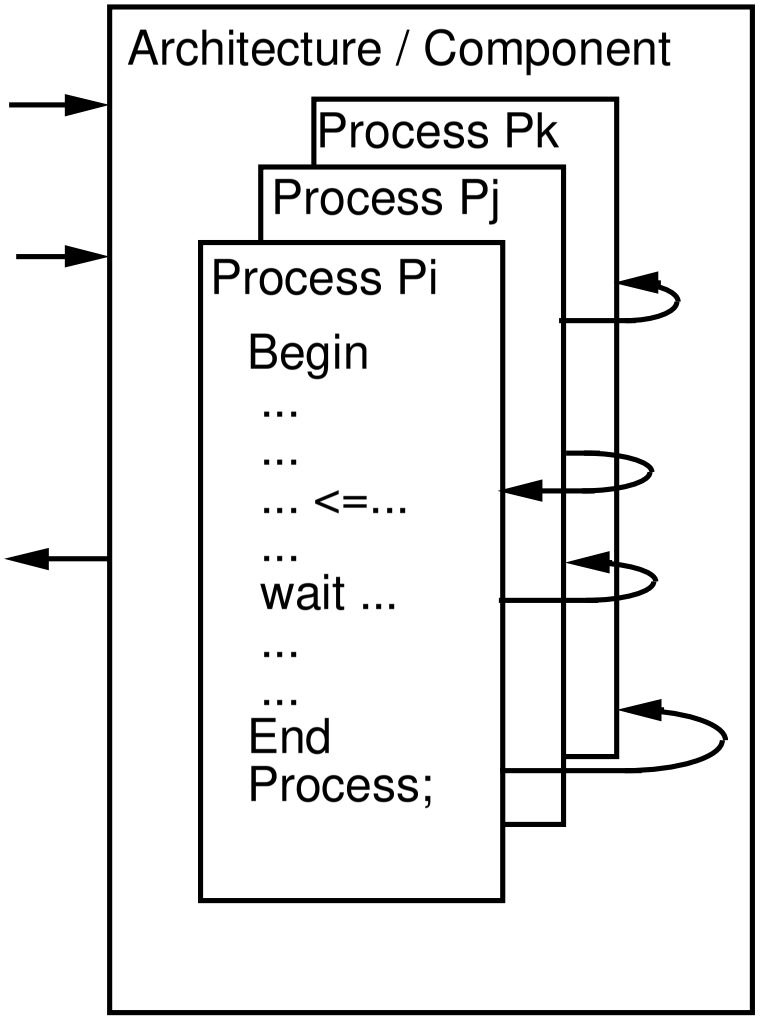
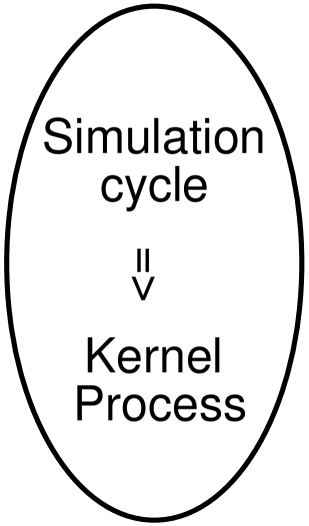
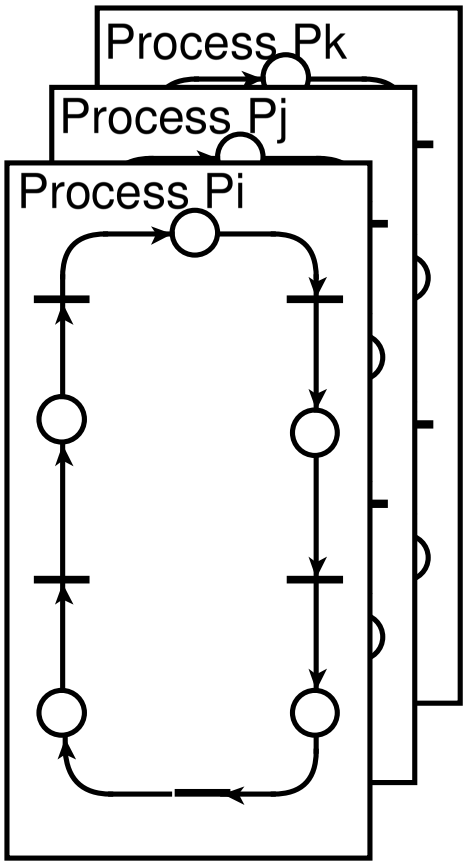
VHSIC Hardware Description Language



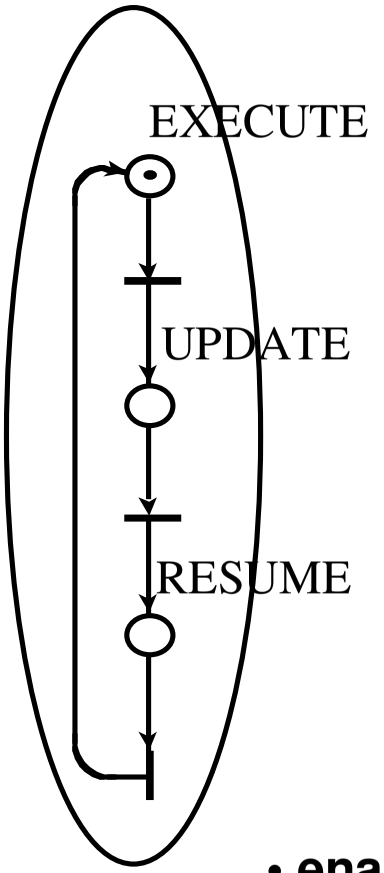


- **IEEE Standard**
- **Simulation Oriented**
- **Tools Avaiability**
- **Description Styles**

Petri Nets



Scheduler

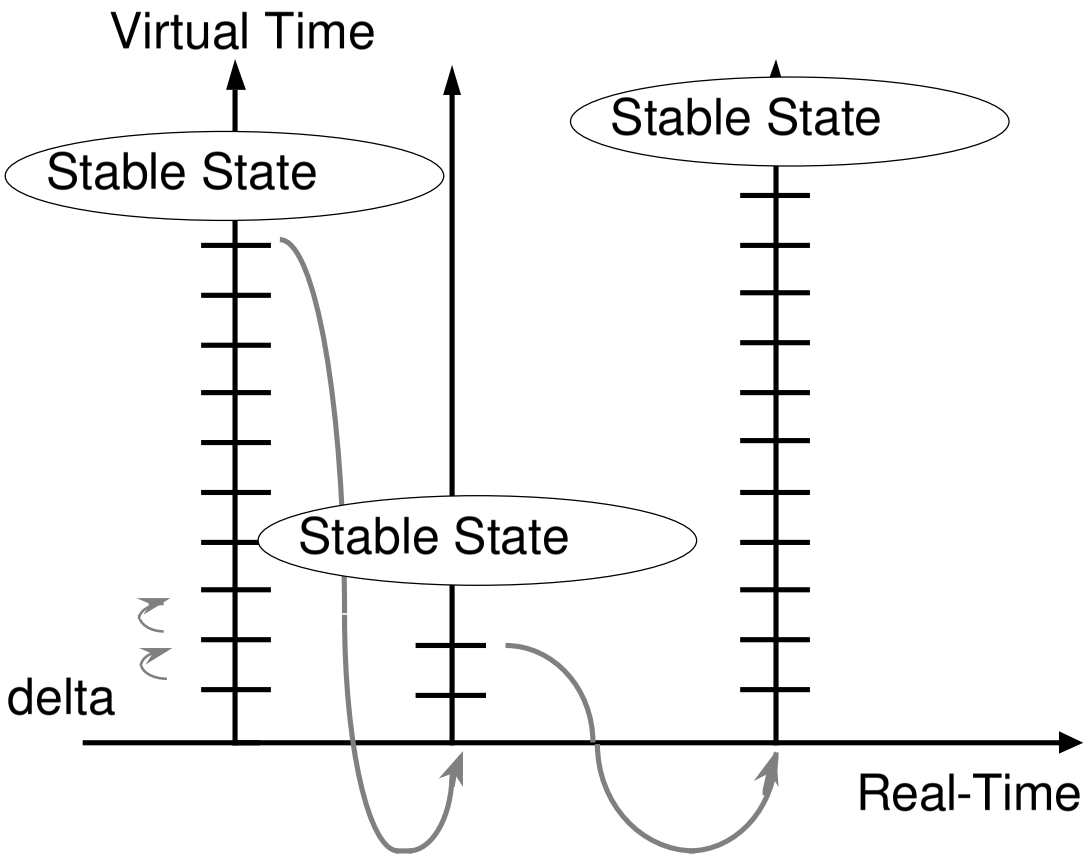


INTERACTION :

associated to each transition

- **delay** $T(t)$
- **guard** $g(t)$
- **enabling phase** *EXE and RES*
- **transformations** $TRF(t) = \{(d, trfd, t)\}$

VHDL Simulation



variable $\mathcal{V}ar = \cup\ var_Pi$

signal $\mathcal{E}ff = \cup\ eff_sig$

$\mathcal{D}rv = \cup\ drv_sig$

$\mathcal{T}rs = \cup\ trs_sig$

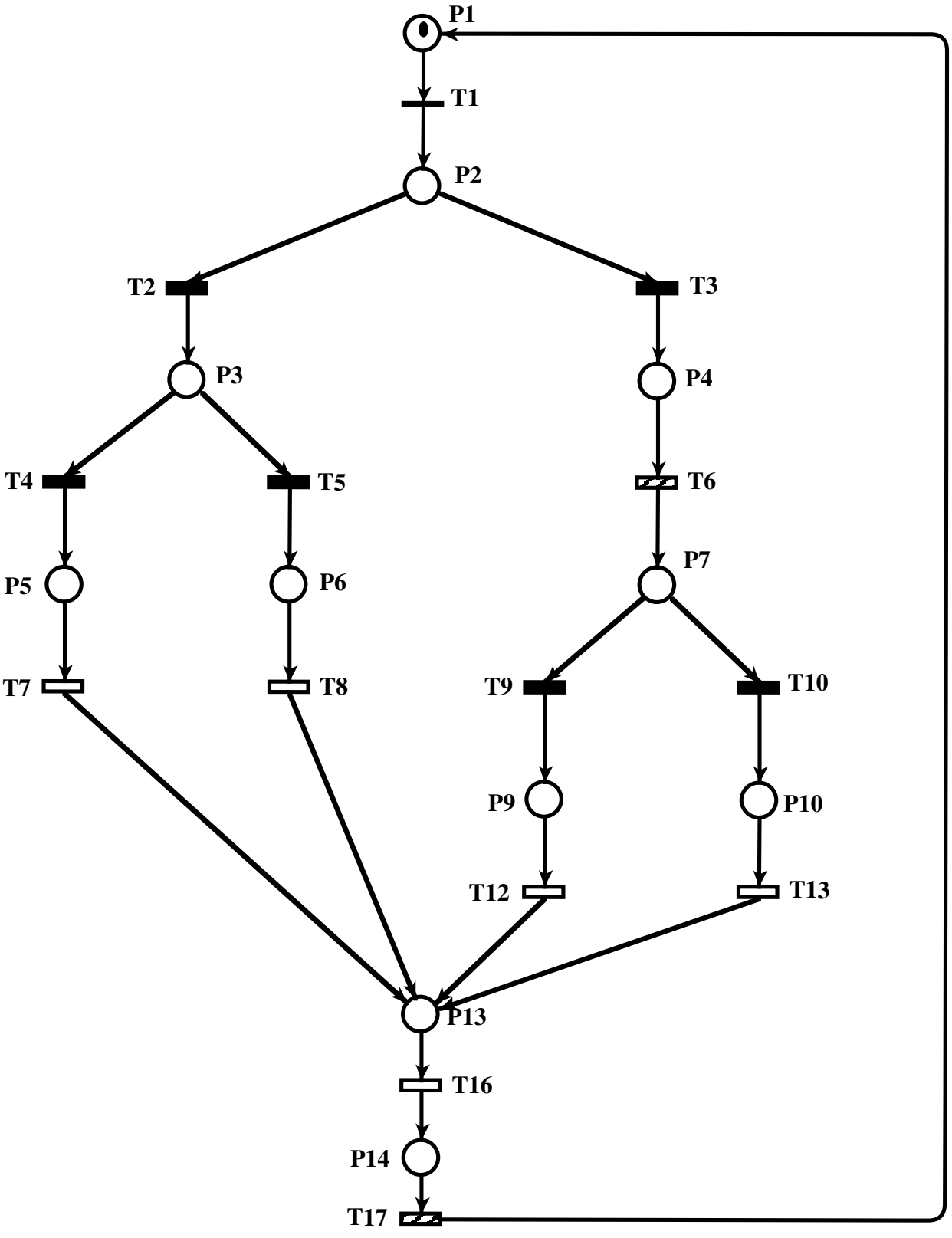
$\mathcal{E}vt = \cup\ evt_sig$

VHDL Simulation

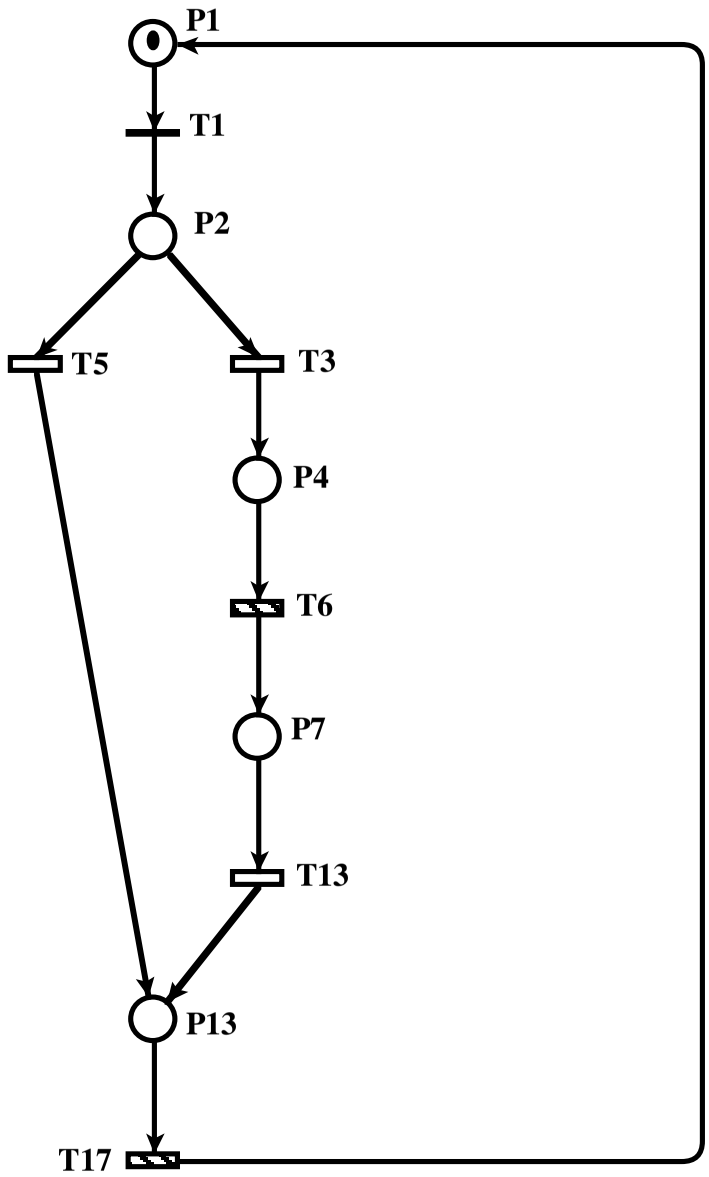
A VHDL PROCESS MONITORING ACCES TO MEMORY UNIT

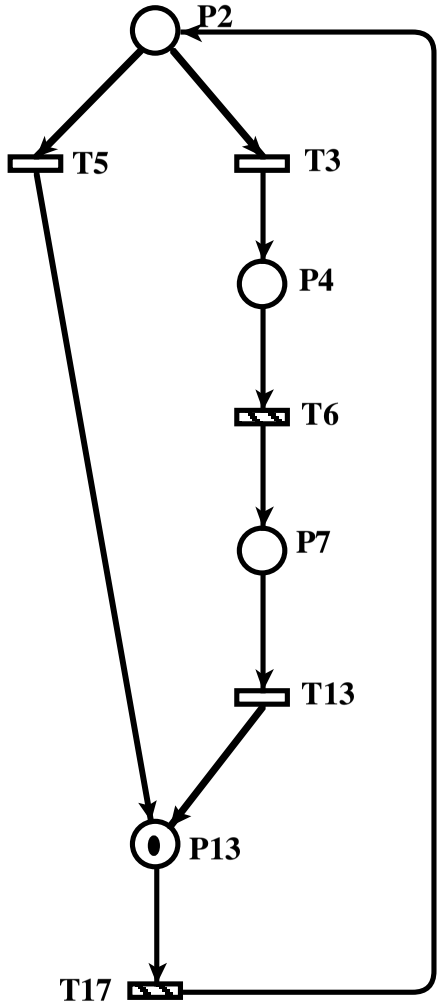
```
MEM_ACCES_MONITOR : process
 variable LAST_ACCES : CLIENT;
begin
 if ( TS_REQUEST = '1' and IQ_REQUEST = '1' ) then
 wait on MEM_ACK until MEM_ACK ='1';-- conflict
  if (LAST_ACCESS = IQ ) then
     LAST_ACESS := TS;
  else
     LAST_ACESS := IQ;
  end if;
  elsif TS_REQUEST = '1' then
     LAST_ACCESS := TS;
   else
     LAST_ACCESS := IQ;
 end if;
 MEM_ACCESS <= LAST_ACCESS;
 wait until (TS_REQUEST ='1' or IQ_REQUEST = '1');

end process MEM_ACCES_MONITOR ;
```

Reduction of Immediate Transitions



Reduced Petri Net





DEFINITIONS

A STATE *S* OF A VHDL PROGRAM

- $\mathcal{P} = \cup \mathcal{P}i$ **Ordinal Counter of each Process Pi**
- $\phi\ Kernel \in \{EXECUTE, UPDATE, RESUME\}$ **the phase of the Kernel Process**
- **Data read and modified by the program**

variable $\mathcal{V}ar = \cup\ var_\mathcal{P}i$

signal $\mathcal{E}ff = \cup\ eff_sig$

$\mathcal{D}rv = \cup\ drv_sig$

$\mathcal{T}rs = \cup\ trs_sig$

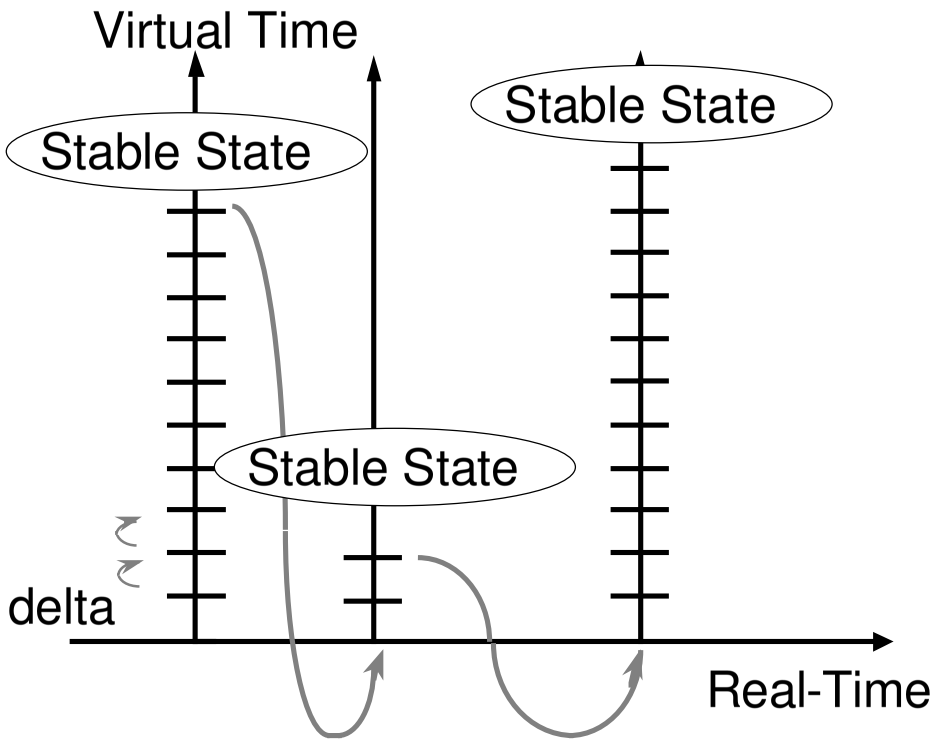
$\mathcal{E}vt = \cup\ evt_sig$

INITIAL STATE *S0*

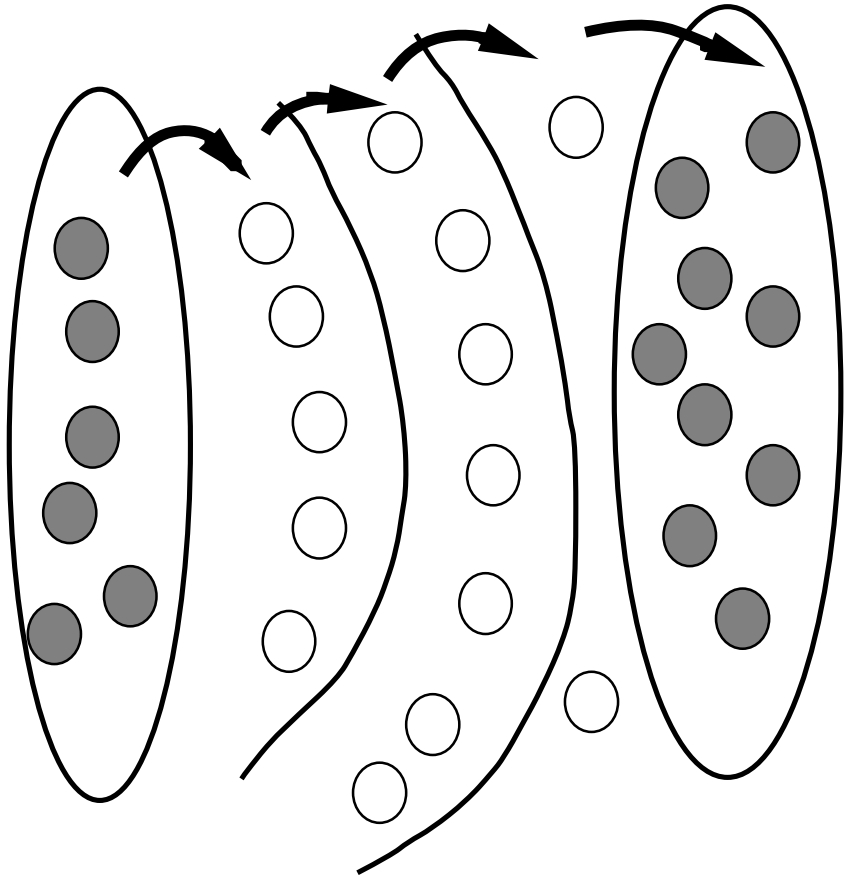
END OF SIMULATION CYCLE STATE : *ESS*

STABLE STATE *SS*

FORWARD REACHABILITY ANALYSIS



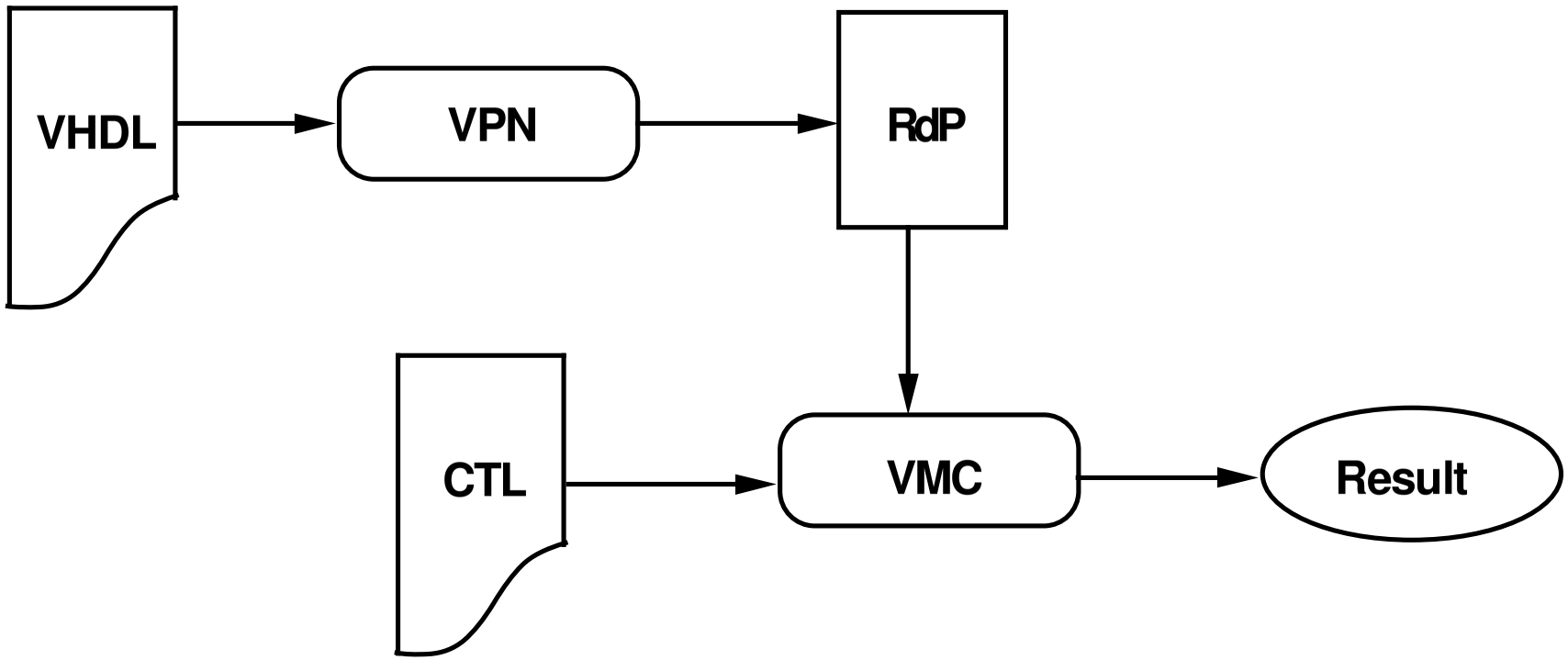
VHDL Simulation



Forward Traversal

2 nested loops =>
Delta cycles up to stabilisation

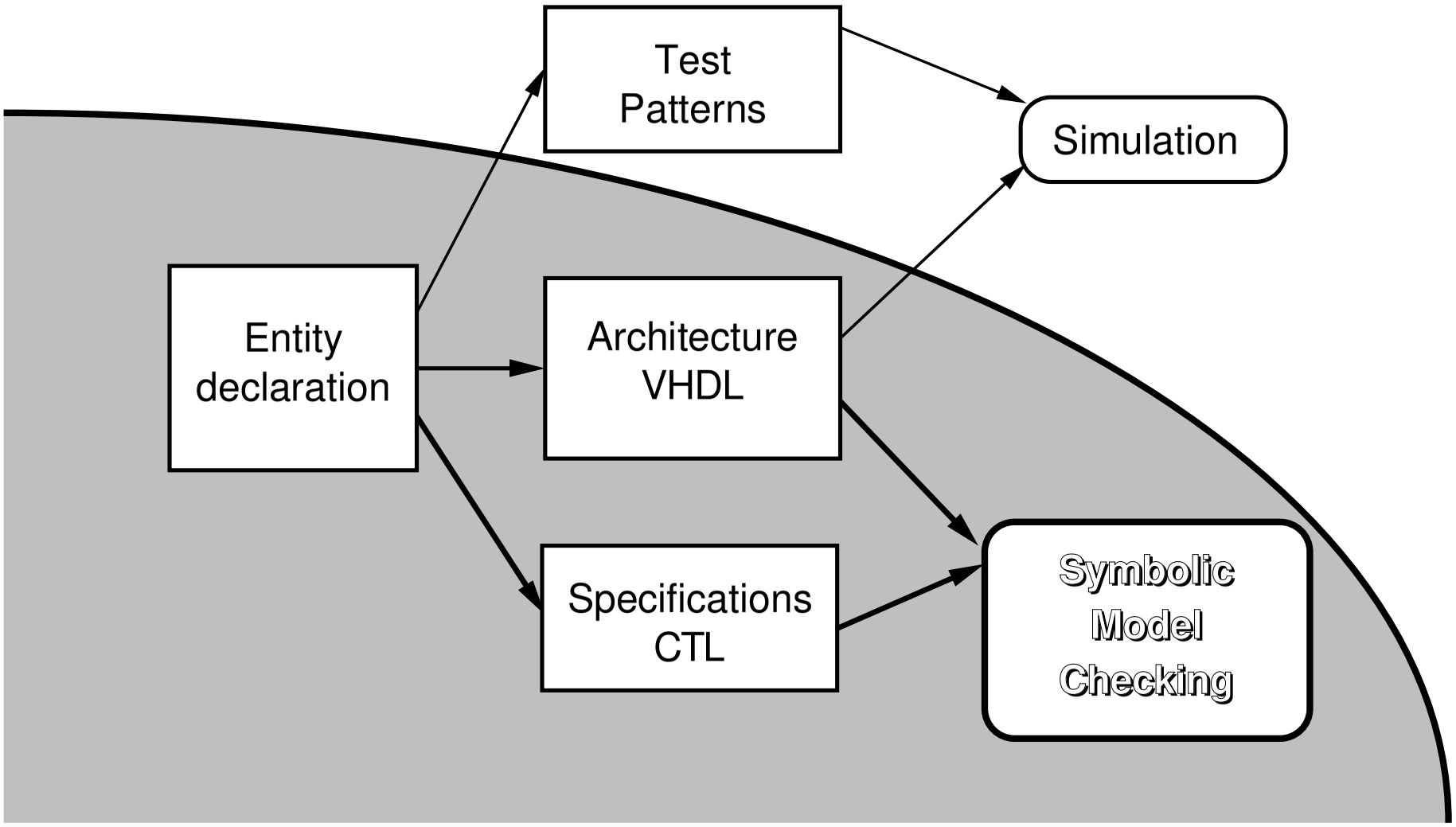
THE AUTOMATIC MODEL CHECKING FRAMEWORK



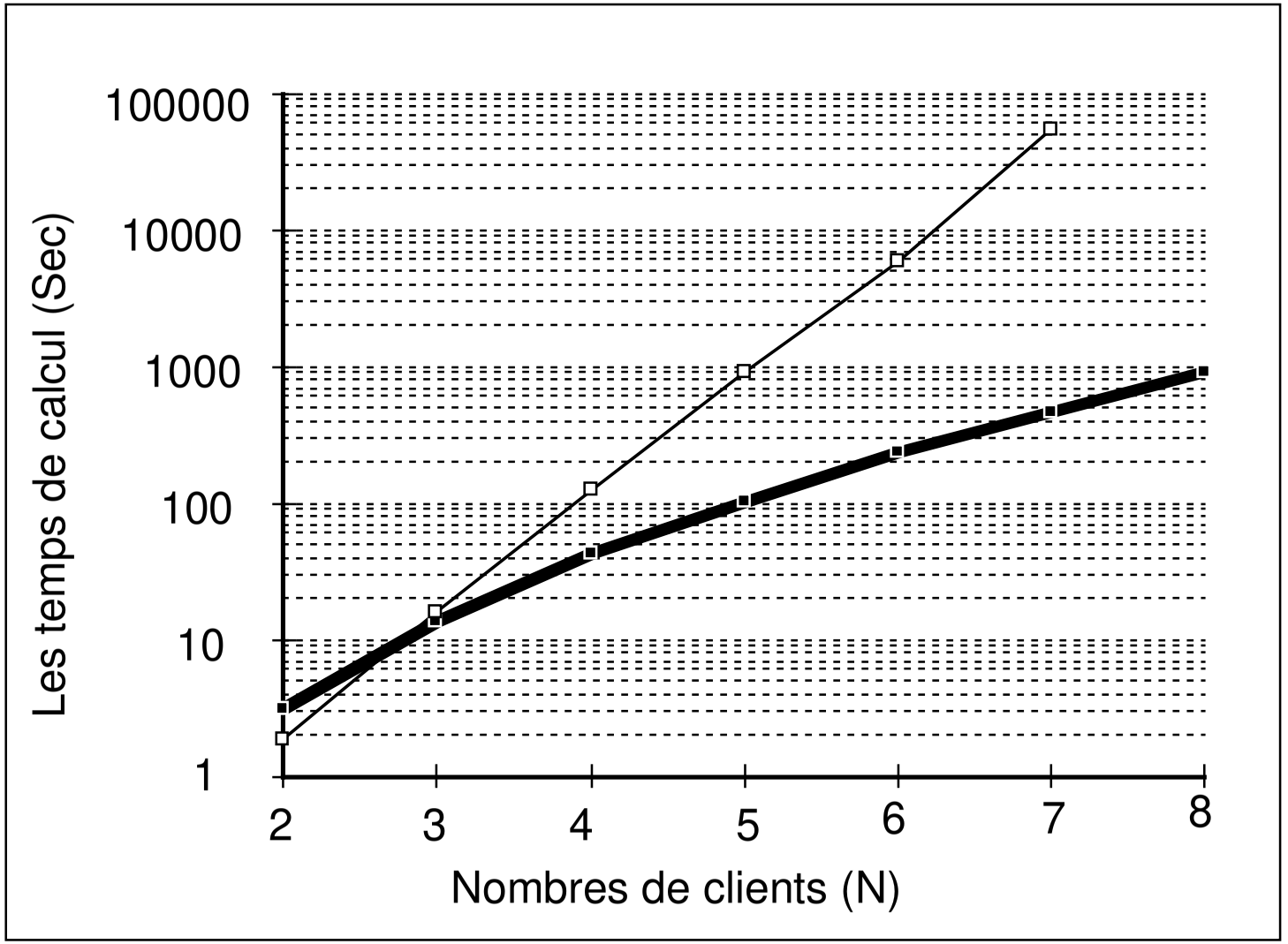
VPN : VHDL to Petri Nets

VMC : VHDL Model Checker

Formal VHDL Design Verification



COMPARAISON OF FORWARD TRAVERSAL ON A SERIES OF DISTRIBUTED ARBITRERS (CLARKE94)

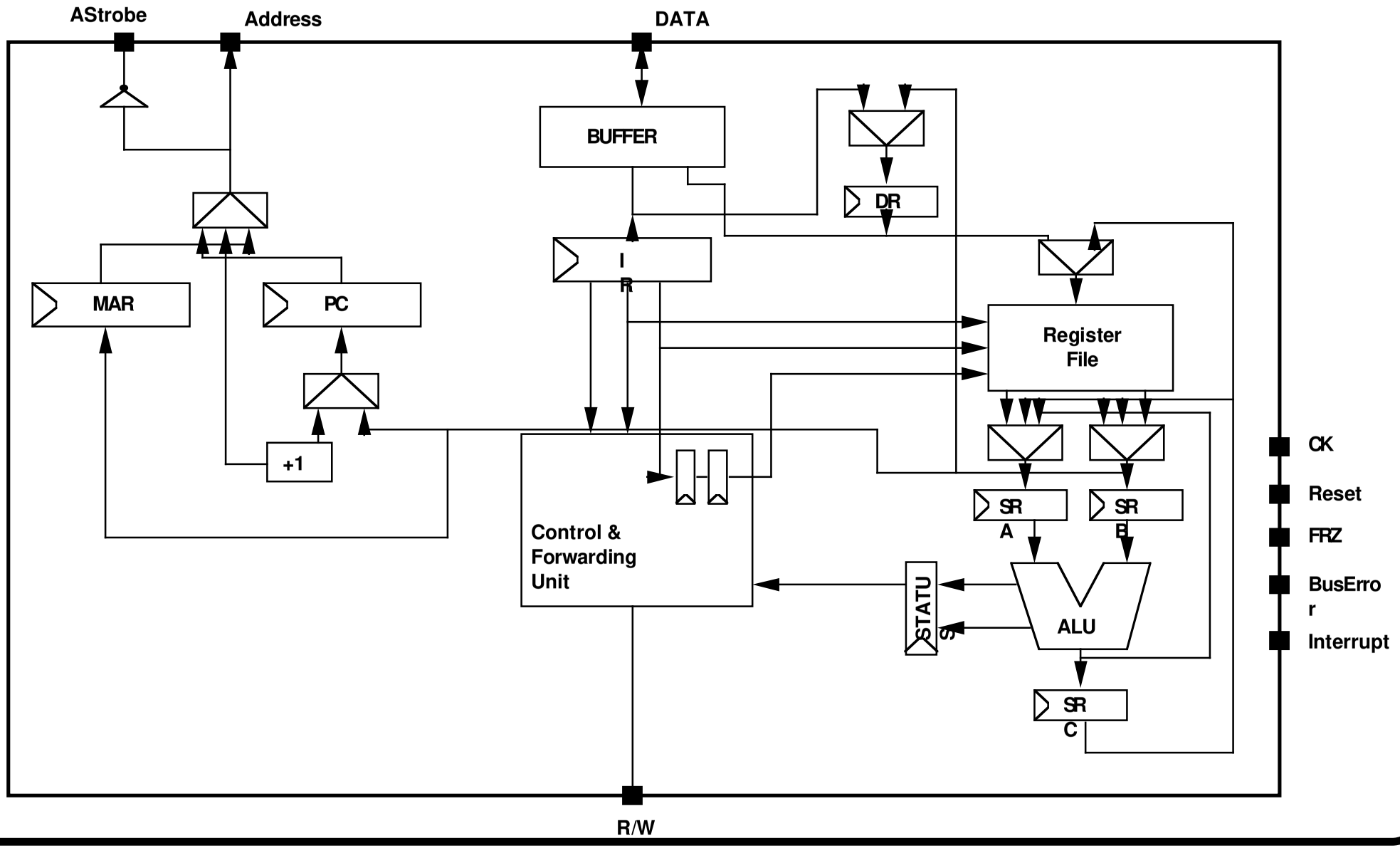


CVA-CVC
CMU
(Déharbe96)

VPN-VMC
LIP6
(Bawa 96)

- External stimuli

INFINITY PROCESSOR



PROCESSOR PROPERTIES

DATA HAZARD

$$\mathbf{AG}\,((\,\texttt{Ctl_Bypass_1} \wedge \texttt{Comp_out_1} \wedge \neg(\,\text{reset} \vee \text{interrupt}\,)) \Rightarrow \mathbf{AX}\,(\,\text{SRA} = \text{alu_out}) \wedge (\,\texttt{Ctl_Bypass_2} \wedge \texttt{Comp_out_2} \wedge \neg(\,\text{reset} \vee \text{interrupt}\,)\,) \Rightarrow \mathbf{AX}\,(\,\text{SRA} = \text{SRC}) \wedge (\,\text{zero})))$$

DATA CONSISTANCY

$$\mathbf{AG}\,((\texttt{Ctl_Stall_1} \Rightarrow \mathbf{AX}\,(\,\text{IR} = \mathbf{AG}\,((\,\texttt{store} \wedge \neg(\,\text{reset} \vee \text{interrupt}\,)) \Rightarrow \mathbf{AX}\,\mathbf{A}\,(\,\neg(\,\text{reset} \vee \text{interrupt}\,)\ \mathbf{UNTIL}\ (\texttt{write_authorized} \wedge (\neg(\,\text{reset} \vee \text{interrupt}\,) \vee \neg\texttt{write_authorized})))$$

DIFFÉRENTES INTERPRETATIONS D"UNE BASCULE D DÉCRITE EN VHDL (par Synopsis)

```
process (ck)
begin
if (ck'event and ck = '1') then
  R <= A;
end if;
end process;
```

UNE BASCULE D

```
process (ck)
begin
if (ck = '1') then
  R <= A;
end if;
end process;
```

UN LATCH

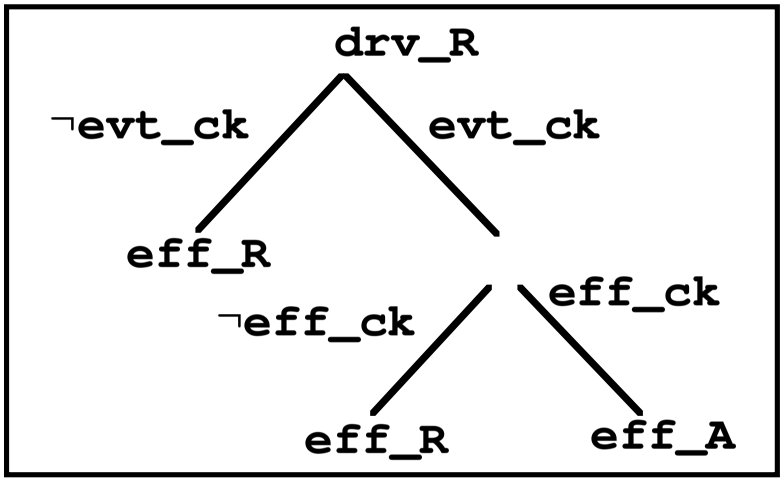
```
process (ck)
Variable T: Bit := '0';
begin
T:= R;
if (ck'event and ck = '1') then
  T := A;
end if;
R <= T;
end process;
```

NON SUPPORTÉ par Synopsis

```
process (ck)
begin
if (ck = '1') then
  R <= A;
  else
  R <= R;
end if;
end process;
```

UN MULTIPLEXEUR

NOTRE APPROCHE
(WYSIWYS)



drv_R <= (evt_ck ∧ eff_CK ∧ eff_A) ∨
(¬ (evt_ck ∧ eff_CK) ∧ eff_R);

- la sémantique de simulation est préservée
- pas de contrainte particulière sur les descriptions
- la reconnaissance des éléments mémorisants est basée sur les BDD
- la sémantique des variables VHDL est correctement interprétée