Presenting:

Some Synchronization Issues When Designing Embedded Systems from Components

by

Albert Benveniste

The Bay Team

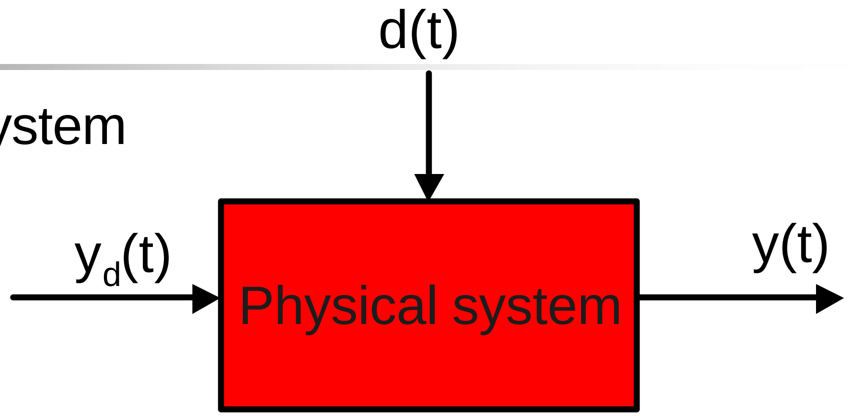
Yanmei Li, Alessandro Pinto, Bruno Sinopoli

Focus

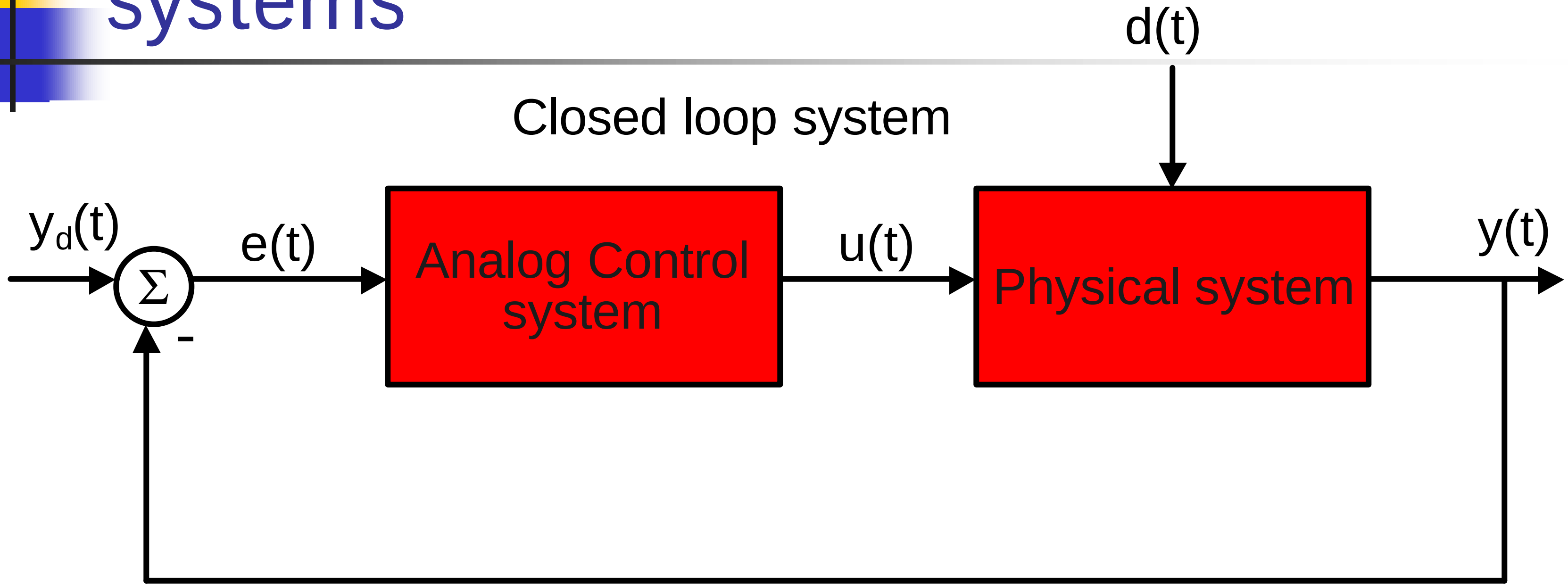
- *This paper looks at issues of synchrony, asynchrony and synchronization that arise in the design of embedded systems*
- Three areas of interest:
  - Hybrid control systems
  - Synchronous hardware design from IP’s
  - Building software or hardware architectures composed of components that interact asynchronously

The issue in continuous time control systems

Open loop system



The issue in continuous time control systems



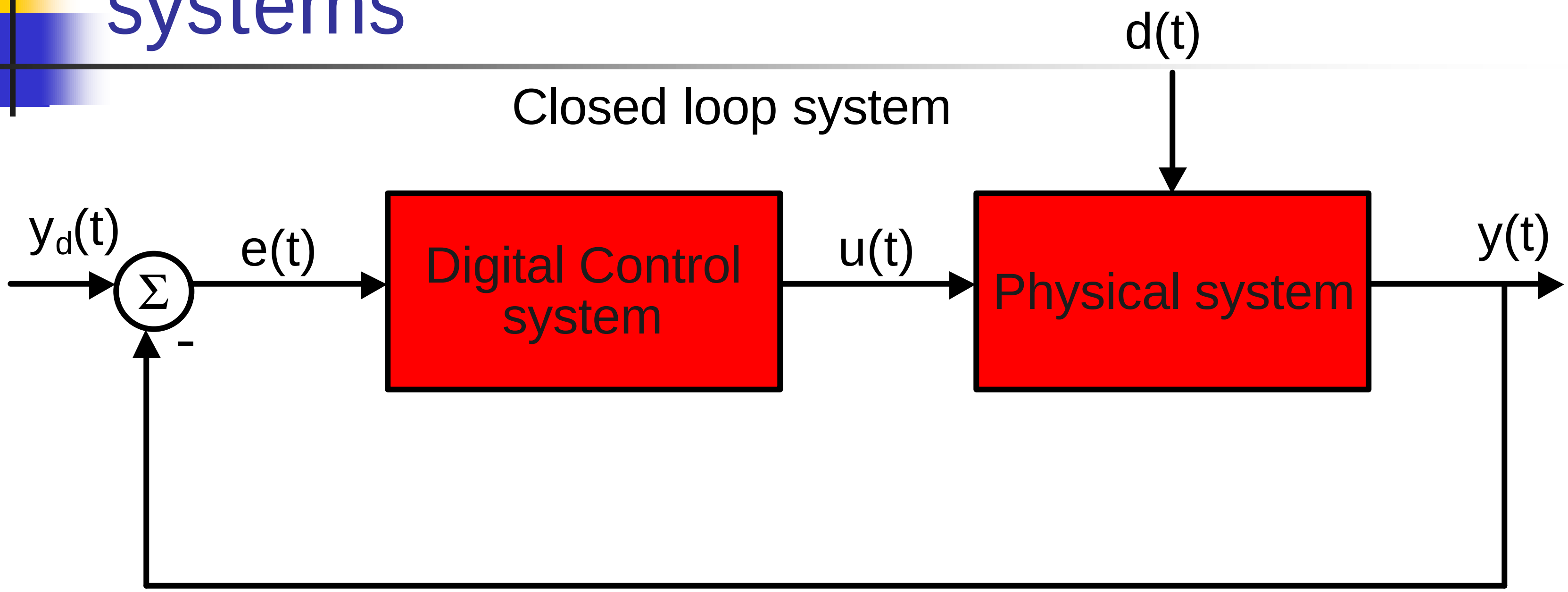
Issues:

- Approximation in space: modeling errors, noisy measurements, unknown disturbance
- Approximation in time: delay from sensing to actuating

Gain, phase margins are design metrics provide robustness with respect to those approximations

The issue in continuous time control systems

Closed loop system


d(t)

$y_d(t)$

Σ

-

e(t)

Digital Control system

u(t)

Physical system

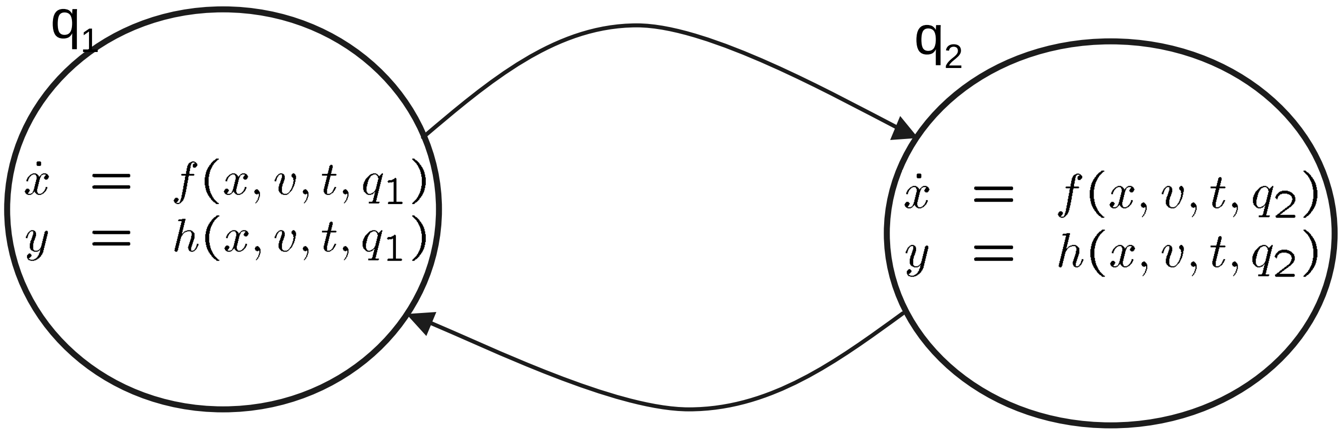
y(t)

Digital control systems makes approximation in time worse due to sampling, A/D, D/A conversions

Bounds on maximum sampling will preserve performance, making the synchronous model still valid

What about hybrid systems?

$$\begin{aligned}
\dot{x} &= f(x, v, t, q) \\
y &= h(x, v, t, q) \\
\alpha &= a(x, u) \\
\alpha &\neq \alpha_{-} \Rightarrow T(q, \alpha, q')
\end{aligned}$$



Time approximation errors can determine quite different behaviors

Why this difference?

- Continuity ensures robustness to time jitters

- A hybrid system, by introducing discreteness, i.e. discontinuity, is inherently susceptible to lack of synchrony in its components. Small errors in state estimation can trigger an undesired change of discrete mode, making behavior highly unpredictable

Synchronous Hardware Design from IP's

- Retiming
- Transformations
- Latency Insensitive Design
- Models
- Basic idea and two problems

Retiming

- Use data-flow graph G to model synchronous hardware:
  - **vertices** figure variables
  - **branches** figure dependencies
    - $u \rightarrow v$: v_k uses u_k for its computation
    - $u \xrightarrow{n} v$: v_k uses u_{k-n} for its computation
    - index n : **weight** of the branch
  - **weight of a path** in the graph is the sum of the weights of its successive branches
- We consider only *well formed* graphs(every circuit has a strictly *positive* weight)

Transformations

- **GOAL:** To find modifications of G that will not change its semantics
- **Two primitive transformations:**
    - *Moving latches around*
    - *Upsampling*

Transformations-1

Moving Latches Around:

- $u_i \xrightarrow{n_i} v \Rightarrow u_i \xrightarrow{n_i - m} v$
- $v \xrightarrow{n_j} w_j \Rightarrow v \xrightarrow{n_j + m} w_j$
  - pick *m* latches from each ingoing branch of *v*, and move them to each outgoing branch of it
  - this transformation does not change the map: $(u_i\,,\ i = 1,\ldots,p) \rightarrow (w_j\,,\ j = 1,\ldots,q)$
  - $\forall i=1,\ldots,p$: $n_i - m \geq 0$ and $\forall j=1,\ldots,q$: $n_j + m \geq 0$
- It is a semantic preserving transformation

Transformations-2

Upsampling:

- Pick an integer $J > 1$
- Set $v_m' :=$ if m = kJ then v_k else t
  - t means *non_informative,* not belonging to any useful domain,it represents a don’t care
- Ignoring t in v' generates v
- If we perform this globally using the same integer J at all vertices, we preserve the semantics

Latency Insensitive Design

- At early stages of the design,both IP’s and the system can be regarded as completely synchronous, i.e., just as a set of modules that communicate by means of channels having “zero-delay”
- At later stages of the design where real clocks are used, it adjusts automatically to any interconnect-delay,on-line

Models-1:strictly synchronous

- A ***state x*** assigns an effective value to each variable $v \in V$
- A ***strictly synchronous behavior*** is a sequence $s = x1, x2, \ldots$ of states
- A ***strictly synchronous process*** is a set of strictly synchronous behaviors
- A ***strictly synchronous signal*** is the sequence of values $s_v = v(x1), v(x2), \ldots$, for $v \in V$ given

Models-2: synchronous

- This model is the same as in the previous case, but every domain of data is enlarged with some *non-informative* value(***t***)
- A ***state x*** assigns an informative or non-informative value to each variable $v \in V$
- A ***synchronous behavior*** is a sequence of states
- A ***synchronous process*** is a set of synchronous behaviors
- A ***synchronous signal*** is the sequence of informative or non-informative values $s_v = v(x1), v(x2), \ldots$, for $v \in V$ given

Basic Idea

- We wish to implement a strictly synchronous specification P by means of a synchronous process $P^{/}$, insensitive to latency. Then $P^{/}$ replaces P and will be used as an IP block

Problem-1

How to model that a synchronous process P^I implements a strictly synchronous specification P, while being insensitive to latency?

- Values of variables travel on wires of the design,and this causes latency.Such latency may differ for different variables (since different wires are used)

Problem-1 Solution

- For $v \in V$, pick some signal

  $s_v = v(x1), v(x2), v(x3), \ldots \in P$
- To reflect a wire-dependent latency, the same signal, observed later on along a wire, has (for example) the form

  $s_v^l = \boldsymbol{t}\ , v(x1), \boldsymbol{t}\ , v(x2), \boldsymbol{t}\ , \boldsymbol{t}\ , v(x3), \ldots$
- $\boldsymbol{t}$ can be inserted at arbitrary places of the original signal s_v . This is the mechanism of ***stalling*** a signal
- Map $X_v : s_v^l \longrightarrow s_v$ giving the strict version of a stalled signal

Problem-1 Solution(cont.)

- ***Patient process*** *P* :

  For all $s \in P$, all input signal s_i of s, and all instant k, there exists another behavior *stall* $(s) \in P$, whose *i*-signal coincides with s_i before instant k, has a t-event at k, and can be further stalled after k

- ***Buffer:***
  - A ***single buffer*** is any process which has two variables v_i, v_o, and has the identity process $s_{vo} := s_{vi}$ as corresponding strict process
  - A ***buffer*** is the parallel composition of finite single buffers involving disjoint sets of variables

Problem-1 Solution(cont.)

- **Theorem.** *If* P^l *and* Q^l *are* ***patient*** *processes,and* B,B' *are two buffers,then*

  $$X_V(P^l \,||\, B \,||\, Q^l) = X_V(P^l \,||\, B' \,||\, Q^l) = P \,||\, Q$$

  - P^l and Q^l are two processes having disjoint sets of variables, communicating through a buffer
  - P,Q are the strict processes corresponding to P^l and Q^l
  - $X_V(P^l)$ represents the strict process corresponding to P^l
- Implies that inserting a buffer does not change the corresponding strict process

Problem-2 and Solution

For a strict process P , how to build a patient process P^{l} such that $X_V\,(P^{l}) = P$?

- Enlarge *G* with additional branches:
  (environment) → *u*
  - where $u \in V^{\,i}$ (input variables)
- m_v : the weight of each variable *v* of *G*
- Moving of latches is encoded by the set of weights m_V

Problem-2 Solution(cont.)

- $\forall v \in V$,set initial value for m_V

  $m_v := 0$

- The original data structure to model the circuit is (G, 0)
- (G, m_v) is updated on-line at each reaction according to a update protocol

Problem-2 Solution(cont.)

Update Protocol:

- Case 1(trivial): All inputs of *G* receive informative values for the first round. Then the reaction proceeds as specified by *G* directly,and the circuit waits for a second set of input values
- Case 2: At least one input wire offers a non-informative value t for the first reaction

Problem-2 Solution(cont.)

- Case 2: Assume non-informative value t occurs exactly for one single $u \in V$ i
  - model the reception of a noninformative value on input wire u via the insertion of a **negative** delay in the corresponding input branch of G :

    update G: $[\xrightarrow{0} u\ ] \Rightarrow [\xrightarrow{-1} u\ ]$
  - $(u\rightarrow)$ represents the set of the variables $v \in V$, there exists a path from u to v having zero weight. update the set of weights m_V :

    $\forall\ v \in (u\rightarrow) : \quad m_v := m_v - 1$

    $\forall\ v \notin (u\rightarrow) : \quad m_v := m_v$

Problem-2 Solution(cont.)

- Use retiming rules for $m_v=-1$ at $v \in (u\rightarrow)$:
  - assuming the availability of one latch at the output wires belonging to $(u\rightarrow)$
  - moving these latches backward until a variable not belonging to $(u\rightarrow)$ is reached
- Compensate the negative delay in front of *u* by a positive one, therefore making the whole synchronization correct

Problem-2 Solution(cont.)

- Generalized protocol:
  - update G :

    $s_u(x) = t$: $[\xrightarrow{n} u\ ] => [\xrightarrow{n-1} u\ ]$
  - update m_V :

    $\forall v \in (U_t \longrightarrow)$: $m_v := m_v - 1$

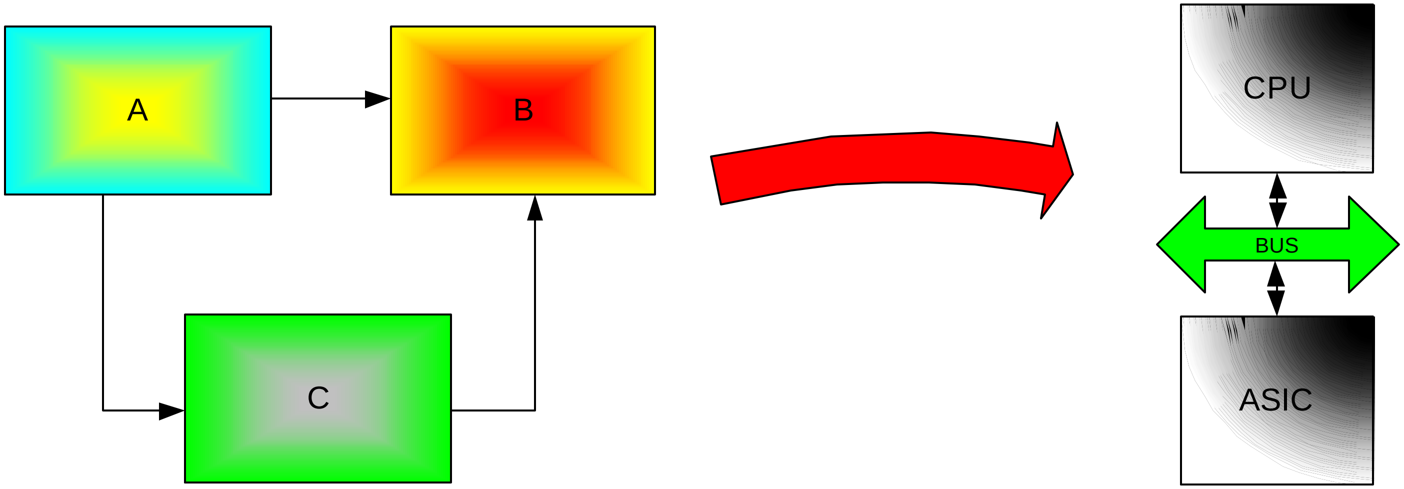
    $\forall v \notin (U_t \longrightarrow)$: $m_v := m_v$

    where $(U_t \longrightarrow) = \bigcup_{u:Su(x)=t} u \longrightarrow$

Latency Insensitive Design

- Take a design based on the assumption that computation in one functional block and communication among blocks take no time (synchronous hypothesis)
  - i.e. the processes corresponding to the functional blocks and their composition are strict
- Replace it with a design where communication does take time and, as a result, signals are delayed, but not changing the sequence of informative events
  - i.e. replace with a set of patient processes

From Synch to GALS



Synchronous Specification

GALS Architecture

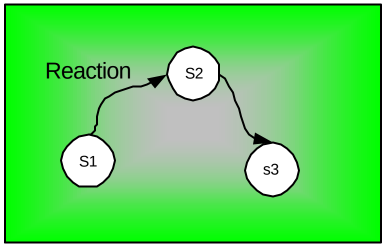
Synchronous Model

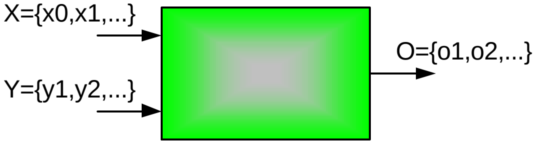
- Processes = Sequence of reactions (R is a set of possible reactions)

$$P = R^{?}$$

- Parallel composition = Pairwise conjunction of reactions (whenever composable)

$$P_1 \parallel P_2 = (R_1 \wedge R_2)^{w}$$





Asynchronous Model

- Signals are Totally Ordered sequences of *informative* events
- Behaviors are *tuple of signals*
- Processes are *set of behaviors*
- Composition is obtained by unifying common signals

$$P_1^a \parallel_a P_2^a = P_1^a \cap P_2^a$$

- The communication is modeled by unbounded FIFOs

Synch vs Asynch

- In Synchronous models
  - “Reaction based ”
  - Absence (⊥) can be sensed and used in the specification of behaviors
  - A global tick exists

- In Asynchronous models
  - “Signal based ”
  - No global tick
  - Reaction cannot be observed anymore
  - ⊥ cannot be sensed

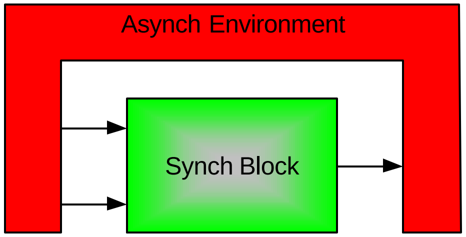
What are the Problems?

- "What if a synchronous block receives its data form an asynchronous environment ? "

- "What if we deploy a synchronous network of synchronous blocks onto a GALS architecture? "

Synch block in Asynch environment

- Input to the synchronous block are not "correct"
  - The Environment provides sequence of totally ordered informative events
  - The Process can sense absence and use it within a state



Desynchronization

- V set of state variables of P
- A state is a valuation of all $v \in V$ ($\perp$ included)
- $s = x_0, x_1, x_2 \ldots$ behavior pf P (sequence of states)
- $v(x)$ valuation of variable v at state x
- $s = (v(x_0))_{v \in V}, (v(x_1))_{v \in V}, (v(x_1))_{v \in V} \ldots = (s_v)_{v \in V}$

- By removing all $\perp$ from s_v we obtain s^a

Endochronicity

- $s \mapsto s^a$ define $P \mapsto P^a$ desynchronization of P
  - This map is unique but not invertible

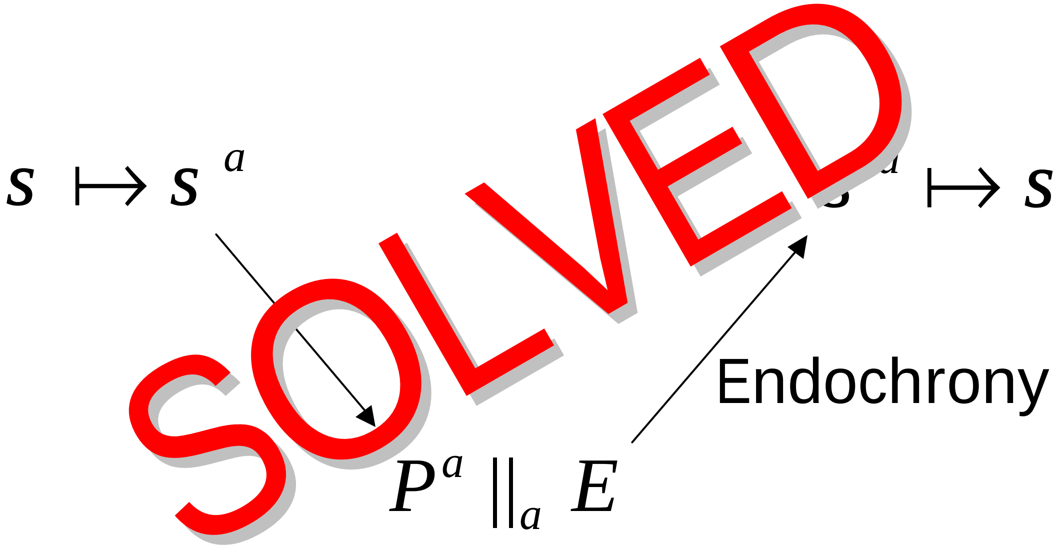
'If P satisfies a special condition called endochrony, then $\forall s^a \in P^a$ there exists a unique $s \in P$ such that $s \mapsto s^a$ holds "

Endochronicity: Properties

- Can be done on-line
- Can be model checked
- Given P, a wrapper W can be found such that P||W is endochronous

Solution to Problem 1

- “What if a synchronous block receives its data form an asynchronous environment ? ”



$$s \mapsto s^a$$

$$s^a \mapsto s$$

$$P^a \parallel_a E$$

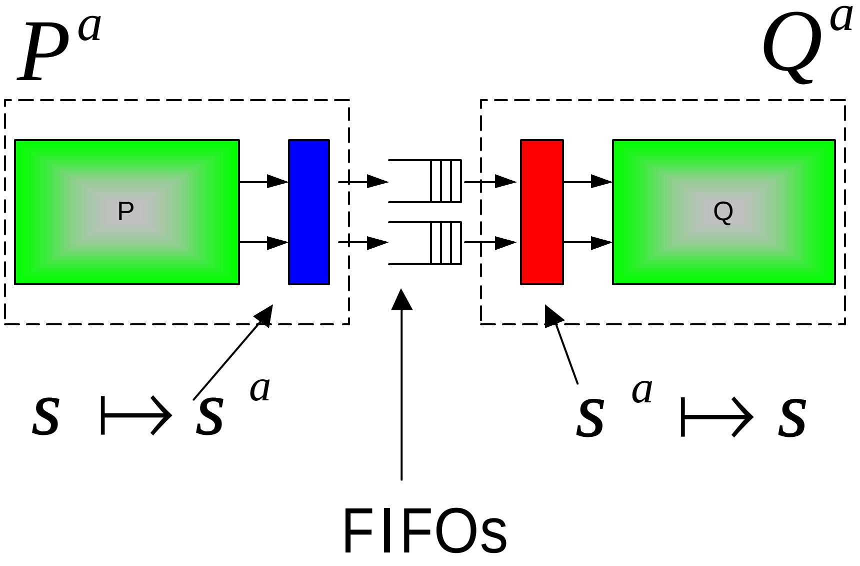
Endochrony

SOLVED

Network of blocks

- We use the desynchronized version of P, Q

$$P^a \|_a Q_a$$



Isochronicity

- In general $(P \parallel Q)^a \subseteq (P^a \parallel_a Q^a)$

- WE want the equality to hold (no spurious behavior due to asynchronous communication)

‘If (P,Q) satisfies a special condition called isochrony then the equality indeed holds ”

Isochronicity: Properties

- It is compositional
- It can be model checked
- For any pair (P,Q) there exists (Wp,Wq) making (P||Wp,Q||Wq) an isochronous pair

Isochronicity: Intuition

- For composition we use $(R_1 \wedge R_2)$
  - In particular common variables are both present or absent
- Weakly Synchronicity
  - A given variable can be present in one component and absent in the other

$$(R_1 \wedge_a R_2)$$

- Isonchronous pair (P,Q):

$$(R_1 \wedge R_2) = (R_1 \wedge_a R_2)$$

Methodology

- Synchronous Network

$$P_1 \| P_2 \| P_3 \| \ldots P_k$$

  - A-Each P_k is endochronous
  - B-$\{P_1, P_2, P_3 \ldots P_k\}$ form an isochronous network

Methodology (cont’d)

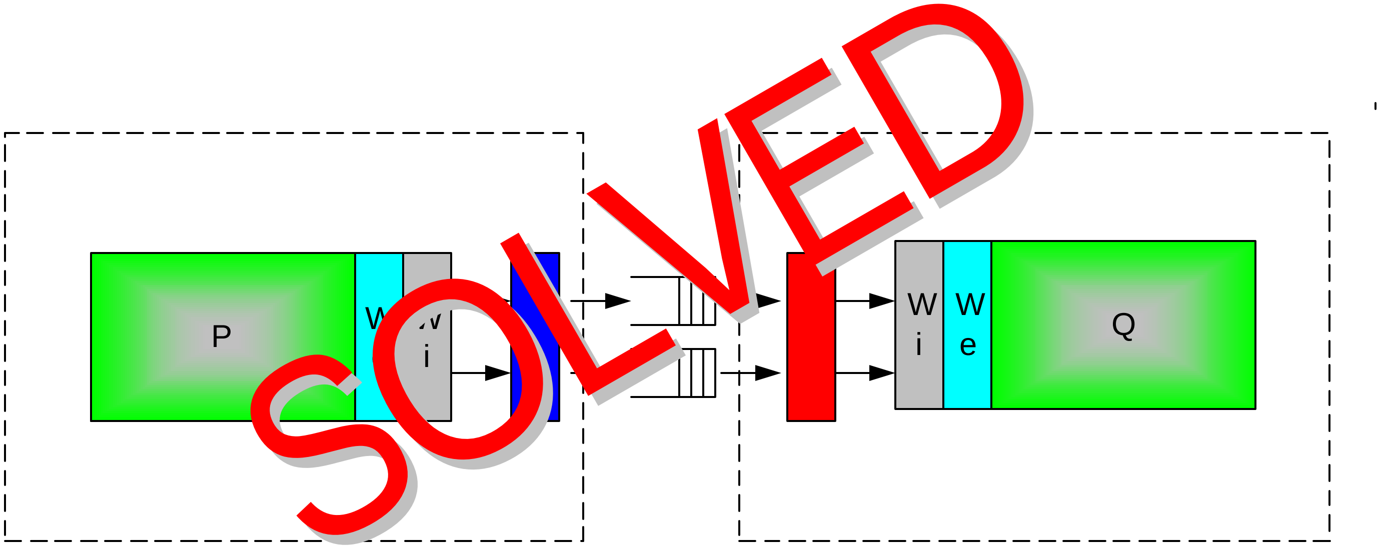
- Isochronicity guarantees that

$$P_1^a \|_a P_2^a \|_a \ldots P_k^a = (P_1 \| P_2 \| \ldots P_k)^a$$

- Endochronicity guarantees that there exists $s^a \mapsto s$
- For each block the original synchronous semantics is preserved

Solution to Problem 2

- "What if we deploy a synchronous network of synchronous blocks onto a GALS architecture? "



Conclusion

- General concern: build a correct by construction methodology for modular architecture
- Similarities
  - Stallable processes ≈ stuttering invariant
  - Equalizer ≈ $s \overset{a}{\longmapsto} s$
- Differences
  - No global clock != global clock
  - Single clock != Milticlock (at the spec. level!)