Static Scheduling of Latency Insensitive Designs
with Lucy-n

Louis Mandel  
LRI, Université Paris-Sud 11  
INRIA Paris-Rocquencourt

Florence Plateau  
LRI, Université Paris-Sud 11  
Presently at Prove & Run

Marc Pouzet  
DI, École Normale Supérieure  
INRIA Paris-Rocquencourt

FMCAD 2011
# Flows and Clocks

\[
x \xrightarrow{w} w = \text{clock}(x)
\]

<table>
<thead>
<tr>
<th>x</th>
<th>2</th>
<th>5</th>
<th>3</th>
<th>7</th>
<th>9</th>
<th>4</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>w = \text{clock}(x)</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>
Sampling

\[
\begin{array}{c|cccccc}
  x & 2 & 5 & 3 & 7 & 9 & \ldots \\
  w_2 & 1 & 0 & 1 & 1 & 0 & \ldots \\
  x \text{ when } w_2 & 2 & 3 & 7 & \ldots \\
  \text{clock}(x \text{ when } w_2) & 1 & 0 & 0 & 1 & 0 & 1 & 0 & \ldots \\
\end{array}
\]

\[
\text{clock}(x \text{ when } w_2) = \text{clock}(x) \text{ on } w_2
\]

Definition:

\[
\begin{align*}
0w_1 \text{ on } w_2 & \overset{\text{def}}{=} 0(w_1 \text{ on } w_2) \\
1w_1 \text{ on } 1w_2 & \overset{\text{def}}{=} 1(w_1 \text{ on } w_2) \\
1w_1 \text{ on } 0w_2 & \overset{\text{def}}{=} 0(w_1 \text{ on } w_2)
\end{align*}
\]
Composition

\[ x \]
\[ w \]
\[ y \]
\[ w \]

\[ + \]
\[ z \]
\[ w \]

\[
\begin{array}{ccccccc}
  x & 2 & 5 & 3 & 7 & 9 & 4 & 6 \\
  y & 5 & 3 & 2 & 2 & 0 & 2 & 1 \\
  z = x + y & 7 & 8 & 5 & 9 & 9 & 6 & 7 \\
\end{array}
\]

\[ clock(x) = clock(y) = clock(z) \]
### Composition

\[ z = x + y \]

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>5</th>
<th>3</th>
<th>7</th>
<th>9</th>
<th>4</th>
<th>6</th>
<th>...</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>5</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>...</td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>...</td>
</tr>
</tbody>
</table>

**Diagram:**
- Inputs: \( x, w, y, w' \)
- Output: \( z \)
- Calculation: \( z = x + y \)
Communication through a bounded buffer:

- **the input’s clock** must be adaptable to **the output’s clock**

\[ w_1 <: w_2 \]

Adaptability relation:

- **Precedence**: writings must occur before readings
- **Synchronizability**: writings and readings must have the same rate
let node plus_plus (x,y) = o where
  rec z = x + y
  and t = z when (10)
  and t' = buffer(t)
  and r = y when (01)
  and o = t' + r
let node plus_plus (x,y) = o where
  rec z = x + y
  and t = z when (10)
  and t' = buffer(t)
  and r = y when (01)
  and o = t' + r

val plus_plus : (int * int) -> int
val plus_plus :: forall 'a. ('a * 'a) -> 'a on (01)

Buffer line 7, characters 11-21: size = 1
Application to Latency Insensitive Designs
Latency Insensitive Design [Carloni et al. 2001]

Method used to design synchronous circuits that tolerate data transfer latency

- design synchronous IPs and interconnect them
  - at each instant, each IP is activated
  - at each activation, an IP consumes a token on each input and produces a token on each output
  - data transfer between each IP takes one instant

- add relay stations on the wires and shell wrappers around IPs
  - relay-station = split a wire into two pieces
  - shell wrapper = buffers on inputs + a controller to activate the IP

Question: when do IPs have to be activated by their controller?
Scheduling Latency Insensitive Design

Existing answers:

- **elastic circuits dynamic schedule** [Carloni et al. 2001, Carmona et al. 2009]:
  - every wire is transformed into a channel carrying data and control bits
  - the wrappers dynamically decide activation of IPs by analysing control bits and applying an ASAP strategy
  - a back pressure protocol must be used to avoid buffer overflows

- **static schedule** [Casu et al. 2004, Boucaron et al. 2007, Carmona et al. 2009]:
  - computation of an explicit schedule
  - avoids additional control paths and runtime overhead of dynamic schedule
  - maximizes rate (by computing sufficient buffer sizes)
  - minimizes buffer sizes (by choosing other strategies than ASAP)
Modeling Latency Insensitive Designs with Lucy-n

Wire

\[
\begin{align*}
\text{delay} & \quad \text{delay x} \\
\text{0(1) on w} & \\
\end{align*}
\]

Relay station

\[
\begin{align*}
\text{relay} & \quad \text{relay x} \\
w & \\
\end{align*}
\]

Shell wrapper

\[
\begin{align*}
\text{IP} & \\
\text{with } w_1 \ll: w \text{ and } w_2 \ll: w \\
\end{align*}
\]
Example: composition of \texttt{ip\_A} and \texttt{ip\_B}

\begin{center}
\includegraphics[width=\textwidth]{example_diagram.png}
\end{center}

\textit{Schedule computed by the compiler}

\texttt{val ip\_AB :: forall 'a. 'a on (10) \rightarrow 'a on (01)
Example: composition of \texttt{ip\_A} and \texttt{ip\_B}

\begin{itemize}
\item Schedule computed by the compiler
\end{itemize}

\begin{verbatim}
val ip\_AB :: forall 'a. 'a on (10) -> 'a on (01)
\end{verbatim}

Better throughput obtained with the help of the user (option \texttt{-nbones 2}):

\begin{verbatim}
val ip\_AB :: forall 'a. 'a on (110) -> 'a on (011)
\end{verbatim}
Schedule computed by the compiler

\[
\text{val ip\textunderscore AAB} :: \text{forall } \, \text{'a}. \, \text{'a on (1100)} \rightarrow \text{'a on 0001(1001)}
\]

The Lucy-n compiler can schedule IPs that do not necessarily consume a token on each input and produce a token on each output at each activation.
MPEG-2 video encoder [Carloni et al. 2002, Casu et al. 2004]