Encoding Latency Insensitive Design in Lucy-n: first experiments

Louis Mandel Florence Plateau
Parkas Team

École Normale Supérieure
Université Paris-Sud 11
INRIA

Synchron 2010 Workshop, Fréjus – 01/12/2010

Latency Insensitive Design [CMSV01]

Method used to design synchronous circuits that tolerate data transfer latency

- design synchronous IPs and interconnect them (Figure 1)
  - 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
- add relay stations on the wires and shell wrappers around IPs (Figure 2)
  - relay-station $=$ non initialized register
  - shell wrapper $=$ buffers on inputs $+$ a controller to activate the IP
  - Question: when do IPs have to be activated ?

Scheduling Latency Insensitive Design [CMSV01]

Existing answers:

- elastic circuits dynamic schedule [KCKO06]:
  - 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 backpressure protocol must be used to avoid buffer overflows
- k-periodic static schedule [BdSM07]:
  - computation of an explicit schedule
  - avoids additionnal control pathes and runtime overhead of dynamic schedule
  - maximizes rate (by computing sufficient buffer sizes)
  - minimizes buffer sizes (by choosing other strategies than ASAP)

Scheduling Latency Insensitive Design with Lucy-n

Idea inspired by [BdSM07] (Figure 3) :

- encode the circuit with latencies in a Lucy-n progam
- get schedules and buffer sizes from clock typing

Scheduling Latency Insensitive Design with Lucy-n

Encoding a synchronous circuit (`synchronous_circuit.ls`):

- an IP is a node of type $\forall\alpha.\ (\alpha \times \ldots \times \alpha) \rightarrow (\alpha \times \ldots \times \alpha)$
- an unitialized register is encoded by a `strict_buffer`
  - Example:

    ```
    let node f x = strict_buffer (x when (100))

    val f :: forall 'a. 'a -> 'a on (010)
    ```

  - if w_1 is the clock of the input of `strict_buffer` and w_2 the clock of its output, they are subject to a strict precedence constraint:
    $\forall j,\ \mathcal{I}_{w_1}(j) < \mathcal{I}_{w_2}(j)$.
- an initialised register is encoded by a merge and a `strict_buffer`

  ```
  x' = (merge 1(0) init (strict_buffer x))
  ```

Scheduling Latency Insensitive Design with Lucy-n

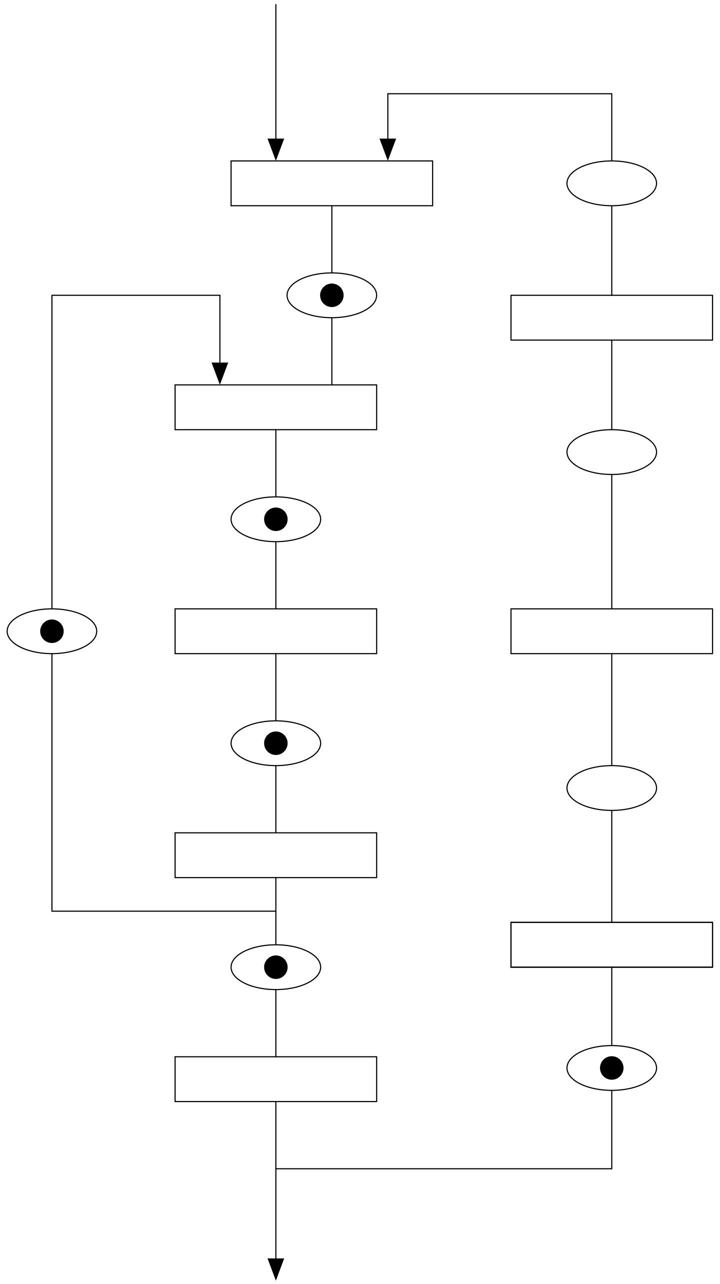
Encoding a circuit with latencies (`circuit_with_latencies.ls`):

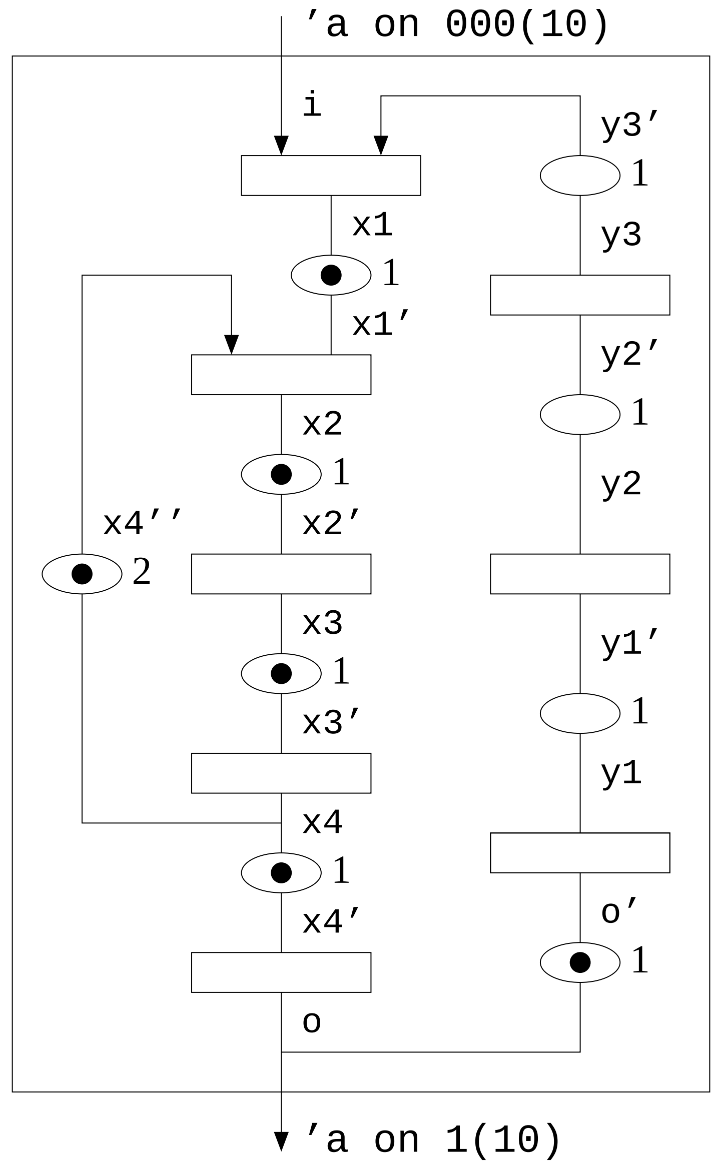
- buffers on input wires are encoded with the `buffer` primitive
- relay-stations are encoded with the `strict_buffer` primitive

Comparision with Aoste team's work

- They translate: LID $\rightarrow$ Marked Graphs $\rightarrow$ synchronous model
- We translate: LID $\rightarrow$ n-synchronous model
- Remark that we can translate: Marked Graphs $\rightarrow$ n-synchronous model

Jean-Vivien Millo Example [Mil08]





```
let node f i = o where
  rec x1 = ip2(i, y3')
  and x1' = merge 1(0) true (strict_buffer x1)
  and x2 = ip2(x4'', x1')
  and x2' = merge 1(0) true (strict_buffer x2)
  and x3 = ip1(x2')
  and x3' = merge 1(0) true (strict_buffer x3)
  and x4 = ip1(x3')
  and x4' = merge 1(0) true (strict_buffer x4)
  and x4'' = merge 1(0) true (strict_buffer x4)
  and o = ip1(x4')
  and o' = merge 1(0) true (strict_buffer o)
  and y1 = ip1(o')
  and y1' = strict_buffer y1
  and y2 = ip1(y1')
  and y2' = strict_buffer y2
  and y3 = ip1(y2')
  and y3' = strict_buffer y3
```

Contribution of our method about scheduling LID (temporary conclusion)

- "Quality" of schedules

  Rate : Boucaron's algorithm [Bou07] finds schedules with faster rates. We do not find these schedules because we search for the schedule with the shortest periodic pattern (in term of number of $1s$).
  If we help the typer by asking him to search for a longer pattern, we find comparable results w.r.t. the rate.

  Buffer sizes: Millo's algorithm [Mil08] minimizes buffer sizes (for a fixed rate) by searching for balanced schedules, with a semi-automatic method (prefixes of schedules must be found by hand). We are not able to find such schedules.
  If we ask the compiler to minimize buffer sizes on the considered example, we find schedules needing as few buffer places as Millo, but these places are more often occupied.

- Modularity: Our method allows to compose in a latency insensitive way statically scheduled LIDs .

References

[BdSM07] Julien Boucaron, Robert de Simone, and Jean-Vivien Millo. Formal methods for scheduling of latency-insensitive designs. *EURASIP Journal on Embedded Systems*, Issue 1(ISSN:1687-3955):8 – 8, January 2007.

[Bou07] Julien Boucaron. *Modélisation formelle de systèmes Insensibles à la Latence et ordonnancement*. PhD thesis, Université de Nice Sophia-Antipolis, 2007.

[CMSV01] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli. Theory of latency-insensitive design. *IEEE Trans. on CAD of Integrated Circuits and Systems*, 20(9):1059–1076, 2001.

[KCKO06] S. Krstic, J. Cortadella, M. Kishinevsky, and J. O'Leary. Synchronous elastic networks. In *Proceedings of the Formal Methods in Computer Aided Design*, 2006.

[Mil08] Jean-Vivien Millo. *Ordonnancements périodiques dans les réseaux de processus : Application à la conception insensible aux latences*. PhD thesis, Université de Nice-Sophia Antipolis, Décembre 2008.