CTL-property transformations along an incremental design process

Cécile Braunstein - Emmanuelle Encrenaz

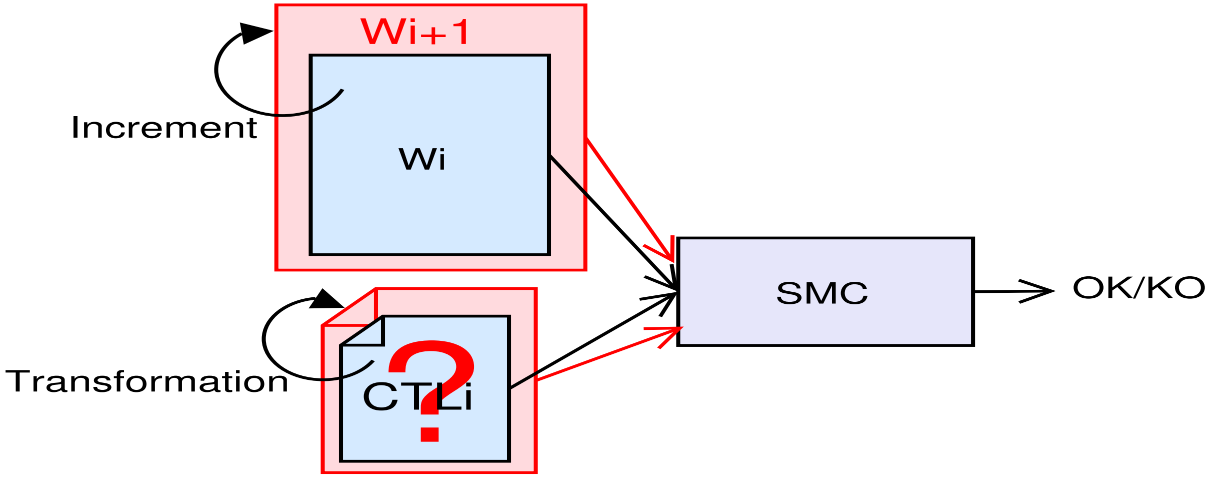
LIP6 - ASIM

Université Pierre et Marie Curie, CNRS UMR 7606

Paris, France

Context

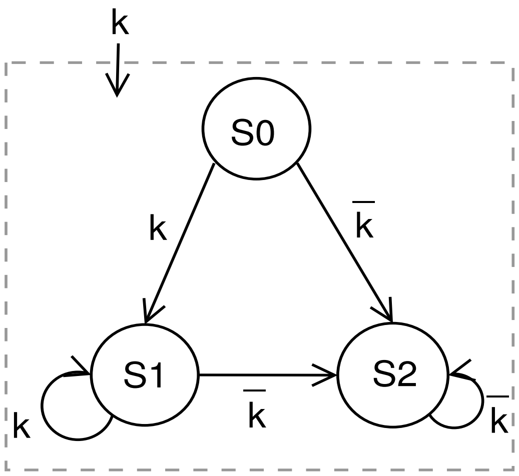
- ❏ Incremental strategy : **successive additions of new behaviours**
  - ➢ Observation of hardware designers (pipeline flow)
  - ➢ Specification by CTL formulae and symbolic model checking



- ❏ ACTL Property preservation $W_{i+1} \longrightarrow W_i$ (Grumberg and Long 1991)
- ❏ ECTL Property preservation $W_i \longrightarrow W_{i+1}$ (Loiseaux and Graf 1995)
- ❏ Complementary to Refinement strategy (B-Method Abrial)
- ❏ Differs to Feature integration (M. C. Plath and M. D. Ryan, D. Méry and D. Cansell)

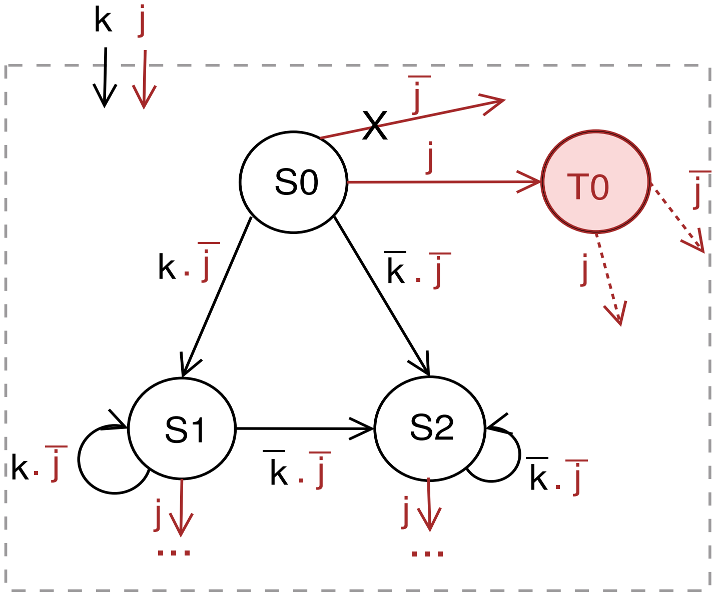
Increment Definition

- ❑ Increment INC is a set of new events
  - ➢ Each event has quiet values and active values (val_qt,val_act)
  - ➢ No new initial state
  - ➢ No behaviour overriding



INC

New signal j added
quiet value : {0}
active value : {1}


k j
$\overline{j}$
X
S0
j
T0
$\overline{j}$
j
k . $\overline{j}$
$\overline{k}$. $\overline{j}$
S1
S2
k . $\overline{j}$
$\overline{k}$. $\overline{j}$
$\overline{k}$. $\overline{j}$
j
...
j
...

Wi : complete deterministic Moore Machine

Wi+1 : complete deterministic Moore Machine

- ➠ W_{i+1} simulates W_i
- ➠ $K(W_{i+1})$ simulates $K(W_i)$ (Kripke structure)
- ➠ $K(W_{i+1})$ includes $K(W_i)$ with the state that were in $K(W_i)$ tagged with the quiet value

CTL-property transformations

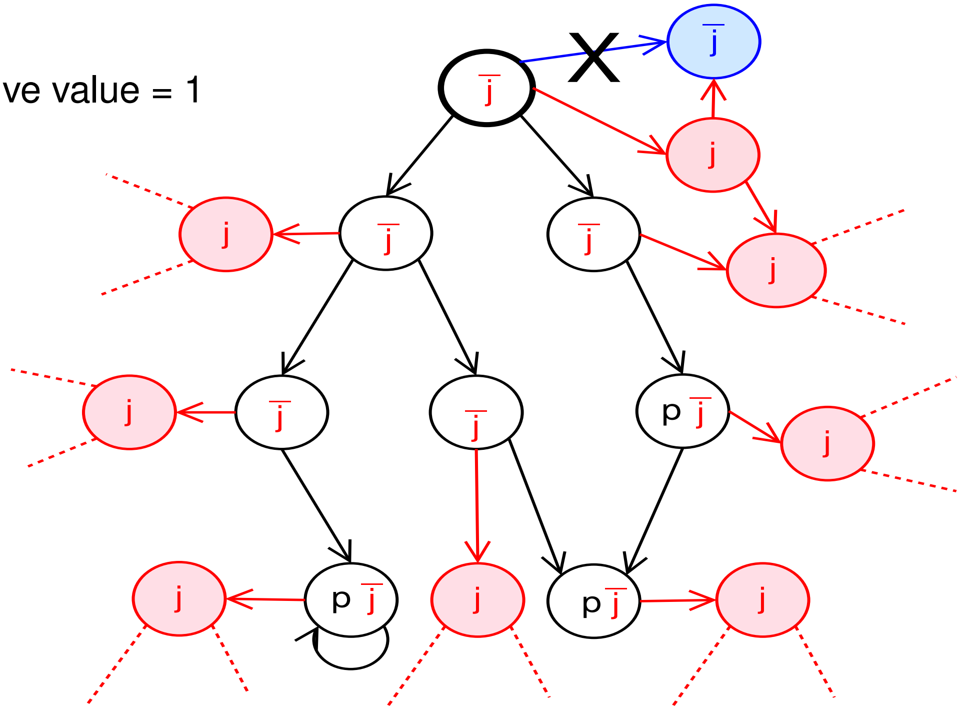
- Goal : Let be φ such that W_i,$s_0 \models \varphi$, what is φ' such that :
  K(W_i),$s_0 \models \varphi \Leftrightarrow K(W_{i+1})$,$s_0$' $\models \varphi$' with $K(W_{i+1})$ obtained by increment from $K(W_i)$
- Principle : Reduction of the computational tree explored

Example : transformation of AFp

$K(W_i) \models$ AFp

INC : j,quiet value = 0,active value = 1

$K(W_{i+1}) \models$ AF(p or j)



Results and Concluding remarks

- Transformation rules
  - All CTL operators are transformable (bi-implication)
  - All CTL formulae are transformable by recursively applying the transformation
  - The transformed CTL formulae have the same complexity as the initial ones
- Application to a concrete component design (VCI-PI protocol converter)
  - System with 330-450 boolean variables
  - Transformations applying on 80 properties automatically
  - The transformed properties do not increase significally the time of verification
- Further studies :
  - Taking advantage of the increment graph structure
  - The opposite analysis
  - Assume-Guarantee verification process

Transformations rules

Φ = p ⇔ Φ' = p

Φ = not f ⇔ Φ' = not f'

Φ = EXf ⇔ Φ' = (e = val_qt) $\Rightarrow$ EXf'

Φ = EFf ⇔ Φ' = E((e = val_qt) U f')

Φ = EGf ⇔ Φ' = EG((e = val_qt) and f')

Φ = Ef U g ⇔ Φ' = E(((e = val_qt) and f') U g')

Φ = AXf ⇔ Φ' = (e = val_qt) $\Rightarrow$ AXf'

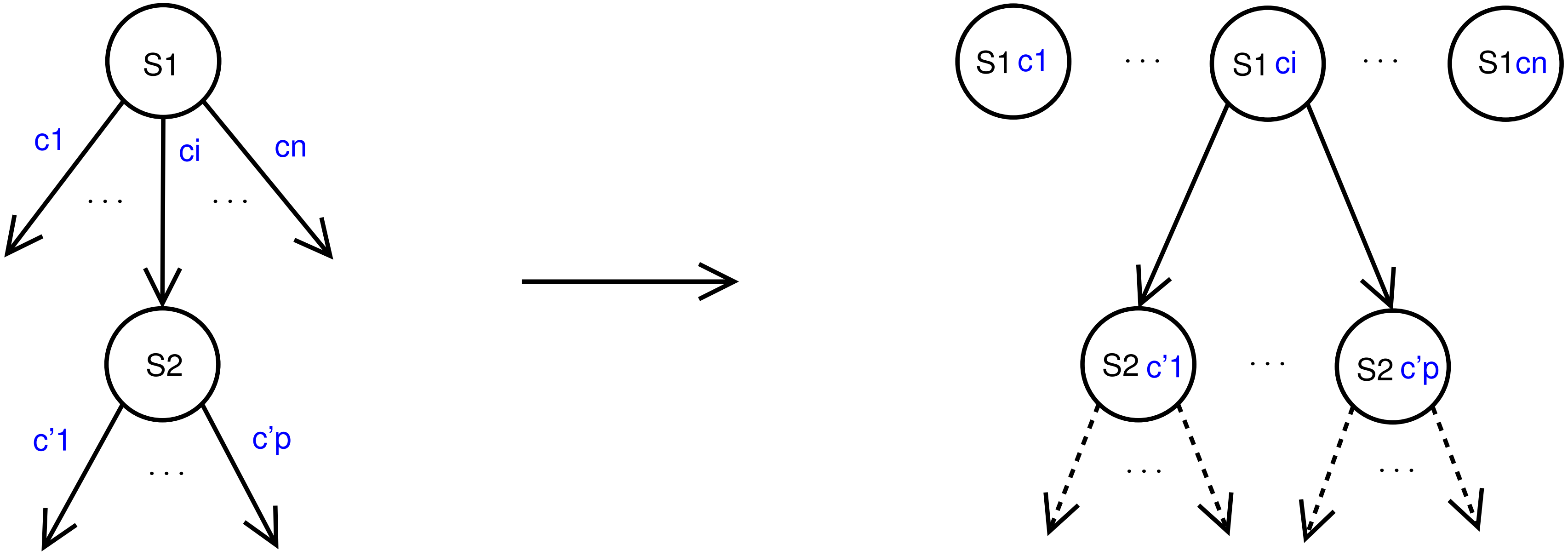
Φ = AFf ⇔ Φ' = AF((e $\neq$ val_qt) or f')

Φ = Af U g ⇔ Φ' = A(((e = val_qt) and f') U ((e $\neq$val_qt) or g'))

Φ = AGf ⇔ Φ' = A(((e = val_qt) and f') W (e $\neq$ val_qt))

Φ = Af W g ⇔ Φ' = A(f' W (g' or (e $\neq$ val_qt)))

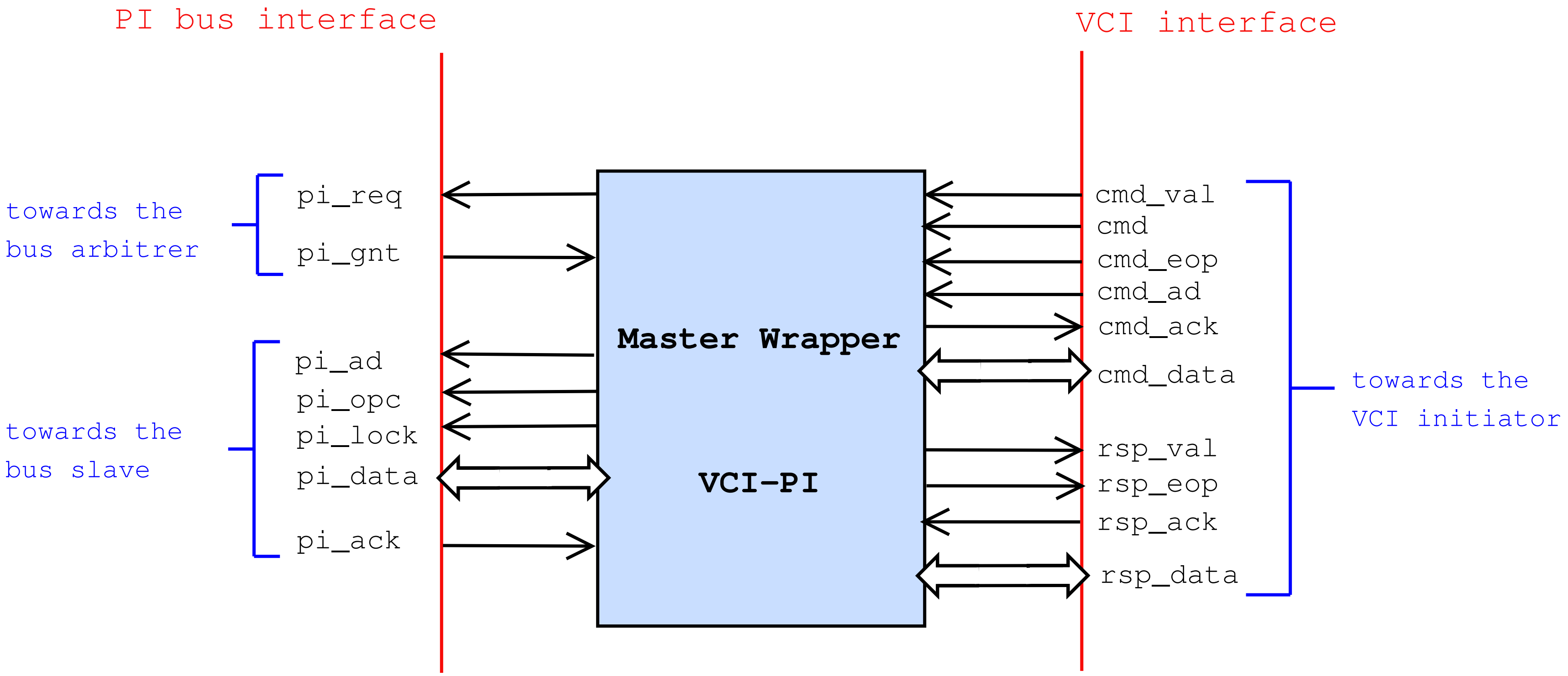
Transformation of a Moore machine into a Kripke structure



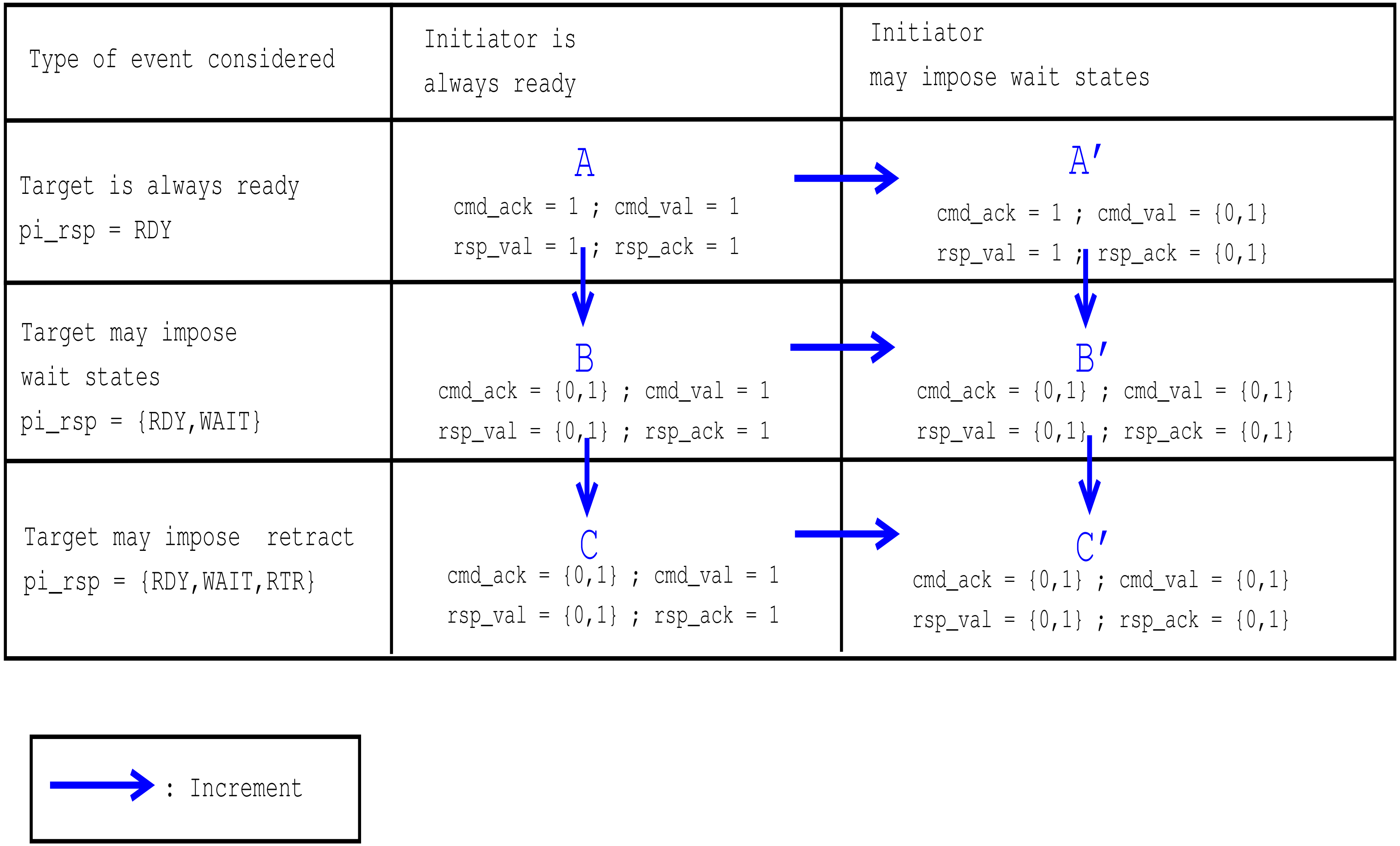
Moore Machine

Kripke Structure

VCI-PI Wrapper interface



Wrappers Hierarchy



Type of event considered	Initiator is always ready	Initiator may impose wait states

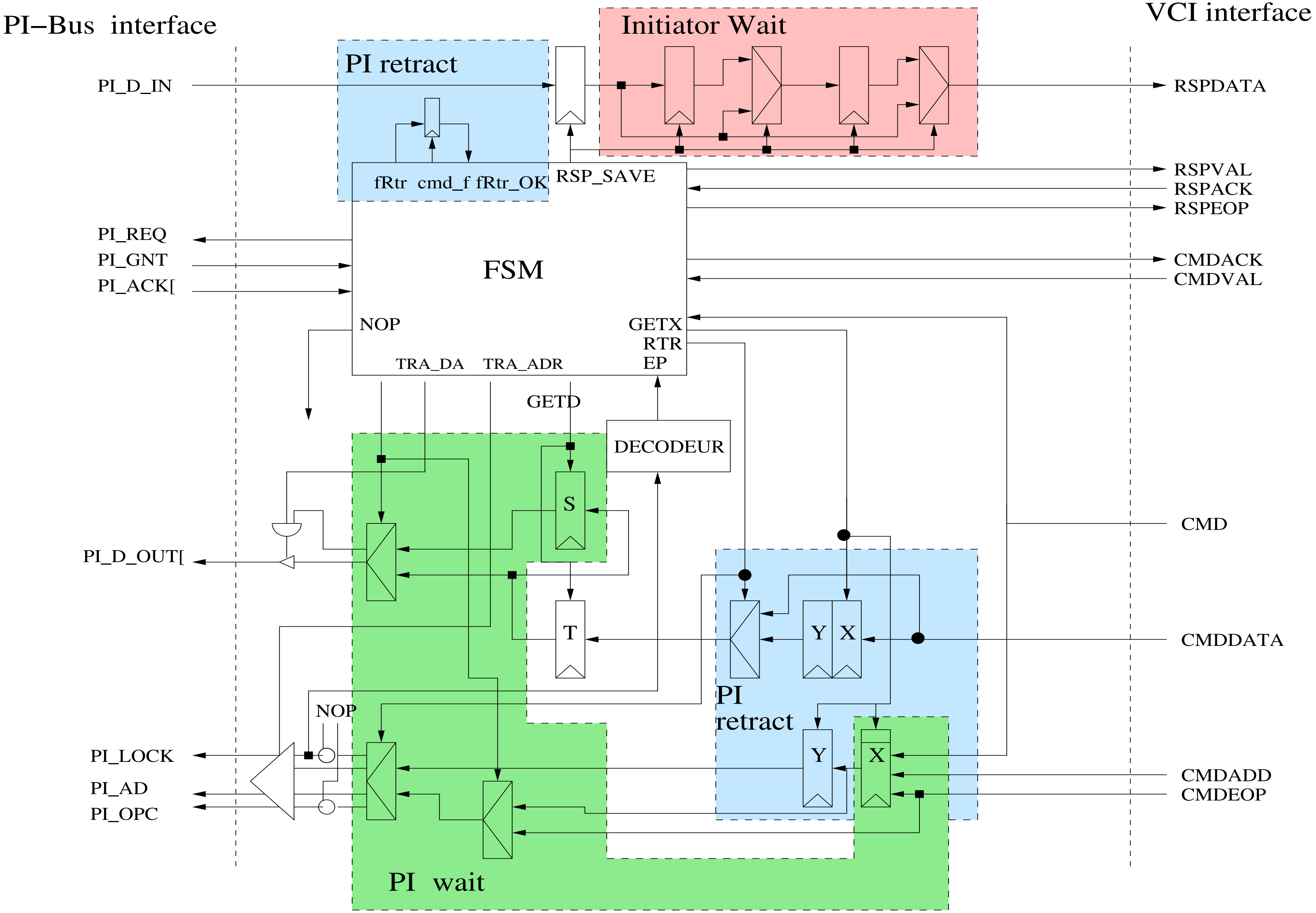
Target is always ready
pi_rsp = RDY
	A
	cmd_ack = 1 ; cmd_val = 1
	rsp_val = 1 ; rsp_ack = 1
		A′
		cmd_ack = 1 ; cmd_val = {0,1}
		rsp_val = 1 ; rsp_ack = {0,1}

Target may impose wait states
pi_rsp = {RDY,WAIT}
	B
	cmd_ack = {0,1} ; cmd_val = 1
	rsp_val = {0,1} ; rsp_ack = 1
		B′
		cmd_ack = {0,1} ; cmd_val = {0,1}
		rsp_val = {0,1} ; rsp_ack = {0,1}

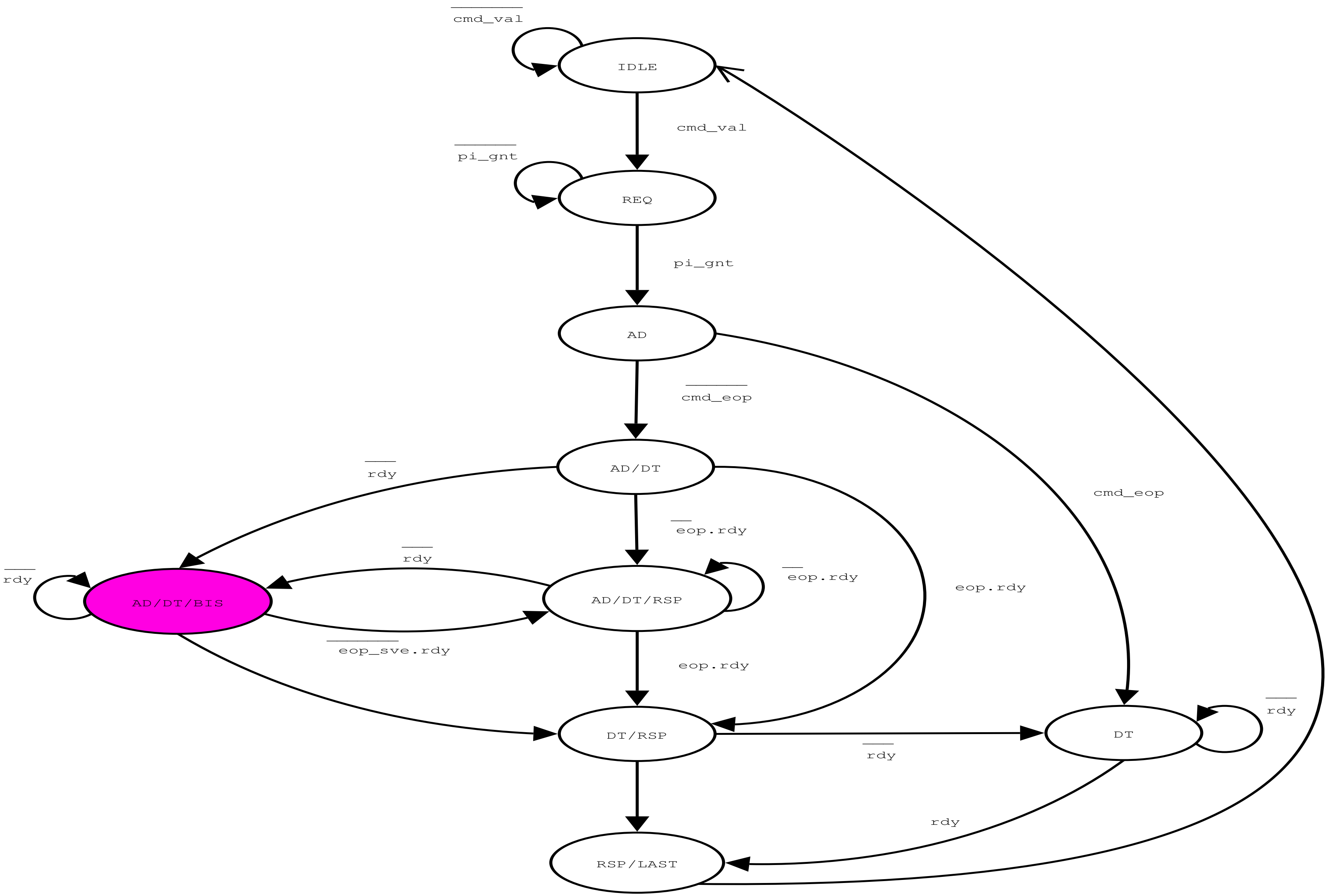
Target may impose retract
pi_rsp = {RDY,WAIT,RTR}
	C
	cmd_ack = {0,1} ; cmd_val = 1
	rsp_val = {0,1} ; rsp_ack = 1
		C′
		cmd_ack = {0,1} ; cmd_val = {0,1}
		rsp_val = {0,1} ; rsp_ack = {0,1}

→ : Increment

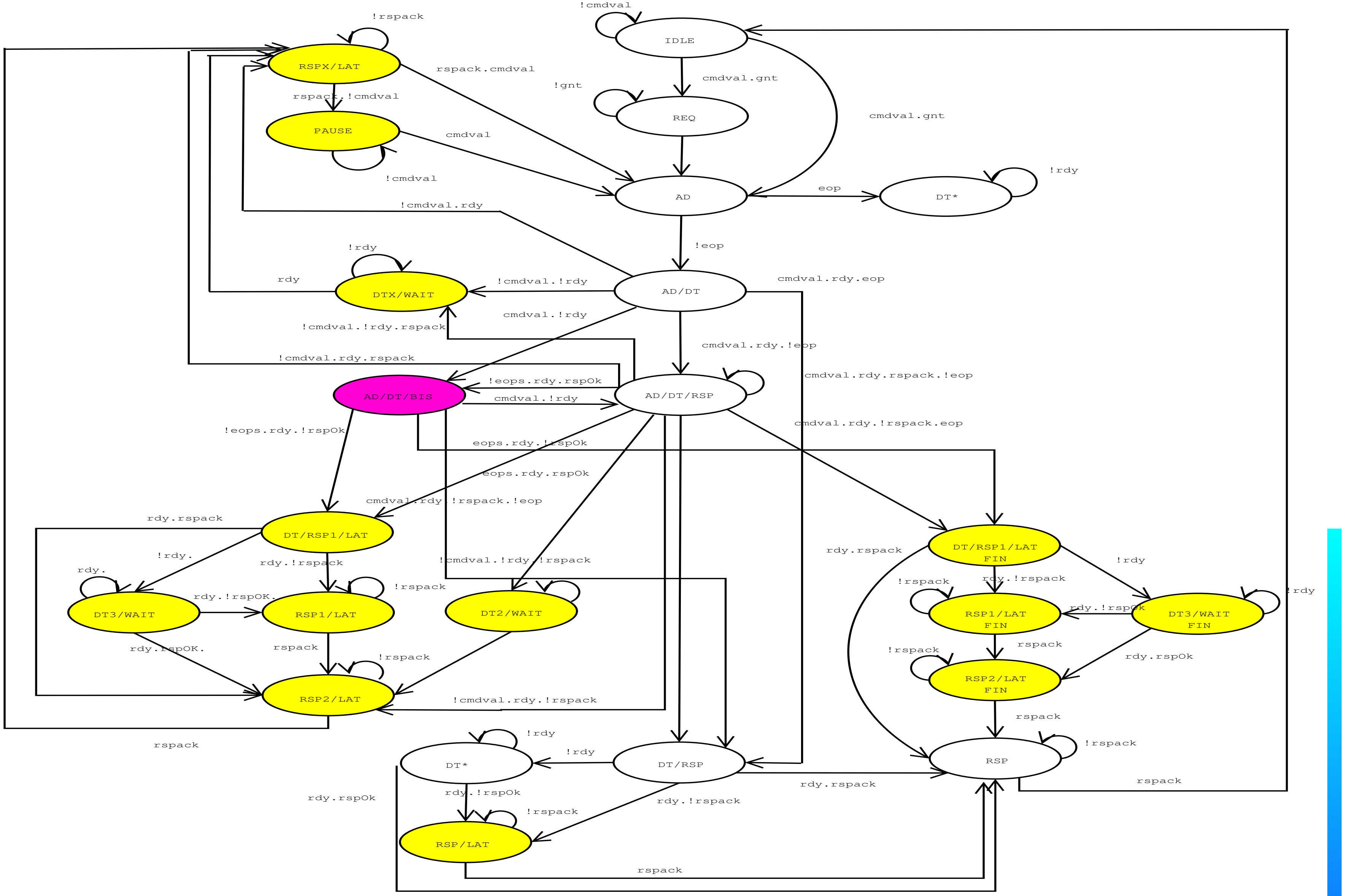
VCI-PI Wrappers Datapath (C)



VCI-PI Wrappers FSM (B')



VCI-PI Wrappers FSM(A and B)


!cmdval
IDLE
!rspack
RSPX/LAT
rspack.cmdval
cmdval.gnt
!gnt
REQ
rspack.!cmdval
PAUSE
cmdval
cmdval.gnt
!cmdval
AD
eop
DT*
!rdy
!cmdval.rdy
!eop
!rdy
rdy
DTX/WAIT
!cmdval.!rdy
AD/DT
cmdval.rdy.eop
cmdval.!rdy
!cmdval.!rdy.rspack
cmdval.rdy.!eop
!cmdval.rdy.rspack
cmdval.rdy.rspack.!eop
!eops.rdy.rspOk
AD/DT/BIS
cmdval.!rdy
AD/DT/RSP
!eops.rdy.!rspOk
cmdval.rdy.!rspack.eop
eops.rdy.!rspOk
eops.rdy.rspOk
cmdval.rdy.!rspack.!eop
rdy.rspack
DT/RSP1/LAT
rdy.rspack
DT/RSP1/LAT FIN
!rdy.
rdy.!rspack
!cmdval.!rdy.!rspack
!rdy
rdy.
!rspack
rdy.!rspack
rdy.!rspOK.
!rspack
DT3/WAIT
RSP1/LAT
DT2/WAIT
RSP1/LAT FIN
rdy.!rspOk
DT3/WAIT FIN
!rdy
rdy.rspOK.
rspack
!rspack
rspack
!rspack
rdy.rspOk
RSP2/LAT
!cmdval.rdy.!rspack
RSP2/LAT FIN
rspack
rspack
!rdy
!rspack
DT*
!rdy
DT/RSP
RSP
rspack
rdy.rspOk
rdy.!rspOk
rdy.rspack
rdy.!rspack
!rspack
RSP/LAT
rspack