

# Synchronizing Periodic Clocks in Kahn Networks

Albert Cohen<sup>1</sup>, Marc Duranton<sup>2</sup>, Christine Eisenbeis<sup>1</sup>, Claire  
Pagetti<sup>1</sup> and Marc Pouzet<sup>3</sup>

<sup>1</sup>Inria Futurs, Orsay France

<sup>2</sup>Philips Research Laboratories, Eindhoven, The Netherlands

<sup>3</sup>Université Pierre et Marie Curie, Paris, France

March 6th, 2005

# Context

**Domain:** real-time video processing (TV boxes)  
tera-operations per second (on pixel components)  
**Conception:** specific hardware (ASIC)  
**Evolution:** mixing hardware/software because of costs, variability of supported algorithms.

**Domain-specific designs:** general-purpose architectures and compilers are not suitable. Wish: higher compute density and programmability  $\implies$  an appropriate programming language and compiler.

**Synchronous paradigm:** generation of custom hardware and software systems with *correct-by-construction structural properties*, including real-time and resource constraints.

# Multiple Clock Domains

**synchronous hypothesis:** a common clock for all registers, and an overall predictable hardware where communications and computations can be proven to take less than a clock-cycle.

**system-on-chip:** is divided into multiple, asynchronous clock domains: *Globally Asynchronous Locally Synchronous* (GALS)

**multiple clock domains** modular designs with separate compilation phases, for a single system with multiple input/output associated with different real-time clocks;

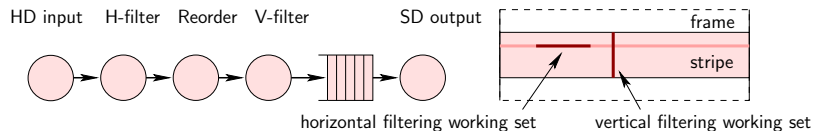
**Kahn Process Networks** (KPN) model for processes communicating through unbounded FIFO buffers.

# Downscaler

high definition (HD) → standard definition (SD)  
1920 × 1080 pixels → 720 × 480

**horizontal filter:** number of pixels in a line from 1920 pixels  
downto 720 pixels,

**vertical filter:** number of lines from 1080 downto 480



## Real-Time Constraints

the input and output processes: 30Hz.

HD pixels arrive at  $30 \times 1920 \times 1080 = 62,208,000\text{Hz}$

SD pixels at  $30 \times 720 \times 480 = 10,368,000\text{Hz}$  (6 times slower)

HF: 8:3

Reorder: stores 6 lines, transposes them by column of 6 pixels

VF: 9:4

## Required Features of the Language

automatically produces an efficient code for an embedded architecture, checking that real-time constraints are satisfied and optimizing the total memory resources to store the intermediate data and the code itself.

1. a proof that, according to worst-case execution time hypotheses, the frame and pixel rate will be sustained;
2. an evaluation of the delay introduced by the downscaler in the video processing chain, i.e., the delay before the output process starts receiving pixels;
3. a proof that the system has bounded memory requirements;
4. an evaluation of memory requirements, to store data within the processes, and to buffer the stream produced by the vertical filter in front of the output process.

# Synchronous Languages (Esterel, Lustre and Signal)

- ▶ dedicated to hard real-time critical systems
- ▶ concurrent *but* deterministic model
- ▶ hardware and software compilers, static analysis, verification tools

## **But too much restrictive for our video applications**

- ▶ a synchronous program define a 0-synchronous KPN (no buffering)
- ▶ adding buffer (by hand) is feasible but error-prone
- ▶ we can relax synchrony and define  $N$ -synchrony
- ▶  $N$ -synchronous programs can be transformed into 0-synchronous ones

# Outline

## Review

Introduction

Example

## Clocks as Infinite Binary Words

Definitions

Synchronizability

## The Programming Language

Syntax

Clock Calculus

Relaxed Clock Calculus

# Infinite Binary Words

*infinite binary words:*  $(0 + 1)^\omega$

*infinite periodic binary words*  $\Sigma_2^*$ :

$$\begin{aligned} w &::= u.(v) \\ v &::= 0 \mid 1 \mid 0.v \mid 1.v \\ u &::= \epsilon \mid 0 \mid 1 \mid 0.u \mid 1.u \end{aligned}$$

with  $(v) = \lim_n v^n$  is the periodic repetition of the pattern  $v$ .

$|w|$  length of  $w$ ,  $|w|_1$  number of 1,  $|w|_0$  number of 0,  $w[n]$   $n$ -th letter,  $w[1..n]$  prefix of length  $n$ .  $w \sqsubseteq w' \Rightarrow \exists v$  binary word, such that  $w.v = w'$

$[w]_p$  the position of the  $p$ -th 1.  $w_1 \prec w_2$  iff  $\forall p \geq 1, [w_1]_p \leq [w_2]_p$

# Clocks

*clocks*

$$\begin{aligned} \text{clk} &::= w \mid \text{clk on } w \\ w &\in (0 + 1)^\omega \end{aligned}$$

on:

$$\begin{aligned} 0.w \text{ on } w' &= 0. (w \text{ on } w') \\ 1.w \text{ on } 0.w' &= 0. (w \text{ on } w') \\ 1.w \text{ on } 1.w' &= 1. (w \text{ on } w') \end{aligned}$$

algorithm

$w'' = w \text{ on } w'$  with  $\forall n \in \mathbb{N}, w''[n] = w[n] \wedge w'[|w[1..n]|_1]$ .

on-associativity

Let  $w_1, w_2$  and  $w_3$  be three infinite binary words. Then  
 $w_1 \text{ on } (w_2 \text{ on } w_3) = (w_1 \text{ on } w_2) \text{ on } w_3$ .

# Periodic Clocks

Periodic Clocks:

$$clk ::= w \mid clk \text{ on } w \\ w \in \Sigma_2^*$$

We can always write  $clk_1 = a.(b)$  and  $clk_2 = c.(d)$  with  $|a| = |c| = \max(|u|, |u'|)$  and  $|b| = |d| = \text{lcm}(|v|, |v'|)$  where  $\text{lcm}$  is the least common divisor.

For instance,  $010(001100)$  and  $10001(10)$  become  $01000(110000)$  and  $10001(101010)$

# Clock Signatures

- ▶ A synchronous process transforming an input into an output is given a *clock signature*.
- ▶ This signature is a *type* and is an abstraction of the way the process consumes its inputs and produces its output.
- ▶  $\alpha \rightarrow \alpha$  on  $w$ , it means that for all valuation  $w' \in \Sigma_2^*$  of the variable  $\alpha$ , the output has the clock  $w'$  on  $w$ .

# Downscaler Signatures

- input process is on the base clock, i.e., the binary word (1)
- HF:  $\alpha \rightarrow \alpha$  on (10100100).
- reordering process delays the output of  $5 \times 720 \times 8/3 = 7680$  cycles. The clock signature  $\alpha \rightarrow 0^{7680}\alpha$ .
- VF:
 
$$\alpha \rightarrow \alpha \text{ on } (1^{720}0^{720}1^{720}0^{1440}1^{720}0^{1440}1^{720})$$

simplification:  $\alpha \rightarrow \alpha$  on (101001001).
- output's process clock (100000).

# Synchronizing Clocks

$f_1 : \alpha_1 \rightarrow C_1[\alpha_1]$  and  $f_2 : \alpha_2 \rightarrow C_2[\alpha_2]$   
then composition  $f = f_2 \circ f_1 : \alpha_1 \rightarrow C_2[C_1[\alpha_1]]$

Required:  $output = buffer(ver(reorder(hor(input))))$

- ▶  $hor(input)$ : (1) on (10100100) = (10100100).
- ▶  $reorder(hor(input))$ :  $0^{7680}(10100100)$ .
- ▶  $ver(reorder(hor(input)))$ :  $0^{7680}(10100100)$  on (101001001) =  $0^{7680}(100001000000010000000100) \neq (100000)$ .

# Synchronizability

$clk_i$  are *synchronizable*,  $clk_1 \bowtie clk_2$ , iff there exists  $d, d' \in \mathbb{N}$  such that  $clk_1 \prec 0^d \cdot clk_2$  and  $clk_2 \prec 0^{d'} \cdot clk_1$ .

It means that we can delay  $clk_1$  by  $d'$  ticks so that the 1 of  $clk_2$  occur before the 1 of  $clk_1$  and conversely.

1. 11(01) and (10) are synchronizable;
2. 11(0) and (0) are not synchronizable;
3. (010) and (10) are not synchronizable since there are too much reads or too much writes (infinite buffer).

$clk_1 \bowtie clk_2 \Rightarrow \exists$  two synchronous processes, called buffers  $b_1$  and  $b_2$  such that  $b_1(clk_1) = 0^d \cdot clk_2$  and  $b_2(clk_2) = 0^{d'} \cdot clk_1$ .

# Synchronizability: Periodic Clocks

$clk_1 \bowtie clk_2$  iff

$$\begin{cases} \frac{|v|_1}{|v'|_1} = \frac{|v|}{|v'|} \text{ if } |v'|_1 > 0 \\ |u| = |u|_1 \wedge |v|_1 = 0 \text{ otherwise} \end{cases}$$

The first condition means that are in average the same number of writes and reads in  $(v)$  and  $(v')$ . The second condition deals with the particular case of finite streams where there must be precisely the same number of writes and reads.

## Delaying a Clock

Delay the reads after the writes

$$delay = \min\{l \mid clk_1 \prec 0^l . clk_2\}$$

$clk_1 = u(v)$  and  $clk_2 = u'(v')$  with  $|u| = |u'|$  and  $|v| = |v'|$ . Then  $delay = \max(d', 0)$  where

$$d' = \max \{ |w| - |w'| \mid w.1 \sqsubseteq uv, w'.1 \sqsubseteq u'v', |w'|_1 = |w|_1 \}$$

For instance if  $clk_1 = 000001$  and  $clk_2 = 001000$ , then  $d = 3$  is reached with  $w = 000001$  and  $w' = 00$ .

Downscaler:

100001	000000	010000	000100
<b>000</b> 100	000100	000100	000100

# Buffer Size

$clk_1 \prec 0^d.clk_2$ . We write  $clk_1 = u(v)$  and  $0^d.clk_2 = u'(v')$  with  $|u| = |u'|$  and  $|v| = |v'|$ .

The minimal buffer size  $n$  satisfies:

$$n = \max\{(|w|_1 - |w'|_1) \mid w \sqsubseteq uv, w' \sqsubseteq u'v'; |w| = |w'|\}$$

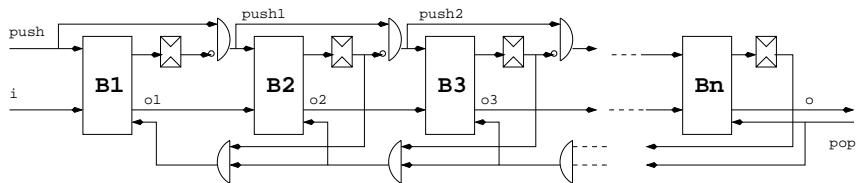
Communication from  $clk_1$  to  $clk_2$  is called  *$n$ -synchronous*.

**Downscaler:** simplified version buffer size = 1 and general version = 400.

# Buffer Construction

A buffer is a synchronous program

- ▶ either compute statically all the intermediate periodic clocks controlling each register of a  $N$ -buffer (but cubic size)
- ▶ or simply implement it as a sequence of connected one buffer (linear size)



# A Synchronous Data-flow Kernel

Reminiscent to Lustre and Lucid Sychrone

$$e ::= x \mid i \mid (e, e) \mid e \text{ where } x = e \mid e(e) \\ \mid e \text{ fby } e \mid e \text{ when } pe \mid \text{merge } pe \ e \ e \\ \mid \text{fst } e \mid \text{snd } e$$

$$d ::= \text{node } x(x) = e \mid d; d$$

$$dp ::= \text{period } p = pe \mid dp; dp$$

$$pe ::= p \mid w \mid pe \text{ on } pe \mid \text{not } pe \mid pe \text{ or } pe \mid pe \ \& \ pe$$

## Example

*node hf*  $p = o$  *where*

$o1 = p$

*and*  $o2 = 0$  *fb*  $o1$

*and*  $o3 = 0$  *fb*  $o2$  *and*  $o4 = 0$  *fb*  $o3$

*and*  $o5 = 0$  *fb*  $o4$  *and*  $o6 = 0$  *fb*  $o5$

*and*  $o = (o1 + o2 + o3 + o4 + o5 + o6)/6$  *when* (10100100)

*node vf*  $(i1, i2, i3, i4, i5, i6) = o$  *where*

$o = (i1 + i2 + i3 + i4 + i5 + i6)/6$  *when* (101001001)

*node main*  $(i : (1)) = (o : 0^{7683}(100000))$  *where*

$t = fh\ i$  *and*  $(i1, i2, i3, i4, i5, i6) = buff1(t)$

*and*  $o = vf(i1, i2, i3, i4, i5, i6);;$

# Clock Calculus I

Clock calculus as a type system  $\rightarrow$  judgments of the form  $P, H \vdash e : ct$  meaning that “the expression  $e$  has clock type  $ct$  in the environment of periods  $P$  and the environment  $H$ ”.

$$\sigma ::= \forall \alpha_1, \dots, \alpha_m. ct$$

$$ct ::= ct \rightarrow ct \mid ct \times ct \mid ck$$

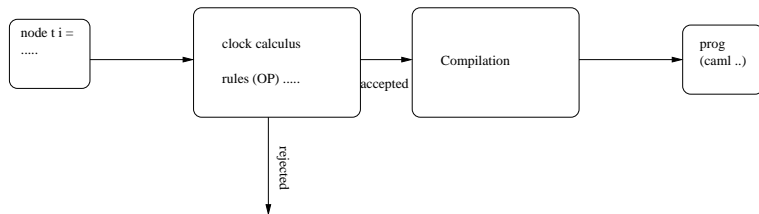
$$ck ::= \text{base} \mid ck \text{ on } pe \mid \alpha$$

$$H ::= [x_1 : \sigma_1, \dots, x_m : \sigma_m]$$

$$P ::= [p_1 : pe_1, \dots, p_n : pe_n]$$

clock schemes ( $\sigma$ ), unquantified clock types ( $ct$ ), clock type variables ( $\alpha$ ), functional clock types ( $ct \rightarrow ct$ ), products ( $ct \times ct$ ), or stream clocks ( $ck$ ), base clock (base), sampled clock ( $ck \text{ on } pe$ ), clock variable ( $\alpha$ ).

# 0-Synchrony Compilation



# Relaxed Clock Calculus

$n$ -synchronous programs  $\leftrightarrow$  programs which can be executed using buffers of size at most  $n$ .

Kahn networks  $\leftrightarrow$   $\infty$ -synchronous programs.

Synchronous programs  $\leftrightarrow$  0-synchronous programs.

0-synchrony can be checked using standard Milner-type system [ICFP'96, Emsoft'03]

Extension of the clock calculus with a **sub-typing rule**:

$$\text{(SUB)} \quad \frac{P, H \vdash_s e : ck \text{ on } pe_1 \quad pe_1 \prec pe_2}{P, H \vdash_s e : ck \text{ on } pe_2}$$

Because  $(\{0 + 1\}^\omega, \prec)$  is a complete lattice, we can follow Aiken & Wimmers technique to implement the system

# Example

node  $f(x) = y$  where  $y = (x \text{ when } (01)) + (x \text{ when } 1(10))$

(01) and 1(10) can be synchronized using a buffer of size 1. We can apply the rule:

$$\frac{P, H \vdash_s x \text{ when } 1(10) : \alpha \text{ on } 1(10) \quad 1(10) \prec (01)}{P, H \vdash_s x \text{ when } (01) : \alpha \text{ on } (01)}$$

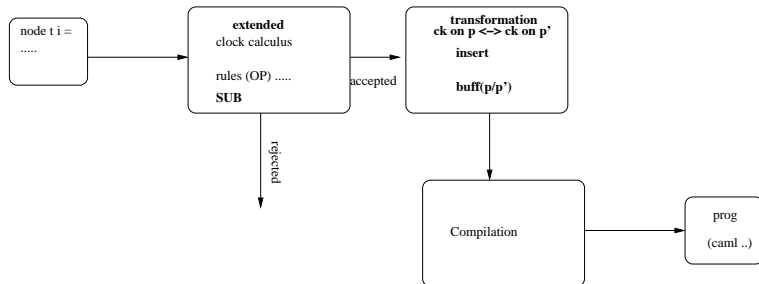
and then, classical rules apply and we get the final signature:

$$f : \forall \alpha. \alpha \rightarrow \alpha \text{ on } (01)$$

# Translation Semantics

programs accepted with the relaxed clock calculus  $\rightarrow$  synchronous programs which are accepted by the original clock calculus through a program transformation which insert a buffer every time the (SUB) is applied.

# Relaxed Synchrony Compilation



# Conclusion

- ▶ design of real-time applications: strong correctness requirements, decomposition into modular components communicating thanks to a buffering mechanism;
- ▶ global system is synchronous but hard by hand;
- ▶ extended synchronous framework: automatic generation of the synchronous buffers which are semantically (as defined by Kahn) guaranteed correct.
- ▶ periodic clocks;
- ▶ synchronous functional programming language.

## Future Work

- ▶ algebraic characterization of clocks (diadic numbers);
- ▶ connection between retiming and delay insertion;
- ▶ towards a criterion of optimization of the buffers (here we choose a particular solution but no unicity);