Synchronizing Periodic Clocks in Kahn Networks

Albert Cohen[1], Marc Duranton[2], Christine Eisenbeis[1], Claire Pagetti[1] and Marc Pouzet[3]
(submitted LCTES)

[1]Inria Futurs, Orsay France

[2]Philips Research Laboratories, Eindhoven, The Netherlands

[3]Université Pierre et Marie Curie, Paris, France

February the 3rd, 2005

Context

Domain: real-time video processing
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 $\Longrightarrow$ 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;

our assumption: execution layer with a global clock.

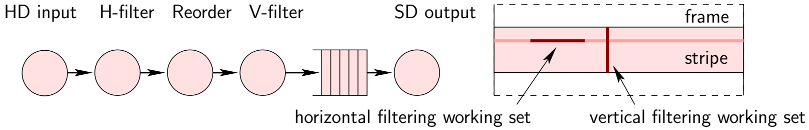
Kahn Process Networks (KPN) model for processes communicating through unbounded FIFO buffers.

Downscaler

high definition (HD) $\longrightarrow$ 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


HD input
H-filter
Reorder
V-filter
SD output
frame
stripe
horizontal filtering working set
vertical filtering working set

Real-Time Constraints

the input and output processes: 30Hz.
HD pixels arrive at $30 \times 1920 \times 1080 = 62,208,000 Hz$
SD pixels at $30 \times 720 \times 480 = 10,368,000 Hz$ (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.

Outline

Infinite Binary Words

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

infinite periodic binary words Σ_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

$$\begin{array}{l} clk ::= w \mid clk \textit{ on } w \\ w \in (0+1)^\omega \end{array}$$

on:

$$\begin{array}{lcl} 0.w \textit{ on } w' & = & 0.\,(w \textit{ on } w') \\ 1.w \textit{ on } 0.w' & = & 0.\,(w \textit{ on } w') \\ 1.w \textit{ on } 1.w' & = & 1.\,(w \textit{ on } w') \end{array}$$

algorithm
$w'' = w \textit{ on } w'$ 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 $w_1 \textit{ on } (w_2 \textit{ on } w_3) = (w_1 \textit{ on } w_2) \textit{ on } w_3$.

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 $|b| = |d| = lcm(|v|, |v'|)$ where lcm is the least common divisor.
For instance, 010(001100) and 10001(10) become 01000(110000) and 10001(101010)

Clock Signatures

A synchronous process transforms an input clock into an output clock. This transformation is encoded in the process *clock signature*
$\alpha \longrightarrow \alpha$ *on* w, it means that for all valuation $w' \in \Sigma_2^*$ of the variable α, the output has the clock w' *on* w.

Downscaler Signatures

1. input process is the binary word (1)
2. HF: $\alpha \rightarrow \alpha \textit{ on } (10100100)$.
3. reordering process delays the output of $5 \times 720 \times 8/3 = 7680$ cycles. The clock signature $\alpha \rightarrow 0^{7680}\alpha$.
4. VF:
$$\alpha \rightarrow \alpha \textit{ on } (1^{720}0^{720}1^{720}0^{1440}1^{720}0^{1440}1^{720})$$
simplification: $\alpha \rightarrow \alpha \textit{ on } (101001001)$.
5. 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 $f = f_2(f_1) : \alpha_1 \rightarrow C_2[C_1[\alpha_1]]$

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

- $hor(input)$: (1) *on* $(10100100) = (10100100)$.
- $reorder(hor(input))$: $0^{7680}(10100100)$.
- $vert(reorder(hor(input)))$: $0^{7680}(10100100)$ *on* $(101001001) = 0^{7680}(1000010000000010000000100) \neq (100000)$.

Synchronizability

clk_i are *synchronizable*, $clk_1 \bowtie clk_2$, iff there exists $d, d' \in \mathbb{N}$ such that $clk_1 \prec 0^d.clk_2$ and $clk_2 \prec 0^{d'}.clk_1$.
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 $b_1(clk_1) = 0^d.clk_2$ and $b_2(clk_2) = 0^{d'}.clk_1$.

Synchronizability: Periodic Clocks

$clk_1 \bowtie clk_2$ iff

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

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:

$$\begin{array}{cccc} 100001 & 000000 & 010000 & 000100 \\ \mathbf{000}100 & 000100 & 000100 & 000100 \end{array}$$

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

`NOP` — $w[j] = 0$ and $w'[j] = 0$: Nothing happens in the buffer: $clk_i[j] = 0$, $w_i[j] = w_i[j-1]$; registers x_i are left unchanged.

`PUSH` — $w[j] = 1$ and $w'[j] = 0$: Some data is written into the buffer and stored in register x_1, all the data in the buffer being pushed from x_i into x_{i+1}. Thus $x_i = x_{i-1}$ and $x_1 = input$, $\forall i > 2, w_i[j] = w_{i-1}[j-1]$, $w_1[j] = 1$ and $clk_i[j] = 0$.

Buffer Construction

`POP` — $w[j] = 0$ and $w'[j] = 1$: Let $p = max\{0\} \cup \{1 \leq i \leq n | w_i[j-1] = 1\}$. If p is zero, then no register stores any data at cycle j: input data must be bypassed directly to the output, crossing the wire clocked by clk_0, setting $clk_i[j] = 0$ for $i > 0$ and $clk_0[j] = 1$, $w_i[j] = w_i[j-1]$. Conversely, if $p > 0$, $\forall i \neq p, clk_i[j] = 0$, $clk_p[j] = 1$, $\forall i \neq p, w_i[j] = w_i[j-1]$ and $w_p[j] = 0$. Registers x_i are left unchanged (notice this is not symmetric to the `PUSH` operation).

`POP; PUSH` — $w[j] = 1$ and $w'[j] = 1$: A `POP` is performed, followed by a `PUSH`, as defined in the two previous cases.

A Synchronous Data-flow Kernel

$$\begin{array}{rcl} e & ::= & x \mid i \mid (e, e) \mid e \;\texttt{where}\; x = e \mid e(e) \\ & & \mid e \;\texttt{fby}\; e \mid e \;\texttt{when}\; pe \mid \texttt{merge}\; pe\; e\; e \\ & & \mid \texttt{fst}\; e \mid \texttt{snd}\; e \end{array}$$

Expressions (e), constants (i), variables (x), pairs (e, e), local definitions (e `where` $x = e$), applications ($e(e)$), initialized delays (e `fby` e).

A Synchronous Data-flow Kernel

$$\begin{array}{rcl} e & ::= & x \mid i \mid (e, e) \mid e \ \texttt{where}\ x = e \mid e(e) \\ & & \mid e \ \texttt{fby}\ e \mid e \ \texttt{when}\ pe \mid \texttt{merge}\ pe\ e\ e \\ & & \mid \texttt{fst}\ e \mid \texttt{snd}\ e \end{array}$$

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

stream functions: $\texttt{node}\ x(x) = e$

A Synchronous Data-flow Kernel

$$
\begin{array}{lcl}
e & ::= & x \mid i \mid (e, e) \mid e \; \texttt{where} \; x = e \mid e(e) \\
 & & \mid e \; \texttt{fby} \; e \mid e \; \texttt{when} \; pe \mid \texttt{merge} \; pe \; e \; e \\
 & & \mid \texttt{fst} \; e \mid \texttt{snd} \; e \\
 & & \\
d & ::= & \texttt{node} \; x(x) = e \mid d; d \\
 & & \\
dp & ::= & \texttt{period} \; p = pe \mid dp; dp
\end{array}
$$

periods: $\texttt{period} \; p = pe$

A Synchronous Data-flow Kernel

$$
\begin{array}{lcl}
e & ::= & x \mid i \mid (e,e) \mid e \ \texttt{where}\ x = e \mid e(e) \\
 & & \mid e \ \texttt{fby}\ e \mid e \ \texttt{when}\ pe \mid \texttt{merge}\ pe\ e\ e \\
 & & \mid \texttt{fst}\ e \mid \texttt{snd}\ e \\
 & & \\
d & ::= & \texttt{node}\ x(x) = e \mid d; d \\
 & & \\
dp & ::= & \texttt{period}\ p = pe \mid dp; dp \\
 & & \\
pe & ::= & p \mid w \mid pe \ \texttt{on}\ pe \mid \texttt{not}\ pe \mid pe \ \texttt{or}\ pe \mid pe \ \&\ pe
\end{array}
$$

Example

```
node hf p = o where
        o1= p
        and o2= 0 fby o1
        and o3= 0 fby o2 and o4= 0 fby o3
        and o5= 0 fby o4 and o6= 0 fby 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);;
```

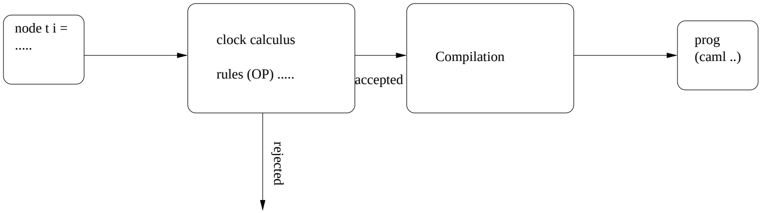
Clock Calculus I

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”.

$$\begin{array}{lcl}
\sigma & ::= & \forall \alpha_1, ..., \alpha_m . ct \\
ct & ::= & ct \rightarrow ct \mid ct \times ct \mid ck \\
ck & ::= & \texttt{base} \mid ck \texttt{ on } pe \mid \alpha \\
 & & \\
H & ::= & [x_1 : \sigma_1, ..., x_m : \sigma_m] \\
P & ::= & [p_1 : pe_1, ..., p_n : pe_n]
\end{array}$$

clock schemes (σ), unquantified clock types (ct), clock type variables (α), functional clock types ($ct \rightarrow ct$), products ($ct \times ct$), or stream clocks (ck), base clock (base), sampled clock (ck on pe), clock variable (α).

0-Synchrony Compilation



Relaxed Clock Calculus

n-synchronous programs $\leftrightarrow$ programs which can be executed using buffers of size at most n.
Kahn networks $\leftrightarrow$ ∞-synchronous programs.
Synchronous programs $\leftrightarrow$ 0-synchronous programs.

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

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

Example

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

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

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

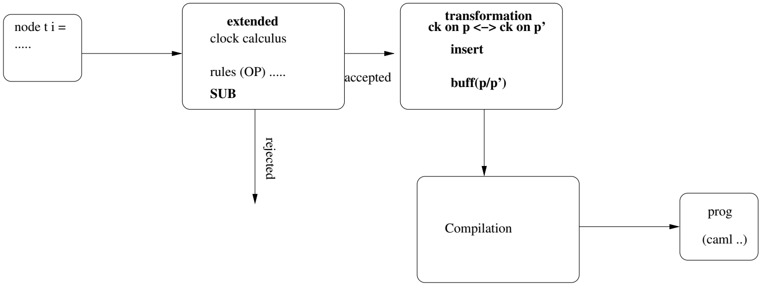
and then, classical rules apply and we get the final signature:

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

Translation Semantics

programs accepted with the relaxed clock calculus $\longrightarrow$ 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



Buffer in Lucid size 1

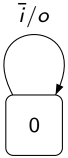
```
let buffer1 (push, pop, i) = (empty, o) where
    rec o = if pempty then i
        else pmemo
    and memo = if push then i else pmemo
    and pmemo = 0 fby memo
    and empty = if push then if pop then pempty
        else false
        else if pop then true
        else pempty
    and pempty = true fby empty
```

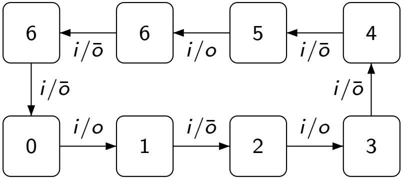
Alphabet

$\Sigma = \{i/o, i/\bar{o}, \bar{i}/o, \bar{i}, /\bar{o}\}$.

- the symbol i stands for an input occurs,
- the symbol o stands for an output occurs, so that for instance the event i/o means that an input and an output occur simultaneously,
- $\bar{i}$ stands for no input occurs and the symbol $\bar{o}$ stands for no output occurs.

Example





$\mathcal{A}_0 = \textit{input}$ $\mathcal{A}_1 = \textit{horizontal filter}$

$L \subseteq \Sigma^*$ can be seen as $L_1 \times L_2 \subseteq \{0,1\}^* \times \{0,1\}^*$ with $l \leftrightarrow 1$ and $\bar{l} \leftrightarrow 0$ with $l = i, o$

Synchronization

$$\begin{tikzcd}\boxed{q} \arrow[r, "a/o"] & \boxed{q'} & \| & \boxed{s} \arrow[r, "i/b"] & \boxed{s'} & = & \boxed{(q,s)} \arrow[r, "a/b"] & \boxed{(q',s')}\end{tikzcd}$$

$$\begin{tikzcd}\boxed{q} \arrow[r, "a/\bar{o}"] & \boxed{q'} & \| & \boxed{s} \arrow[r, "\bar{i}/b"] & \boxed{s'} & = & \boxed{(q,s)} \arrow[r, "a/b"] & \boxed{(q',s')}\end{tikzcd}$$

$L(\mathcal{A}_1) \equiv L_1^1 \times L_2^1$ and $L(\mathcal{A}_2) \equiv L_1^2 \times L_2^2$. Then $\mathcal{A} = \mathcal{A}_1 \parallel \mathcal{A}_2$ recognizes $L(\mathcal{A}) \equiv L_1 \times L_2$. We have $L_1 \to L_1 \mathit{ on } L_2 = L_1^1 \to (L_1^1 \mathit{ on } L_2^1 \mathit{ on } (L_1^2 \mathit{ on } L_2^2))$. This means that $L_1 = L_1^1$ and $L_2 = L_2^1 \mathit{ on } (L_1^2 \mathit{ on } L_2^2)$.

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);
- delays in the language.