Automatic Production of Globally Asynchronous Locally Synchronous Systems

Alain GIRAULT **INRIA Rhône-Alpes**

and

Clément MÉNIER **ENS Lyon**

Outline

1. GALS Systems
2. Related Work
3. Program model
4. Our method in details
5. Perspectives

GALS Systems

Acronym for "Globally Asynchronous Locally Synchronous"

In software : paradigm for composing blocks and making them communicate asynchronously

➥ Used in embedded systems

In hardware : circuits designed as sets of synchronous blocks communicating asynchronously

➥ No need to distribute the clock $\Longrightarrow$ saves power

Our goal : automatically obtain GALS systems from a centralised program

Automatically Distributed Programs

Why distribute ?

➥ physical constraints, fault-tolerance, performance...

Advantages of automatic distribution :

- less error-prone than by hand
- possibility to debug & validate before distribution
- ...

Related Work

The closest is *Berry & Sentovich'2000* :
"Implementation of constructive synchronous circuits as a network of CFSMs in POLIS"

Main differences with our work :

1. Partitioning of the circuit into N clusters is by hand
   Our partitioning is automatic

2. They partition the circuit, that is the control part
   We partition the data part and replicate the control part

Program Model : Synchronous Circuits

Program = synchronous sequential circuit driving a table of actions

A control part and a data part :

- Control part = synchronous sequential boolean circuit
- Data part = table of external actions
  - ➥ manipulate inputs, outputs, and typed variables (integers, reals...)

A program has a set of input and output signals

Signals can be pure or valued

Valued signals are associated to a local typed variable

It can be obtained from Esterel $\Longrightarrow$ so called SC internal format

VHDL code can be generated from it

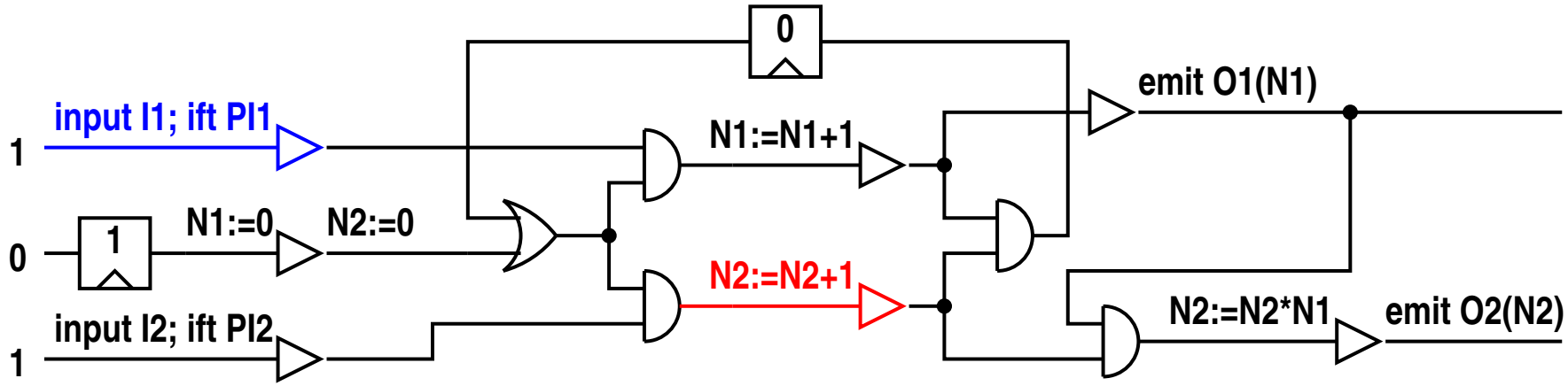
Program Model : Properties

The control structure is :

- Parallel : there are several control paths
- Implicit : the state is coded in the registers
- Dynamic : the control depends on the data

Important property : any given variable can only be modified in one parallel branch (same as in Esterel)

An Example : FOO



input I1 ; ift PI1
input I2 ; ift PI2
N1 := 0

N2 := N2 + 1
N2 := N2 * N1
emit O1(N1)

N2 := 0
emit O2(N2)
N1 := N1 + 1

Distribution Method

1. Design a centralised system

2. Compile it into a single synchronous circuit

3. Distribute it into N communicating synchronous circuits

We focus here on the point 3 : the automatic distribution

Distribution Specifications

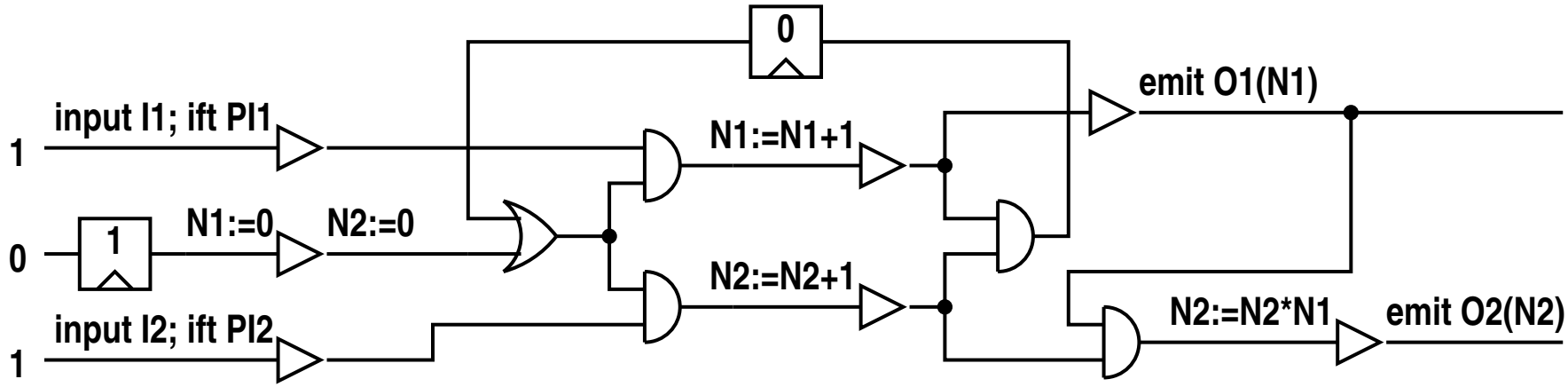
Must be provided by the user :

- The desired number of computing locations

- The localisation of each input and output

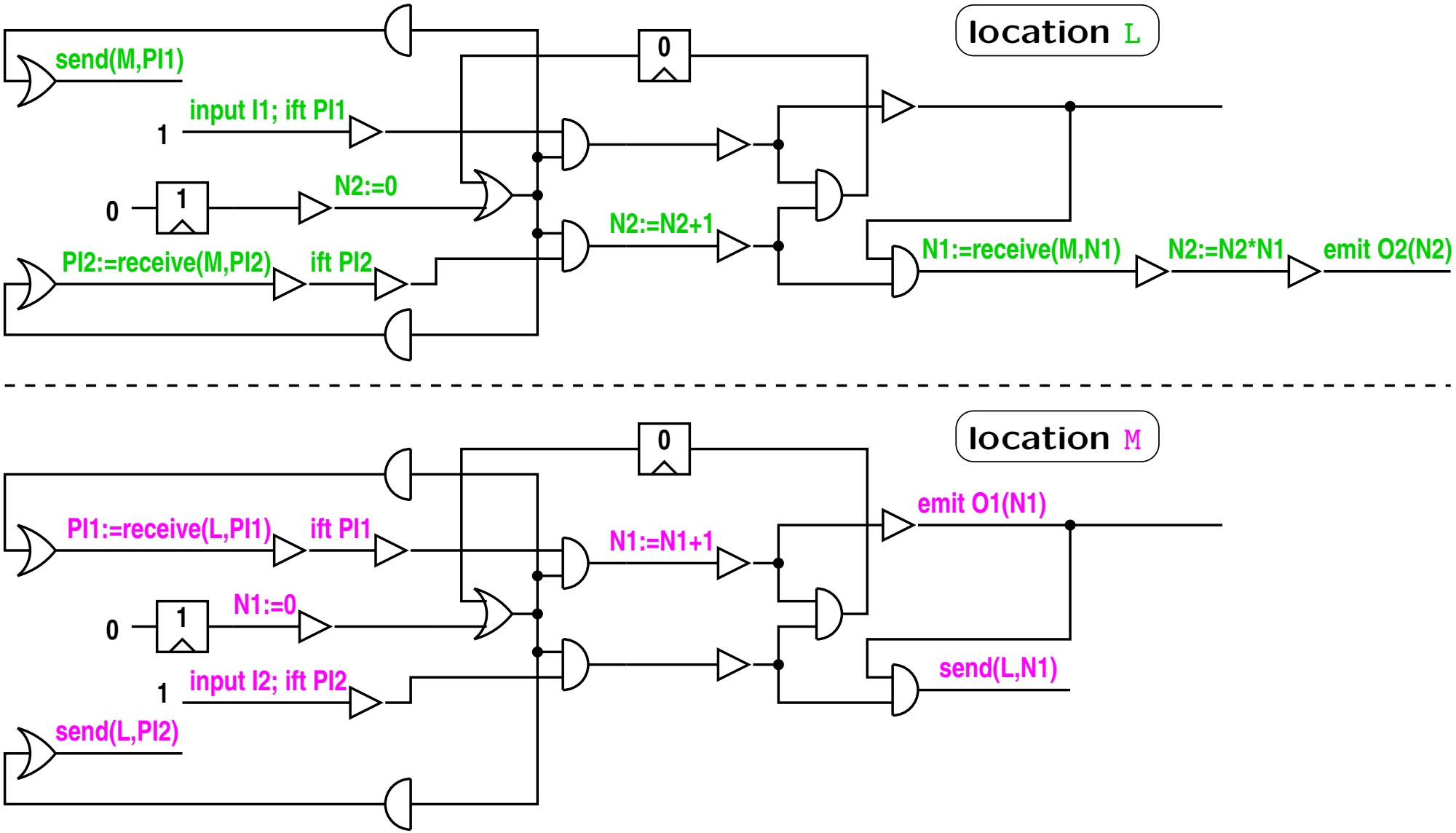
  ➥ derived from the physical localisation of the sensors and actuators

Distribution Specifications of FOO



location L	location M
I1,O2	I2,O1

Where are we Heading ?



Distribution Algorithm : Principle

Based on past work : *Caspi, Girault, & Pilaud'1999*

➥ Replicate the control part and partition the data part

1. Localise each action to get N virtual circuits
2. Solve the distant variables problem for each virtual circuit
3. Project each virtual circuit to get one actual circuit
4. Solve the distant inputs problem

We obtain N circuits communicating harmoniously
➥ without inter-blocking and with the same functional behaviour

Communication Primitives

Asynchronous communications

➥ Two FIFO queues associated with each pair of locations and each variable

➥ Each queue is identified by a triplet $\langle \texttt{src}, \texttt{var}, \texttt{dst} \rangle$

Two communication primitives :

- On location `src` : `send(dst,var)` non blocking
- On location `dst` : `var:=receive(src,var)` blocking when empty

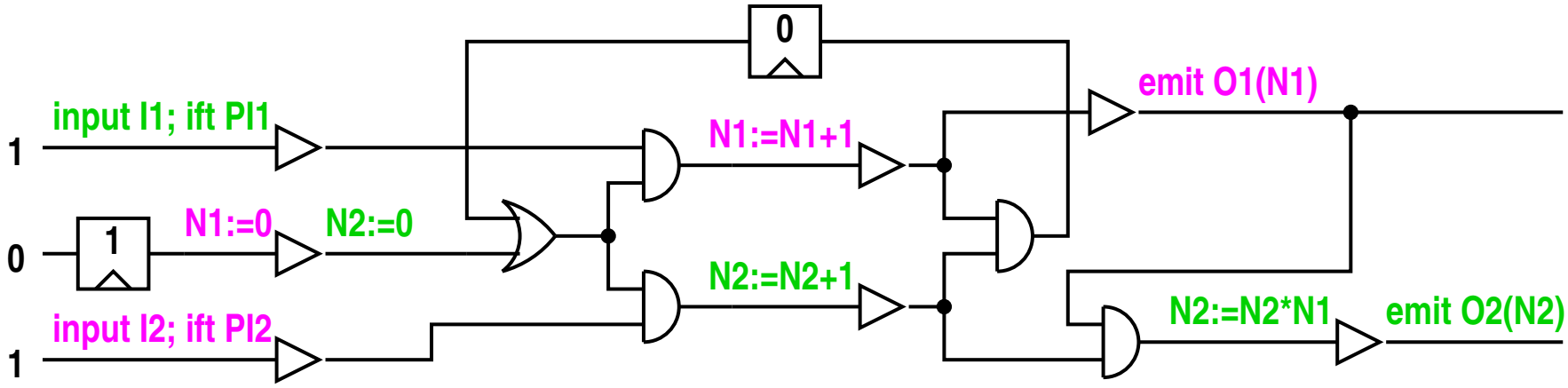
Localisation of the Actions

Only the data part is partitioned : the control part is replicated

loc.	action
L	`input I1 ; ift PI1`
M	`input I2 ; ift PI2`
M	`N1 := 0`

loc.	action
L	`N2 := N2 + 1`
L	`N2 := N2 * N1`
M	`emit O1(N1)`

loc.	action
L	`N2 := 0`
L	`emit O2(N2)`
M	`N1 := N1 + 1`



Two Problems to Solve

1. Distant variables problem :

➥ Not computed locally

➥ We add `send` and `receive` actions

2. Distant inputs problem :

➥ Not received locally

But : input signals convey two informations : value and presence

And : an `ift` net is required to modify the control flow according to the input's presence

➥ We add input simulation blocks

Solving the Distant Variables Problem

We apply a simple algorithm to solve the data dependencies to each buffered path (sequential path) :

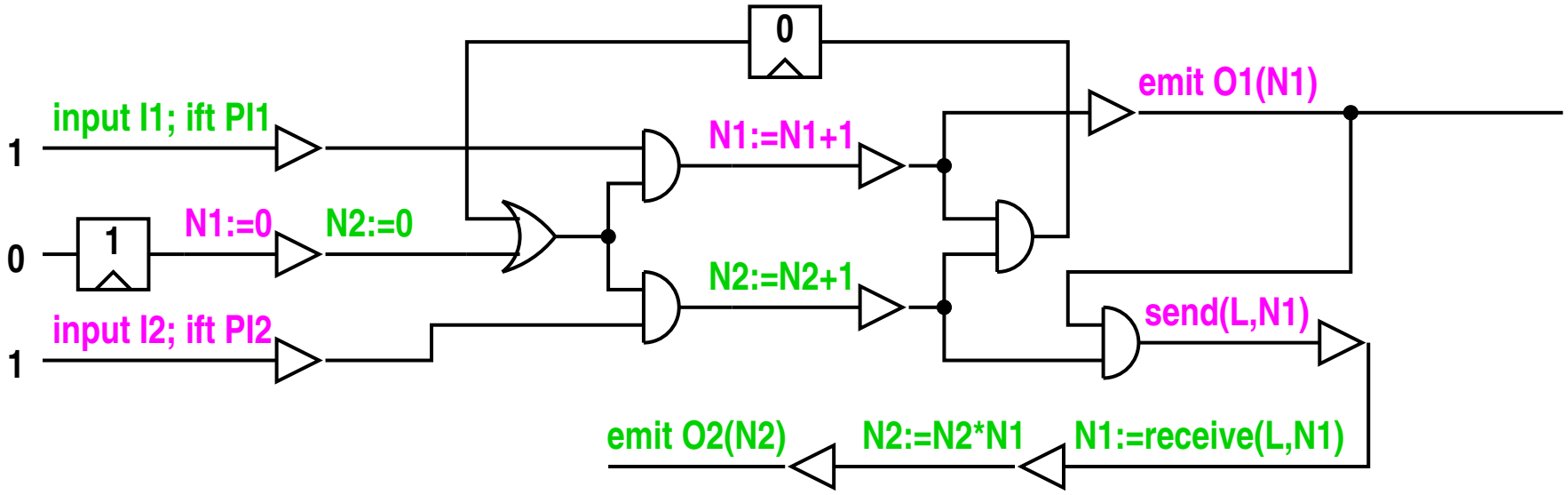
1. Isolate a buffered path and mark its root and tail

2. Insert the `send` actions in the buffered path
   ➥ Traverse the path backward to insert the `send` actions asap

3. Insert the `receive` actions in the buffered path
   ➥ Traverse the path forward to insert the `receive` actions alap

4. Proceed to the unmarked successor nets of the tail

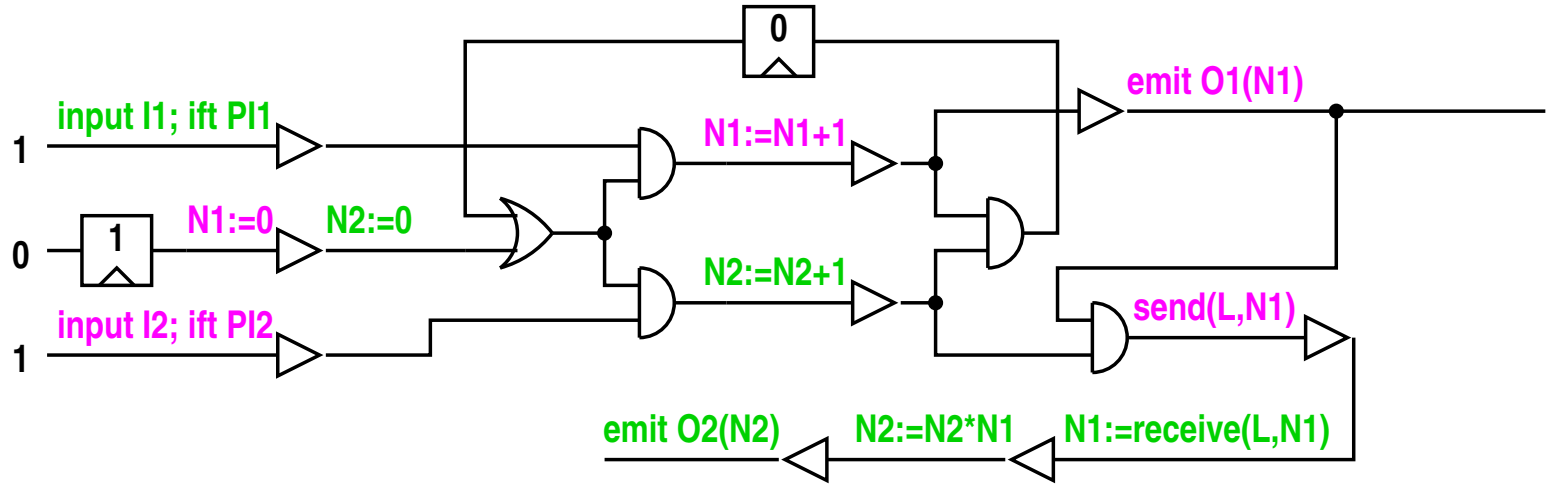
Partial Result for FOO



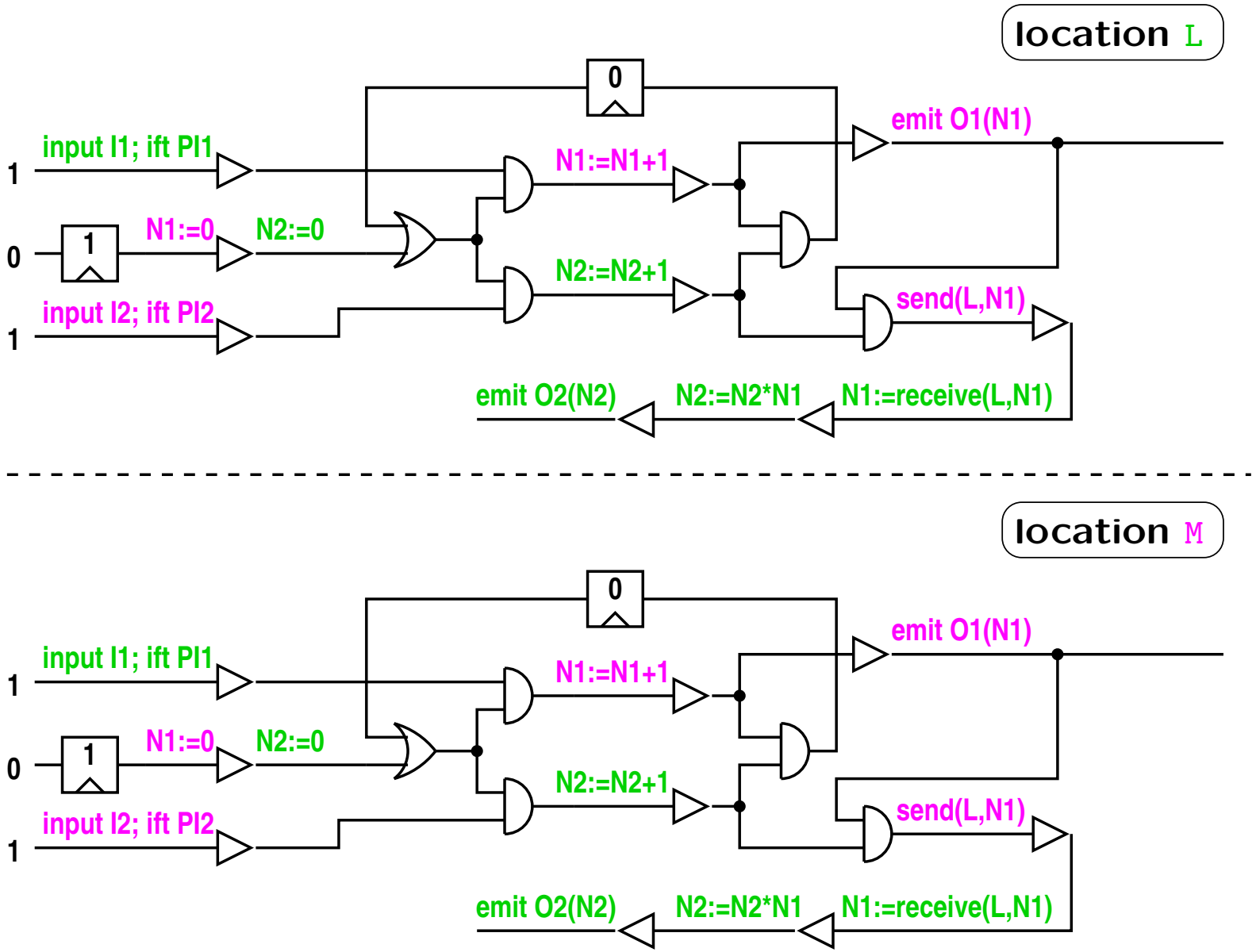
This is still one circuit representing two virtual circuits

The next step is to project onto two actual circuits

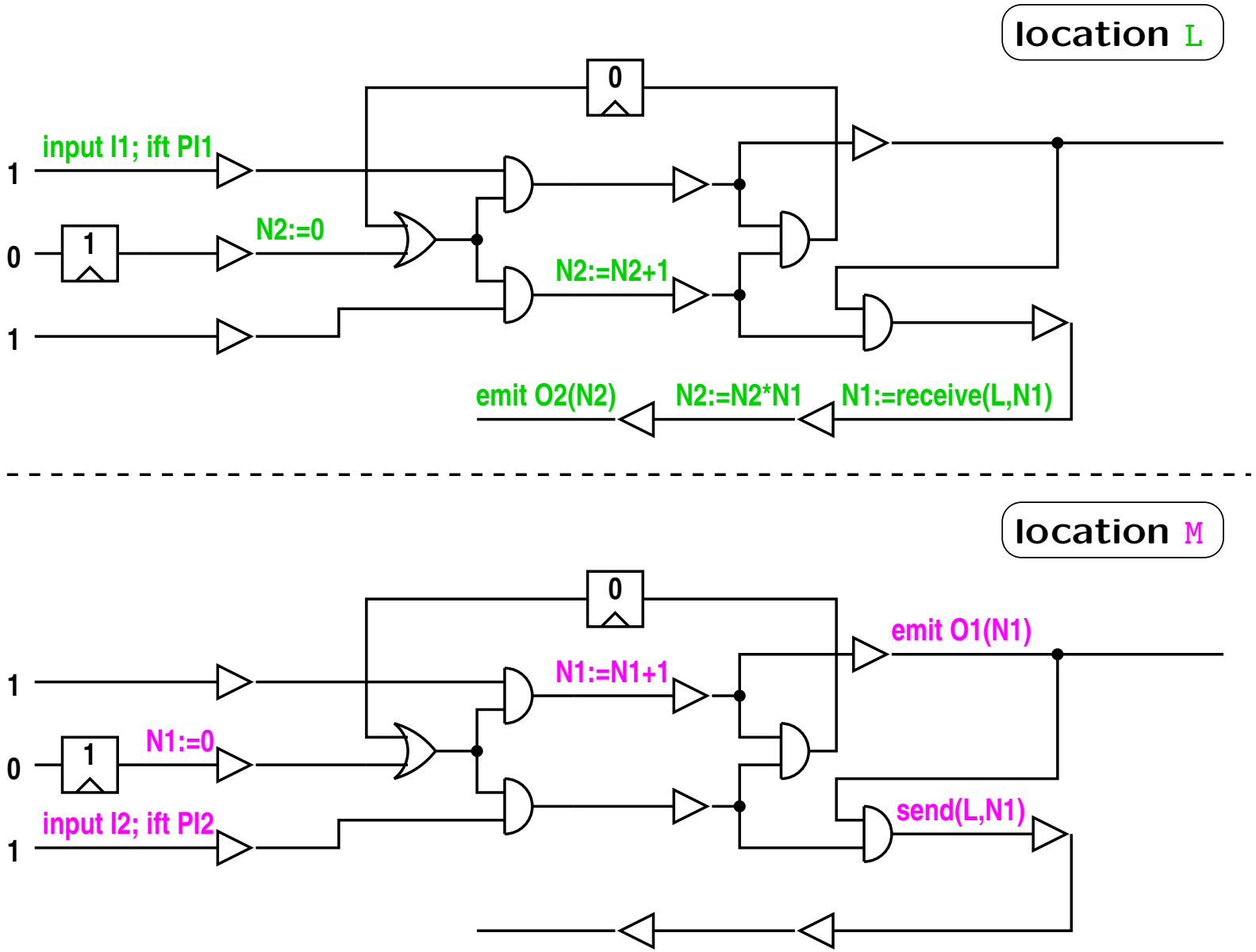
Projection for FOO



Projection for FOO



Projection for FOO



Solving the Distant Inputs Problem

Reminder : input signals convey two informations : the value and the presence

And : an `ift` net is required to modify the control flow according to the input's presence

Our goal is to send the presence information only to those computing locations that need them :

1. Detect the impure input-dependent nets and their needed inputs

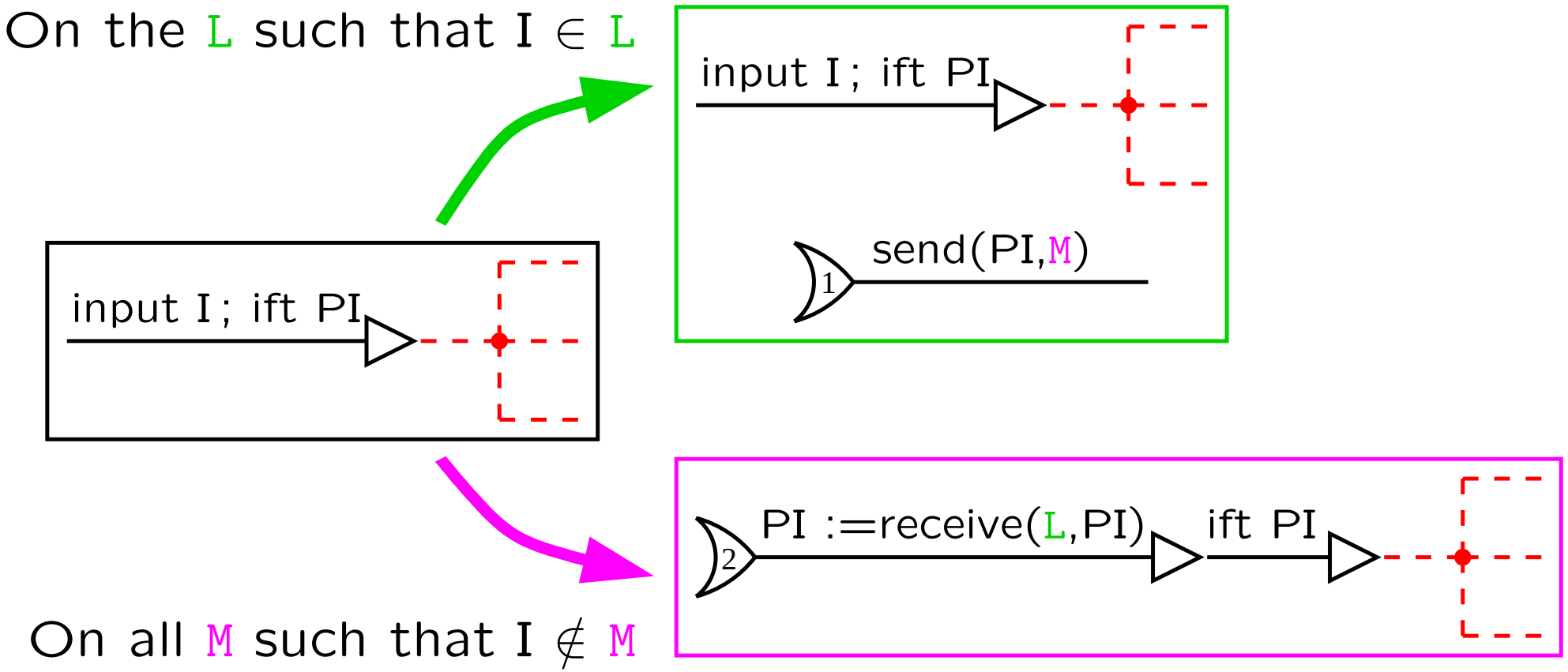
   ➥ Circuit traversal to compute for each net the set

   `Input` = {`needed inputs`}

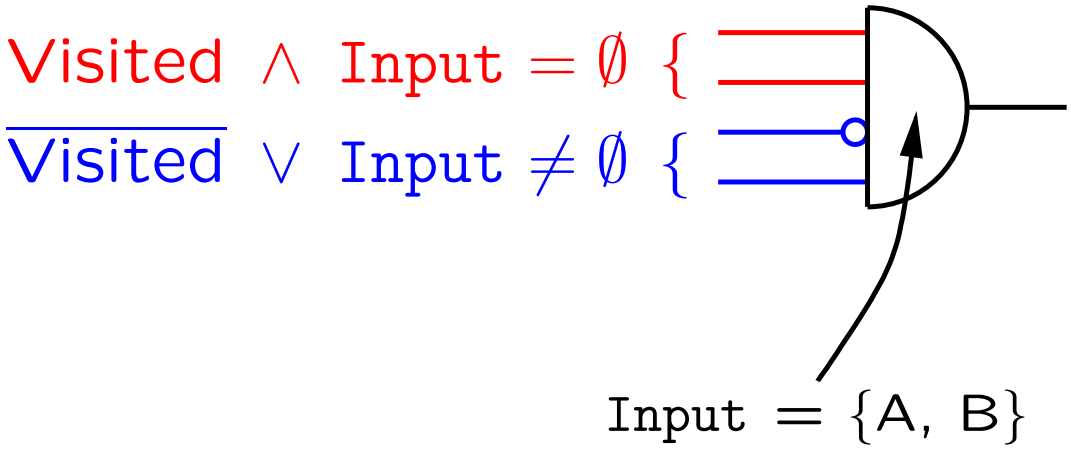
2. Create the simulation blocks for the input nets

3. Connect the nets detected at step 1 to the required simulation blocks

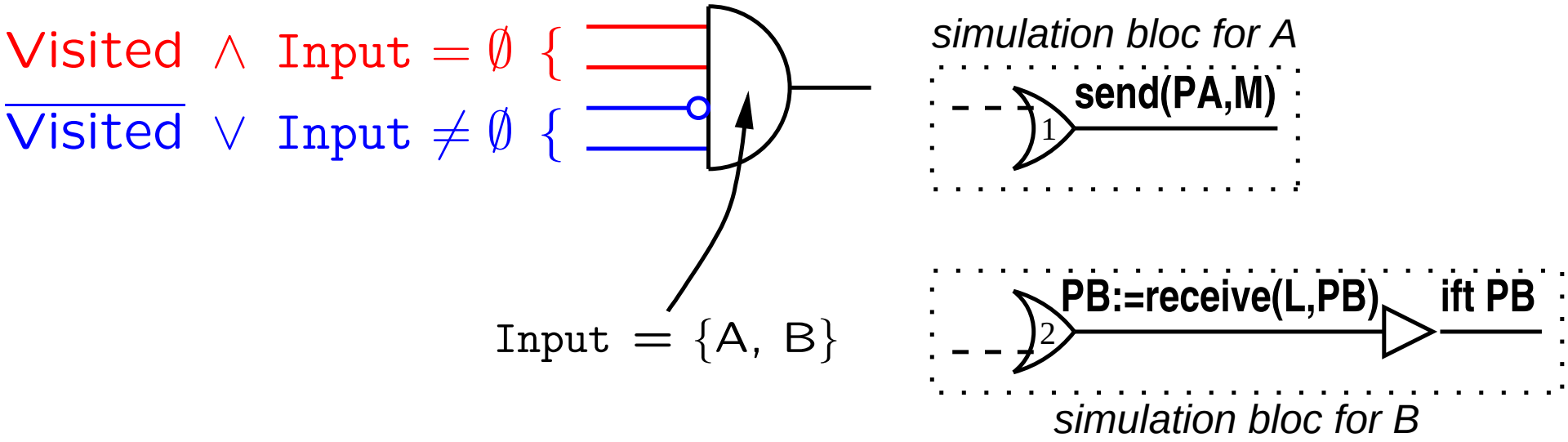
Creation of the Input Simulation Blocks



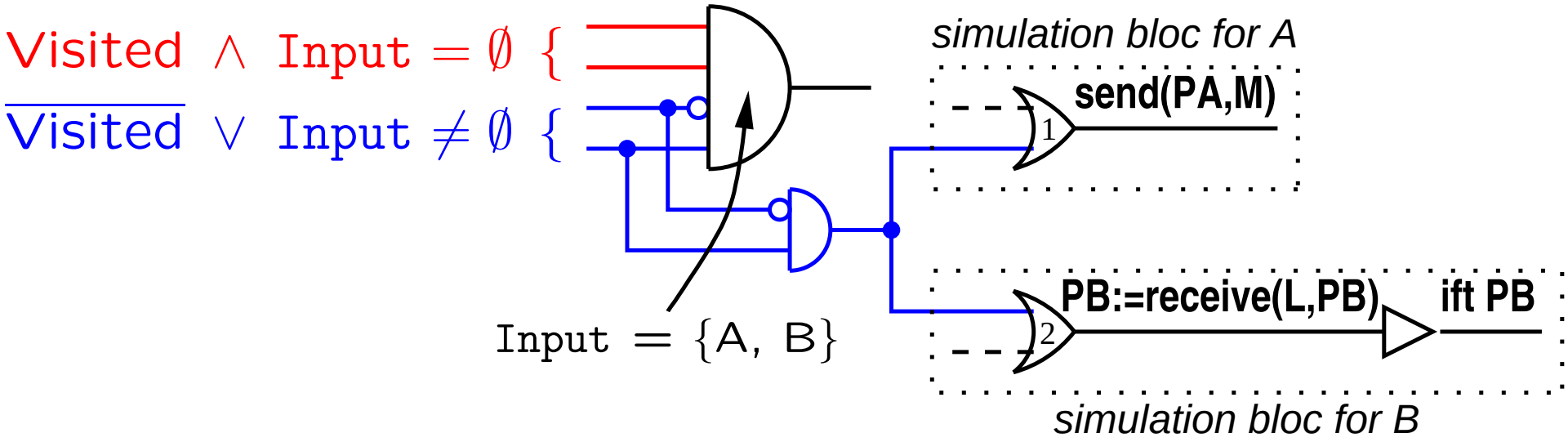
Connection of the Input Simulation Blocks



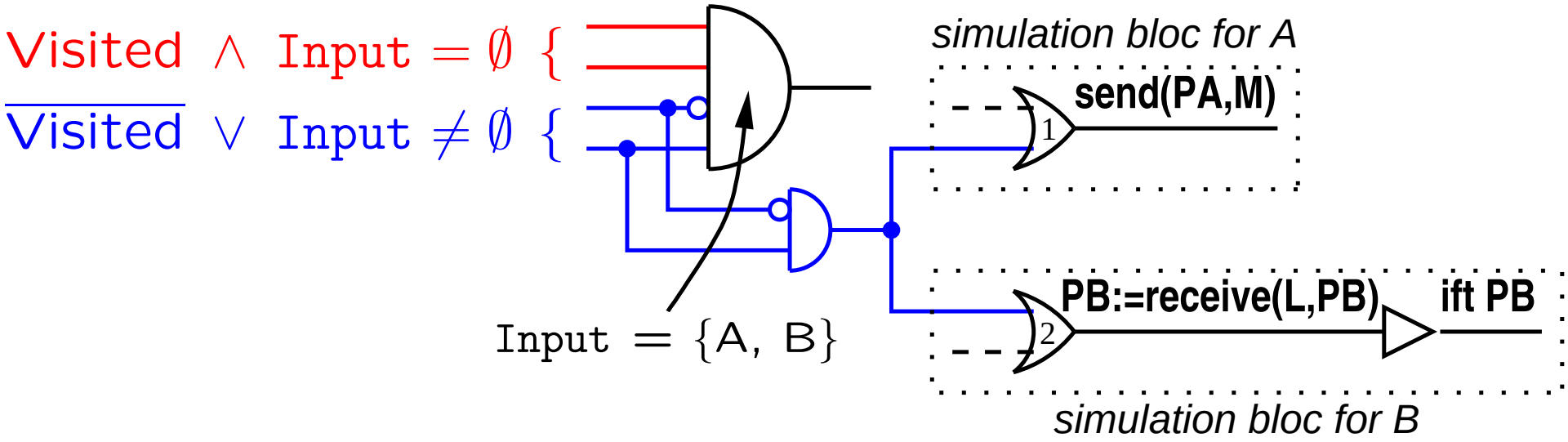
Connection of the Input Simulation Blocks



Connection of the Input Simulation Blocks

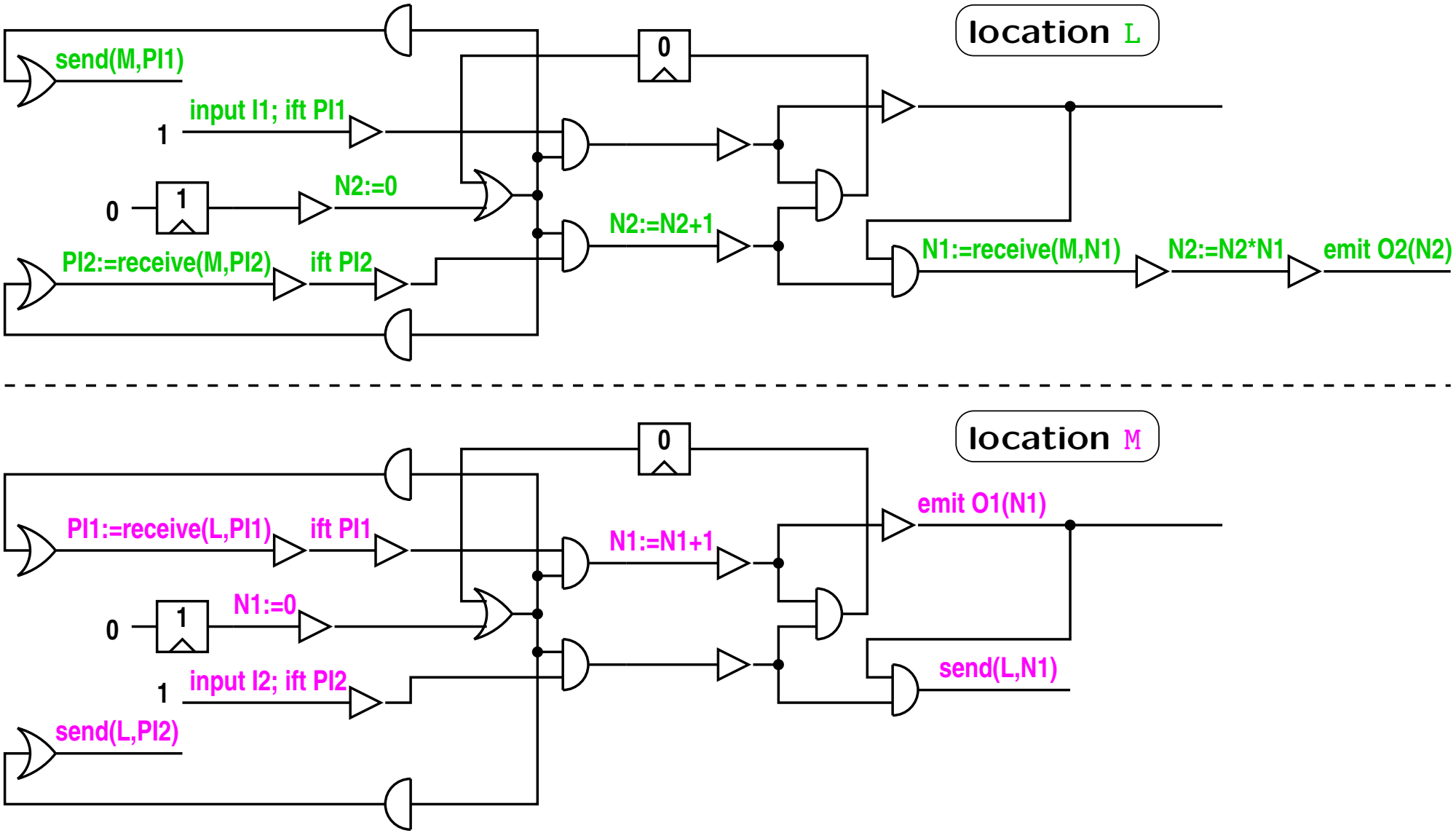


Connection of the Input Simulation Blocks



Connection of an OR gate is similar

Final Result for FOO



Conclusions and Future Research

This methods works and a tool exists : `screp`

➥ `http://www.inrialpes.fr/bip/people/girault/Screp`

Only interesting if the data part is big (because the control part is replicated)

Short-term perspectives : hardware/software codesign, post-distribution optimisations, ...

The most interesting perspective is to mix this approach with *Berry & Sentovich'2000* :

- Accepting as inputs cyclic constructive circuits
- Automatic partitioning of the circuit into N clusters
- Partitioning both the data part and the control part