

Modeling and Validating Globally Asynchronous Design in Synchronous Frameworks

M. Mousavi¹, P. Le Guernic², J.P. Talpin², S.K. Shukla³, T. Basten¹

¹Eindhoven University of Technology, ²INRIA/IRISA Rennes,

³Virginia Tech.

DATE Conference, February 2004,
Paris, France

Overview

→ [Introduction](#)

2. Defining Synchrony and Asynchrony
3. Asynchrony Using FIFOs
4. Bounding the Fifo Size
6. Conclusion and Future Perspective

1. Introduction: Motivation

We cannot afford **global synchrony**, because of:

1. **clock drift** across the chip surface (small scales);
2. synchronizing problem across **distributed** architectures (large scales);
3. new **platforms** (e.g., NoC) forcing towards global asynchrony.

1. Introduction: Motivation

But, **local synchrony** is still interesting, due to :

1. **simpler** design process;
2. more **tools** and **experience**;
3. possibility of rigorous **testing** and **verification**.

I. Introduction: Motivation

We cannot afford **global synchrony** and **local synchrony** is still interesting.

Polychrony:
A framework for **synchronous** design with **different clock rates**.

1. Introduction: Motivation

Benefits of using Polychrony:

1. **Formal** underlying theory;
2. simple and efficient **semantics**;
3. existence of mature design and verification **tools**.

I. Introduction: Motivation

Our proposal:

Modeling and Validating **GALS** Designs in **Polychrony**

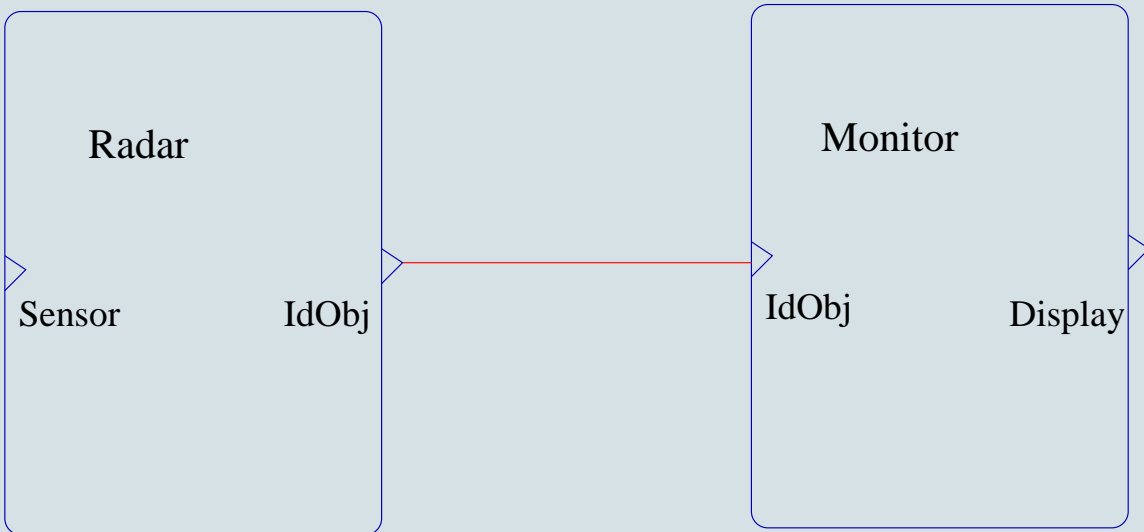
Thus:

1. Breaking the **synchronous barrier**;
2. but still benefiting from synchronous **simplicity** and **tools**.

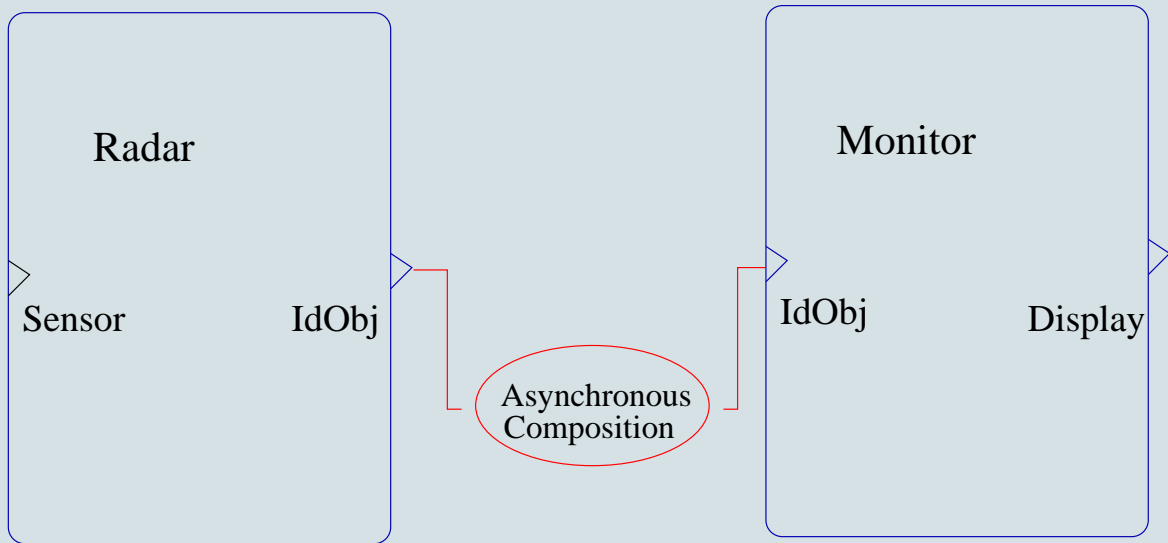
Challenge:

There is no truly **asynchronous composition** in Polychrony.

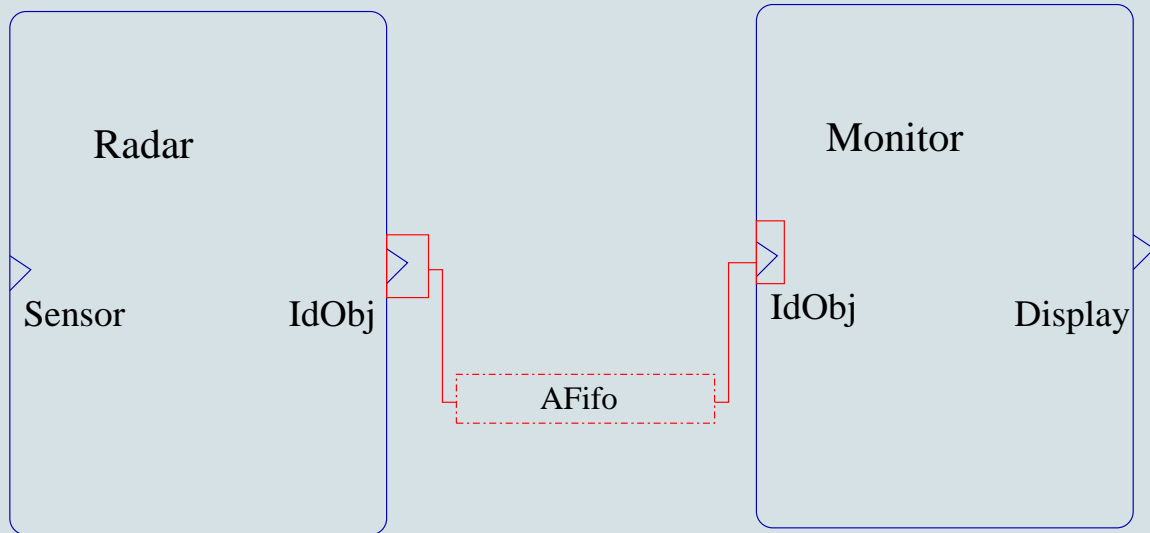
I. Introduction: Overview of the Process



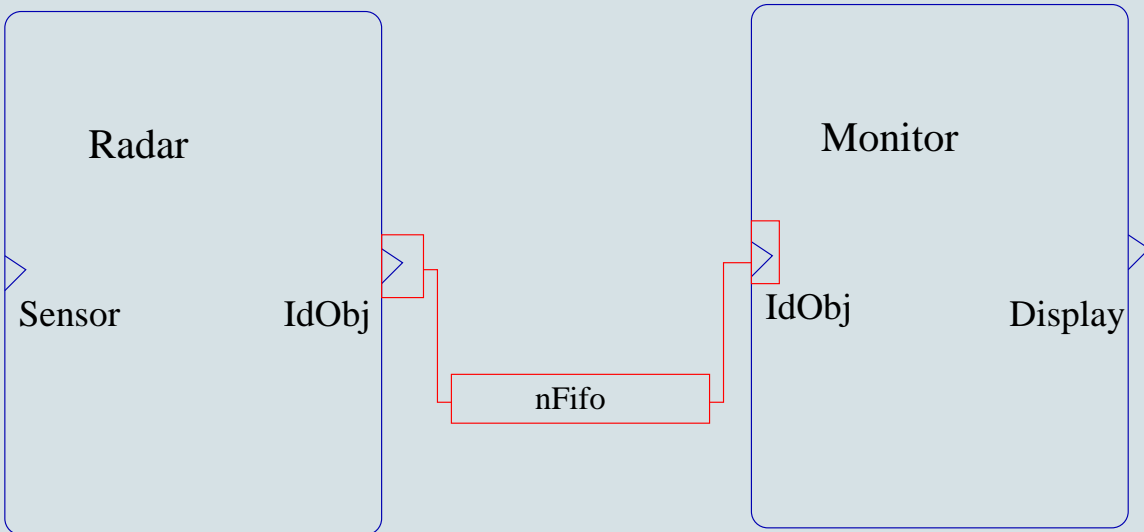
I. Introduction: Overview of the Process



I. Introduction: Overview of the Process



I. Introduction: Overview of the Process



Overview

✓ Introduction

→ Defining Synchrony and Asynchrony

3. Asynchrony Using FIFOs

4. Bounding the Fifo Size

5. Conclusion and Future Perspective

2. Defining Synchrony and Asynchrony

A plain synchronous behavior:

$y :$	t	ff	ff	...
$z :$	ff	t	t	...
$f(y, z) :$	t	t	t	...

2. Defining Synchrony and Asynchrony

A polychronous behavior:

$T :$	t_1	t_2	t_3	t_4	t_5	t_6
$y :$	tt	ff		ff		...
$z :$		ff	tt			...
$f(y, z) :$	tt	tt				...

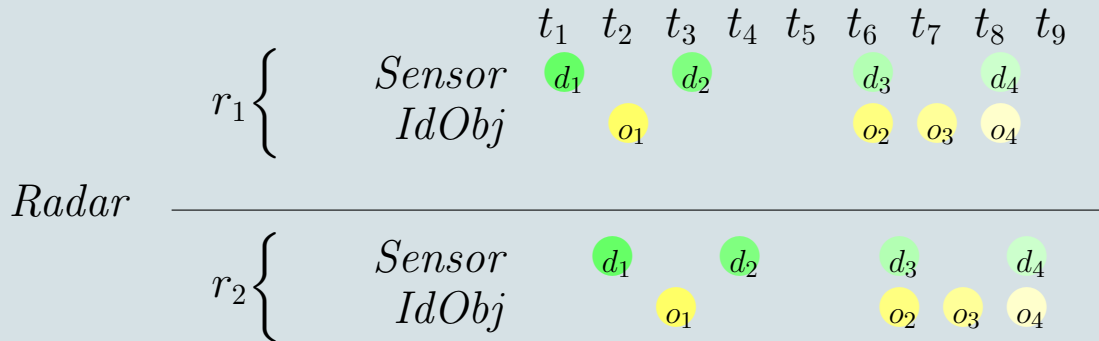
N.B., tags are:

1. partial order
2. representation of causality and synchronization.

They do not represent real time.

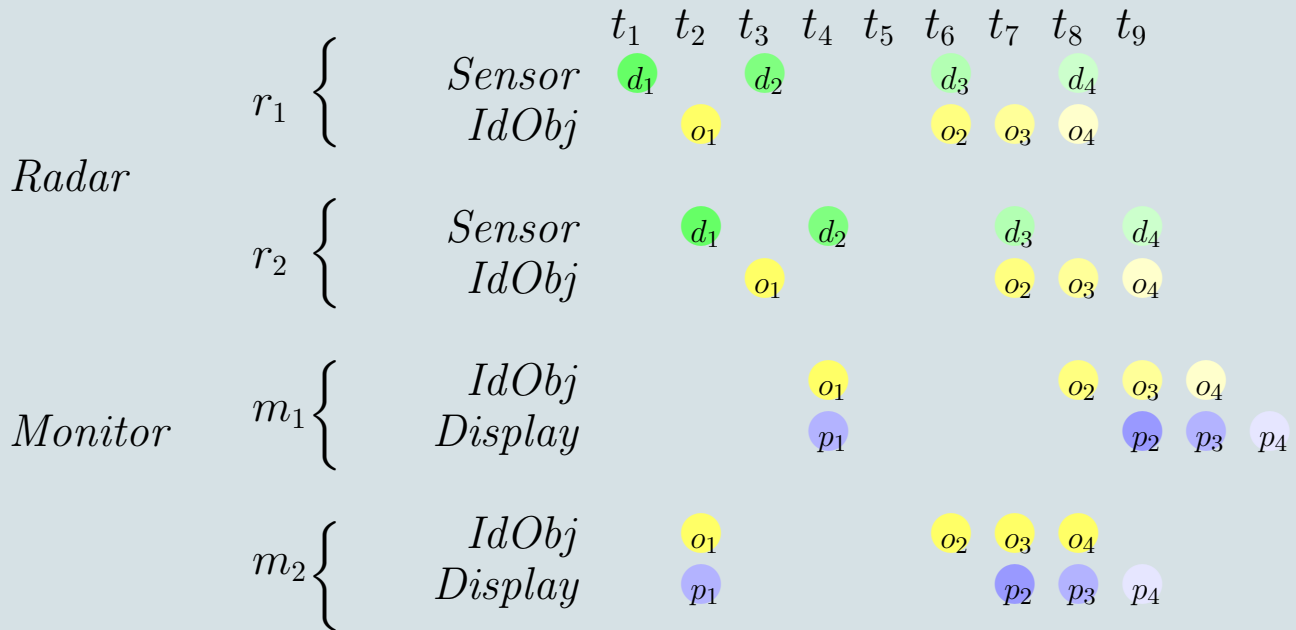
2. Defining Synchrony and Asynchrony

A polychronous process:



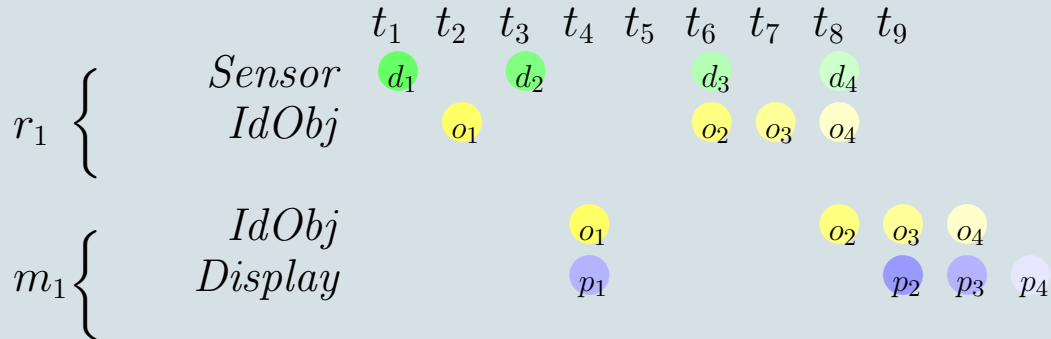
2. Defining Synchrony and Asynchrony

A polychronous process:



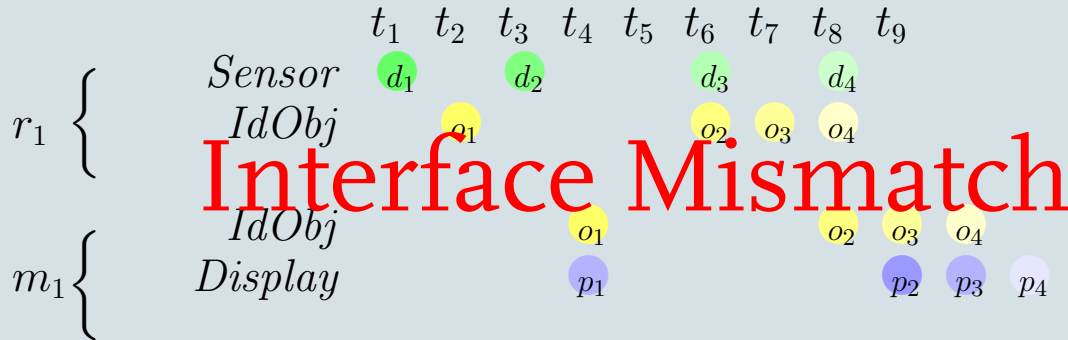
2. Defining Synchrony and Asynchrony

Synchronous composition of behaviors ($r_1 \parallel_s m_1$):



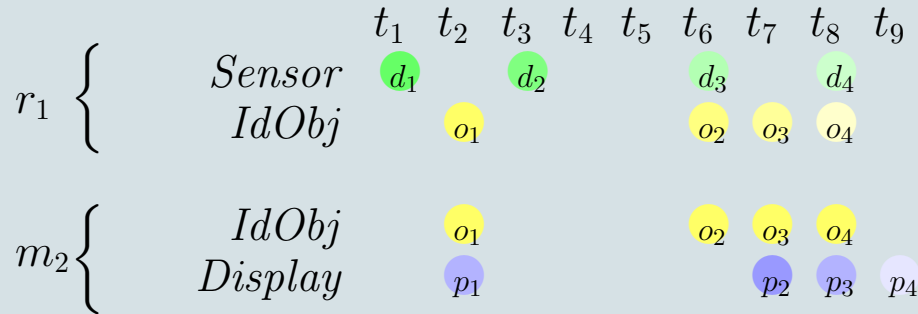
2. Defining Synchrony and Asynchrony

Synchronous composition of behaviors $(r_1 \parallel_s m_1)$:



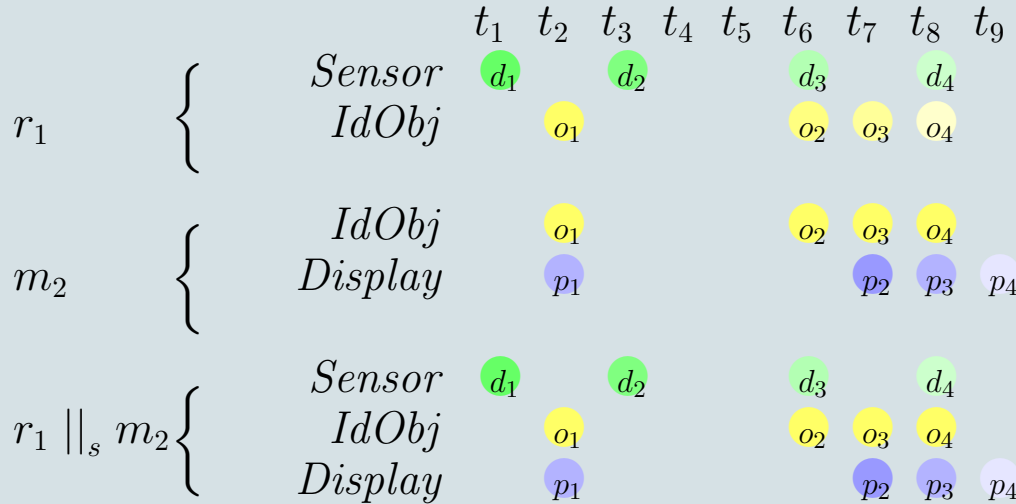
2. Defining Synchrony and Asynchrony

Synchronous composition of behaviors ($r_1 \parallel_s m_2$):



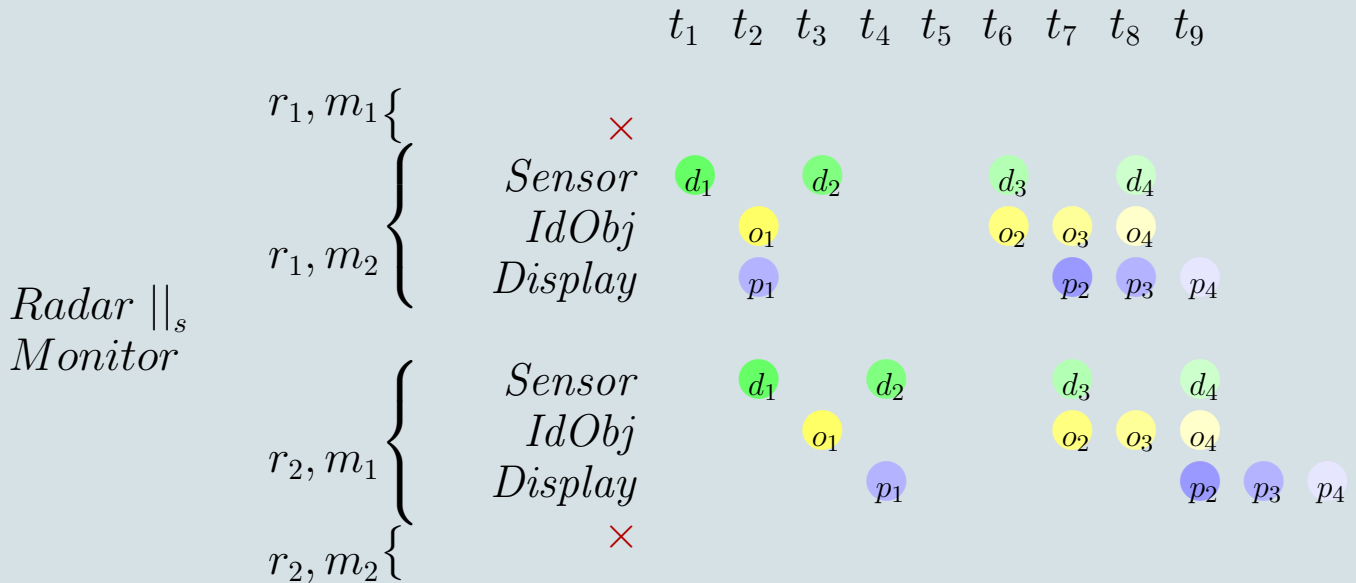
2. Defining Synchrony and Asynchrony

Synchronous composition of behaviors ($r_1 \parallel_s m_2$):



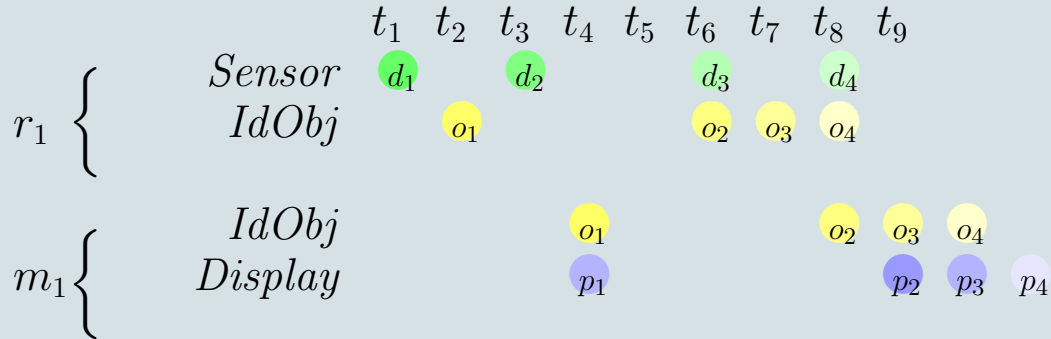
2. Defining Synchrony and Asynchrony

Synchronous composition of behaviors ($Radar \parallel_s Display$):



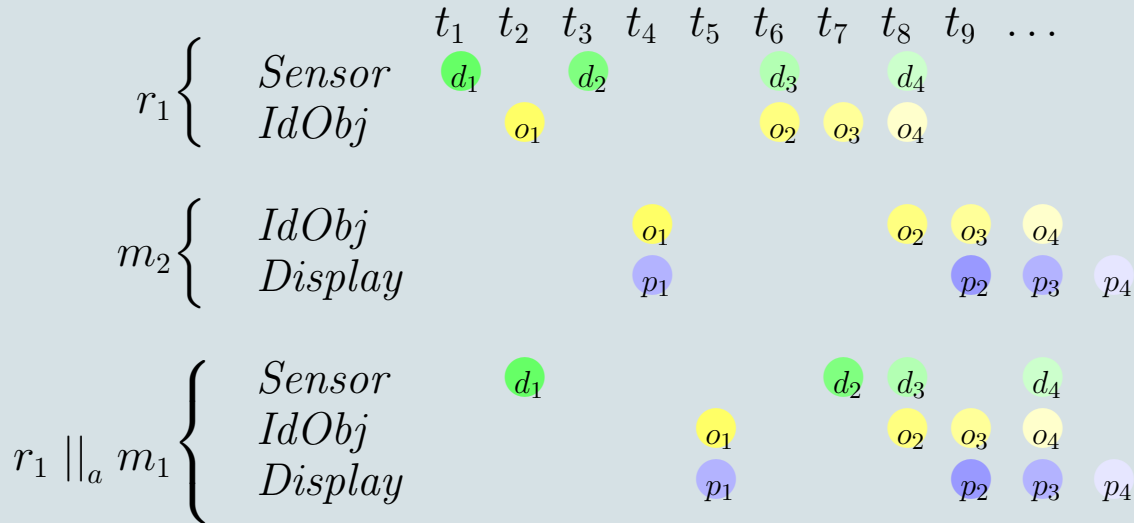
2. Defining Synchrony and Asynchrony

Asynchronous composition of behaviors $(r_1 \parallel_a m_1)$:



2. Defining Synchrony and Asynchrony

Asynchronous composition of behaviors ($r_1 \parallel_a m_1$):



Overview

- ✓ Introduction
- ✓ Defining Synchrony and Asynchrony
- Asynchrony Using FIFOs
- 4. Bounding the FIFO Size
- 5. Conclusion and Future Perspective

3. Asynchrony Using FIFOs

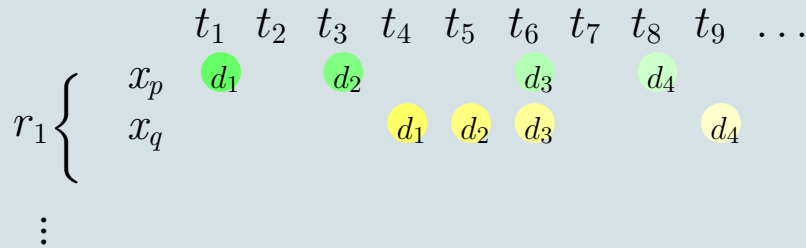
Asynchronizing process:

1. replacing all **common signal** names with **fresh and unique** ones;
2. **inserting asynchronous FIFO channels** connecting the common signals (using their new names);
3. **synchronous composition** of components and FIFO channels.

3. Asynchrony Using FIFOs

Asynchronizing process

Asynchronous FIFO $x_p \rightarrow x_q$:



Containing all behaviors in which x_q is a stretching of x_p

3. Asynchrony Using FIFOs

Theorem:

The system resulting of **asynchronizing** process has the same set of **behaviors** as **asynchronous** composition of components.

3. Asynchrony Using FIFOs

Theorem:

The system resulting of **asynchronizing** process has the same set of **behaviors** as **asynchronous** composition of components.

But, asynchronous FIFO has the same problems as asynchronous composition:

1. not (practically) **implementable**
2. not easy to **reason** about

3. Asynchrony Using FIFOs

Theorem:

The system resulting of **asynchronizing** process has the same set of **behaviors** as **asynchronous** composition of components.

But, asynchronous FIFO has the same problems as asynchronous composition:

1. not (practically) **implementable**
2. not easy to **reason** about

Solution: Replacing **asynchronous** FIFOs with **bounded** ones **when possible**

Overview

- ✓ Introduction
- ✓ Defining Synchrony and Asynchrony
- ✓ Asynchrony Using FIFOs
- **Bounding the FIFO Size**
- 5. Conclusion and Future Perspective

4. Bounding the FIFO size

One-place buffer can be implemented in **SIGNAL** (a polychronous language), as follows:

```
data      = (msgIn when (not full)) default  
            (pre 0 data)  
msgOut = data when (^msgIn default full)  
in       = ^msgIn default ff  
out      = ^msgOut default ff  
full     = ((pre in ff) ^ not(pre out ff)) default  
            (pre full ff)  
^data    = (^msgIn default ^msgOut) default ^full  
^full    = ^in = ^out
```

4. Bounding the FIFO size

A sample behavior of one-pace buffer:

<i>msgIn</i>		1		2			3
<i>in</i>	ff	#	#		ff	ff	#
<i>full</i>	ff	ff	#		#	ff	ff
<i>data</i>	0	1	1		1	1	3
<i>out</i>	ff	ff	ff		#	ff	#
<i>msgOut</i>					1		3

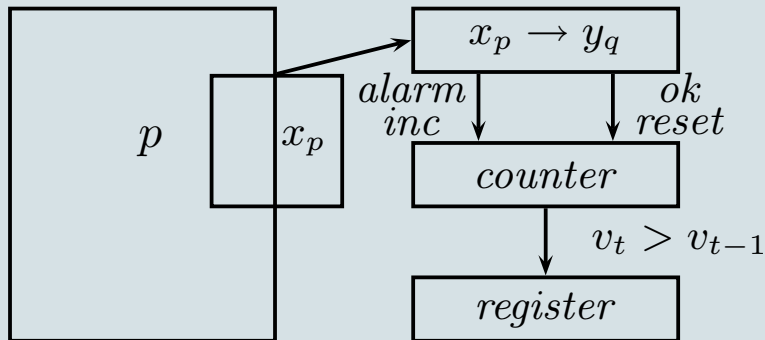
4. Bounding the FIFO size

One-place buffers are composed to get an n-place buffer:

$$\begin{aligned}
 nFifo_{x_0 \rightarrow x_n} &= 1Fifo_{x_0 \rightarrow x_1} [full_1, in_1, out_1 / full, in, out] \\
 &\parallel_s \dots \\
 &\parallel_s 1Fifo_{x_{n-1} \rightarrow x_n} [full_n, in_n, out_n / full, in, out] \\
 in_i &= (in_{i-1} \text{ when } not(full_i)) \text{ default } in_i \\
 out_i &= (out_{i+1} \text{ when } full_i) \text{ default } out_i \quad (0 < i < n) \\
 alarm &= (full_1 \wedge \dots \wedge full_n) \text{ when } x_P \\
 ok &= not(alarm)
 \end{aligned}$$

4. Bounding the FIFO size

The following instrumentation accounts for insufficient buffer size:



4. Bounding the FIFO size

By feeding a set of **common inputs** to the instrumented channel network, one can find **an optimal buffer size** for each channel (w.r.t the given inputs).

For unanticipated inputs, provide a **feedback loop** (handshaking mechanism) or **degrade service level** of the consumer as the buffer gets full (using alarm signals).

Theorem:

If **no alarm** signal is raised for a set of inputs, then the **channel network** implements **asynchronous** behavior.

Overview

- ✓ Introduction
- ✓ Defining Synchrony and Asynchrony
- ✓ Asynchrony Using FIFOs
- ✓ Bounding the FIFO Size in Practice
- Conclusion and Future Perspective

Conclusion and Future Perspective

What we have done:

1. A reference **framework for asynchrony** in **synchronous** languages
2. necessary and sufficient conditions on **semantics** to **implement** asynchronous designs
3. **implementing prototypes** of necessary ingredients in **SIGNAL** polychronous language
4. thus, providing the possibility of **design and verification** of GALS designs inside **Polychrony toolkit**

Conclusion and Future Perspective

What we plan to do:

1. Applying the theory in [practice](#)
2. providing sufficient conditions on [SIGNAL syntax](#) to implement asynchronous designs
3. thus, support for [automatic generation of GALS designs](#) inside Polychrony toolkit

Overview

- ✓ Introduction
- ✓ Defining Synchrony and Asynchrony
- ✓ Asynchrony Using FIFOs
- ✓ Bounding the FIFO Size in Practice
- ✓ Conclusion and Future Perspective

Thank You!