embedded, real-time, and low power systems)

Asynchrony in concurrent systems

- Inevitable
  - High degrees of integration mean that distribution of global clocks becomes impractical, so even on the same chip there will be asynchrony
  - Distributed systems naturally have local clocks. Synchronizing such clocks can be problematic.
  - Self-timed systems (esp. useful for low power) have no regularly pulsing clocks.

Asynchrony in concurrent systems

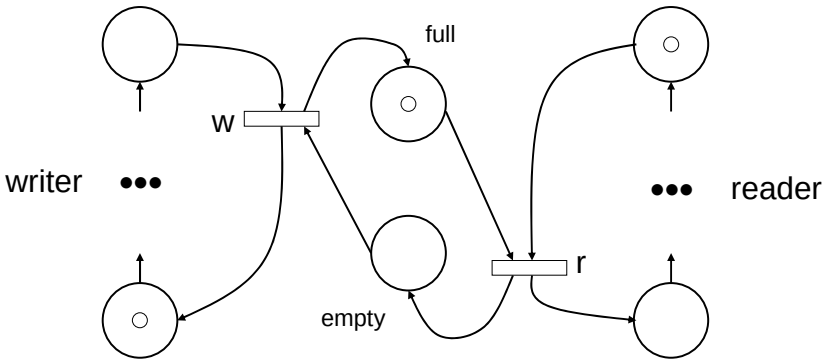
- Real-time elements
  - Most obvious characteristic is timing predictability
  - Such processes thus should not be delayed by outside influences, e.g. data communications with another process
  - Data communications should therefore follow “not obliged to wait for data” protocols

Asynchrony in concurrent systems

- Low power elements
  - In such things as battery powered remote sensors, timing for data items can depend on two factors
  - Some sensors only produce data when they detect changes in the input signal
  - Other sensors may produce data when their communication partners request data from them
  - Each requires a different approach to data communication synchrony

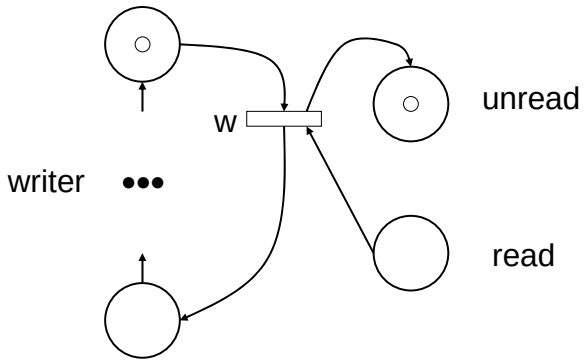
Asynchronous data communication

- A buffer increases scope for asynchrony



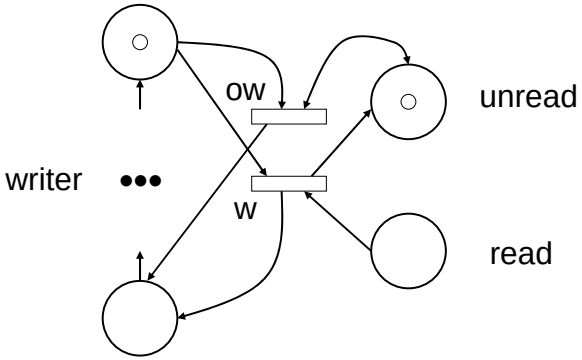
ACMs

- Writer may be required to wait



ACMs

- Writer may not be required to wait

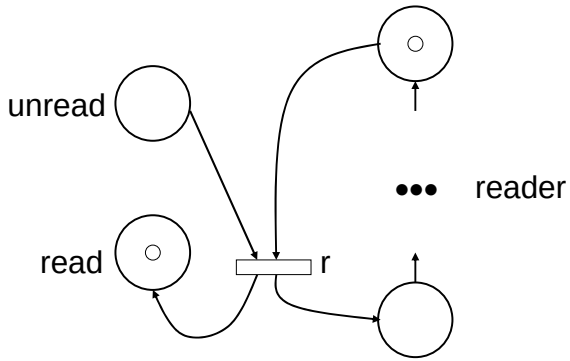


ACMs

- May be required to wait on both sides
  - Traditional buffer in computers
  - Full data continuity (no overwriting or re-reading)
  - Message data
  - Increasing the size of the buffer increases quantitative asynchrony, at the expense of latency (true for all ACMs)

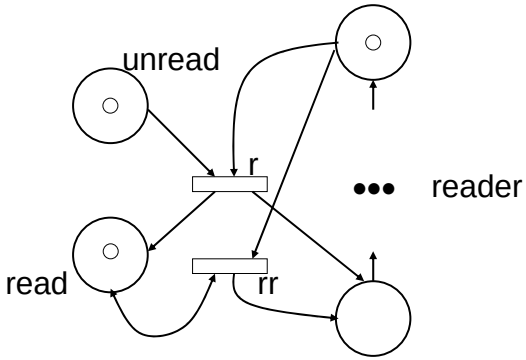
ACMs

- Reader may be required to wait



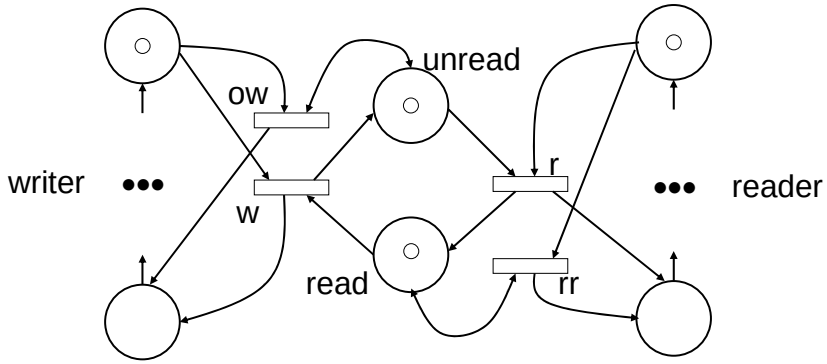
ACMs

- Reader may not be required to wait



ACMs

- No wait on either side



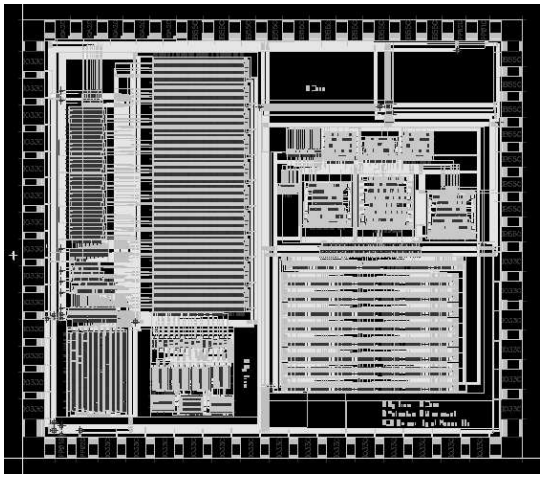
ACMs

- No wait on either side
  - Connecting together two independently timed processes (e.g. real-time processes)
  - Reference data
  - Clock example

ACMs

- May be required to wait on one side, either reading or writing
  - Connecting self-motivated timing with reactive timing
  - Signal or command data
  - One side real-time the other low power, etc.

An asynchronous demonstrator chip designed at Newcastle



Contains:

- •ACM
- •Priority arbiter
- •Async A/D converter

These and other components can be used to design complex self-timed neural networks

How to communicate/interact in brain-like machines?

- Dynamic connectivity seems to be the biggest problem, does it?
- Is it feasible to multiplex 10^{15} of synapses on a limited interconnect fabric of a chip or wafer?
- What mechanisms are used for timing in the brain? Is it cell-based, layer-based, sensor-tiggered, … - where are “motive powers”?
- What mechanisms are used for tolerating data errors and losses in the brain?

Building a brain-like machine (smart widget)

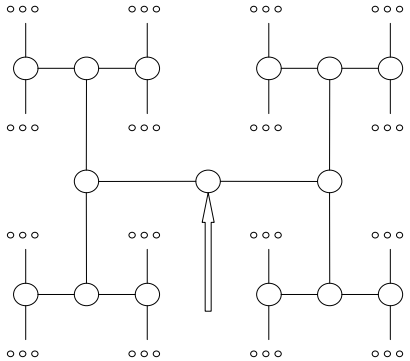
- Produce a detailed functional/algorithmic description – neuro-info-scientist task
- Map it into a silicon (or some emerging technology) space – system designer task

Finding a good mapping is a key!

Finding a good mapping

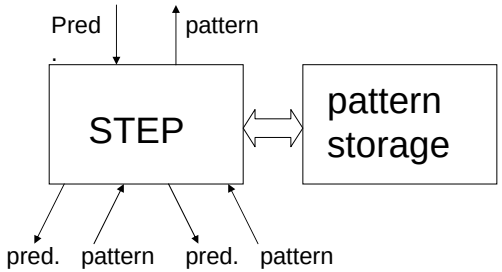
- Refine the functional description:
  - Find a good set of representations for data (most efficient for temporal/spatial integration)
  - Find a good implementation architecture;
    - Interconnection (static and programmable) structure
    - Timing mechanisms
- Produce specifications for all functional blocks and map to (say) library cells in the given technology

H tree (good for 2D layouts)



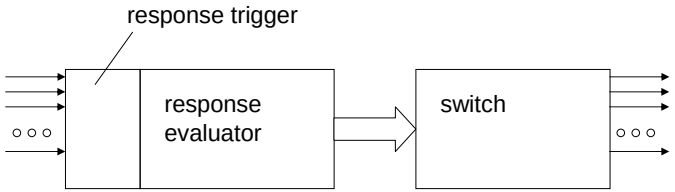
Can be combined with local routes and shortcuts between branches and custers

Asynchronous counterflow pipeline
(with data loss and repetition)



STEP: self-timed event processor

STEP: self-timed event processor



priority arbiter + switch