

Esterel et SCADE, de la recherche à l'industrie

3. Urgences scientifiques posées par l'industrie

Gérard Berry

Collège de France

Chaire Algorithmes, machines et langages

gerard.berry@college-de-france.fr

Cours 3, Inria Sophia-Méditerranée, 29/01/2014

Trois questions brûlantes

- On ne peut pas se servir à fond du langage, car il rejette des programmes évidemment corrects pour **erreur de causalité** !
- Tous les systèmes sur puce sont maintenant **multi-horloges** et utilisent le **clock-gating**.
Quelle est votre offre en ce domaine?
- Nos évaluations par la R&D sont bonnes, mais pour passer en production il est impératif de traiter les **ECOs**. Comment faites vous cela?

Trois questions brûlantes

- On ne peut pas se servir à fond du langage, car il rejette des programmes évidemment corrects pour **erreur de causalité** !
- Tous les systèmes sur puce sont maintenant **multi-horloges** et utilisent le clock-gating. Quelle est votre offre en ce domaine?
- Nos évaluations par la R&D sont bonnes, mais pour passer en production il est impératif de traiter les **ECOs**. Comment faites vous cela?

La causalité (1984-2012)

En 1768, la France acheta la Corse, trois ans avant la naissance de Napoléon, pour être sûr qu'il serait français.

Je prie mes héritiers de faire procéder à une autopsie, car je tiens absolument à connaître les causes de ma mort.

Vu son nom, Gilles Kahn était bien forcé de travailler sur les réseaux de Kahn !

Causalité : logique, statique, dynamique ?

~~if X then nothing else emit X end $X = \text{not } X$~~

~~if X then emit X end $X = X$~~

non-déterminisme ou **non-causalité** ?

~~if X then emit X else emit X end $X = X$ or not X~~

déterminisme mais **causalité** ?



Solution 1 : retards + acyclicité

- Exiger l'acyclicité du graphe de dépendance
 - if X then emit Y end \Rightarrow dépendance $Y \leftarrow X$
 - calculer les **dépendances potentielles** par analyse statique
 - **erreur** si le graphe des dépendances est **acyclique**
- Pour casser les cycles, introduire des délais
 - **pre**(X) de Lustre : valeur à l'instant d'avant
if **pre**(X) then emit X end \leftarrow **OK**
 - **next**(X) de TLA : émettre à l'instant d'après
if X then emit **next** X end \leftarrow **OK**

Contrainte OK en flot de données (Lustre / SCADE)
et (hélas) quasi-obligatoire en synthèse de circuits
Mais **trop rigide en contrôle complexe !**

Exemple naturellement cyclique (SW)

E. Ledinot, Dassault aviation, séminaire du 16/04/2013

```
input TOP_HORLOGE, INDICATEUR_ON ;  
relation TOP_HORLOGE # INDICATEUR_ON ;  
signal FIN, REARMEMENT in  
  abort  
    await TOP_HORLOGE ; emit FIN end  
  when REARMEMENT  
||  
  abort  
    every INDICATEUR_ON ; emit REARMEMENT  
  when FIN  
end signal
```



TOP_HORLOGE # INDICATEUR_ON \Rightarrow FIN # REARMEMENT

Analyse explicite fine (Gonthier, 1987)

- Calcul de **potentiels d'émission** de signaux utilisant le développement explicite d'un automate, dépendant de l'état et de l'entrée

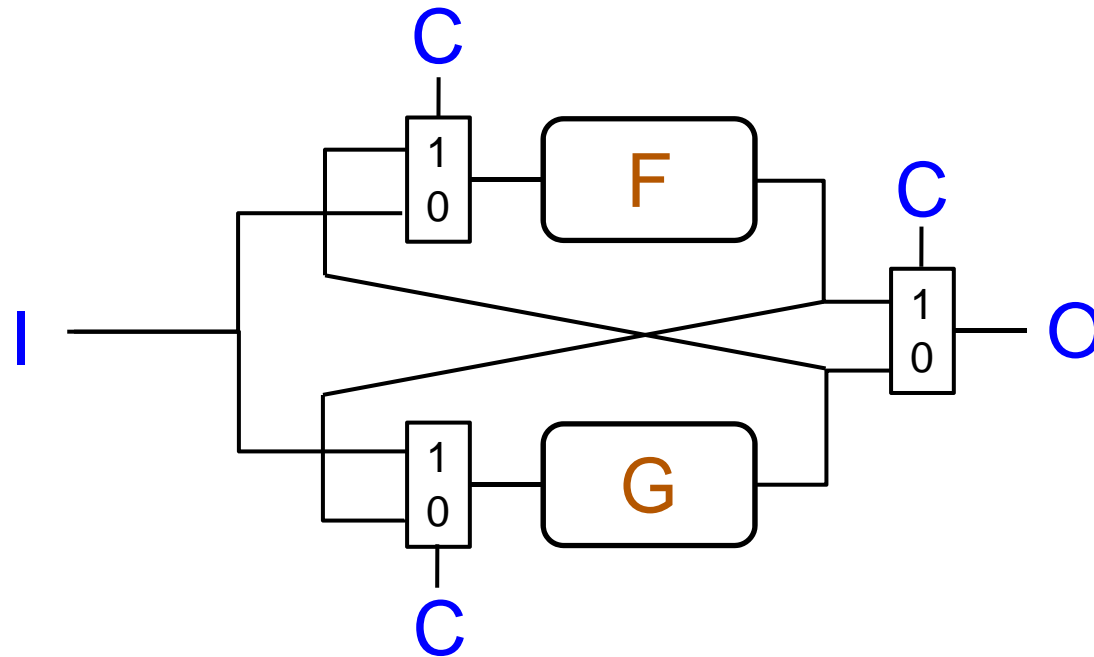
=> **Accepte les programmes Dassault**

- Mais deux inconvénients importants
 - pas de caractérisation simple des programmes acceptés ou rejetés
 - **ne passe pas à l'échelle**, comme toute génération d'automates déterministes

Pb: trouver une solution pour Esterel v4 / v5

Partage de ressources \Rightarrow cycles combinatoires

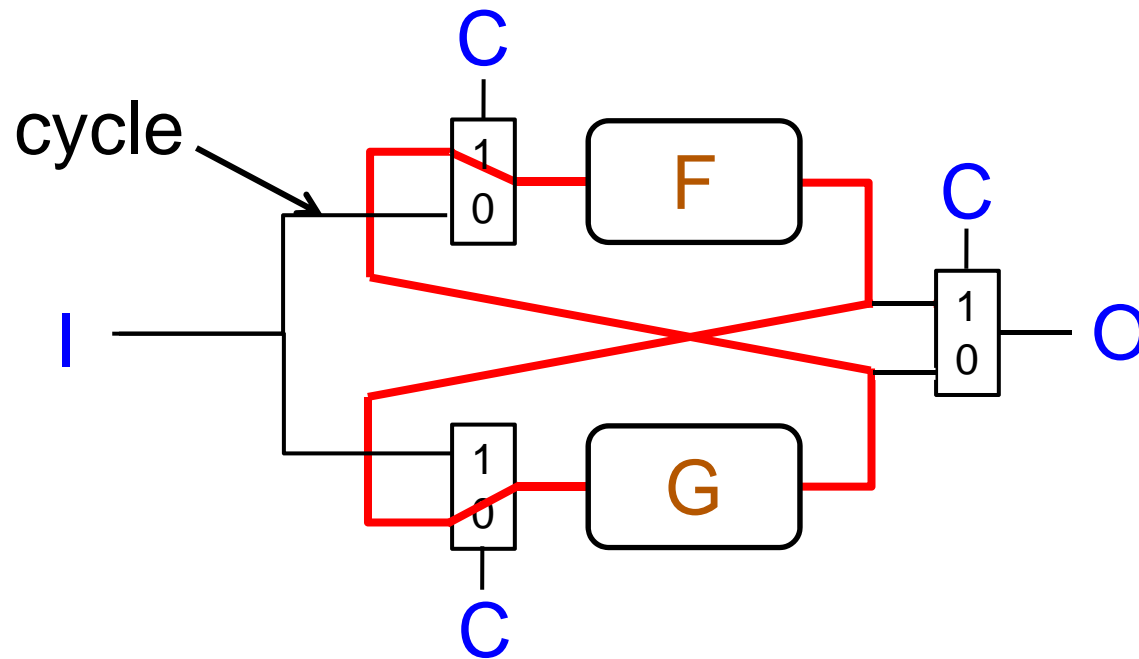
$O = \text{if } C \text{ then } F(G(I)) \text{ else } G(F(I))$



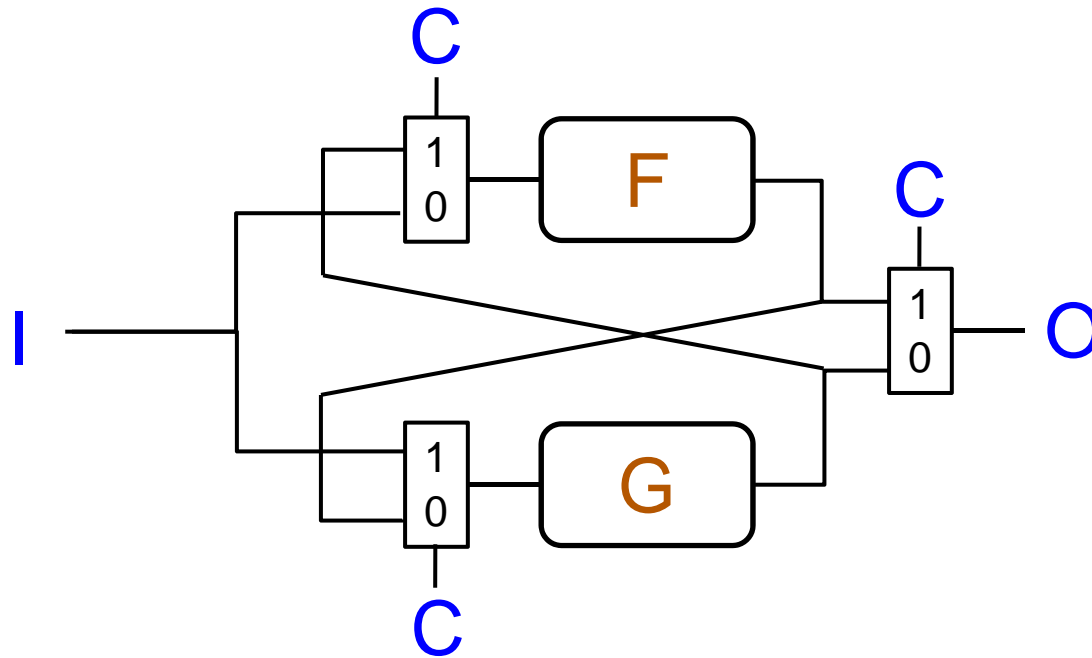
Sharad Malik, *Analysis of Cyclic Combinational Circuits*
IEEE Transactions on Computer-Aided Design of Integrated
Circuits and Systems, VOL. 13, NO. 7, JULY 1994

Partage de ressources \Rightarrow cycles combinatoires

$O = \text{if } C \text{ then } F(G(I)) \text{ else } G(F(I))$

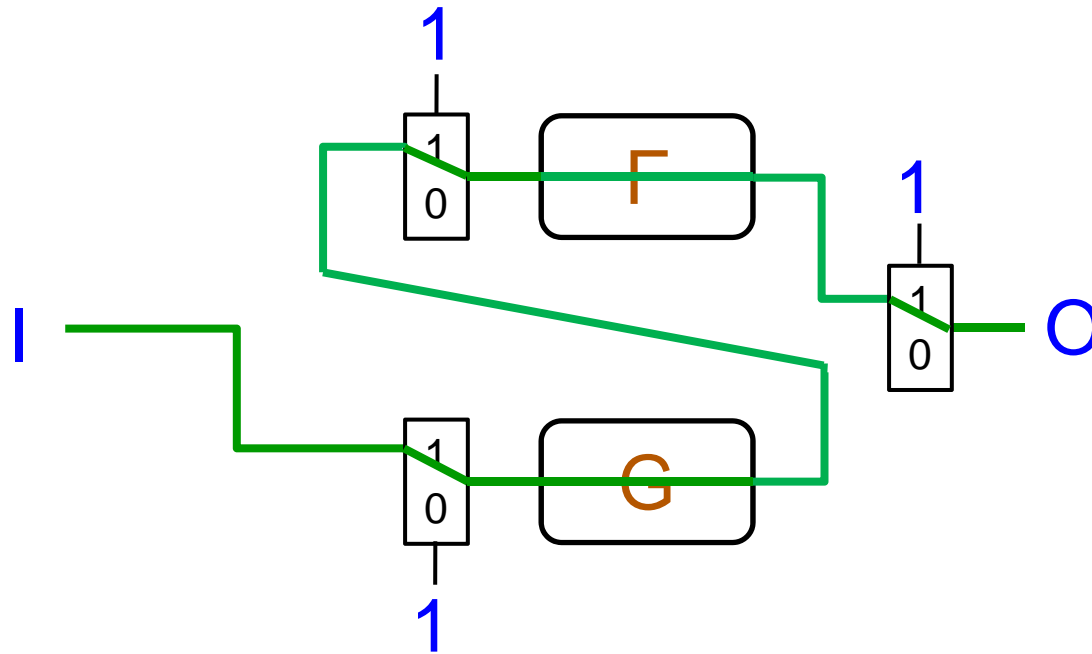


Partage de ressources \Rightarrow cycles combinatoires



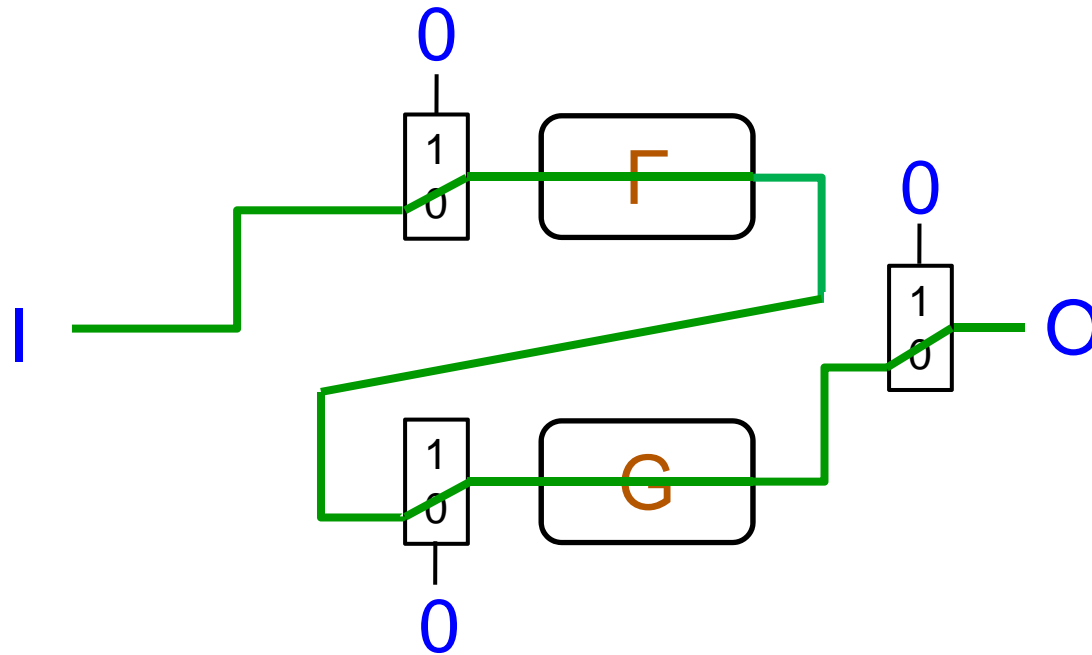
Partage de ressources \Rightarrow cycles combinatoires

$$C = 1 \Rightarrow O = \text{if } C \text{ then } F(G(I)) \text{ else } G(F(I))$$



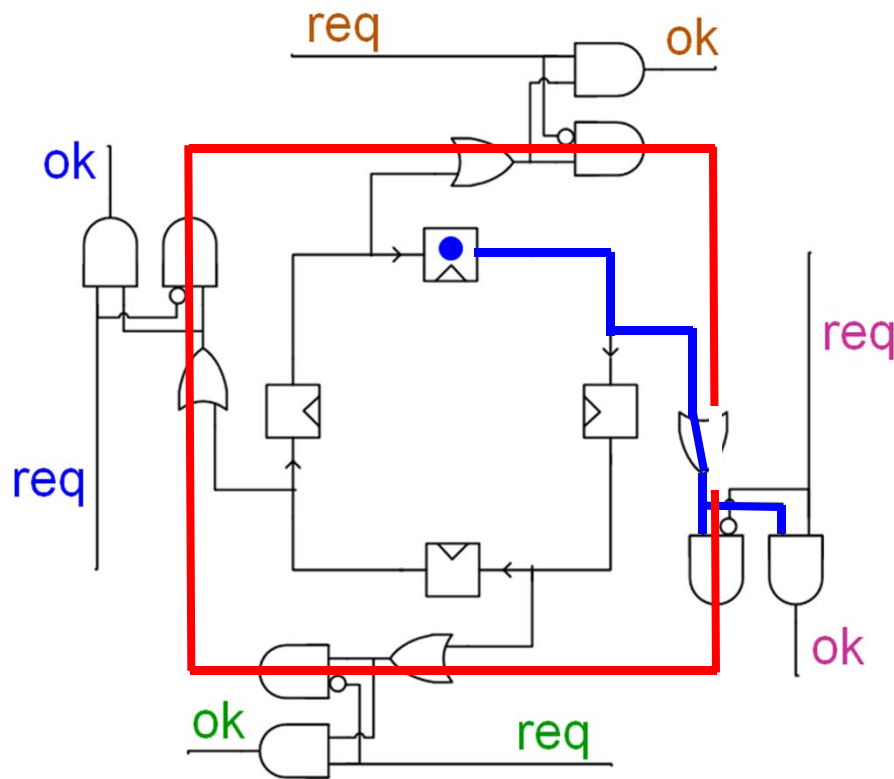
Partage de ressources \Rightarrow cycles combinatoires

$C = 0 \Rightarrow O = \text{if } C \text{ then } F(G(I)) \text{ else } G(F(I))$

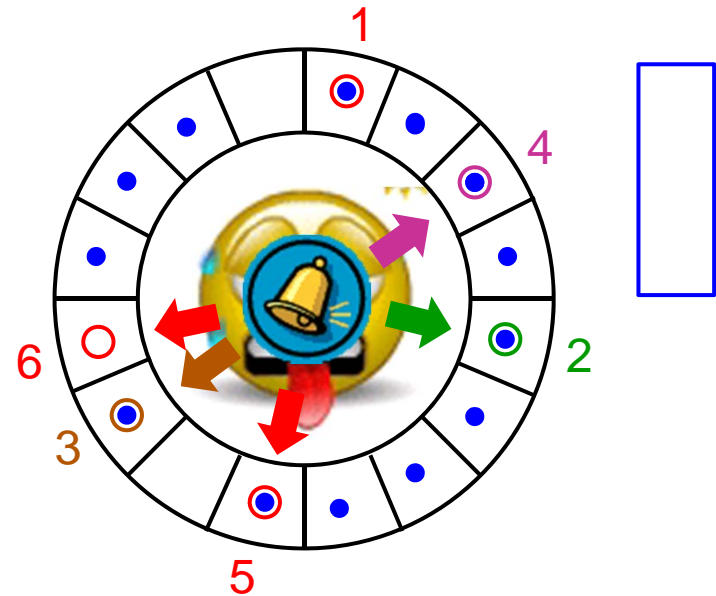


Le cycle ne pose ni problème logique,
ni problème électrique !

Circuits séquentiels cycliques OK



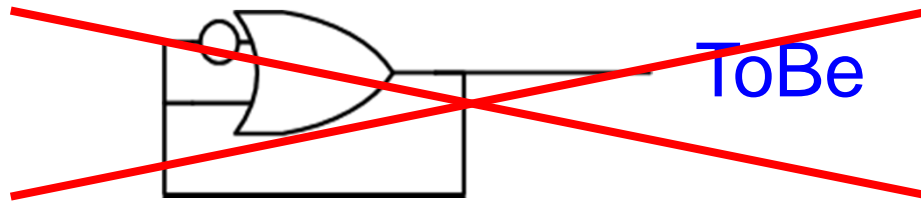
Allocateur de ressources
round-robin cyclique
(R. de Simone)



ILD
(Instruction Length Decoder)

Electricité = Logique constructive

Hamlet : ToBe = ToBe or not ToBe



- Calcule 1 en logique classique
mais pas en logique constructive (sans tiers exclu) !
- Se stabilise électriquement à 1 pour certains délais des portes et fils, mais pas pour tous les délais !

Théorème (Mendler-Shiple-Berry), cours 26/03/2014 :
électriquement stable pour tous délais \Leftrightarrow constructif

Anagramme renversante?

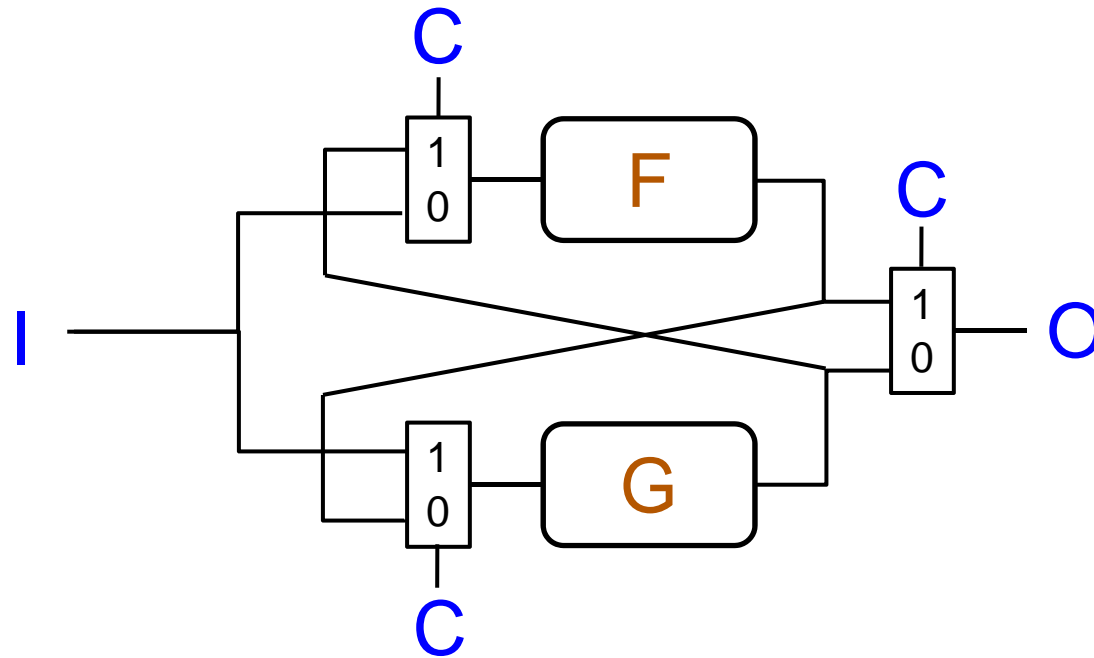
Etre ou ne pas être, voilà la question !
Oui, et la poser n'est que vanité orale !

*Anagrammes renversantes
ou le sens caché du monde*
E. Klein, J. Perry-Salkow et M. Donatien

Et bien non !

Partage de ressources \Rightarrow cycles combinatoires

$O = \text{if } C \text{ then } F(G(I)) \text{ else } G(F(I))$



Trouvé correct par la sémantique constructive
Pour F , G , H , etc, la version cyclique
est linéaire mais la version acyclique **exponentielle** en taille

Propagation constructive : des faits !

```
emit x
|| if not x then emit y
|| if not y then emit z
```

must

```
emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow x = 1$

cannot

```
emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow y = 0$

car les émetteurs
de y ont disparu

must

```
emit x
|| if not x then emit y
|| if not y then emit z
```

$\Rightarrow z = 1$

Implémentation

- Interprétation : propagation pas à pas des valeurs
 - application directe des règles de la sémantique constructive
 - vrais cycles détectés à l'exécution
- Compilation / vérification par BDD / SAT
 - vérification de la constructivité en fonction des relations d'entrées et des états atteignables
 - abstraction des données (mais calcul exact faisable en SMT)
 - production d'un circuit / logiciel acyclique équivalent
 - vérification de l'équivalence avec un acyclique fait main
 - algorithmes malins (T. Shiple) **mais chers en pratique** (passage à l'échelle non garanti)

Beaucoup d'améliorations algorithmiques possibles

Autre causalité : la programmation réactive

- Interdire la réaction instantanée à l'absence
- Privilégier l'absence sur la présence
- retarder l'accès aux valeurs

F. Boussinot, 1991 : Reactive C, SugarCubes

L. Mandel, 2002 : Reactive ML

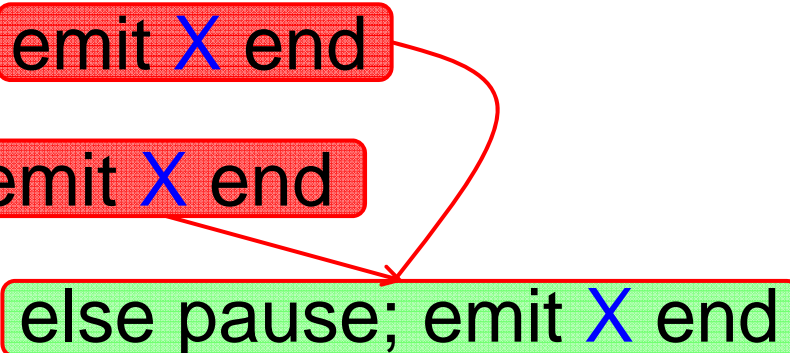
⇒ Tous les programmes ont une sémantique

if X then emit X end \rightarrow X absent

if X then nothing else emit X end

if X then emit X else emit X end

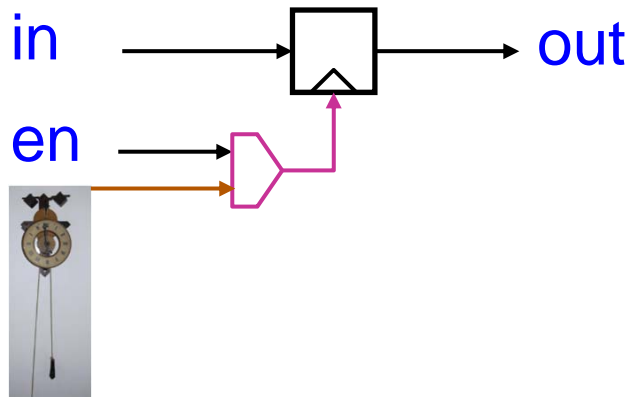
else pause; emit X end



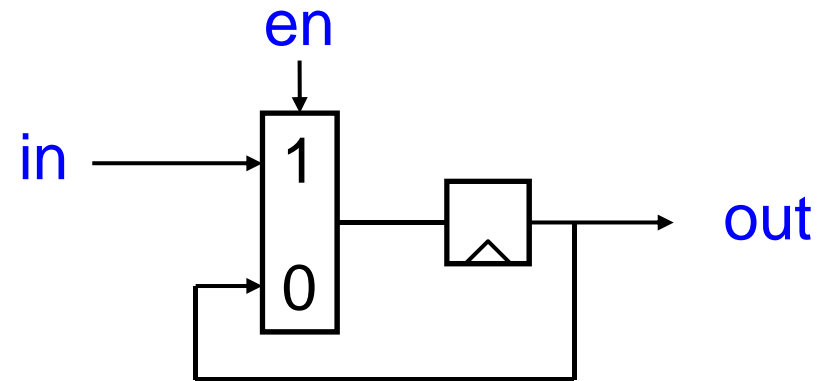
Trois questions brûlantes

- On ne peut pas se servir à fond du langage, car il rejette des programmes évidemment corrects pour **erreur de causalité** !
- Tous les systèmes sur puce sont maintenant **multi-horloges** et utilisent le **clock-gating**.
Quelle est votre offre en ce domaine?
- Nos évaluations par la R&D sont bonnes, mais pour passer en production il est impératif de traiter les **ECOs**. Comment faites vous cela?

Masquage d'horloge (clock gating)



Clock gater (ASIC)



logique de masquage
(FPGA, logiciel)

- Habituellement vu comme **économiseur d'énergie** mais aussi utilisable en algorithmique HW
- En partie automatisé par pattern-matching HDL
- Pas vraiment dans le modèle RTL des circuits

Instruction suspend

```
suspend  
  loop  
    pause ;  
    emit ?X <= pre(?X) + 1  
    pause ;  
  end loop
```

||

```
  loop  
    pause ;  
    emit next ?Y <= ?Y+1 ;  
    pause ;  
  end loop  
when Susp
```

état de départ

Instruction suspend avec *Susp* absent

suspend

loop

pause ;

emit ?X <= pre(?X) + 1

pause ;

end loop

||

loop

pause ;

emit next ?Y <= ?Y+1 ;

pause ;

end loop

when *Susp*

action

état de départ

état d'arrivée

Instruction suspend avec *Susp* présent

suspend

loop

pause ;

emit ?X \leq pre(?X) + 1

pause ;

end loop

||

loop

pause ;

emit next ?Y \leq ?Y+1 ;

pause ;

end loop

when *Susp*

suspension

état de départ

état d'arrivée

inchangé

Sémantique formelle de suspend

$$\begin{array}{c}
 \text{action} \quad \frac{s \notin E \quad p \xrightarrow[E]{E' \quad k} p'}{\quad} \quad \text{nouvel état} \\
 \\
 s \triangleright p \xrightarrow[E]{E' \quad k} s \triangleright p'
 \end{array}$$

$$\begin{array}{c}
 \text{pas d'action} \quad \frac{s \in E}{\quad} \quad \text{même état} \\
 \\
 s \triangleright p \xrightarrow[E]{\emptyset \quad 1} s \triangleright p
 \end{array}$$

K. Schneider : par symétrie, weak suspend

$$\text{action} \quad \frac{s \notin E \quad p \xrightarrow[E]{E' \quad k} p'}{\text{nouvel état}}$$

$$s \supseteq p \xrightarrow[E]{E' \quad k} s \supseteq p'$$

$$\text{action} \quad \frac{s \in E \quad p \xrightarrow[E]{E' \quad k} p'}{\text{même état}}$$

$$s \supseteq p \xrightarrow[E]{E' \quad \max(k, 1)} s \supseteq p$$

Instruction weak suspend

weak suspend

loop

pause ;

emit ?X \leq pre(?X) + 1

pause ;

end loop

||

loop

pause ;

emit next ?Y \leq ?Y+1 ;

pause ;

end loop

when Susp

état de départ

*weak suspend avec **Susp** absent*

weak suspend

loop

pause ;

emit ?X <= pre(?X) + 1

pause ;

end loop

||

loop

pause ;

emit next ?Y <= ?Y+1 ;

pause ;

end loop

when **Susp**

action

état de départ

état d'arrivée

idem suspend

*weak suspend avec **Susp** présent*

weak suspend

loop

pause ;

emit ?X <= pre(?X) + 1

pause ;

end loop

||

loop

pause ;

emit next ?Y <= ?Y+1 ;

pause ;

end loop

when **Susp**

suspension, action

état d'arrivée
inchangé !

weak suspend immédiat

- par défaut, suspension ignorée au premier instant
- variante immédiate pour suspendre au premier instant

weak suspend

p

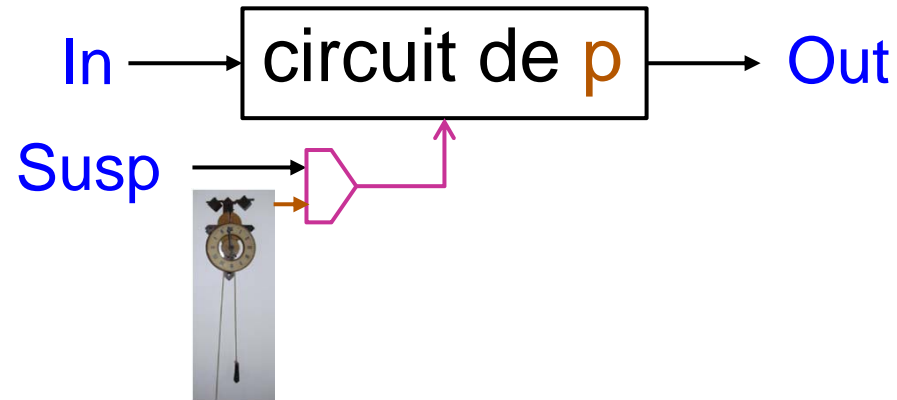
when immediate *S*

```
trap Done in
loop
  trap Immediate in
  {
    p
    ||
    if exp then exit Immediate end
  }
  exit Done
end trap;
pause
end loop
end trap
```

weak suspend = clock gating

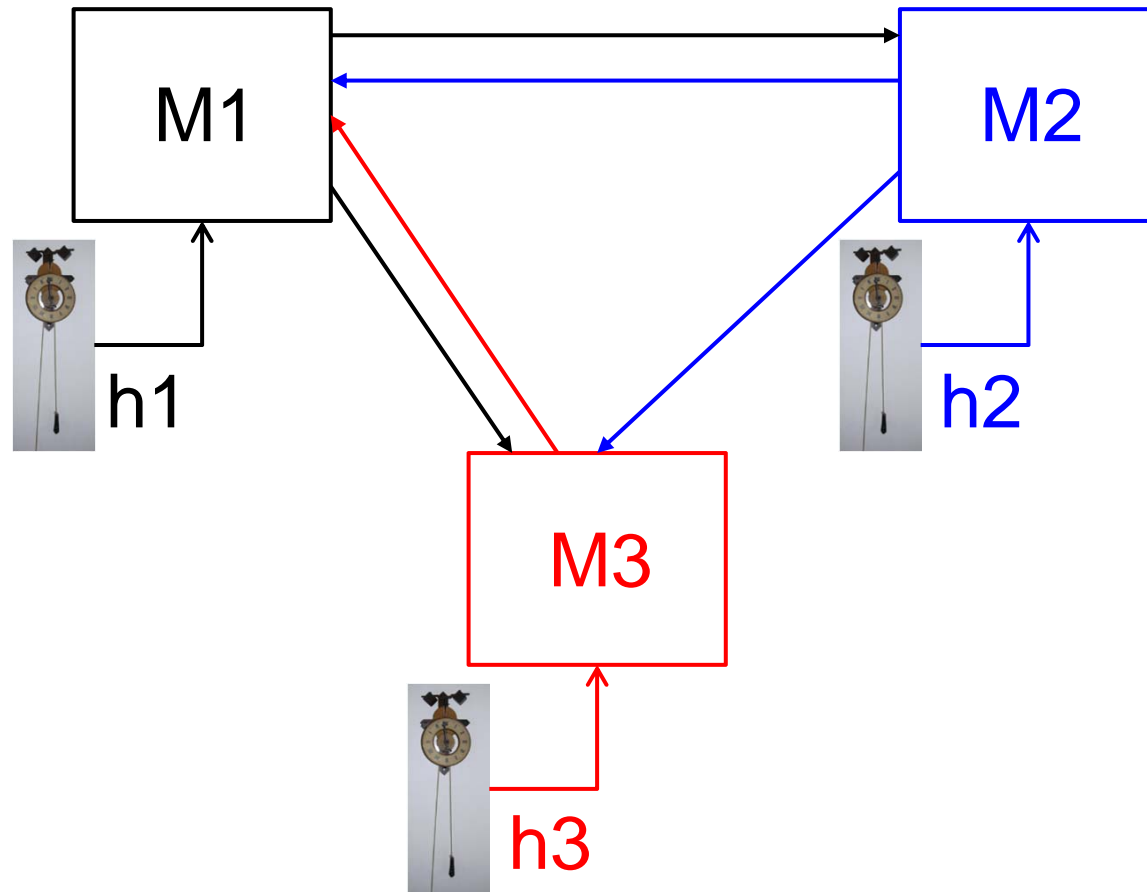
weak suspend

p
when **Susp**

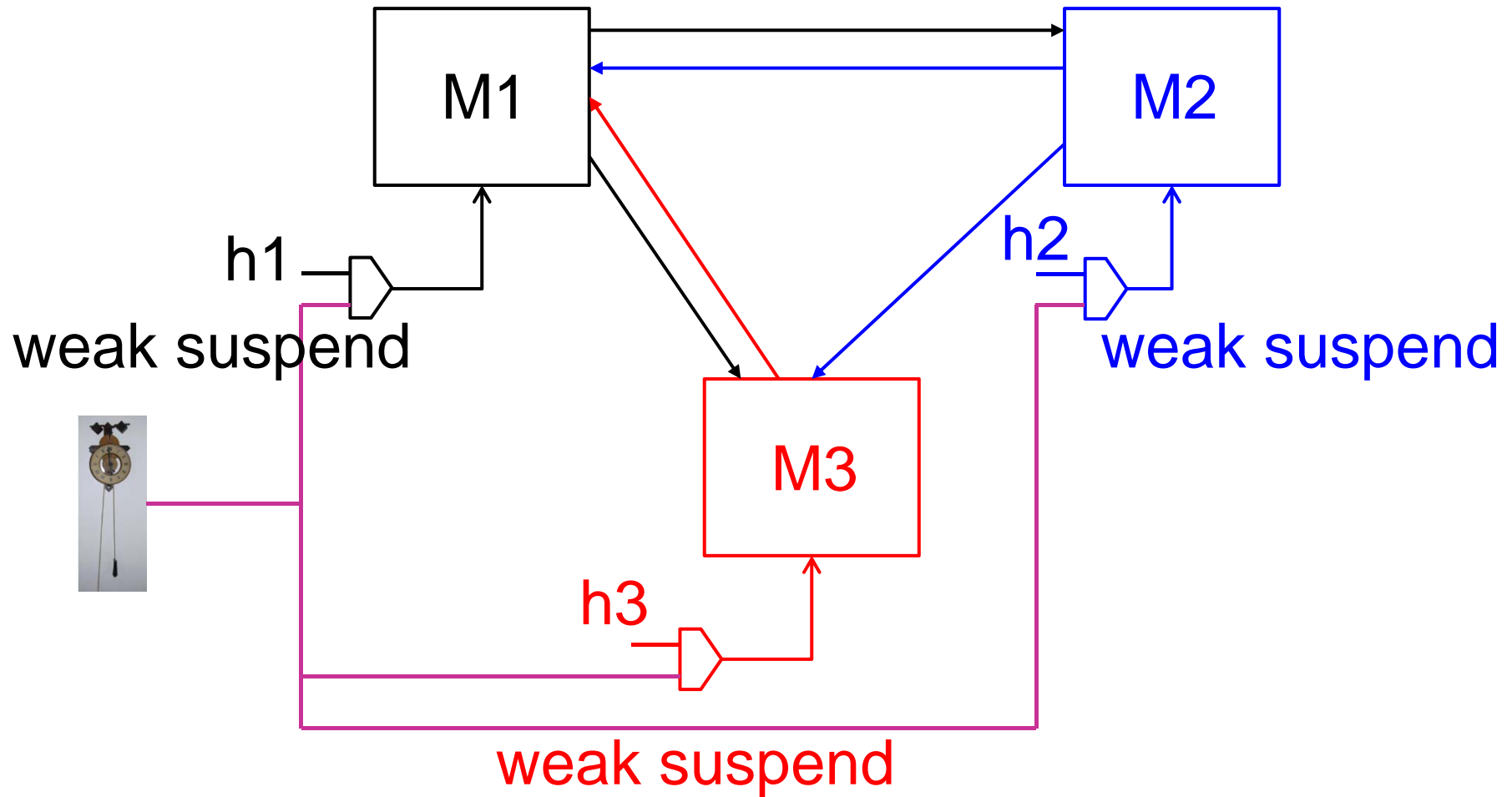


- le clock gating devient un **opérateur standard**, avec une **sémantique standard**
- Il se compile soit par un vrai clock gating, soit par un multiplexeur (option du compilateur)

Circuits multi-horloges GALS



Horloge rapide fictive



Esterel multi-horloges

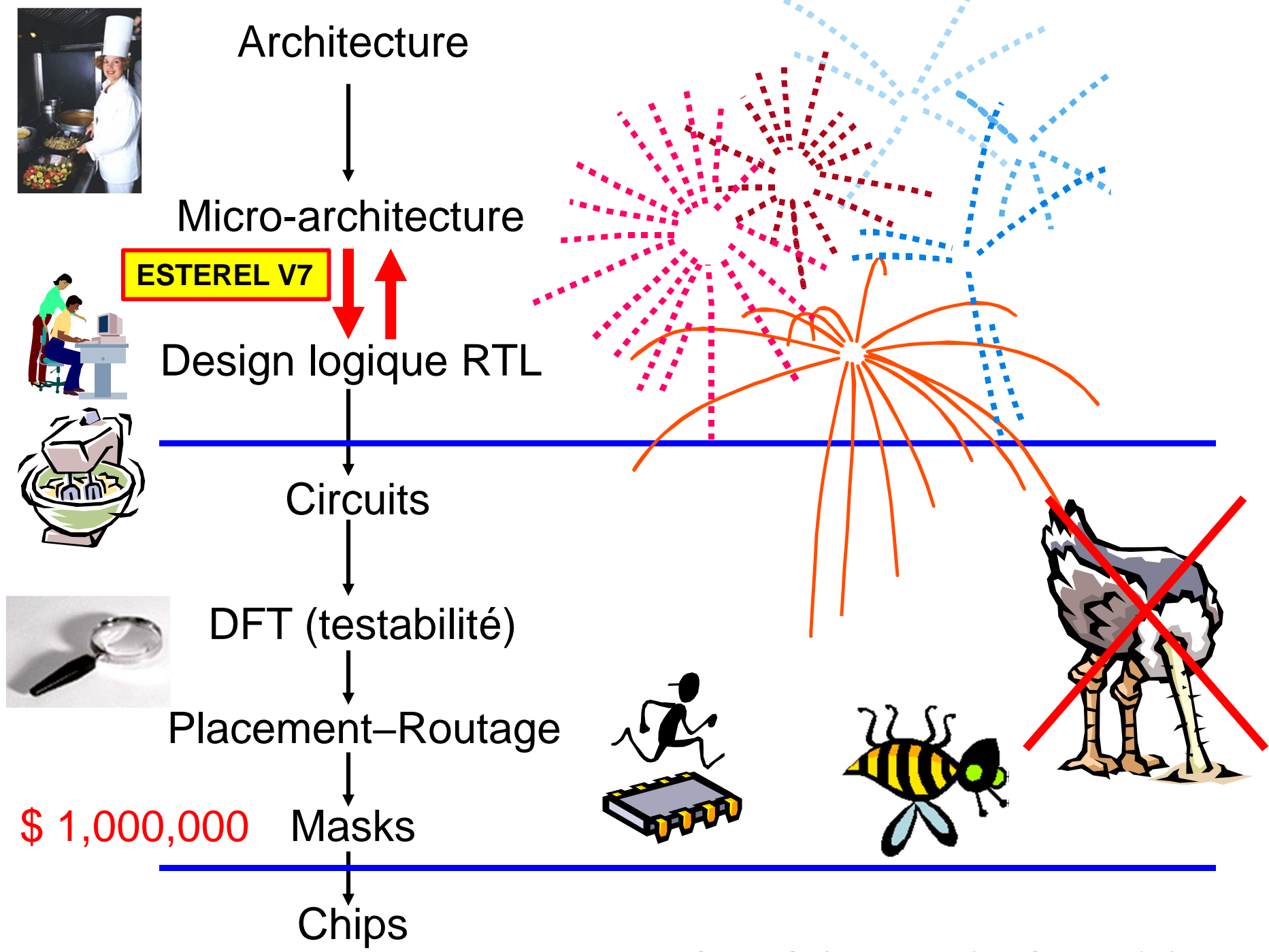
```
multiclock Multi :  
input {h1, h2, h3} : clock ;  
input I, J, K ;  
output X, Y, Z ;  
{  
  run M1 [ clock h1 ]  
||  
  run M2 [clock h2 ]  
||  
  run M3 [clock h3 ]  
}  
end multiclock
```

```
-- horloges dérivées  
signal h4, h5 : clock in  
sustain {  
  h4 <= h1 if I,  
  h5 <= mux(J, h2, h3)  
}  
||  
  ...  
}  
end signal
```

Sémantique mathématique, ASIC multi-horloges
FPGA / logiciel par weak suspend, vérification formelle

Trois questions brûlantes

- On ne peut pas se servir à fond du langage, car il rejette des programmes évidemment corrects pour **erreur de causalité** !
- Tous les systèmes sur puce sont maintenant **multi-horloges** et utilisent le **clock-gating**.
Quelle est votre offre en ce domaine?
- Nos évaluations par la R&D sont bonnes, mais pour passer en production il est impératif de traiter les **ECOs**. Comment faites vous cela ?





Architecture



Micro-architecture

Correction de bug
trouvé sur le circuit

ESTEREL V7

modif source



Design logique RTL



Circuits



resynthèse

DFT (testabilité)



Placement-Routage



Masks

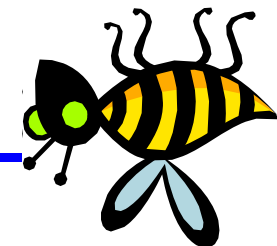
\$ 1,000,000

ECO

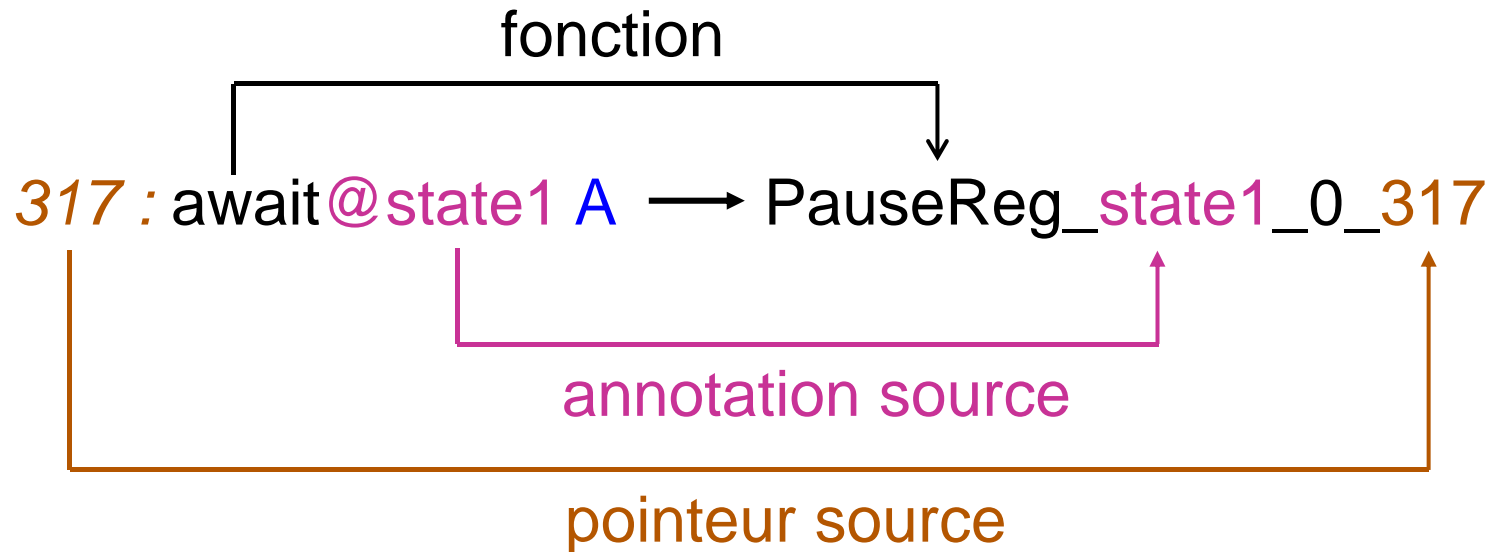


formellement !

Chips



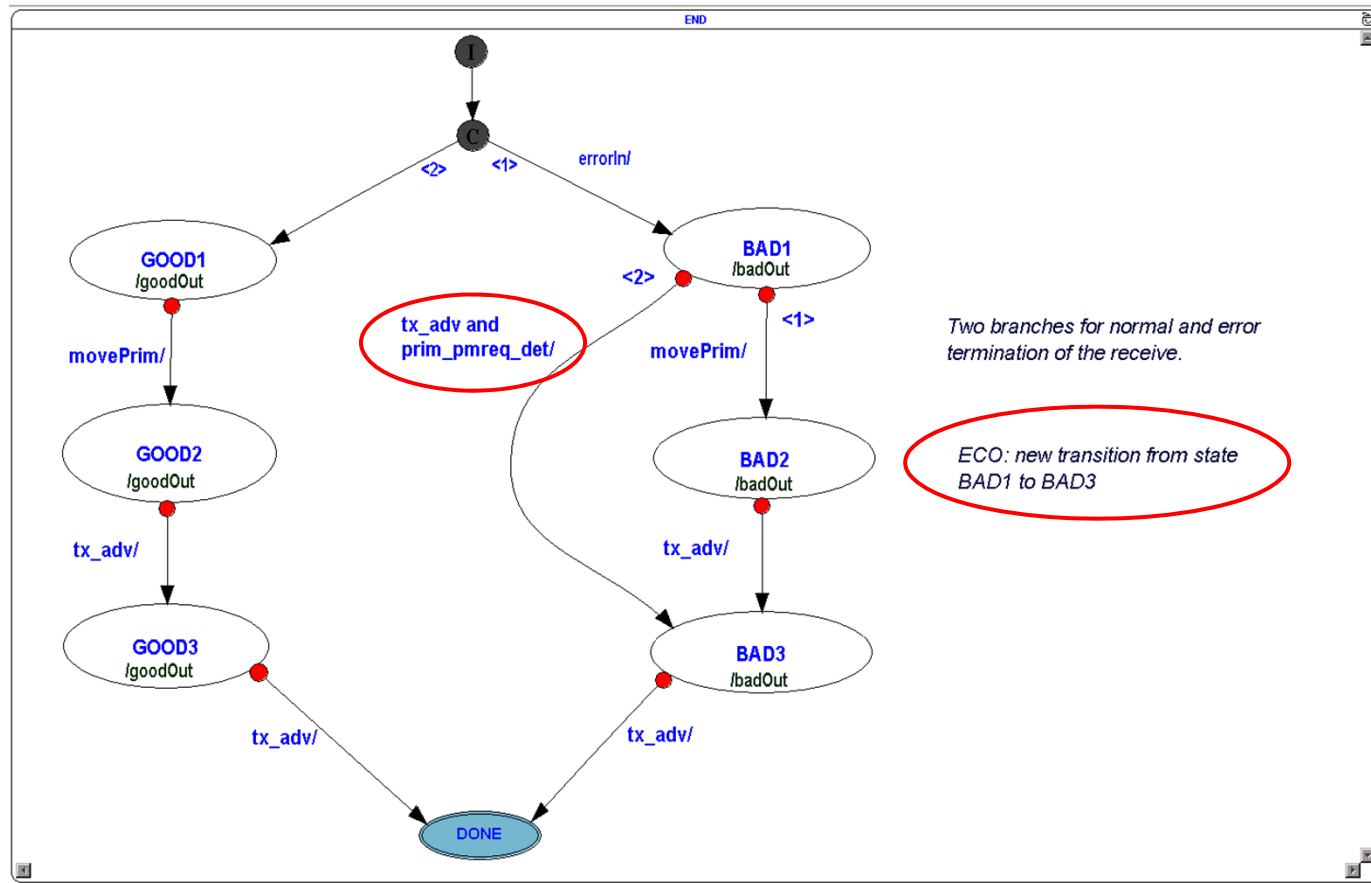
Traçabilité en HDL des noms Esterel



521 : emit A <= B → Status_A_S12_0 :=
Go_521_0 and Status_B_S5_1

Ce type de traçabilité est indispensable
pour la certification du compilateur SCADE

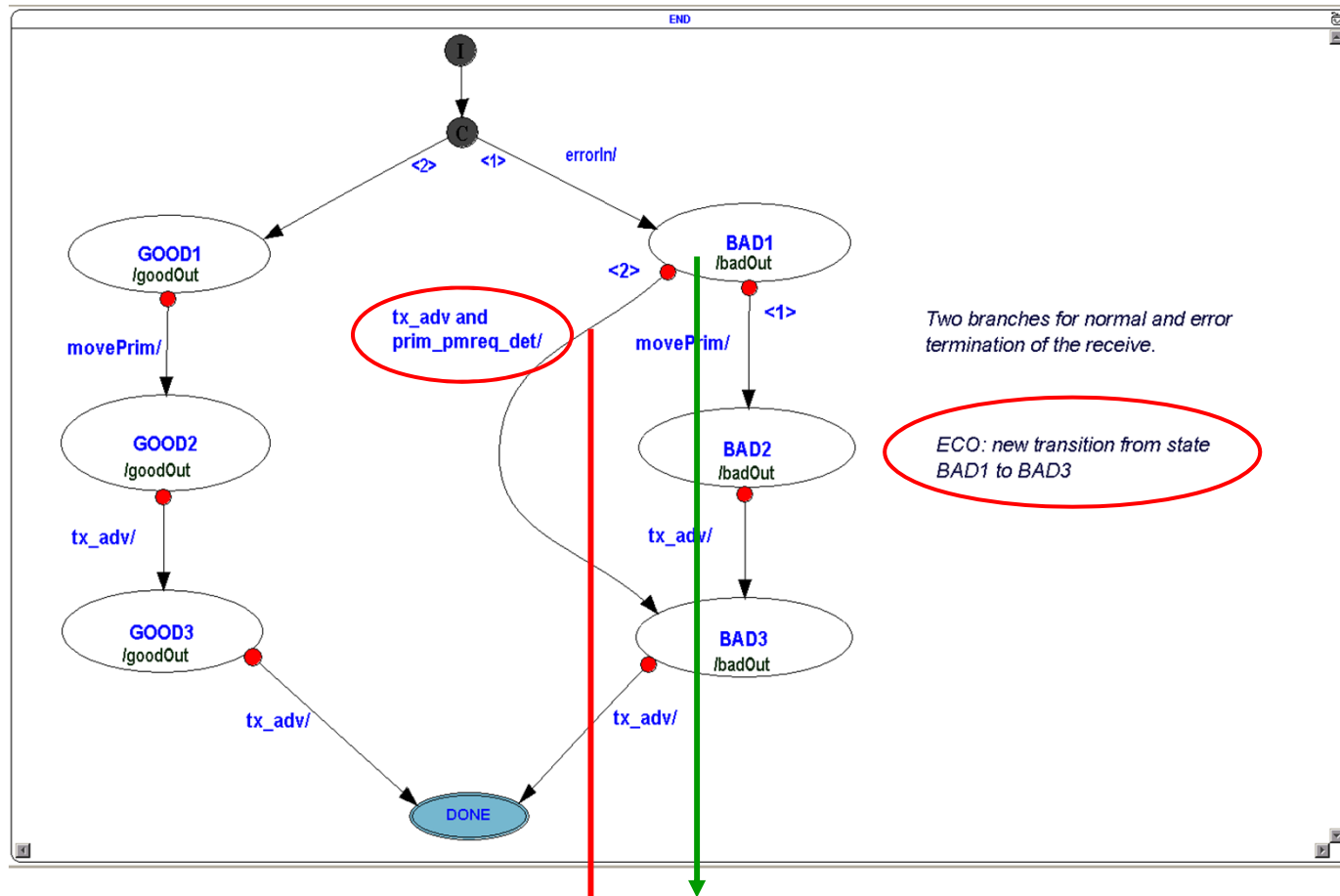
ECO d'automate : ajout d'une transition



Next (state) = Stay (state) or Enter (state)

- modifier la condition Stay de l'état source
- modifier la condition Enter de l'état cible

Modification Stay état source



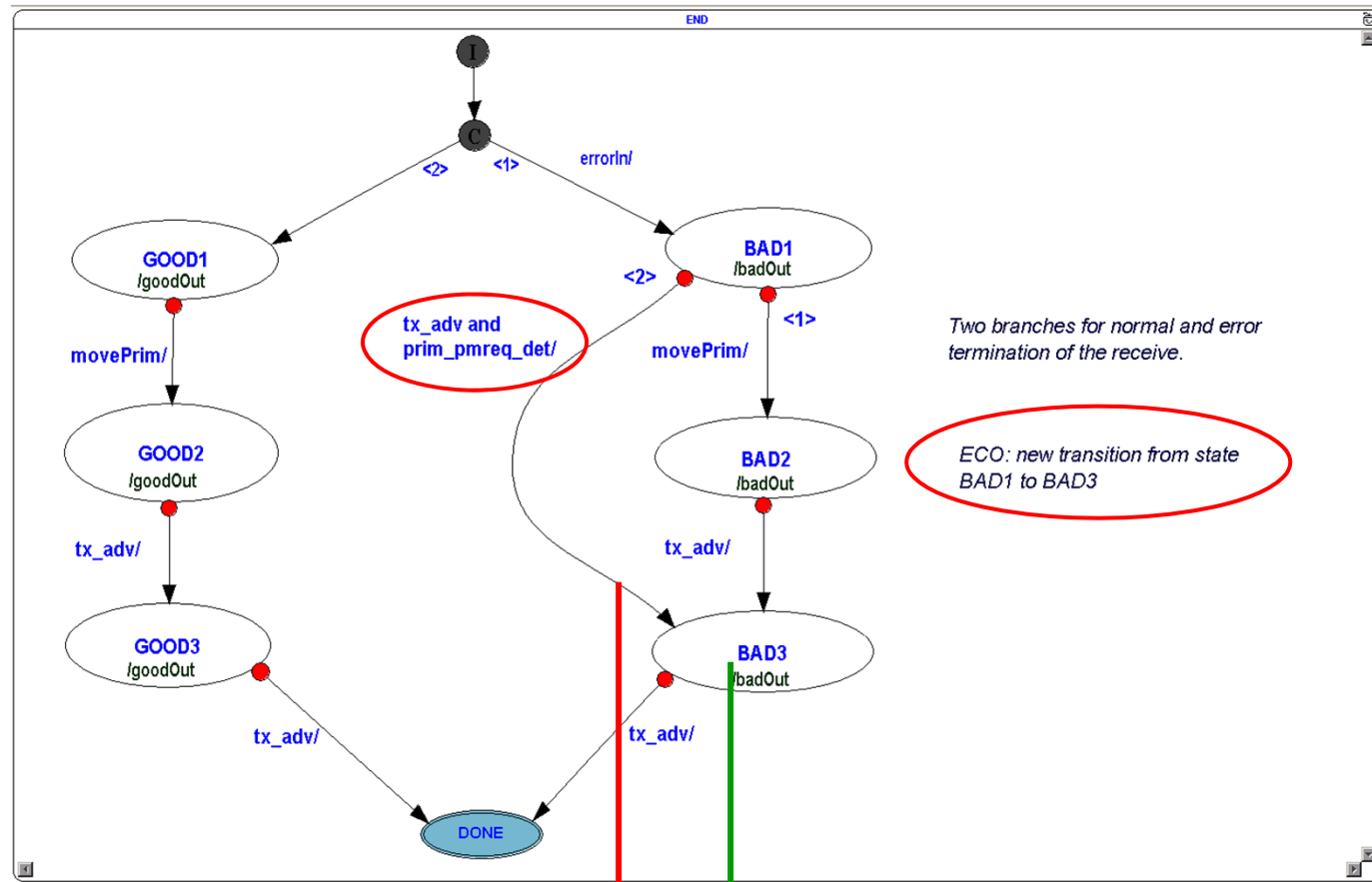
PauseRegIn_BAD1_53_0 =

Go_51_0

-- old next (BAD1)

and not (PauseReg_BAD1_53_0 -- do not take new transition
and tx_adv and prim_pmreq_det)

Modification Enter état cible

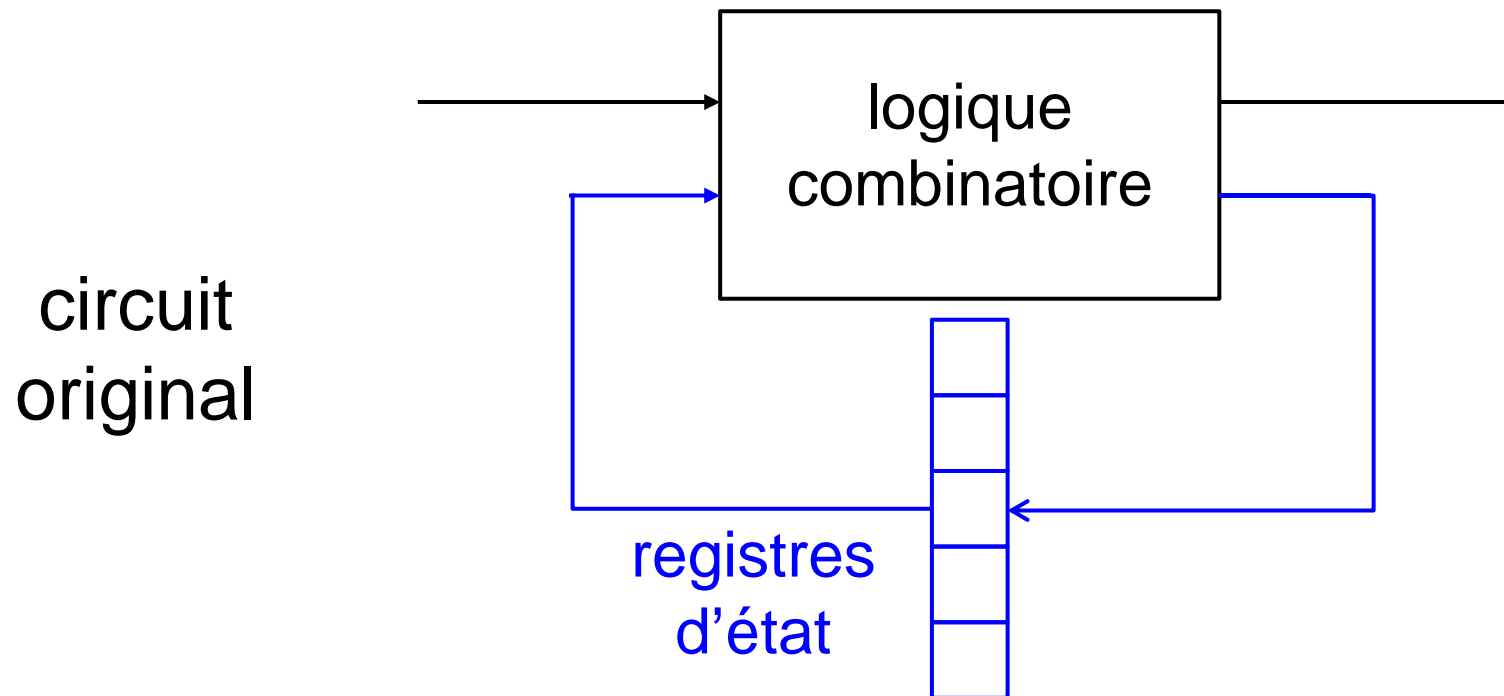


PauseRegIn_BAD3_37_0 := Go_35_0 -- old next (BAD3)
 or (PauseReg_BAD1_53_0 -- in BAD1
 and (tx_adv and prim_pmreq_det) -- trigger true
 and not phy_ready_deasserted -- do not go to IDLE state
 and not Status_movePrim_S15_0); -- do not go to BAD2 state

Mais ce n'est pas si simple....

.... car le circuit généré par Esterel est très optimisé !

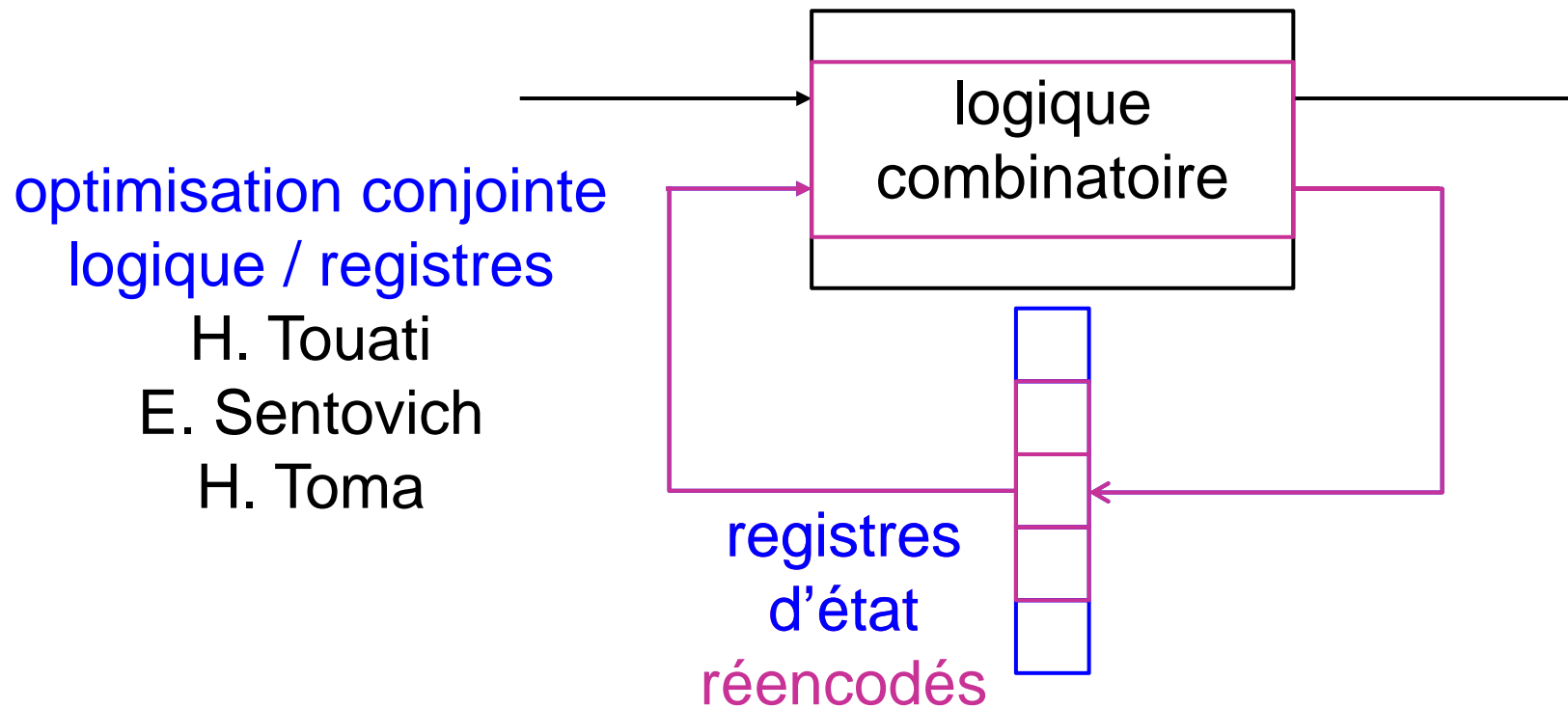
Analogie: patcher l'assembleur engendré par gcc -O4



Mais ce n'est pas si simple....

.... car le circuit généré par Esterel est très optimisé !

Analogie: patcher l'assembleur engendré par gcc -O4



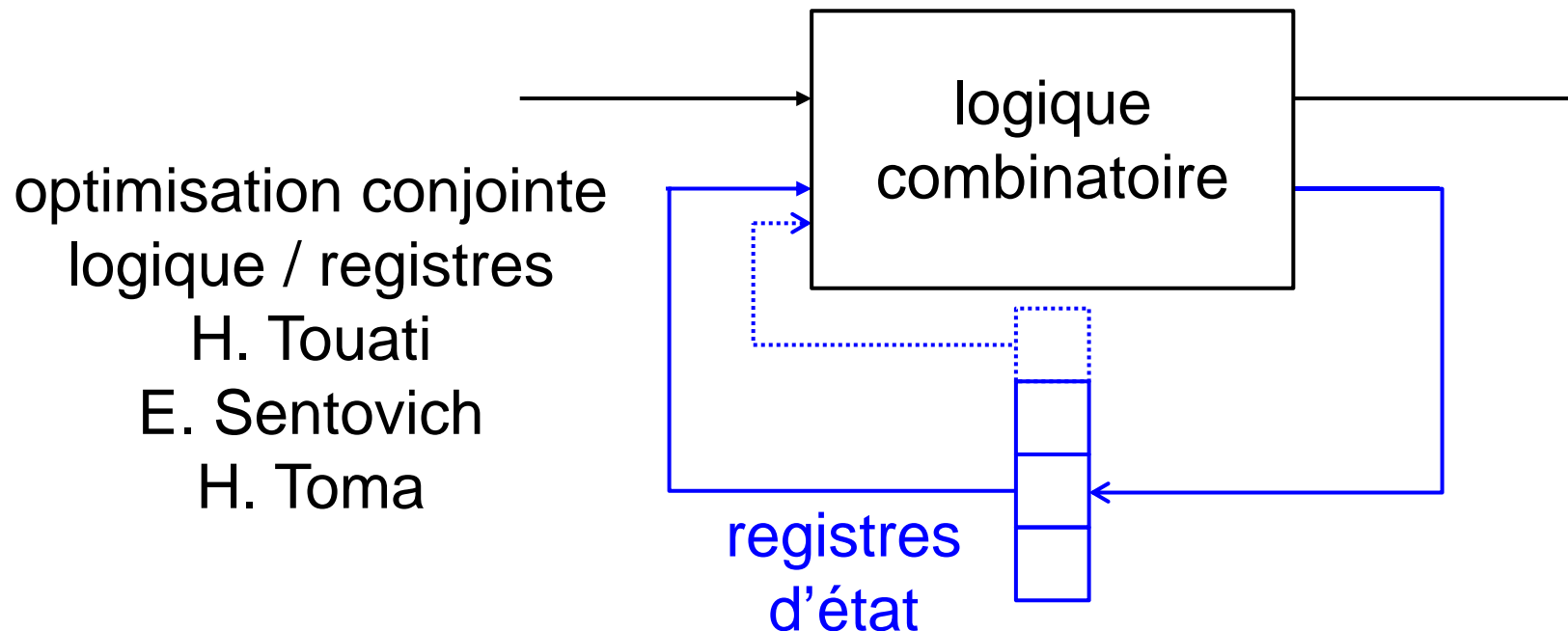
La bonne méthode

- Rendre l'optimisation **réversible** (peu de perte)
- Reconstruire les objets manquants si nécessaire

Mais ce n'est pas si simple....

.... car le circuit généré par Esterel est très optimisé !

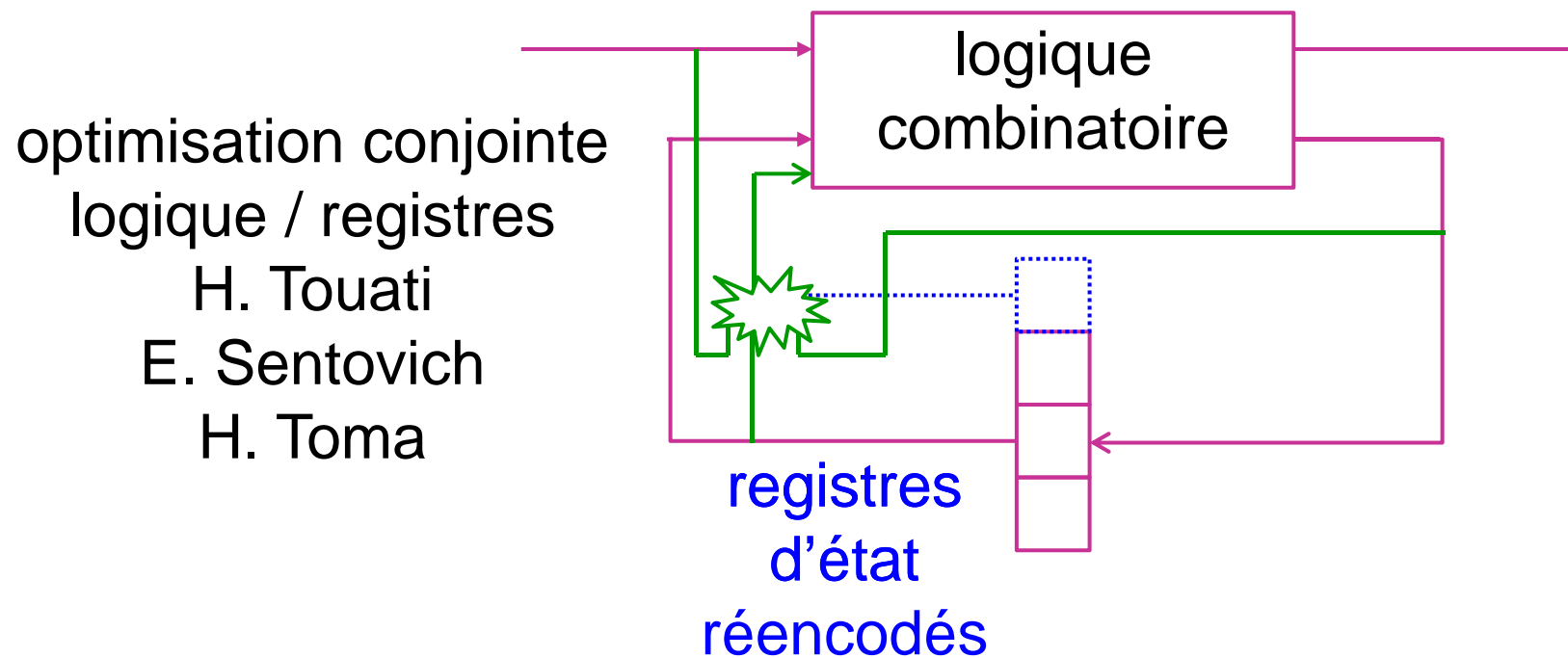
Analogie: patcher l'assembleur engendré par gcc -O4



Mais ce n'est pas si simple....

.... car le circuit généré par Esterel est très optimisé !

Analogie: patcher l'assembleur engendré par gcc -O4



La bonne méthode

- Rendre l'optimisation **réversible** (peu de perte)
- Reconstruire les objets manquants si nécessaire
- Finir l'ECO de façon standard (toujours difficile)
- Prouver formellement la correction finale
nouveau source \Leftrightarrow **nouveau circuit patché**

Ce traitement des ECOs
est **indispensable** pour tout circuit en production
... mais peu connu dans la recherche académique

Effets de bords inattendus

- Pour toute équipe de design, on crée une équipe de vérification
 - Qui est jugée sur le nombre de bug qu'elle trouve
 - Mais les designs ont bien moins de bugs:
 - langage plus clair et formel
 - simulation de haut niveau
 - vérification formelle très tôt dans le cycle
- ⇒ l'équipe de vérification se plaint d'être mal jugée
à cause de l'équipe de design!

Pas facile de convaincre le management
qu'il y gagne au total !

Conclusion

- Quand on a de vrais clients,
on ne peut rien mettre sous le tapis
- Ils sont très exigeants, et ont raison de l'être
- Les problèmes qu'ils rencontrent sont souvent plus durs et surtout plus gros que ceux que les chercheurs se posent eux-mêmes
- Et les résoudre demande de la vraie recherche !

Faire semblant de résoudre les problèmes d'un coup de hack se paye toujours cher par la suite.
Seule l'approche scientifique est vraiment pérenne.

Références (voir page personnelle)

[The Constructive Semantics of Esterel](#)

G. Berry. Livre web, 3.0, 2002.

[Reactive C: An Extension of C to Program Reactive Systems](#)

F. Boussinot

Software Practice and Experience, 21(4), avril 1991, pp. 401--428

[Constructive Boolean Circuits and the Exactness of Timed Ternary Simulation](#)

M. Mendler, T. Shiple et G. Berry

Formal Methods in System Design, Vol.40, No.3, pp. 283-329, Springer (2012)

[Clocking Schemes in Esterel](#)

L. Arditi, G. Berry, M. Kishinevsky et M. Perreaut

Proc. *Designing Correct Circuits DCC'06*, Vienna, Austria

[Late Design Changes \(ECOs\) for Sequentially Optimized Esterel Designs](#)

L. Arditi, G. Berry et M. Kishinevsky

Proc. *Formal Methods in Computer Aided Design (FMCAD'04)*, Austin, Texas, USA