Automatic rate desynchronisation of reactive embedded systems

Paul CASPI, Alain GIRAULT, Xavier NICOLLIN, Daniel PILAUD, and Marc POUZET

INRIA Rhône-Alpes, INPG-VERIMAG, and Orsay/LRI

Grenoble and Paris, FRANCE
Introduction

Embedded reactive programs

- **embedded** so they have limited resources
- **reactive** so they react continuously with their environment
Introduction

Embedded reactive programs

- embedded so they have limited resources
- reactive so they react continuously with their environment

We consider programs whose control structure is a finite state automaton

Put inside a periodic execution loop:

loop each tick
  read inputs
  compute next state
  write outputs
end loop
**Automatic rate desynchronisation**

**Desynchronisation**: to transform one centralised synchronous program into a GALS program

- Each local program is embedded inside its own periodic execution loop

**Automatic**: the user only provides distribution specifications

**Rate desynchronisation**:
- the periods of the execution loops will not be the same and
- not necessarily identical to the period of the initial centralised program
Motivation: long duration tasks

Characteristics:

- Their execution time is long
- Their execution time is known and bounded
- Their maximal execution rate is known and bounded

Examples:

- The CO3N4 nuclear plant control system of Schneider Electric
- The Mars rover pathfinder
A small example

Consider a system with three independent tasks:

- Task A performs slow computations:
  - duration = 8, period = deadline = 32

- Task B performs medium and not urgent computations:
  - duration = 6, period = deadline = 24

- Task C performs fast and urgent computations:
  - duration = 4, period = deadline = 8

How to implement this system?
Manual task slicing

Tasks A and B are sliced into small chunks, which are interleaved with task C.
Manual task slicing

Tasks A and B are sliced into small chunks, which are interleaved with task C.

Very hard and error prone because:
- The slicing is complex
- The implementation must be correct and deadlock-free
Manually programming 3 async. tasks

Tasks A, B, and C are performed by one process each.

The task slicing is done by the scheduler of the underlying RTOS.

But the manual programming is difficult.

Example: the Mars Rover Pathfinder had priority inversion!
Automatic distribution

The user programs a centralised system.

The centralised program is compiled, debugged, and validated.

It is then automatically distributed into three processes.

The correctness ensures that the obtained distributed system is functionally equivalent to the centralised one.
Example: the FILTER program

state 0:
go(CK, IN)
if (CK) then
    RES:=0
    write(RES)
    V:=0
    OUT:=SLOW(IN)
    write(OUT)
    goto 1
else
    RES:=V
    write(RES)
goto 0
endif
Example: the FILTER program

state 0:

go(CK, IN)
if (CK) then
  RES:=0
  write(RES)
  V:=0
  OUT:=SLOW(IN)
  write(OUT)
  goto 1
else
  RES:=V
  write(RES)
  goto 0
endif

state 1:

go(CK, IN)
if (CK) then
  RES:=OUT
  V:=OUT
  OUT:=SLOW(IN)
  write(OUT)
else
  RES:=V
  write(RES)
  goto 1
endif
Example: the **FILTER** program

**state 0:**

\[
\text{go}(\text{CK, IN})
\]

*if (CK) then*

\[
\begin{align*}
\text{RES} & := 0 \\
\text{write}(\text{RES}) \\
\text{V} & := 0 \\
\text{OUT} & := \text{SLOW(IN)} \\
\text{write}(\text{OUT})
\end{align*}
\]

*else*

\[
\begin{align*}
\text{RES} & := \text{V} \\
\text{write}(\text{RES}) \\
\text{goto} 0
\end{align*}
\]

*endif*

**state 1:**

\[
\text{go}(\text{CK, IN})
\]

*if (CK) then*

\[
\begin{align*}
\text{RES} & := \text{OUT} \\
\text{V} & := \text{OUT} \\
\text{OUT} & := \text{SLOW(IN)} \\
\text{write}(\text{OUT})
\end{align*}
\]

*else*

\[
\begin{align*}
\text{RES} & := \text{V} \\
\text{write}(\text{RES}) \\
\text{goto} 1
\end{align*}
\]

*endif*
Example: the **FILTER** program

state 0:
go(CK, IN)
if (CK) then
  RES:=0
  write(RES)
  V:=0
  OUT:=SLOW(IN)
  write(OUT)
goto 1
else
  RES:=V
  write(RES)
goto 0
endif

state 1:
go(CK, IN)
if (CK) then
  RES:=OUT
  V:=OUT
  OUT:=SLOW(IN)
  write(OUT)
else
  RES:=V
endif
write(RES)
goto 1

It has two inputs (the Boolean ck and the integer in) and two outputs (the integers res and out)
Example: the **FILTER** program

**state 0:**
go(CK, IN)
if (CK) then
    RES:=0
    write(RES)
    V:=0
    OUT:=SLOW(IN)
    write(OUT)
goto 1
else
    RES:=V
    write(RES)
goto 0
endif

**state 1:**
go(CK, IN)
if (CK) then
    RES:=OUT
    V:=OUT
    OUT:=SLOW(IN)
    write(OUT)
else
    RES:=V
    endif
    write(RES)
goto 1

It has two inputs (the Boolean **CK** and the integer **IN**) and two outputs (the integers **RES** and **OUT**)

The **go(CK, IN)** action materialises the read input phase
Rates

The FILTER program has two inputs (the Boolean CK and the integer IN) and two outputs (the integers RES and SLOW).

Each input and output has a rate, which is the sequence of logical instants where it exists.

- IN is used only when CK is true, so its rate is CK.
- CK is used at each cycle, so its rate is the base rate.
- OUT is computed each time CK is true, so its rate is CK.
- RES is computed at each cycle, so its rate is the base rate.
A run of the centralised FILTER

CK₁=T

IN₁=13

RES₁=0

OUT₁=42

logical time/state
A run of the centralised FILTER

CK₁ = T
IN₁ = 13
OUT₁ = 42

RES₁ = 0
CK₂ = F
RES₂ = 0

CK₂ = F
2/1

logical time/state

CK₂ = F
1/0

OUT₁ = 42
A run of the centralised FILTER

CK₁=T
IN₁=13
OUT₁=42

RES₁=0
CK₂=F
2/1

RES₂=0
CK₃=F
3/1

RES₃=0

logical time/state

– p.11/35
A run of the centralised FILTER

CK₁=T

CK₂=F

CK₃=F

CK₄=T

IN₁=13

OUT₁=42

RES₁=0

RES₂=0

RES₃=0

RES₄=42

OUT₂=27

logical time/state
A run of the centralised FILTER

CK₁=T

IN₁=13

OUT₁=42

RES₁=0

CK₂=F

2/1

FILTER

RES₂=0

CK₃=F

3/1

FILTER

RES₃=0

CK₄=T

4/1

FILTER

RES₄=42

logical time/state

IN₂=9

OUT₂=27

p.11/35
A run of the centralised FILTER

\[
\begin{align*}
\text{WCET}(\text{SLOW}) & = 7 \\
\text{WCET}(\text{other computations}) & = 1
\end{align*}
\]

\[\implies \text{WCET}(\text{FILTER}) = 8\]

Thus the period of the execution loop (base rate) must be greater than 8
Where are we going?

CK₁ = T
IN₁ = 13
OUT₁ = 42

CK₂ = F
RES₁ = 0
CK₃ = F
RES₂ = 0
CK₄ = T
RES₃ = 0
OUT₂ = 27

IN₂ = 9

logical time/state
Where are we going?

Two tasks running on a single processor:

Task L performs the fast computations
Task M performs the slow computations, sliced into 3 chunks
Where are we going?

Two tasks running on two processors:

RES = 0 0 0 42 42 42 27 27 27
CK = T F F T F T F F

Logical time/state for L

IN = IN1 = 13
OUT = OUT1 = 42

Logical time/state for M

IN = IN2 = 9
OUT = OUT2 = 27

IN = IN3 = 40
OUT = OUT3 = 69

- p.12/35
Our automatic distribution algorithm

Lustre program

Lustre compiler

One centralized automaton

Automatic distributor

[N  communicating automata

(one automaton for each computing location)]

[Caspi, Girault & Pilaud 1999]
Communication primitives

Two FIFO channels for each pair of locations, one in each direction:

- \texttt{send(dst,var)} inserts the value of variable \texttt{var} into the queue directed towards location \texttt{dst}

  Non blocking

- \texttt{var:=receive(src)} extracts the head value from the queue starting at location \texttt{src} and assigns it to variable \texttt{var}

  Blocking when the queue is empty
## Distribution specifications

<table>
<thead>
<tr>
<th>location name</th>
<th>assigned rates</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>base</td>
</tr>
<tr>
<td>M</td>
<td>CK</td>
</tr>
</tbody>
</table>

This part is given by the user
## Distribution specifications

<table>
<thead>
<tr>
<th>location name</th>
<th>assigned rates</th>
<th>inferred inputs &amp; outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>base</td>
<td>CK, RES</td>
</tr>
<tr>
<td>M</td>
<td>CK</td>
<td>IN, OUT</td>
</tr>
</tbody>
</table>

The inferred inputs and outputs are those whose rate matches the assigned rate

\[
\begin{align*}
\text{base} & \quad \{\text{RES, CK}\} \\
\downarrow & \\
\text{CK} & \quad \{\text{IN, OUT}\}
\end{align*}
\]
## Distribution specifications

<table>
<thead>
<tr>
<th>location name</th>
<th>assigned rates</th>
<th>inferred inputs &amp; outputs</th>
<th>inferred location rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>L</td>
<td>base</td>
<td>CK, RES</td>
<td>base</td>
</tr>
<tr>
<td>M</td>
<td>CK</td>
<td>IN, OUT</td>
<td>CK</td>
</tr>
</tbody>
</table>

The inferred rate is the root of the smallest subtree containing all the rates assigned by the user.
First attempt of distribution

state 0

go(CK, IN)

if (CK) then

   RES:=OUT
   V:=OUT
   OUT:=SLOW(IN)
   write(OUT)

else

   RES:=V

endif

write(RES)
goto 1
First attempt of distribution

state 0 -- location L

go(CK, IN)

if (CK) then

RES:=OUT
V:=OUT
OUT:=SLOW(IN)
write(OUT)

else
RES:=V
endif
write(RES)
goto 1

state 0 -- location M

go(CK, IN)

if (CK) then

RES:=OUT
V:=OUT
OUT:=SLOW(IN)
write(OUT)

else
RES:=V
endif
write(RES)
goto 1
First attempt of distribution

state 0 -- location L

if (CK) then
  RES:=OUT
  V:=OUT
else
  RES:=V
endif
write(RES)
goto 1

state 0 -- location M

if (CK) then
  OUT:=SLOW(IN)
  write(OUT)
else
endif

goto 1
First attempt of distribution

state 0 -- location L

<table>
<thead>
<tr>
<th>go (CK)</th>
</tr>
</thead>
<tbody>
<tr>
<td>send (M, CK)</td>
</tr>
<tr>
<td>if (CK) then</td>
</tr>
<tr>
<td>OUT := receive (M)</td>
</tr>
<tr>
<td>RES := OUT</td>
</tr>
<tr>
<td>V := OUT</td>
</tr>
<tr>
<td>else</td>
</tr>
<tr>
<td>RES := V</td>
</tr>
<tr>
<td>endif</td>
</tr>
<tr>
<td>write (RES)</td>
</tr>
<tr>
<td>goto 1</td>
</tr>
</tbody>
</table>

state 0 -- location M

| go (IN) |
| CK := receive (L) |
| if (CK) then |
|   send (L, OUT) |
|   OUT := SLOW (IN) |
|   write (OUT) |
| else |
| endif |
| goto 1 |
First attempt of distribution

The go(CK, IN) has been split into 

\{ go(CK) on location L \\
   go(IN) on location M \} 

---
The value of \( \text{CK} \) is sent by \( \text{L} \) to \( \text{M} \) at each cycle of the base rate.

\( \Rightarrow \) Location \( \text{M} \) runs at the speed of the base rate instead of \( \text{CK} \).

If the communications take 1, then the global WCET is still 8.
How to improve this?

We want location $M$ to run at the speed of $CK$

- This would give enough time for the computation of $SLOW$

- For this, location $L$ must not send $CK$ to location $M$

  - We can use an existing bisimulation for detecting and suppressing branchings like $if(CK)$ on location $M$

  - For this bisimulation to work, the $go(IN)$ action must be moved inside the $then$ branch on location $M$

    Makes sense because $IN$ is expected only when $CK$ is $true$

- The two programs will be logically desynchronized
Moving the go downward

Only the locations whose rate is not the base rate

A simple forward traversal of the program:
Moving the **go** downward

Only the locations whose rate is **not** the base rate

A simple forward traversal of the program:

```plaintext
loc. M (rate CK) – state 0

go(IN)

if (CK) then
  OUT:=SLOW(IN)
  write(OUT)
  goto 1
else
  goto 0
endif
```
Moving the **go** downward

Only the locations whose rate is **not** the base rate

A simple forward traversal of the program:

```
loc. M (rate CK) - state 0
  go(IN)
  if (CK) then
    OUT:=SLOW(IN)
    write(OUT)
    goto 1
  else
    goto 0
  endif
```

```
loc. M (rate CK) - state 0
  if (CK) then
    go(IN)
    OUT:=SLOW(IN)
    write(OUT)
    goto 1
  else
    goto 0
  endif
```
Suppressing useless branchings

Bisimulation fully presented in [Caspi, Fernandez & Girault 1995]
Suppressing useless branchings

Bisimulation fully presented in [Caspi, Fernandez & Girault 1995]
Suppressing useless branchings

Bisimulation fully presented in [Caspi, Fernandez & Girault 1995]
Suppressing useless branchings

Bisimulation fully presented in [Caspi, Fernandez & Girault 1995]

\[
\begin{align*}
go(IN) & \quad \text{OUT} := \text{SLOW}(IN) \\
\text{write}(OUT) & \\
goto 1
\end{align*}
\]
### Final result

<table>
<thead>
<tr>
<th>Location</th>
<th>Action</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>L</strong> (rate base)</td>
<td><strong>state 0:</strong></td>
<td>go(CK) if (CK) then { RES:=0 write(RES) goto 1 } else { RES:=V write(RES) goto 0 }</td>
</tr>
<tr>
<td></td>
<td><strong>state 1:</strong></td>
<td>go(IN) OUT:=SLOW(IN) write(OUT) goto 1</td>
</tr>
<tr>
<td><strong>M</strong> (rate CK)</td>
<td><strong>state 0:</strong></td>
<td>go(CK) if (CK) then { OUT:=receive(M) RES:=OUT V:=OUT goto 1 } else { RES:=V endif write(RES) goto 0</td>
</tr>
<tr>
<td></td>
<td><strong>state 1:</strong></td>
<td>go(IN) send(L, OUT) OUT:=SLOW(IN) write(OUT) goto 1</td>
</tr>
</tbody>
</table>
The period of $\mathbb{L}$ is one third of the period of $\mathbb{M}$.
A run of the newly distributed FILTER

RES = 0 0 0 42 42 42 27 27 27
CK = T F F T F F T F F

1/0 2/1 3/1 4/1 5/1 6/1 7/1 8/1 9/1 Logical time/state for L

M M M M M M M M M

1/0 2/1 3/1 Logical time/state for M

IN₁ = 13
OUT₁ = 42
IN₂ = 9
OUT₂ = 27
IN₃ = 40
OUT₃ = 69

Dummy communications can finally be added to guarantee bounded FIFO queues
Validating the synchronous abstraction

We have to compare the WCET with the execution loop period.

But our program is distributed into \( n \) tasks. So:

- We compute the \( n \) WCET
- We compute the total utilisation factor
- We check the Liu & Layland conditions (mono-processor case)
Validating the synchronous abstraction

We have to compare the WCET with the execution loop period

But our program is distributed into \( n \) tasks. So:

- We compute the \( n \) WCET
- We compute the total utilisation factor
- We check the Liu & Layland conditions (mono-processor case)

<table>
<thead>
<tr>
<th>location</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCET rate</td>
<td>2</td>
<td>8</td>
</tr>
<tr>
<td>rate</td>
<td>5</td>
<td>15</td>
</tr>
</tbody>
</table>
Validating the synchronous abstraction

We have to compare the WCET with the execution loop period

But our program is distributed into $n$ tasks. So:

- We compute the $n$ WCET
- We compute the total utilisation factor
- We check the Liu & Layland conditions (mono-processor case)

<table>
<thead>
<tr>
<th>Location</th>
<th>L</th>
<th>M</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCET rate</td>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

$$\frac{2}{5} + \frac{8}{15} = \frac{14}{15} \leq 1$$
RTOS implementation
RTOS implementation
This mechanism relies on the preemption mechanism of the RTOS!
RTOS implementation
Data-flow analysis

Program of location M

state 0:
- go(IN)
- \text{OUT}:=\text{SLOW}(\text{IN})
- write(\text{OUT})
- goto 1

state 1:
- go(IN)
- send(L,\text{OUT})
- \text{OUT}:=\text{SLOW}(\text{IN})
- write(\text{OUT})
- goto 1
Data-flow analysis

Program of location $M$

state 0:
- go(IN)
- OUT:=SLOW(IN)
- send(L,OUT)
- write(OUT)
- goto 1

state 1:
- go(IN)
- OUT:=SLOW(IN)
- send(L,OUT)
- send(L,OUT)
- write(OUT)
- goto 1
Data-flow analysis

Program of location M

state 0:
- go(IN)
- OUT:=SLOW(IN)
- send(L,OUT)
- write(OUT)
- goto 1

state 1:
- go(IN)
- OUT:=SLOW(IN)
- send(L,OUT)
- write(OUT)
- goto 1
Two applications

1. Clock driven automatic distribution of Lustre programs

2. Automatic rate desynchronisation of Esterel programs

Lustre is synchronous, declarative, data-flow

All objects are flows: infinite sequences of typed data
Each flow has a clock ( = first class abstract type)

The sequence of instants where the flow bears a value

Any Boolean flow defines a new clock: the sequence of instants where it bears the value true

Flows can then be upsampled (current) and downsampled (when)

A program must be correctly clocked

One clock is called the base clock of the program:

the sequence of its activation instants (the Esterel tick)

The set of clocks is a tree whose root is the base clock
node FILTER (CK : bool; (IN : int) when CK)
    returns (RES : int; (OUT : int) when CK);
let
    RES = current ((0 when CK) -> pre OUT);
    OUT = SLOW (IN);
tel.
function SLOW (A : int) returns (B : int);
node FILTER (CK : bool; (IN : int) when CK) returns (RES : int; (OUT : int) when CK); let
  RES = current ((0 when CK) -> pre OUT);
  OUT = SLOW (IN);
tel.
function SLOW (A : int) returns (B : int);

The SLOW function is long duration task
node FILTER (CK : bool; (IN : int) when CK)
  returns (RES : int; (OUT : int) when CK);
let
  RES = current ((0 when CK) -> pre OUT);
  OUT = SLOW (IN);
tel.
function SLOW (A : int) returns (B : int);

The clock tree is:

```
base   {RES, CK}
  ↓
   CK   {IN, OUT}
```
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>IN</td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CK</strong></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>OUT = SLOW(IN)</strong></td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CK</strong></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>OUT = SLOW(IN)</strong></td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td><strong>pre OUT</strong></td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CK</strong></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OUT = SLOW(IN)</strong></td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre OUT</strong></td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0 when CK</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\text{CK})</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>(\text{IN})</td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{OUT} = \text{SLOW(IN)})</td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(\text{pre OUT})</td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>((0\ \text{when }\text{CK}) \rightarrow \text{pre OUT})</td>
<td>0</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### An example of a run of FILTER

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CK</strong></td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
</tr>
<tr>
<td><strong>IN</strong></td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>OUT = SLOW(IN)</strong></td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>pre OUT</strong></td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>0 when CK</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>(0 when CK) -&gt; pre OUT</strong></td>
<td>0</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>RES = current (...)</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>...</td>
</tr>
</tbody>
</table>
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>IN</td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT = SLOW(IN)</td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre OUT</td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 when CK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0 when CK) -&gt; pre OUT</td>
<td>0</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES = current (...)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>...</td>
</tr>
</tbody>
</table>

- These are logical instants
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>IN</td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>OUT = SLOW(IN)</td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>pre OUT</td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>0 when CK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>(0 when CK) -&gt; pre OUT</td>
<td>0</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td>RES = current (...)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>42</td>
<td>42</td>
<td>42</td>
<td>27</td>
<td>27</td>
<td>27</td>
<td>...</td>
</tr>
</tbody>
</table>

- These are logical instants
- **OUT** must be **available at the same** clock cycle of **CK** as **IN**
An example of a run of **FILTER**

<table>
<thead>
<tr>
<th>base clock cycle number</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>CK</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>T</td>
<td>F</td>
<td>F</td>
<td>...</td>
</tr>
<tr>
<td>IN</td>
<td>14</td>
<td>9</td>
<td>23</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>OUT = SLOW(IN)</td>
<td>42</td>
<td>27</td>
<td>69</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>pre OUT</td>
<td>nil</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0 when CK</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(0 when CK) -&gt; pre OUT</td>
<td>0</td>
<td>42</td>
<td>27</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RES = current (...)</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td><strong>42</strong></td>
<td><strong>42</strong></td>
<td><strong>42</strong></td>
<td><strong>27</strong></td>
<td><strong>27</strong></td>
<td><strong>27</strong></td>
<td>...</td>
</tr>
</tbody>
</table>

- These are logical instants
- **OUT** must be available at the same clock cycle of **CK** as **IN**
- **RES** must be available at the next clock cycle of **CK**
Clock-driven automatic distribution

Automatic distribution:

From a centralised source program and some distribution specifications, we build automatically as many programs as required by the user.

Their combined behaviour will be functionnaly equivalent to the behaviour of the initial centralised program.
Clock-driven automatic distribution

Automatic distribution:

From a centralised source program and some distribution specifications, we build automatically as many programs as required by the user.

Their combined behaviour will be functionally equivalent to the behaviour of the initial centralised program.

Clock-driven:

The user specifies which clock goes to which computing location.

Partition of the set of clocks of the centralised source program.

One subset for each desired computing location.
Related work

- **Giotto** compiler: [Henzinger, Horowitz & Kirsch 2001]

- Asynchronous tasks in **Esterel**: [Paris 1992]

- Automatic distribution in **Signal**: [Maffeis 1993],
  [Aubry, Le Guernic, Machard 1996],
  [Benveniste, Caillaud & Le Guernic 2000]

- Distributed implementation of **Lustre** over **TTA**:
  [Caspi, Curic, Maignan, Sofronis, Tripakis & Niebert 2003]
Conclusion and future research

This new distribution method:

- is implemented in the ocrep tool:
  
  [Link](http://www.inrialpes.fr/pop-art/people/girault/Ocrep)

- works equally well with Lustre and Esterel programs

- allows the writing and compiling of synchronous programs with long duration tasks

Some future plans:

- To adapt this method to Decade programs in order to obtain code mobility

  Decade is a dynamic higher-order synchronous data-flow programming language [Colaço et al 2004]