Static Scheduling of Latency Insensitive Designs with Lucy-n

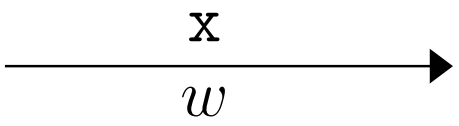
Louis Mandel
LRI, Université Paris-Sud 11
INRIA Paris-Rocquencourt

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

Marc Pouzet
DI, École Normale Supérieure
INRIA Paris-Rocquencourt

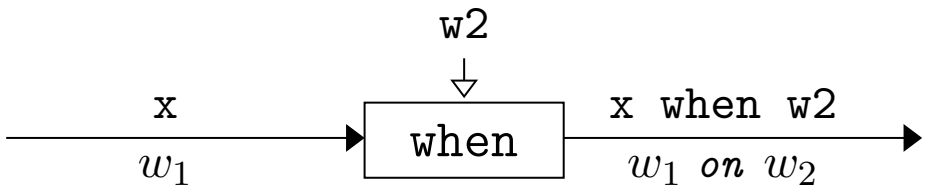
FMCAD 2011

Flows and Clocks



x	2	5		3		7	9	4			6	...
$w = clock(\mathtt{x})$	1	1	0	1	0	1	1	1	0	0	1	...

Sampling



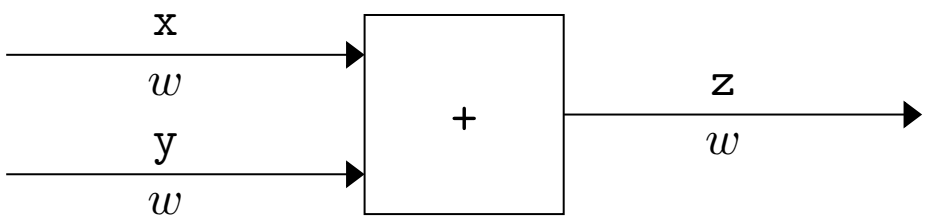
x	2	5		3		7	9	...
w2	1	0		1		1	0	...
x when w2	2			3		7		...
clock(x when w2)	1	0	0	1	0	1	0	...

$$\mathit{clock}(\texttt{x when w2}) = \mathit{clock}(\texttt{x}) \ \mathit{on}\ w_2$$

Definition:

$$\begin{aligned} 0w_1 \ \mathit{on}\ w_2 &\stackrel{\mathit{def}}{=} 0(w_1 \ \mathit{on}\ w_2) \\ 1w_1 \ \mathit{on}\ 1w_2 &\stackrel{\mathit{def}}{=} 1(w_1 \ \mathit{on}\ w_2) \\ 1w_1 \ \mathit{on}\ 0w_2 &\stackrel{\mathit{def}}{=} 0(w_1 \ \mathit{on}\ w_2) \end{aligned}$$

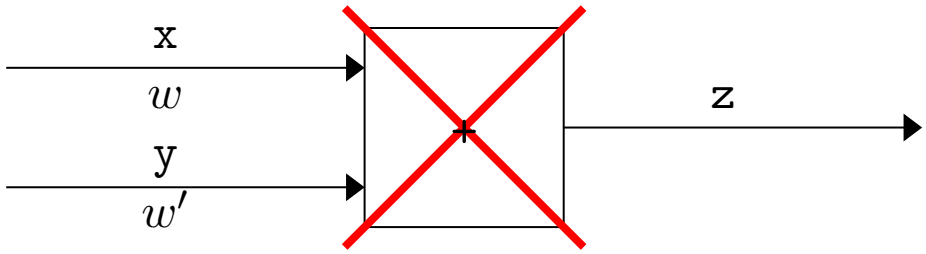
Composition



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	...

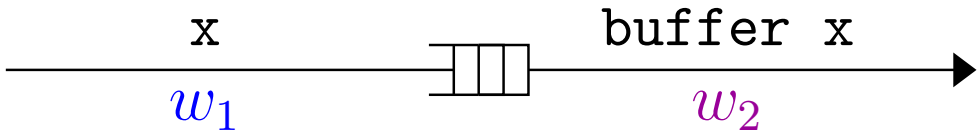
$$clock(\mathrm{x}) = clock(\mathrm{y}) = clock(\mathrm{z})$$

Composition



x	2	5		3		7	9	4			6	...
y	5		3		2	2	0			2	1	...
z = x + y												

Buffering



Communication through a bounded buffer:

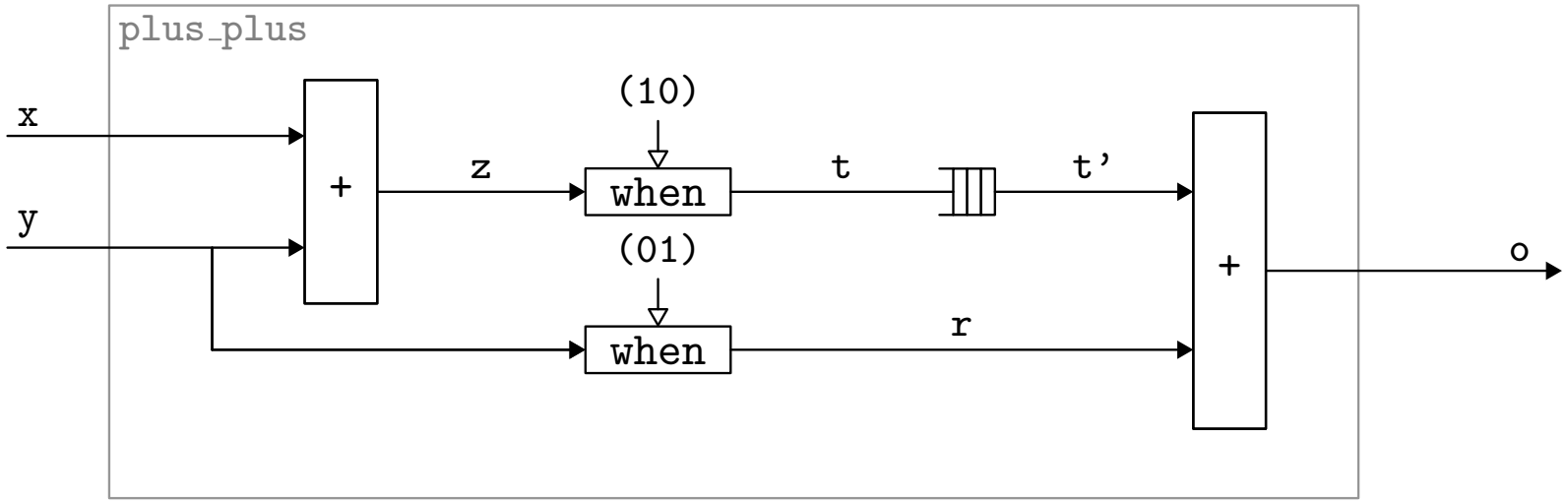
the input's clock must be **adaptable** to the output's clock

$$w_1 \quad <: \quad w_2$$

Adaptability relation:

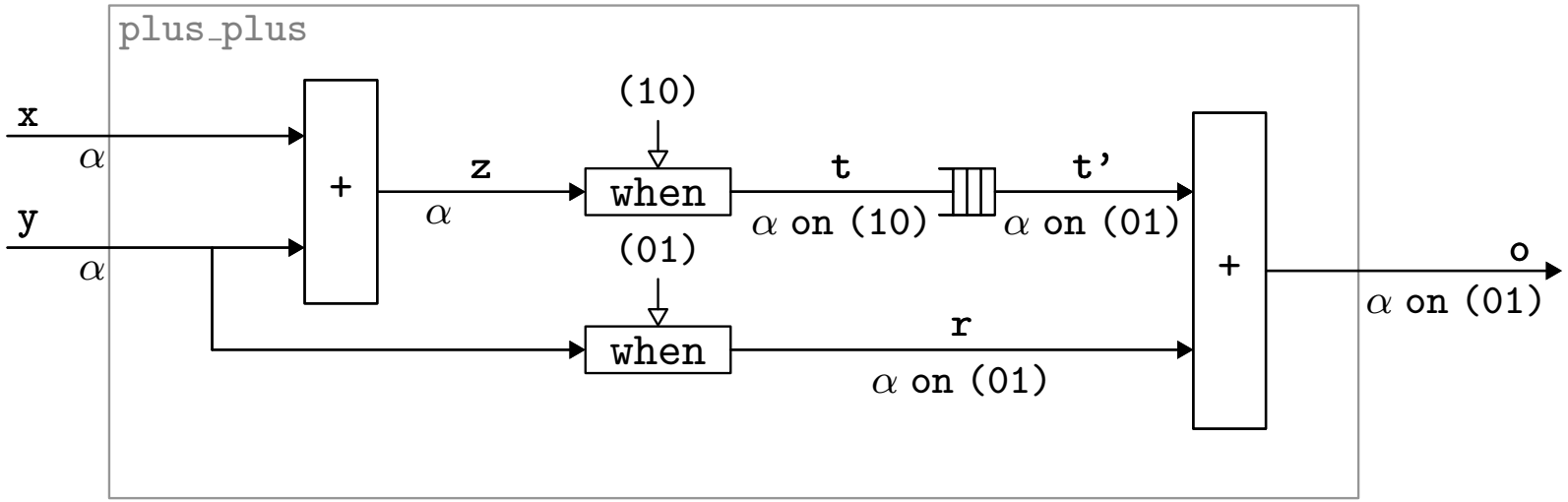
- Precedence: writings must occur before readings
- Synchronizability: writings and readings must have the same rate

Typing




```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when (10)
7    and t'= buffer(t)
8    and r = y when (01)
9    and o = t' + r
```

Typing




```
4  let node plus_plus (x,y) = o where
5    rec z = x + y
6    and t = z when (10)
7    and t'= buffer(t)
8    and r = y when (01)
9    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 ?

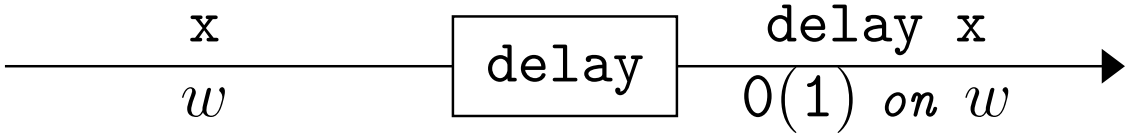
Scheduling Latency Insensitive Design

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 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)

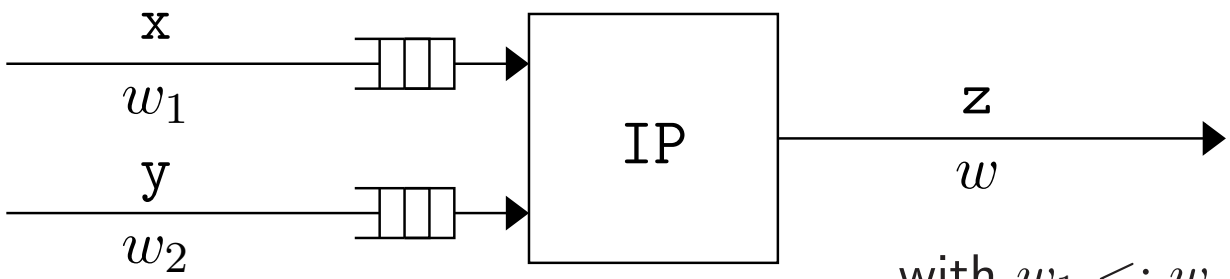
Modeling Latency Insensitive Designs with Lucy-n

Wire



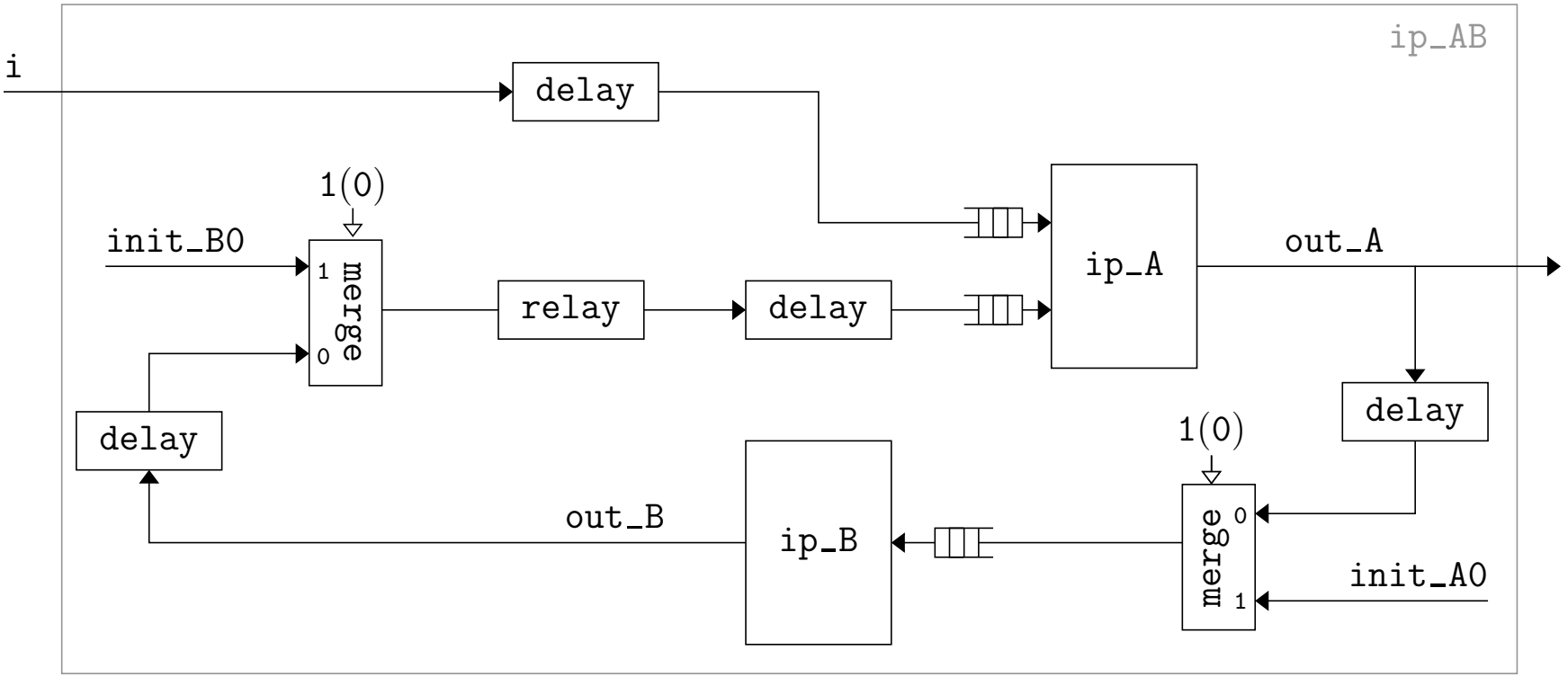
Relay station

Shell wrapper



with $w_1 <: w$ and $w_2 <: w$

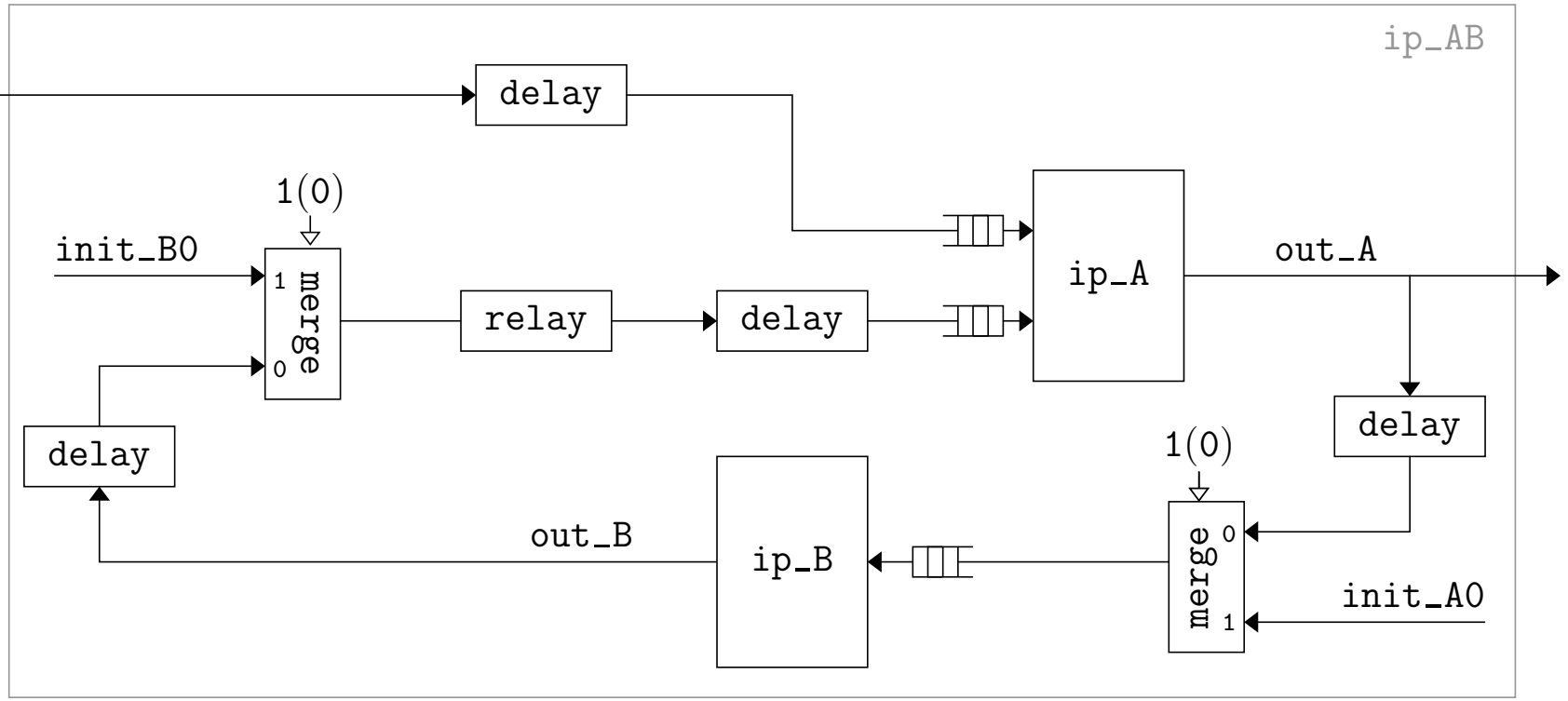
Example: composition of ip_A and ip_B



Schedule computed by the compiler

```
val ip_AB :: forall 'a. 'a on (10) -> 'a on (01)
```

Example: composition of `ip_A` and `ip_B`



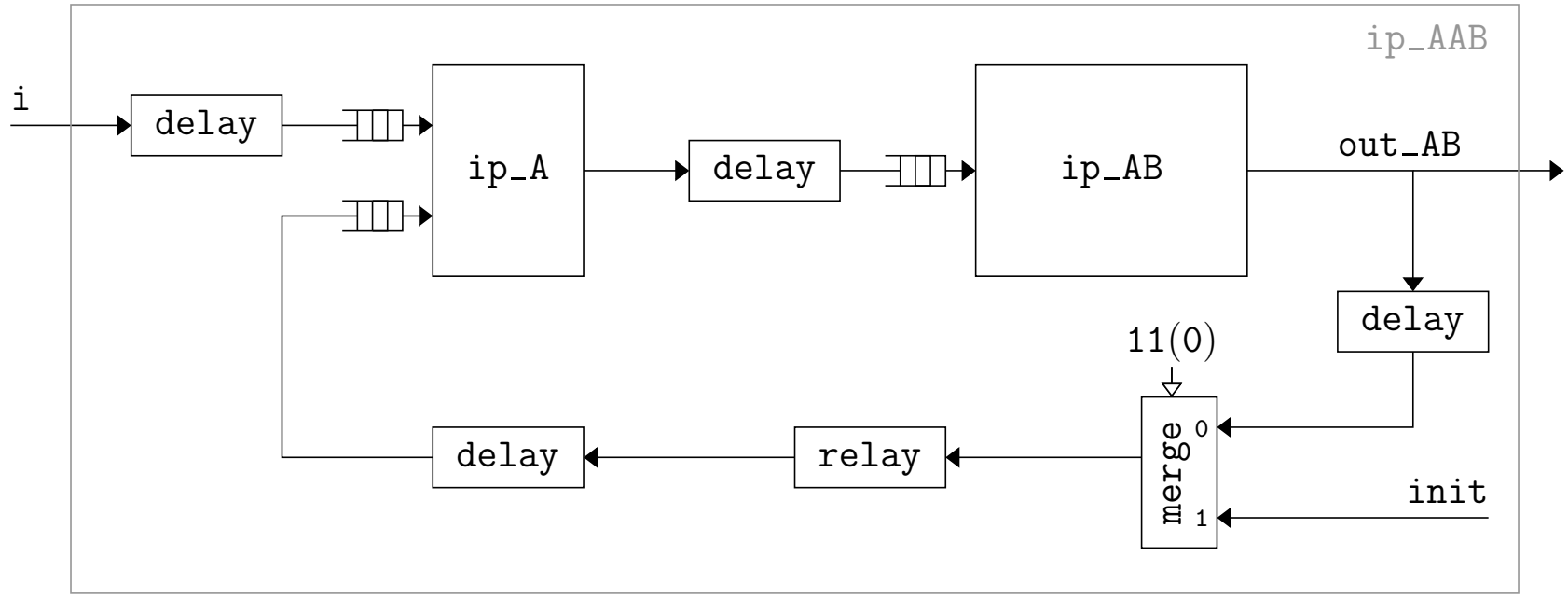
Schedule computed by the compiler

```
val ip_AB :: forall 'a. 'a on (10) -> 'a on (01)
```

Better throughput obtained with the help of the user (option `-nbones 2`):

```
val ip_AB :: forall 'a. 'a on (110) -> 'a on (011)
```

Composition of Statically Scheduled IPs



Schedule computed by the compiler

```
val ip_AAB :: forall 'a. 'a on (1100) -> '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]