Applications des BDDs dans la conception de circuits intégrés

Jean Christophe Madre
Mentor Graphics

Patrick Vuillod
Synopsys

Collège de France

9 Mars 2016

Plan de la présentation

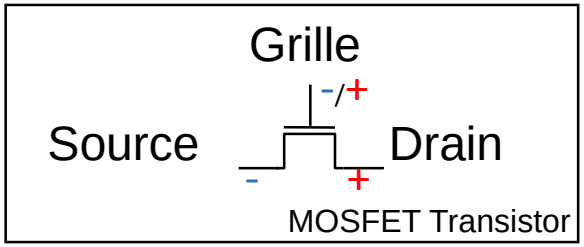
- Conception des circuits numériques
- Synthèse logique
- Réduction de la consommation
- Conclusion

Plan de la présentation

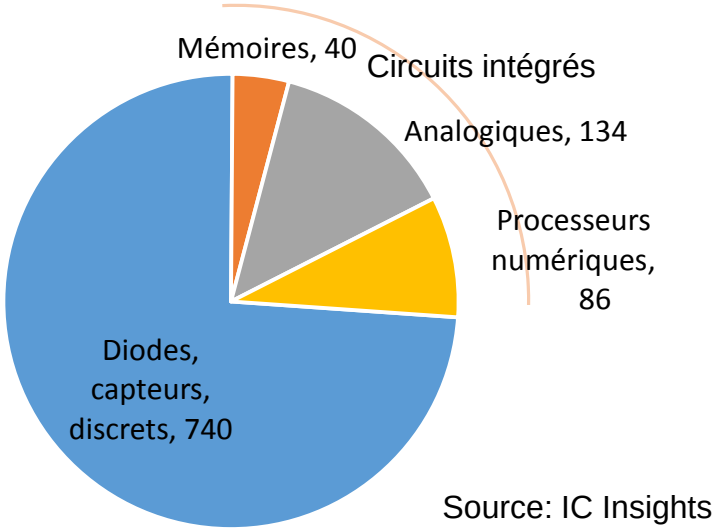
- Conception des circuits numériques
- Synthèse logique
- Réduction de la consommation
- Conclusion

Industrie des semi-conducteurs

- Matériaux dont on peut faire varier dynamiquement la conductivité
- 1947: Invention du transistor



- 1963: Invention de la technologie CMOS (Complementary Metal-Oxyde Semiconductor)
- 2016: 1000E9 composants produits en 2016

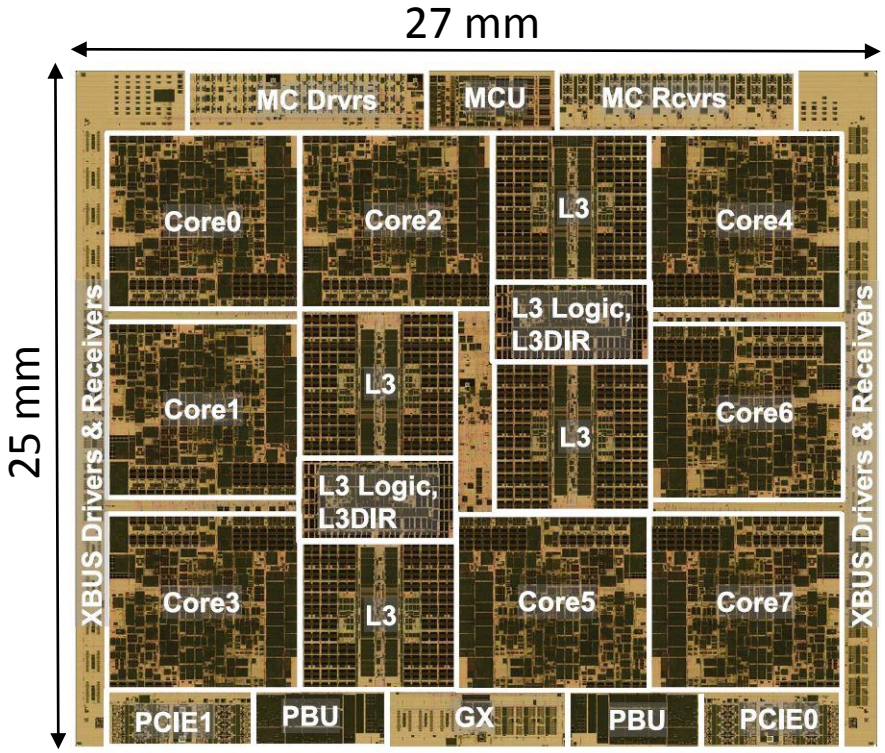


Source: IC Insights

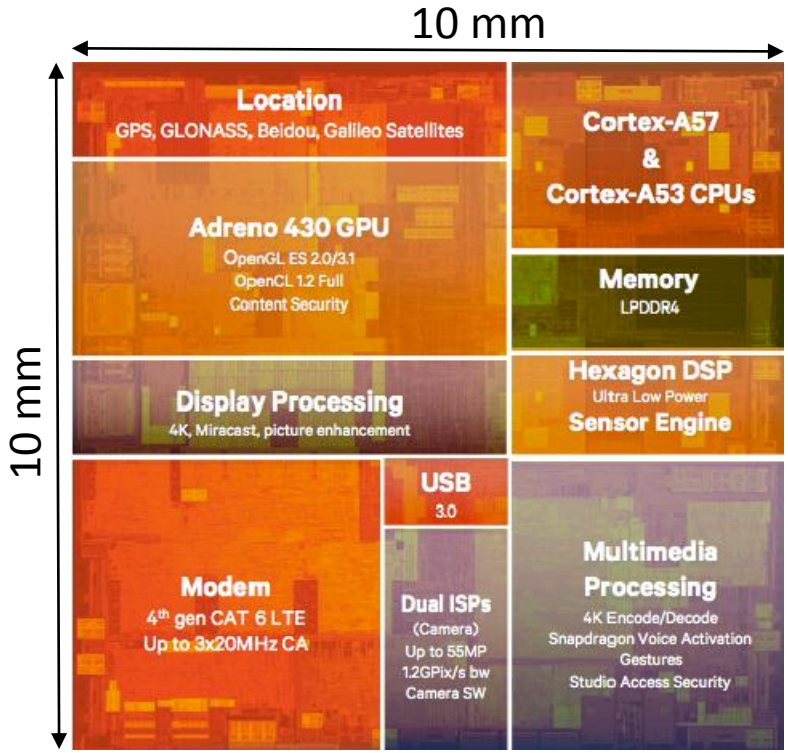
Processeurs numériques

- Composant / circuit intégrant unité(s) de calcul, mémoire, contrôleur d'entrées/sorties
- Modes de production
  - Fabrication par lithographie optique sur silicium
  - Configuration de circuit FPGA (Field Programmable Gate Array)
- État de l'art (2015)
  - Calcul: Intel 18-core Xeon, 5.6E9 transistors, 661mm2, IBM 8-core z13, 4E9 transistors, 678mm2
  - Graphique: AMD Radeon R9, 8.9E9 transistors, 596mm2
  - Mobile: Apple A8, Qualcomm Snapdragon 810, Mediatek MT6595, ~2E9 transistors, ~100mm2
  - FPGA: Xilinx XCVU440, 20E9 transistors (3 circuits)
- Unités de mesure
  - Nano-mètre, femto-seconde, pico-watt, pico-farad

Processeurs numériques (exemples)



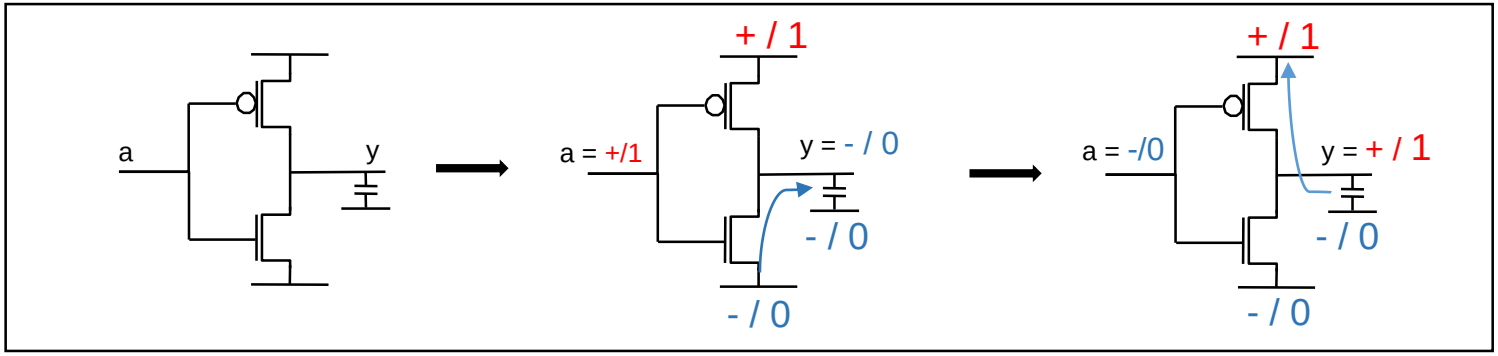
IBM z13
Source: IBM



Qualcomm Snapdragon 810
Source: Qualcomm

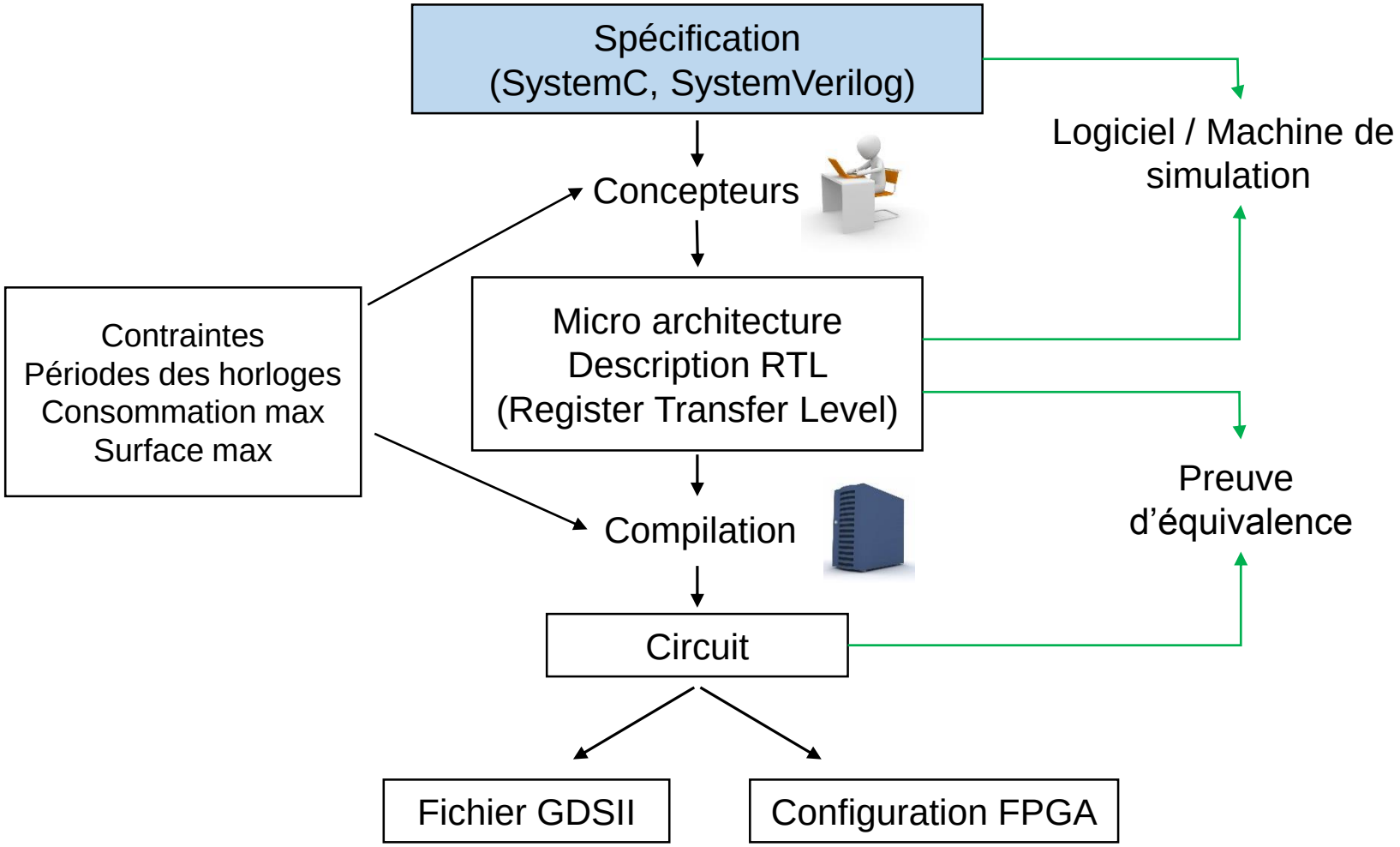
Fonctionnement des circuits numériques

- Machine qui stocke et fait circuler des électrons entre V- et V+
- Etat stable: tous les fils sont soit à V+ soit à V-
- Abstraction 1: + ≡ 1 et - ≡ 0, le circuit devient Booléen
- Inverseur CMOS ($y = \overline{a}$)

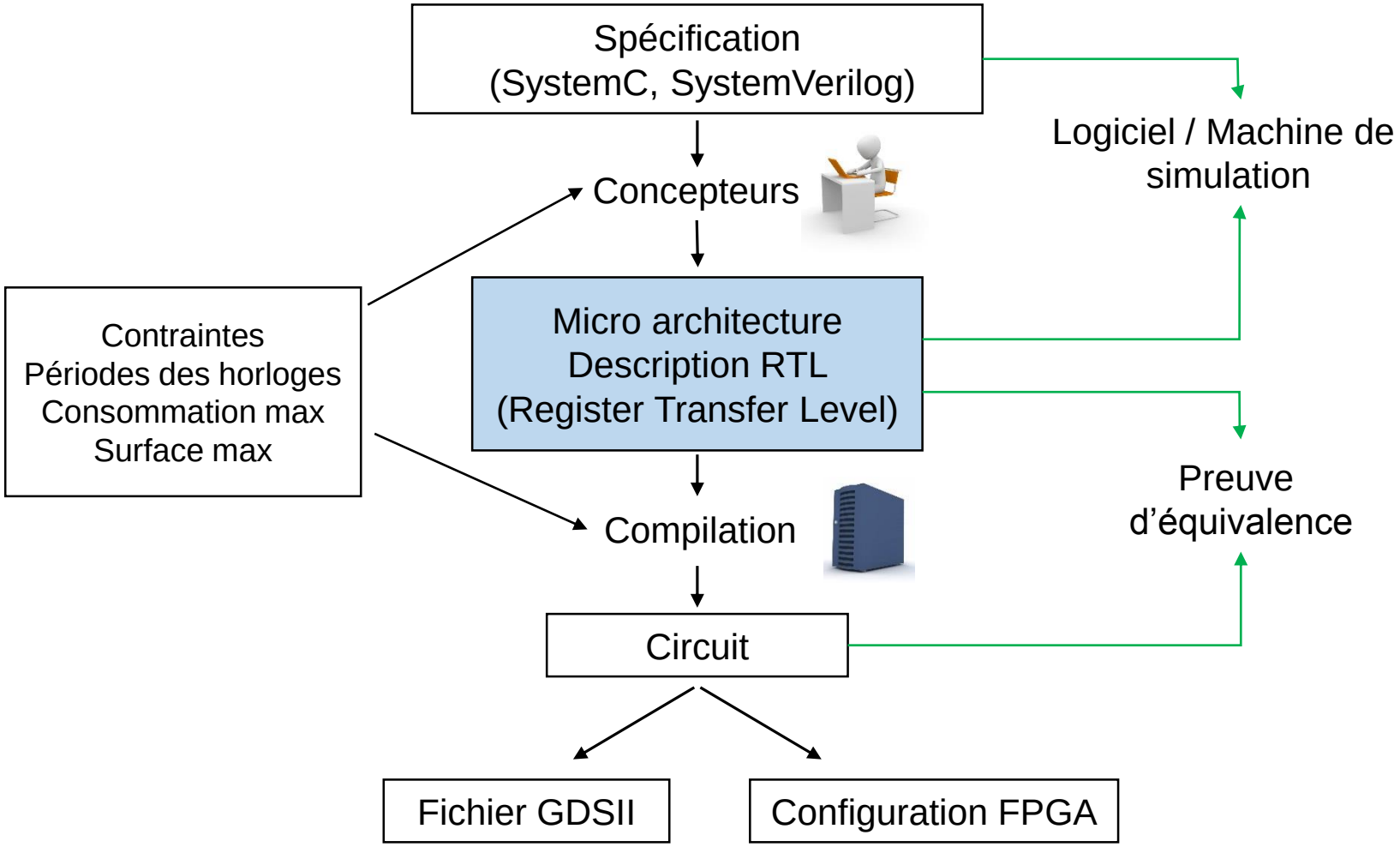


- Abstraction 2: Fonctionnement synchrone
  - Cycle: 'état stable → commutation → état stable' rythmé par une horloge
- Attention!!! : Ces abstractions doivent être vérifiées

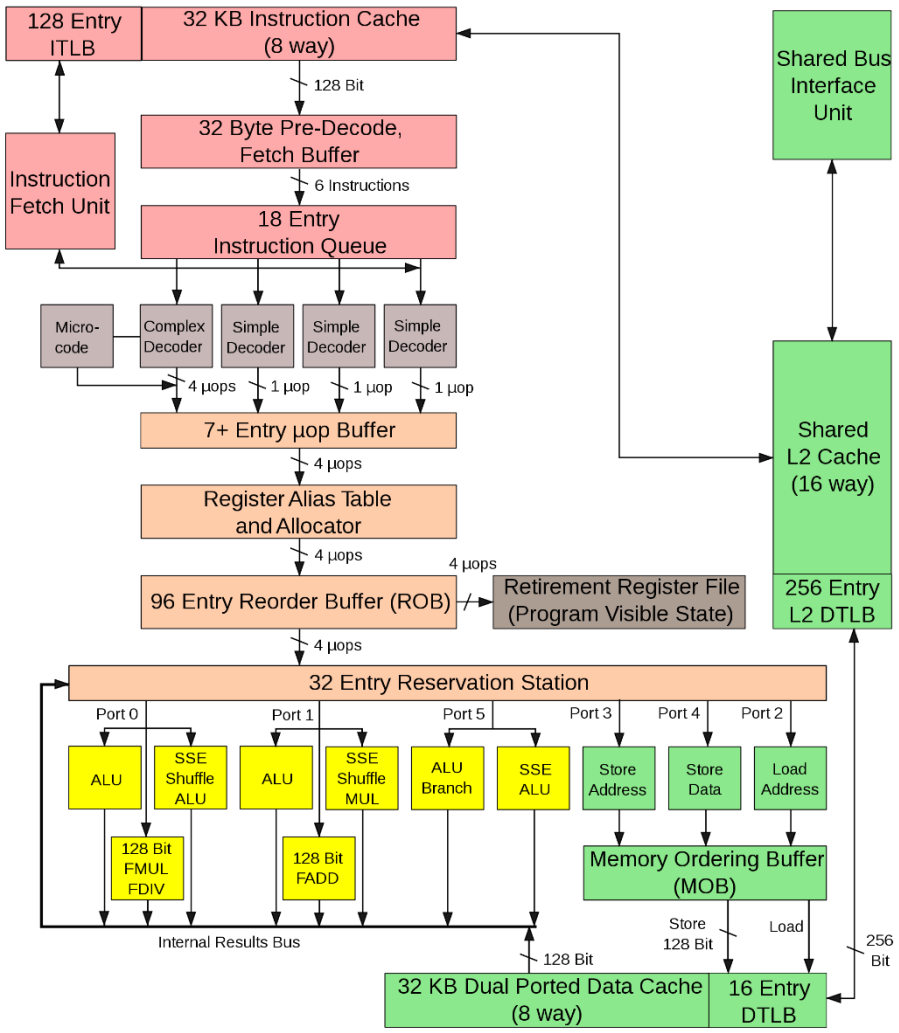
Conception des processeurs numériques



Conception des processeurs numériques (suite)



Conception des processeurs numériques (suite)



Exemple de micro-architecture - Source: Intel

Conception des processeurs numériques (suite)

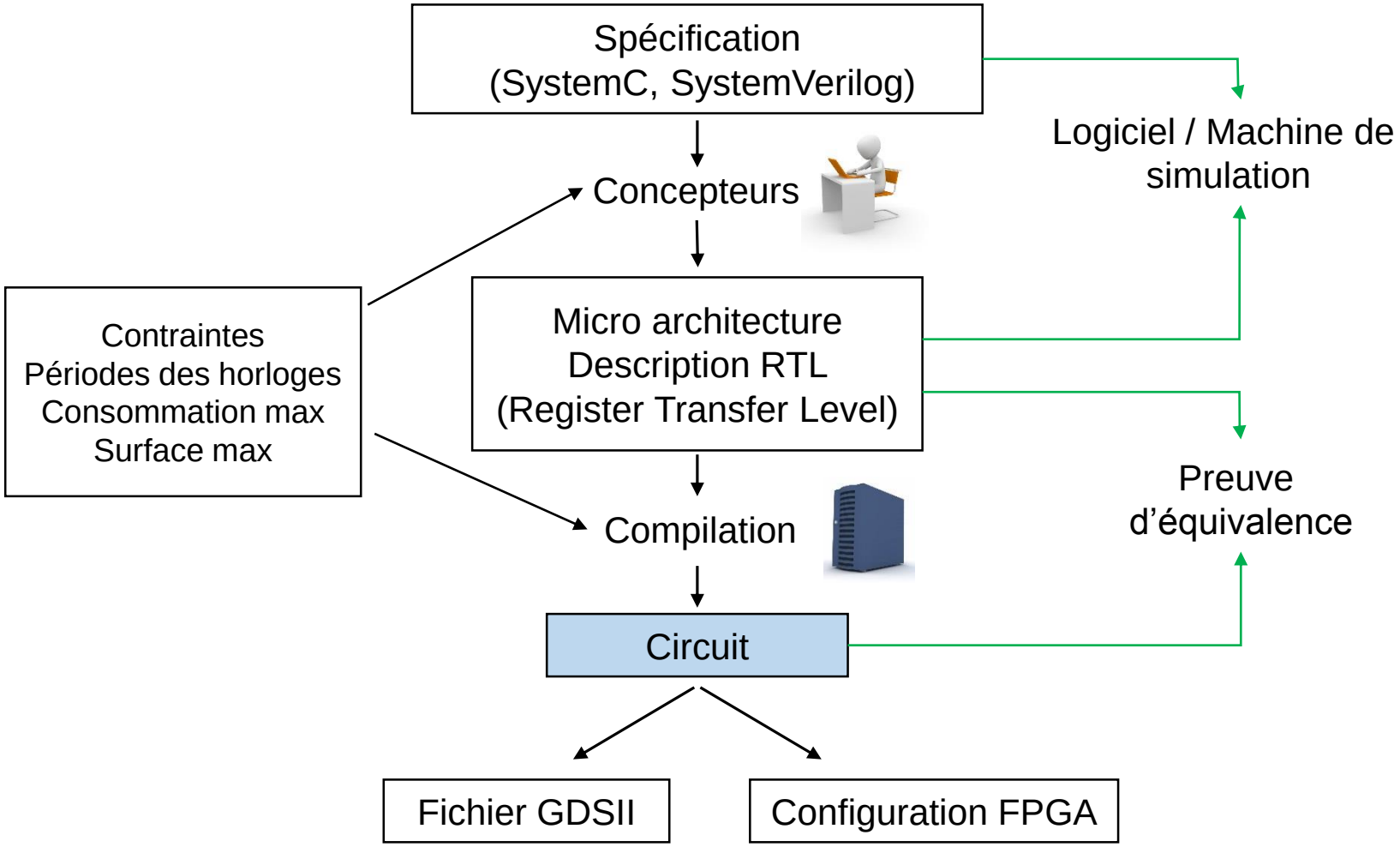
Calcul de la sortie

Calcul de l'état suivant

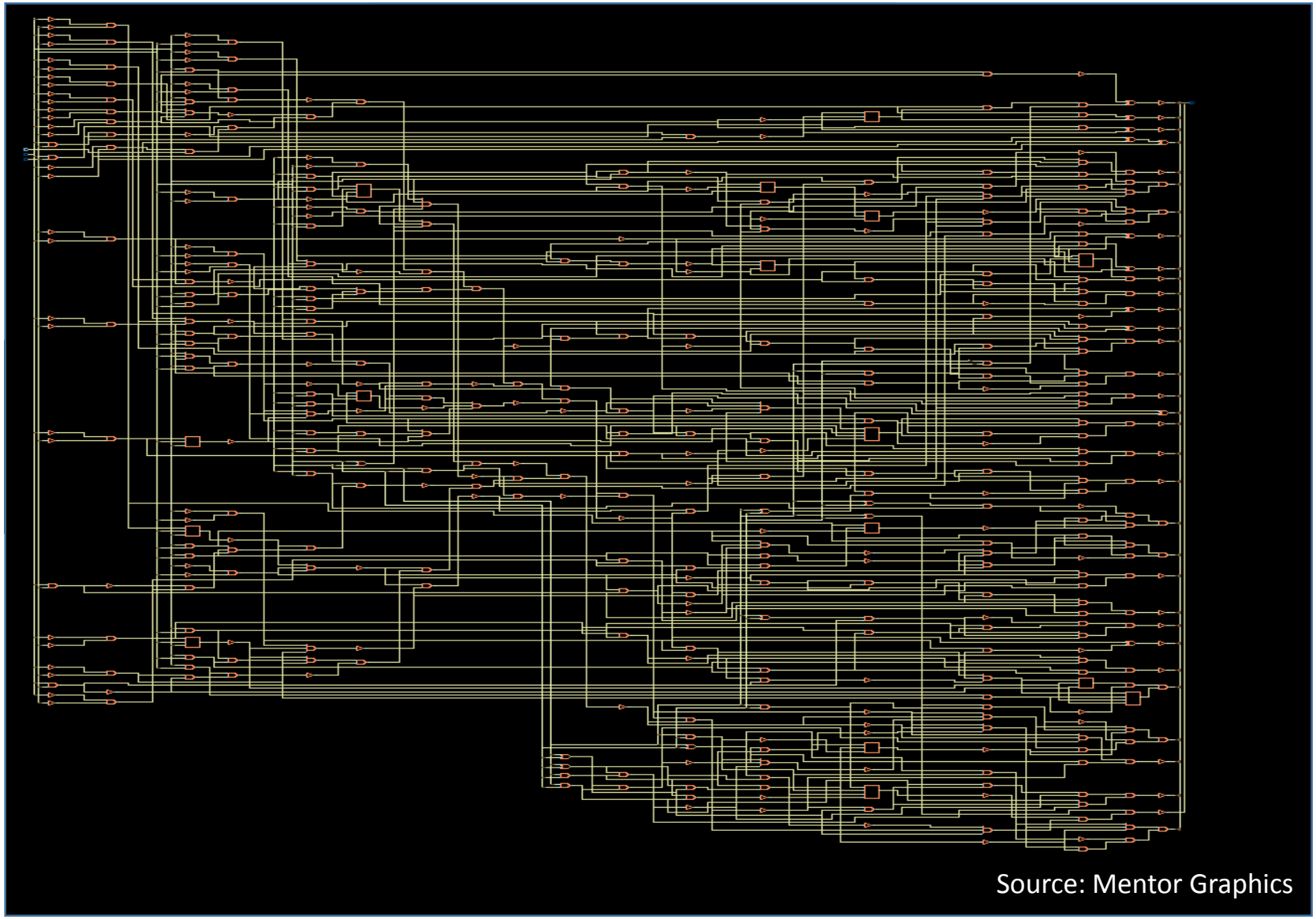
```
module statem(clk, in, reset, out);
input clk, in, reset; output [3:0] out;
reg [1:0] state;
parameter zero=0, one=1, two=2, three=3;

always @(state)
  begin case (state)
    zero:    out = 4'b0000;
    one:     out = 4'b0001;
    two:     out = 4'b0010;
    three:   out = 4'b0100;
    default: out = 4'b0000;
  endcase end

always @(posedge clk or posedge reset)
  begin
    if (reset) state = zero;
    else case (state)
      zero:  state = one;
      one:   if (in) state = zero; else state = two;
      two:   state = three;
      three: state = zero;
    endcase
  end
endmodule
```

Verilog Example - Source: Altera

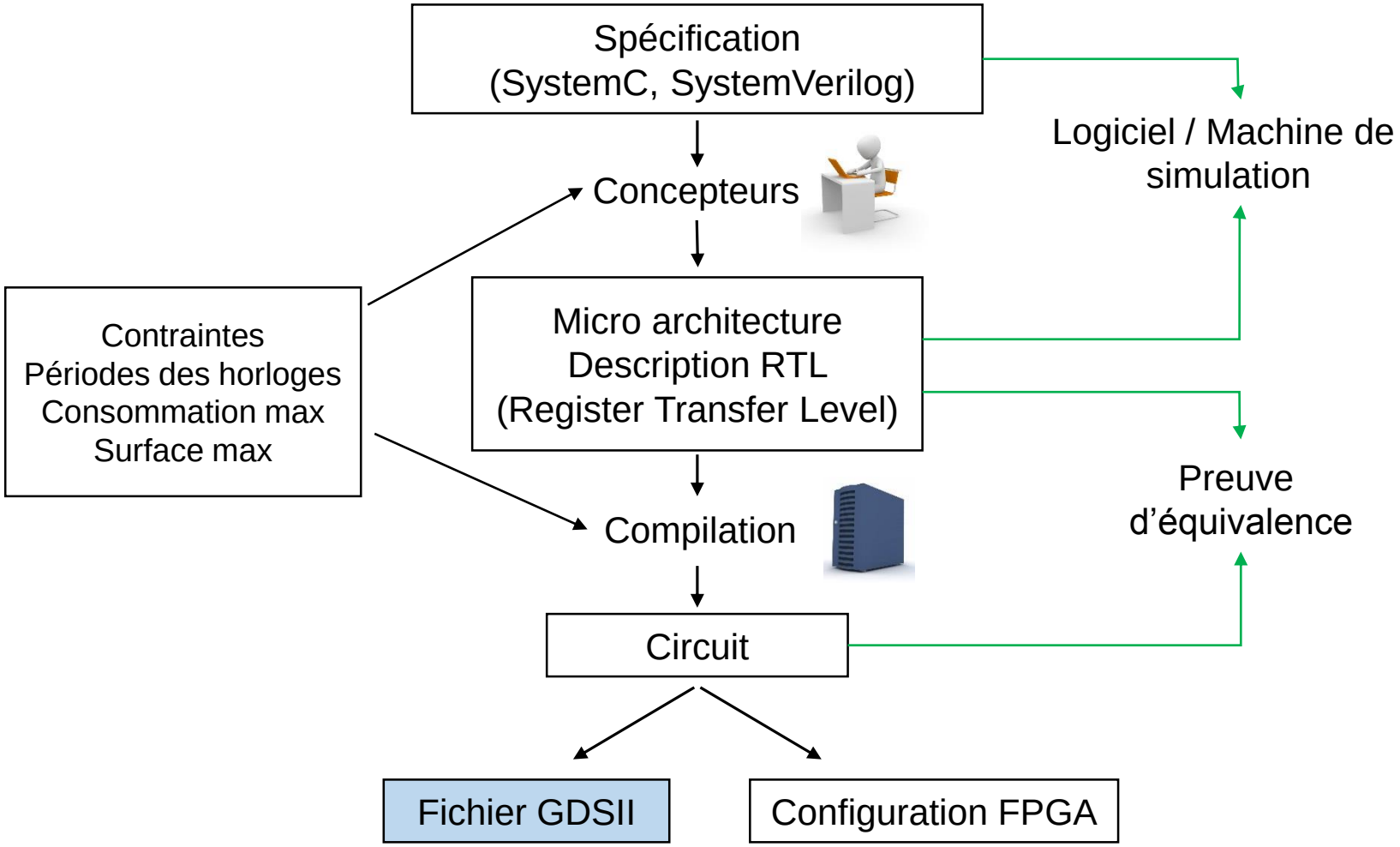
Conception des processeurs numériques (suite)



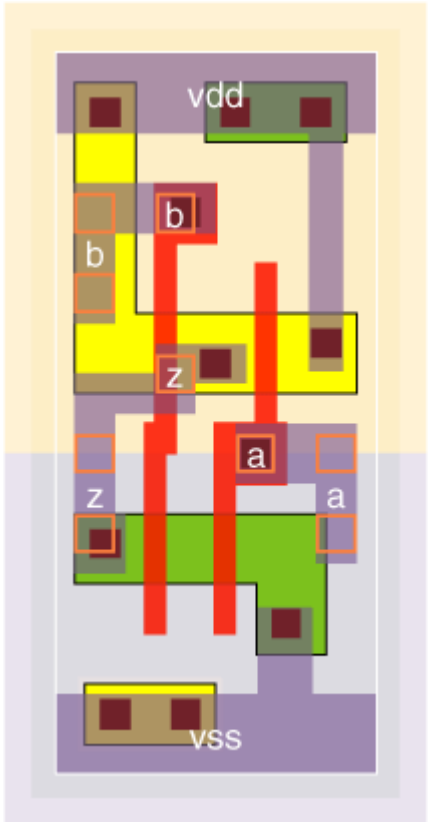
Conception des processeurs numériques (suite)



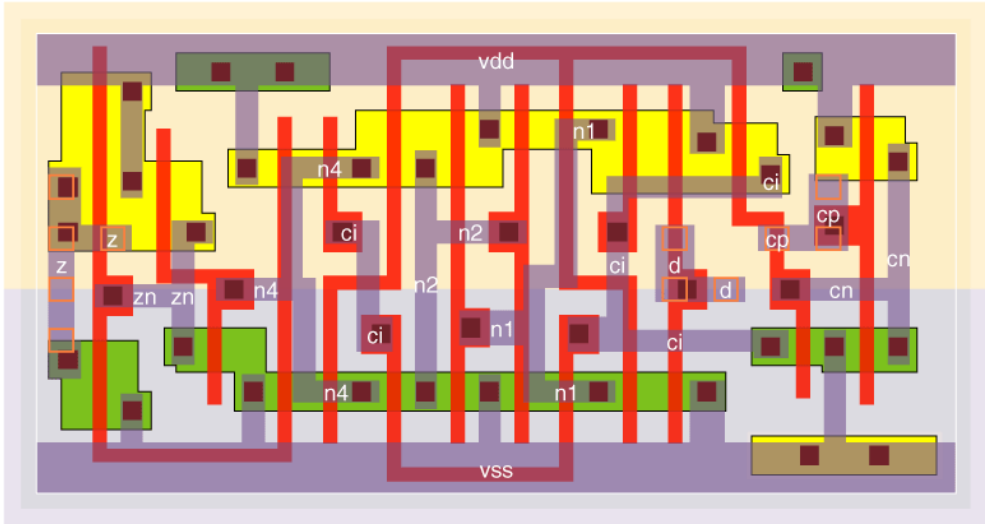
Conception des processeurs numériques (suite)



Conception des processeurs numériques (suite)



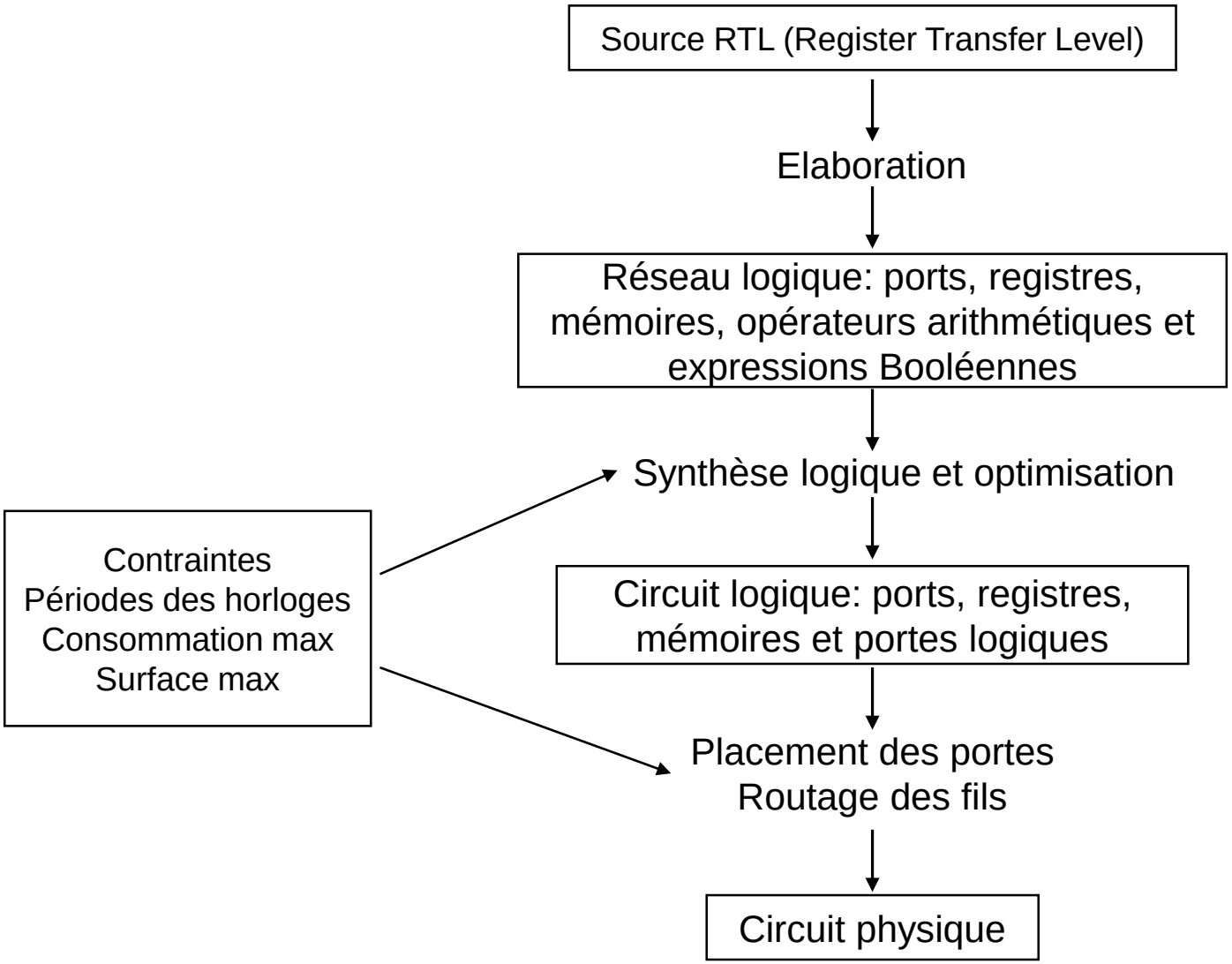
Implémentation
d'une porte NAND2



Implémentation
d'une bascule D

Source: www.vlsitechnology.org

Logiciel de compilation de circuit numérique

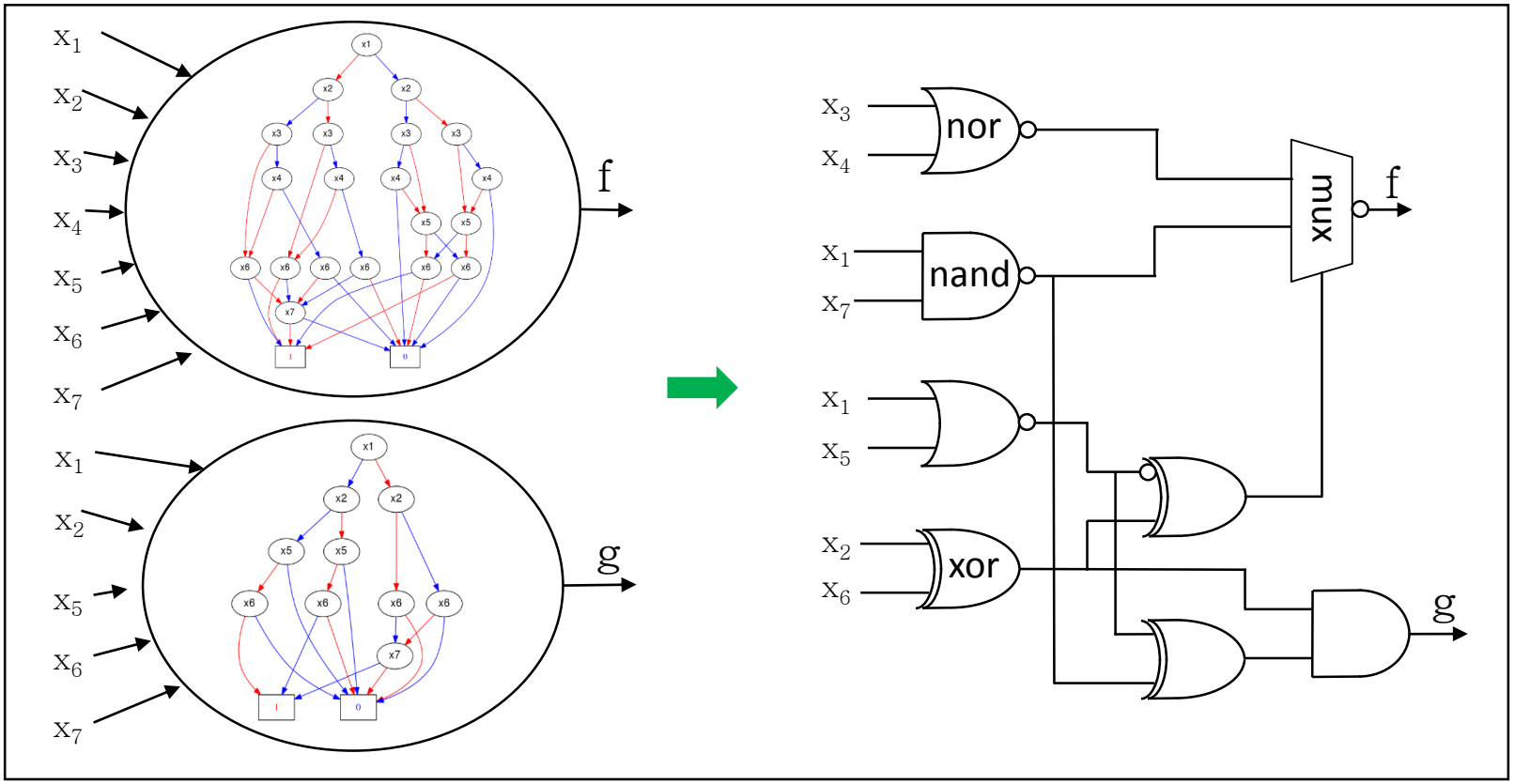


Plan de la présentation

- Conception des circuits numériques
- Synthèse logique
- Réduction de la consommation
- Conclusion

Synthèse logique

- Compilation multi-passes
  - P1 Simplification des expressions Booléennes
  - P2 Factorisation des expressions
  - P3 Génération des portes logiques

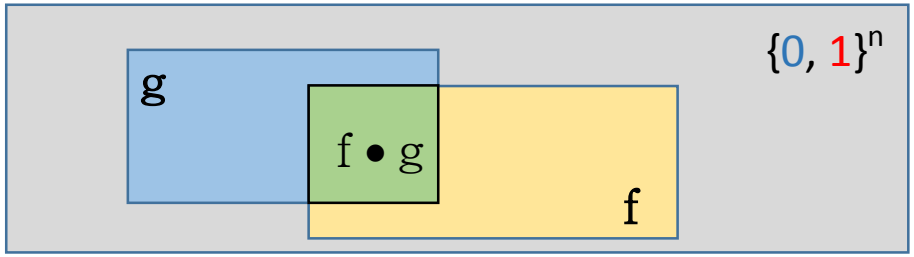


Historique

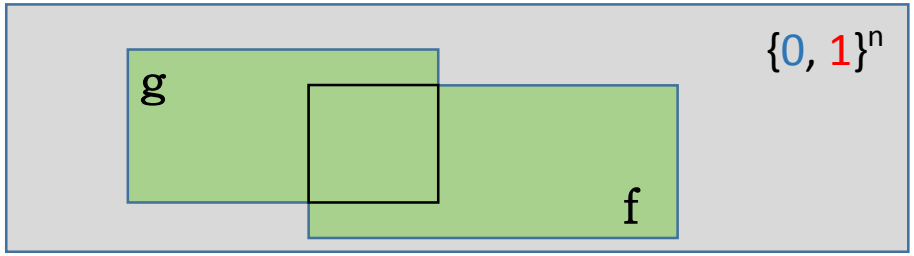
- Utilisation intensive de techniques de calcul Booléen
  - Manipulation d'expressions sous forme DNF/SOP (1980's)
    - SIS, UC Berkeley (CA)
  - Binary Decision Diagrams (BDD) (1990's)
    - BDS, U Amherst (MA), BDS-MAJ (EPF Lausanne)
  - And-Inverter Graphs (AIG) (2000's)
    - ABC, UC Berkeley (CA)
  - Solveurs SAT (2000's)
    - Chaff (U Princeton), Grasp (U Lisbon)

Fonctions Booléennes – rappels

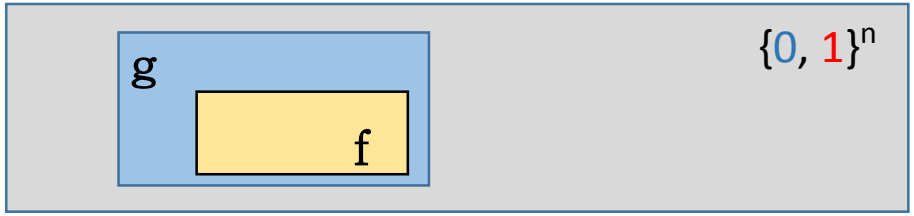
- Fonctions de $\{0, 1\}^n \rightarrow \{0, 1\}$
- $(f \bullet g)(x) = (f \wedge g)(x) = (f \cap g)(x) = f(x) \bullet g(x)$



- $(f + g)(x) = (f \vee g)(x) = (f \cup g)(x) = f(x) + g(x)$



- $f \subset g$ ssi $f \Rightarrow g$ ssi $f^{-1}(1) \subset g^{-1}(1)$

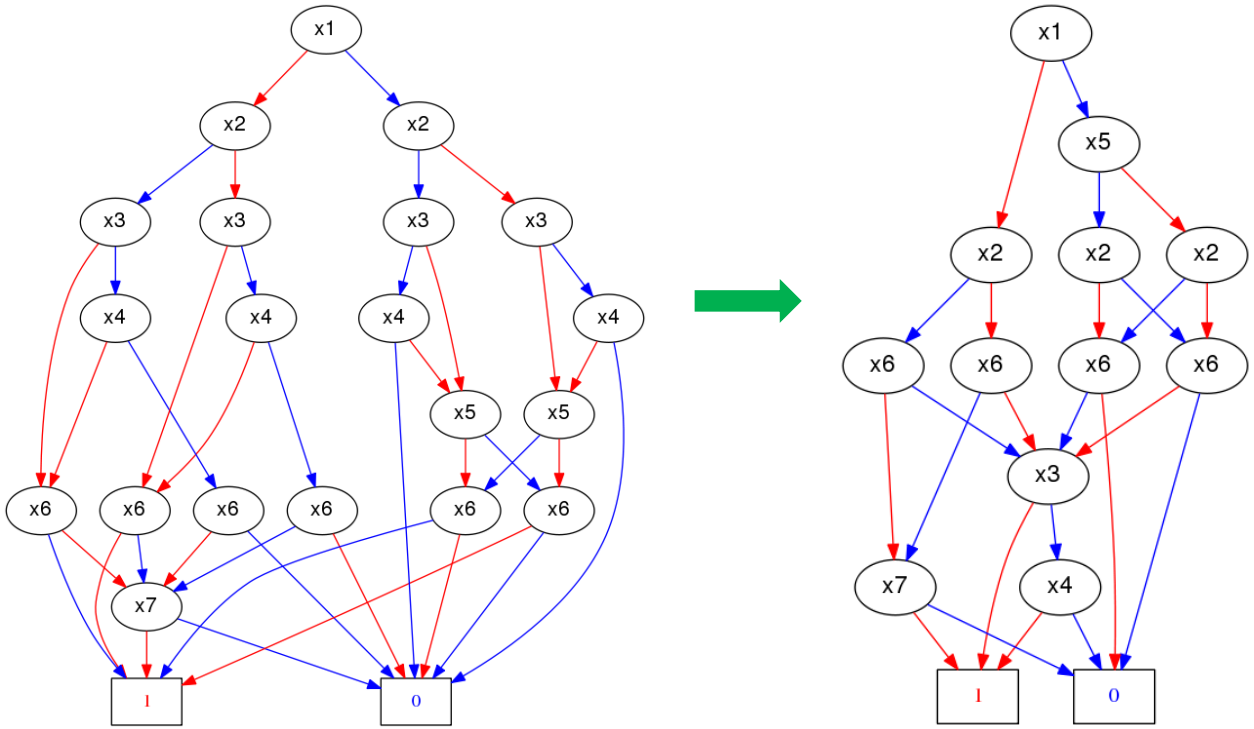


Synthèse logique

- Compilation multi-passes
  - P1 Simplification des expressions Booléennes
  - P2 Factorisation des expressions
  - P3 Génération des portes logiques

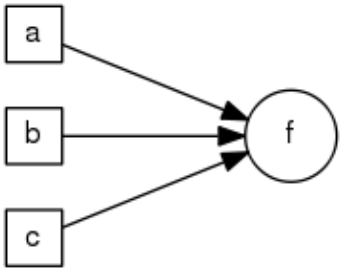
Minimisation de la taille des BDDs

- La surface du circuit est proportionnelle à la taille des BDDs des fonctions des nœuds du réseau
- Ordonnancement des variables
  - Analyse de la structure des fonctions
  - Ré-ordonnancement dynamique des variables



Minimisation de la taille des BDDs (suite)

- Utilisation de ‘care-sets’



  - Supposons que a est toujours égal à c
  - Alors on peut remplacer f par f’ telle que
    - $f'(v_a, v_b, v_c) = f(v_a, v_b, v_c)$ si $v_a = v_c$
    - $f'(v_a, v_b, v_c) = 0$ ou 1 si $v_a \neq v_c$
  - On dit que $cs(a, b, c) = (\bar{a}\bar{c} + ac)$ est un ‘**care-set**’ pour f

a	b	c	f		f’
0	0	0	1	➡	1
0	0	1	0	➡	1
0	1	0	1	➡	1
0	1	1	**1**	➡	1
1	0	0	1	➡	0
1	0	1	0	➡	0
1	1	0	0	➡	0
1	1	1	0	➡	0

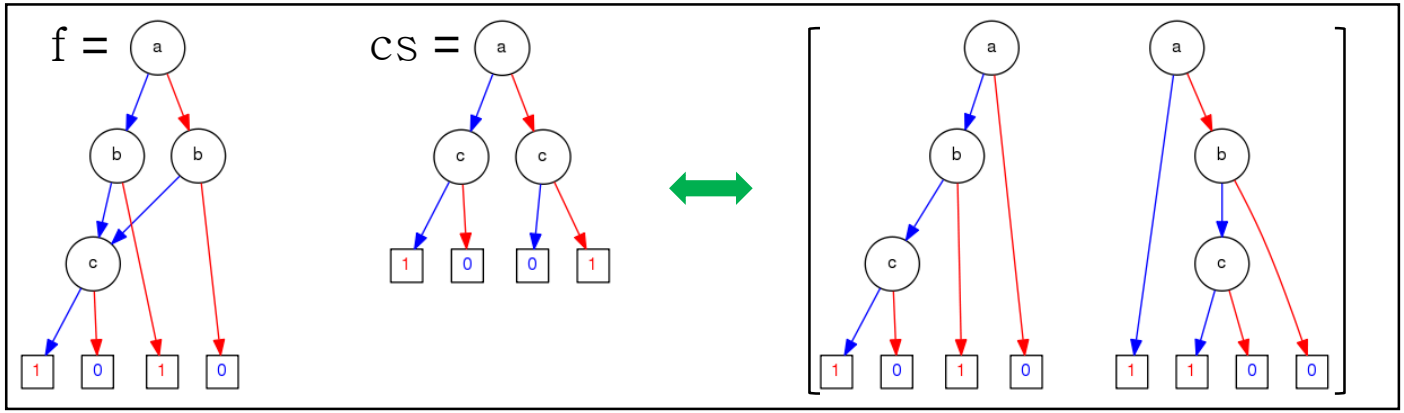
➡ Choix impossible
➡ Choix possible

$f = \bar{a}\bar{c} + \bar{a}b + a\bar{b}\bar{c}$ $\qquad$ $f' = \bar{a}$

- Exemple de ‘care-set’: Etats accessibles d’une FSM

Utilisation de ‘care-sets’ (suite)

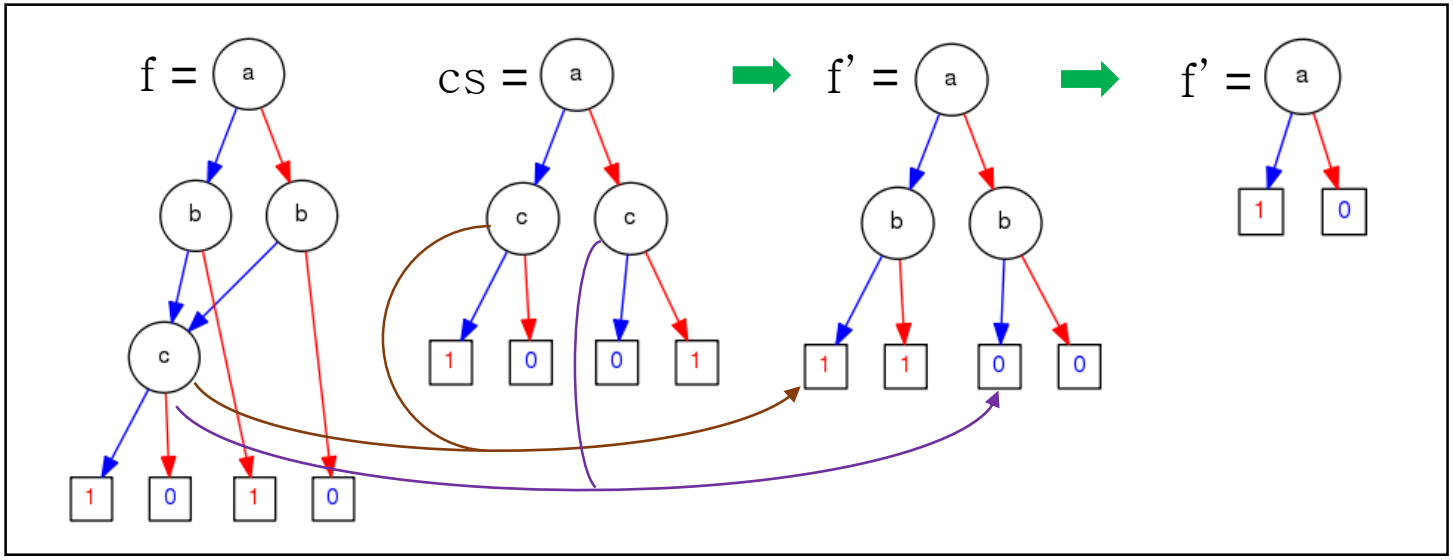
- Problème: étant donnés BDD(f) et BDD(cs), construire f’ telle que
  - Si cs(v_1, …, v_n) = 1 alors f’(v_1, …, v_n) = f(v_1, …, v_n)
  - Si cs(v_1, …, v_n) = 0 alors f’(v_1, …, v_n) = 0 ou bien 1 au choix
  - BDD(f’) plus petit que BDD(f)
- Problème équivalent:
  - Trouver un BDD de taille minimale dans l’intervalle de fonctions [(f • cs) (f + $\overline{cs}$)]



- Problème NP-complet, nombreux algorithmes (Coudert, Shiple, Hong, Oliveira)

Utilisation de ‘care-sets’ (suite)

- Opérateur *Restrict* (Coudert)
  - $R(f, 0) \rightarrow 0$
  - $R(f, 1) \rightarrow f$
  - $R(0/1, cs) \rightarrow 0/1$
  - $R(f, f) \rightarrow 1$
  - $R((x, f_0, f_1), cs) \rightarrow (x, R(f_0, cs), R(f_1, cs))$
  - $R(f, (x, cs_0, cs_1)) \rightarrow R(f, cs_0 + cs_1))$
  - $R((x, f_0, f_1), (x, cs_0, cs_1)) \rightarrow (x, R(f_0, cs_0), R(f_1, cs_1))$

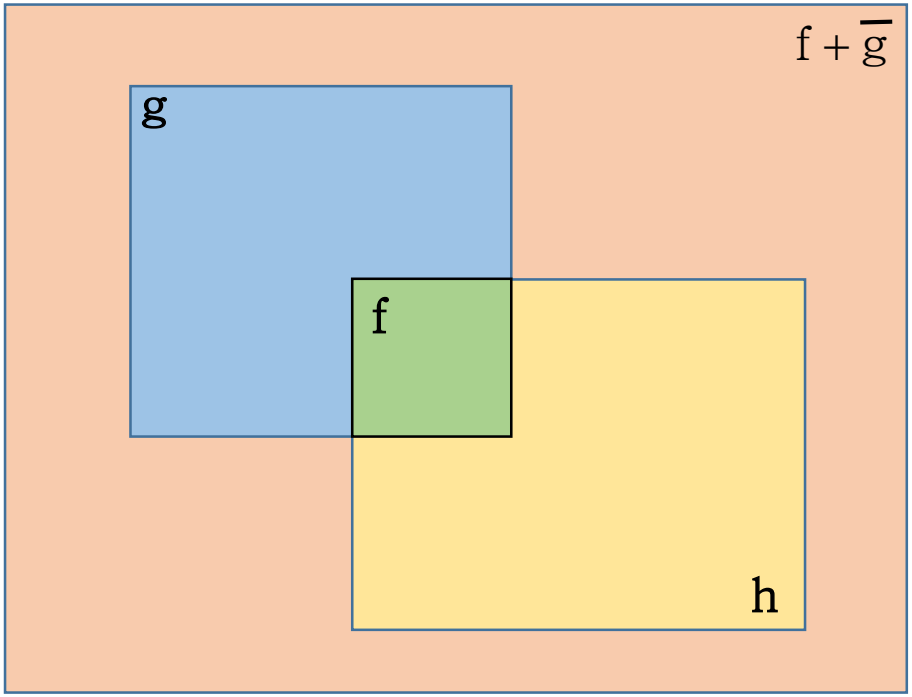


Synthèse logique

- Compilation multi-passes
  - P1 Simplification des expressions Booléennes
  - P2 Factorisation des expressions
  - P3 Génération des portes logiques

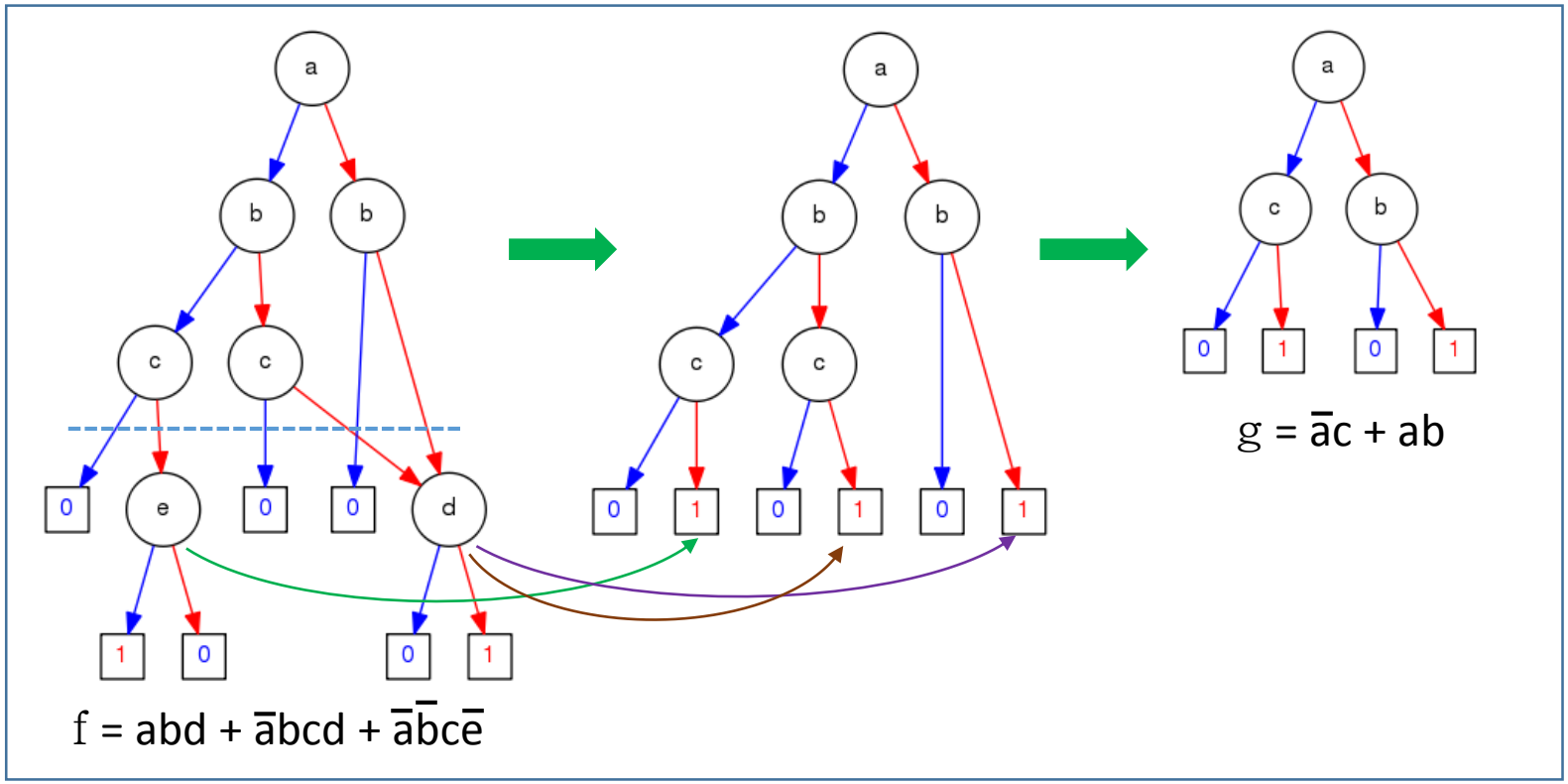
Factorisation de BDD

- Factorisation sous forme conjonctive
  - Une fonction f peut être factorisée sous la forme
    $f = g \bullet h$
    ssi
    $f \subset g$ et h est dans $[f\ (f + \overline{g})]$



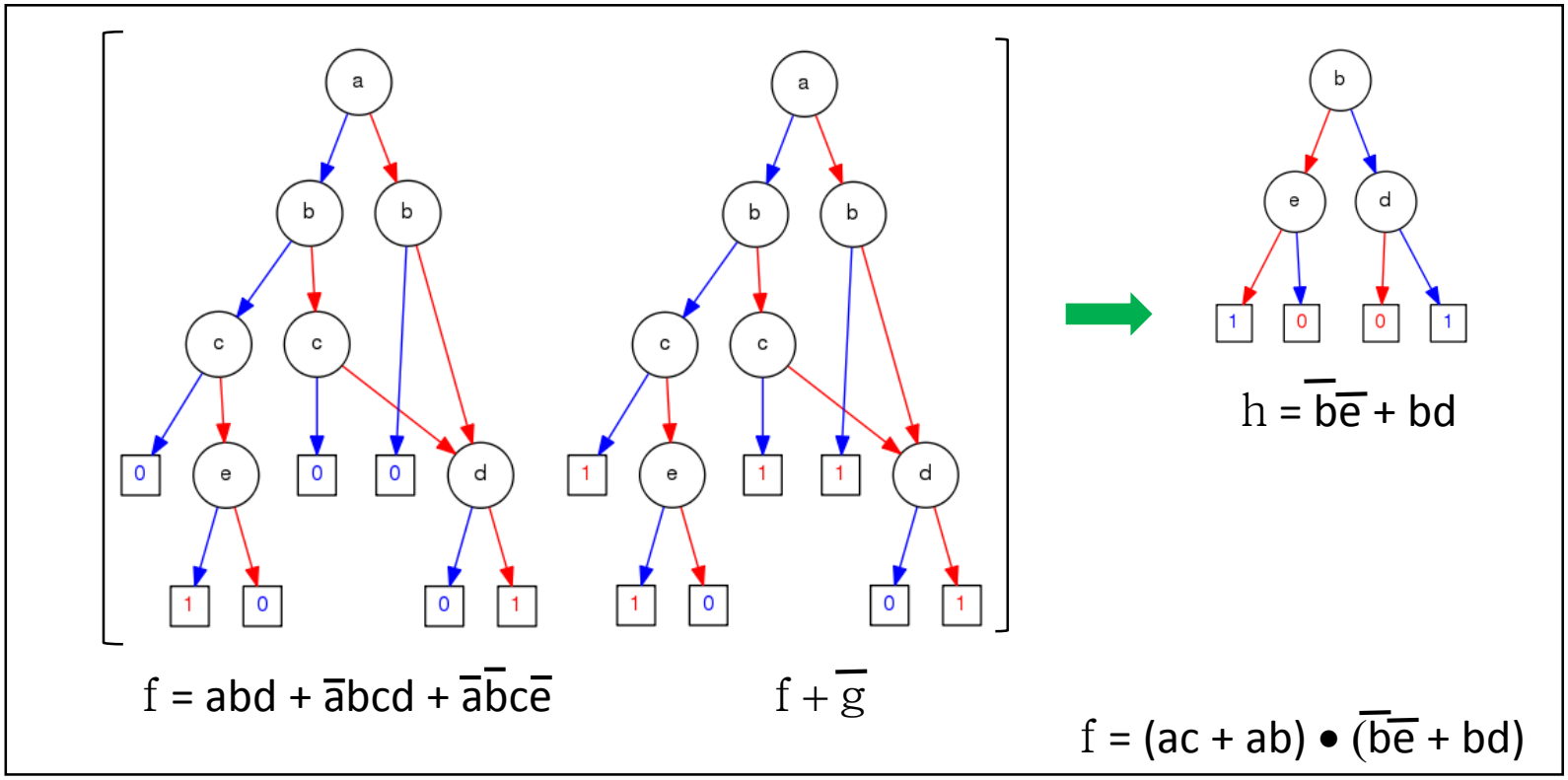
Factorisation sous forme conjonctive (suite)

- Algorithme
  - Couper le BDD de f horizontalement
  - Remplacer les nœuds non terminaux sous la coupure par 1
  - Le BDD produit est g



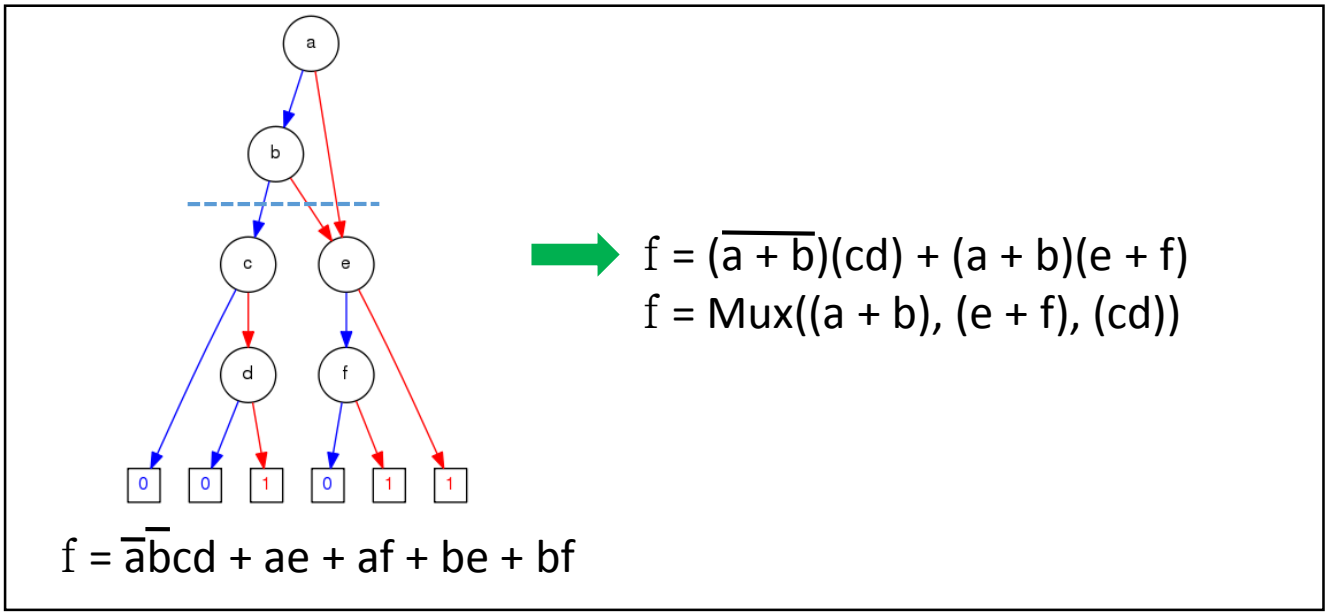
Factorisation sous forme conjonctive (suite)

- Algorithme (suite)
  - Créer le BDD de $f + \overline{g}$
  - Choisir le BDD de h dans l'intervalle $[f \quad (f + \overline{g})]$



Autres factorisations

- Factorisation sous forme disjonctive $g+h$
  - f peut être factorisée sous la forme $f = g+h$ ssi $g \subset f$ et h est dans l'intervalle $[(f - g)\ f]$
  - Quasiment identique au cas précédent
- Factorisation sous forme Mux(g, h_1, h_2)
  - f peut être factorisée sous la forme $f = gh_1 + \overline{g}h_2$ ssi il existe une coupure du BDD de f de la forme $\{h_1, h_2\}$



Autres factorisations (suite)

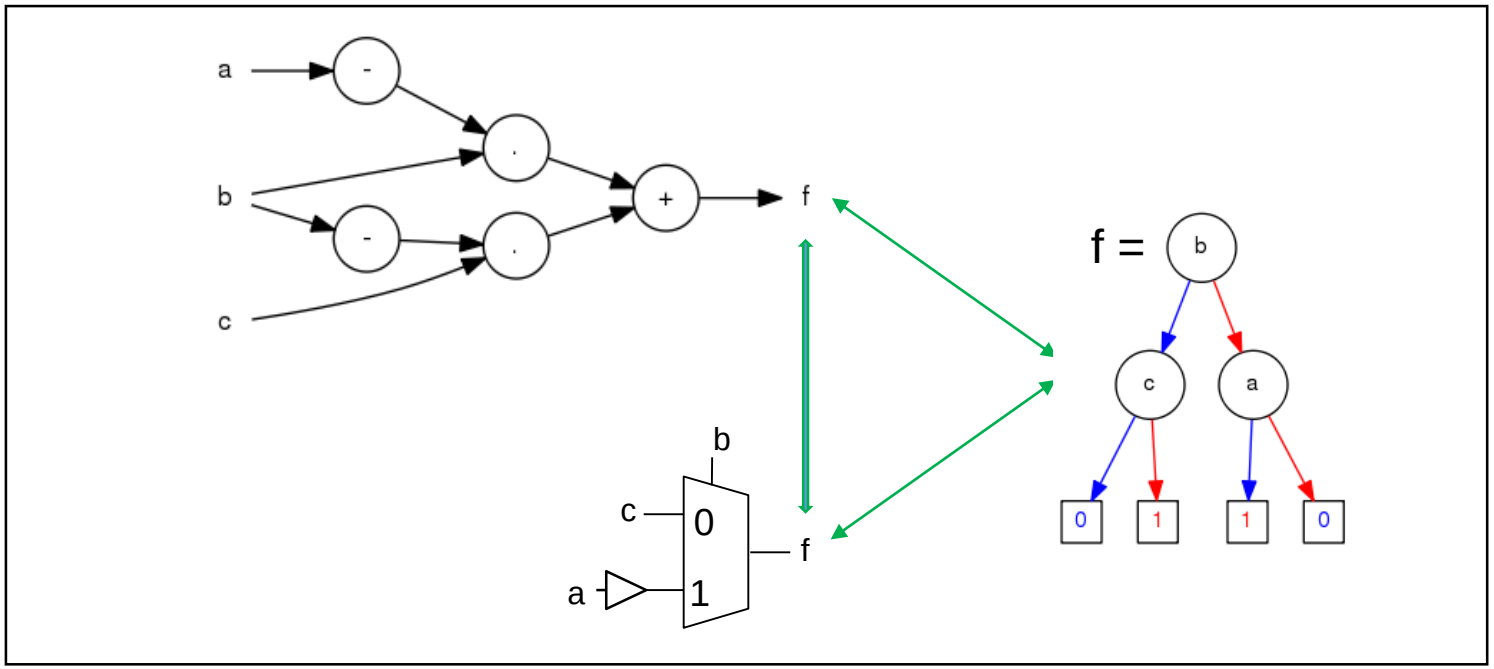
- Factorisation sous forme $g \oplus h$
  - f peut être factorisée sous la forme $f = g \oplus h$ ssi il existe une coupure $\{h, \bar{h}\}$
  - Quasiment identique au cas du Mux
- Factorisation sous forme Maj(f, g, h) = $fg + fh + gh$
  - Très utile mais plus complexe (Amarú)

Synthèse logique

- Compilation multi-passes
  - P1 Simplification des expressions Booléennes
  - P2 Factorisation des expressions
  - P3 Génération des portes logiques

Génération des portes logiques

- Entrée: réseau composé de nœuds (+, •, ⊕, Mux, Maj)
- Sortie: circuit composé de portes logiques
- Problème central:
  - Déterminer si un sous ensemble de nœuds est l'instance d'une porte de la bibliothèque ('Matching Problem')
  - Résolu très simplement grâce à la canonicité des BDDs



Plan de la présentation

- Conception des circuits numériques
- Synthèse logique
- Réduction de la consommation
- Conclusion

Optimisation logique

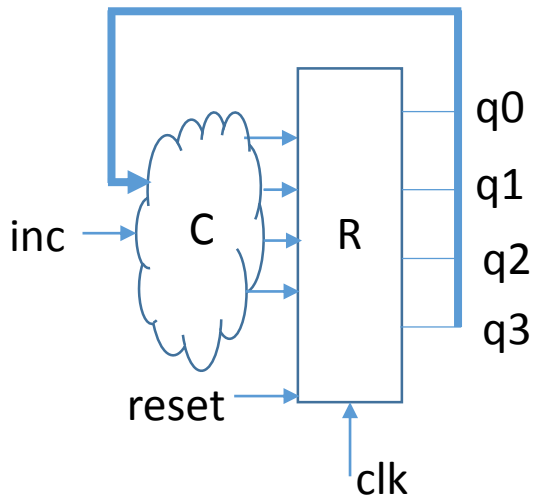
- Entrée
  - Circuit composé de portes de la bibliothèque
  - Contraintes: fréquence des horloges, surface, consommation
- Sortie
  - Circuit fonctionnellement équivalent qui respecte les contraintes
- Types d'optimisation
  - Re-synthèse d'une partie du circuit
    - Séquentielle
    - Combinatoire
  - Gate sizing: remplacement de portes par des portes de même fonctionnalité mais avec différentes caractéristiques (plus petite, plus rapide, meilleure consommation)
  - Buffering: amplification des signaux le long des longs fils, répartition de la capacité en aval d'une porte

Réduction de la consommation

- Objectifs
  - Temps de cycle de batteries, dissipation de chaleur
- Types de consommation
  - Statique (fuites)
  - Dynamique (commutation)
- Techniques de réduction
  - Technologie
    - Tension de fonctionnement (v^2)
    - Taille des transistors, choix des transistors (SOI)
  - Architecture
    - Mise en veille de tout ou partie du circuit
  - Optimisation
    - Fuites (réduction de surface, substitution de portes)
    - Commutation (partie séquentielle)

Réduction de la consommation dynamique

- Circuit = registres + portes combinatoires
- Quand fait-on circuler des électrons?
  - Registres: à chaque commutation de l'horloge
    - Distribution des horloges et commutation des registres
  - Combinatoire: propagation des valeurs des registres
- Exemple: compteur

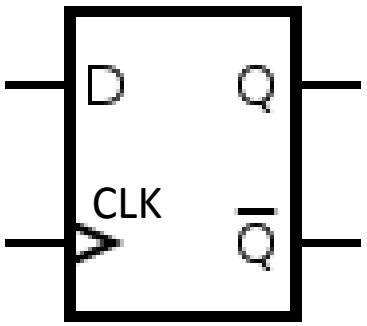


Temps	Inc	q	C	R
0	0	0000	inactif	actif
1	0	0000	inactif	actif
2	1	0001	actif	actif
3	1	0010	actif	actif
4	1	0011	actif	actif
5	0	0011	inactif	actif
6	0	0011	inactif	actif

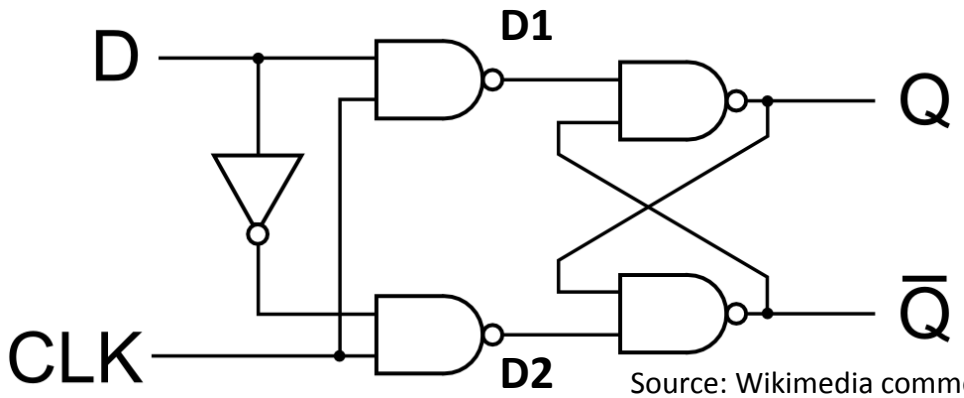
Activité des signaux et portes du compteur

Consommation dynamique de la bascule

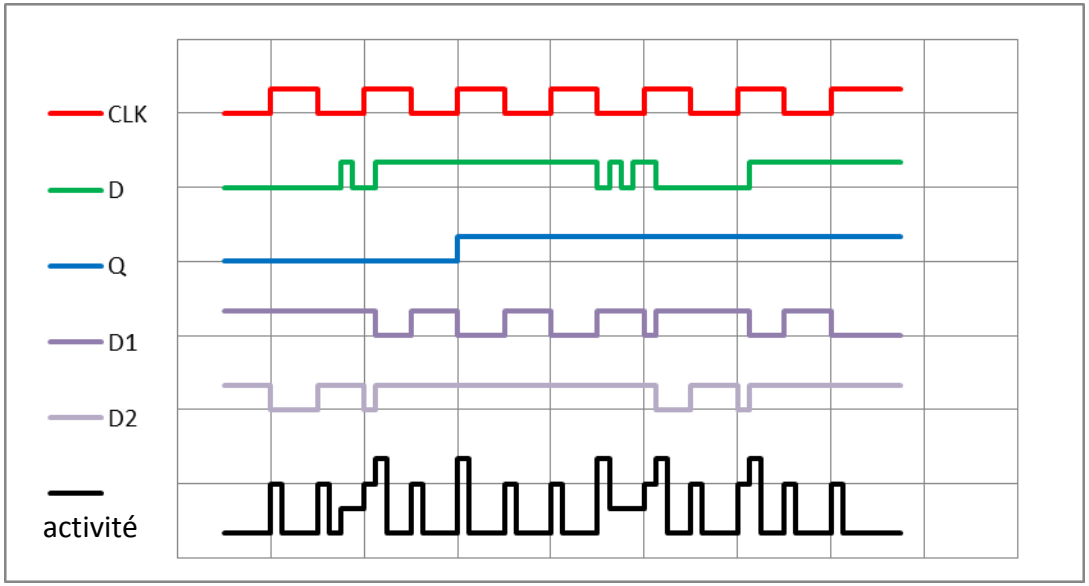
- Bascule: circuit logique qui peut garder une valeur en mémoire.



Source: Wikimedia commons



Source: Wikimedia commons



Synthèse d'horloges dérivées

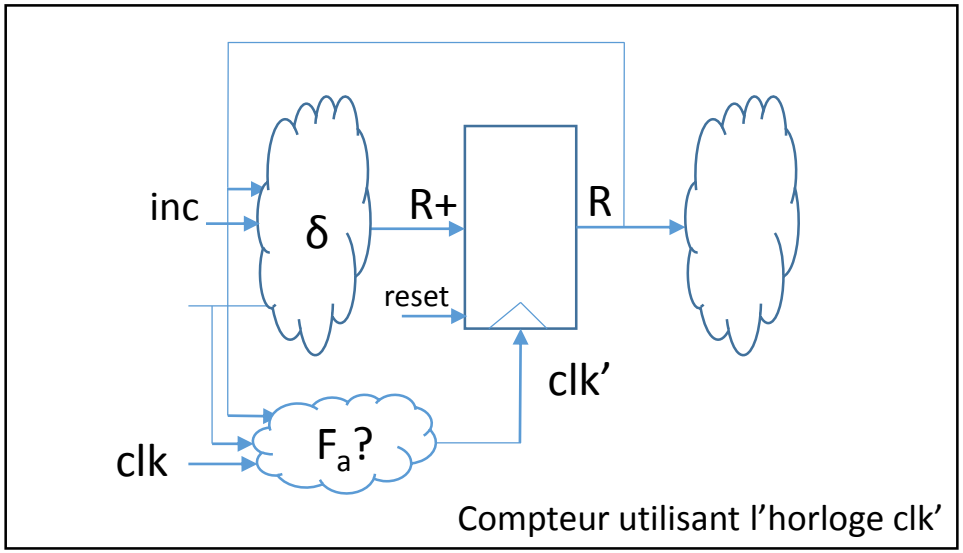
- Objectif
  - Réduire la commutation des entrées CLK des bascules
  - Réduire la commutation à l'intérieur des bascules
- Technique: synthèse d'horloges dérivées
  - Commutent moins
  - Conservent la fonctionnalité et la fréquence d'opération

Temps	Inc	Q	C	Clk'	R
0	0	0000	inactif	0	inactif
1	0	0000	inactif	0	inactif
2	1	0001	actif (1)	1	actif
3	1	0010	actif (2)	1	actif
4	1	0011	actif (1)	1	actif
5	0	0011	inactif	0	inactif
6	0	0011	inactif	0	inactif

Activité des signaux et portes du compteur

Équation d'une horloge dérivée

- Étant donné un ensemble de registres R
- Valeur suivante des registres R^+
- Quand $R^+ \neq R$, il faut activer l'horloge (fonction A)
- Sinon, on a le choix d'activer ou pas l'horloge
- Équation de l'horloge dérivée:
  - *clk'* = *clk* • F_a avec F_a dans [A=($R^+ \neq R$) 1]



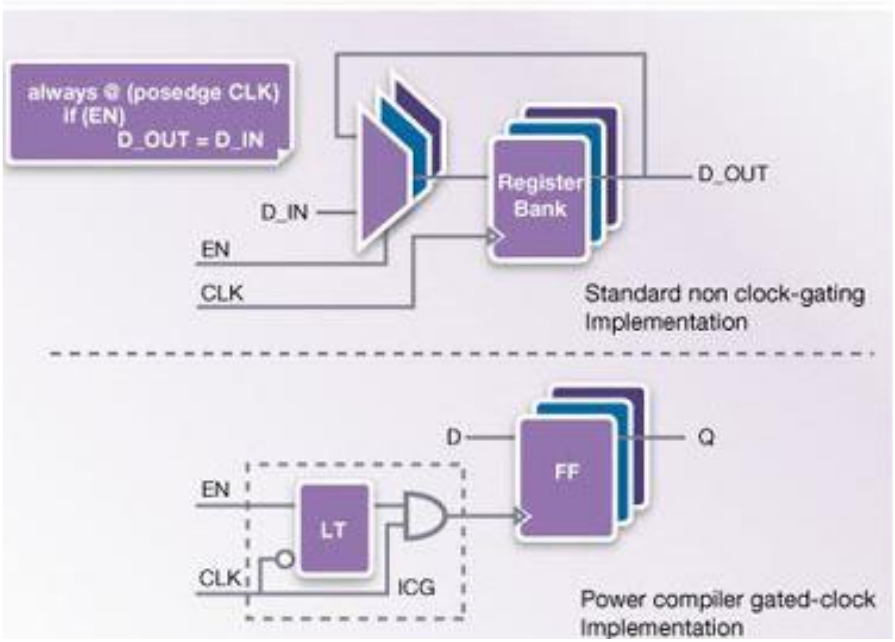
Compteur utilisant l'horloge clk'

Équation d'une horloge dérivée (suite)

- Registres $R = \{r_1 \dots r_k\}$, entrées $X = \{x_1 \dots x_n\}$
- R^+ est donné par: $\delta(R, X) : \{0,1\}^{k+n} \rightarrow \{0,1\}^k$
- Équation de la fonction d'activation:
  - $A(X, R) = \delta(R, X) \neq R$
  - $A(X, R) = \sum_{i=1}^{k} (\delta_i(R, X) \neq r_i)$
  - $A(X, R) = \sum_{i=1}^{k} (\delta_i(R, X) \oplus r_i)$
- Le BDD de A se calcule en fonction du BDD de δ
- La taille du BDD de A est une borne supérieure de la taille du circuit de dérivation

Synthèse du circuit de dérivation

- Entrée: Signal clk et une fonction d'activation F_a
- Sortie: Circuit de dérivation
- Processus
  - Synthèse du circuit qui produit F_a
  - Combinaison avec clk pour produire clk'

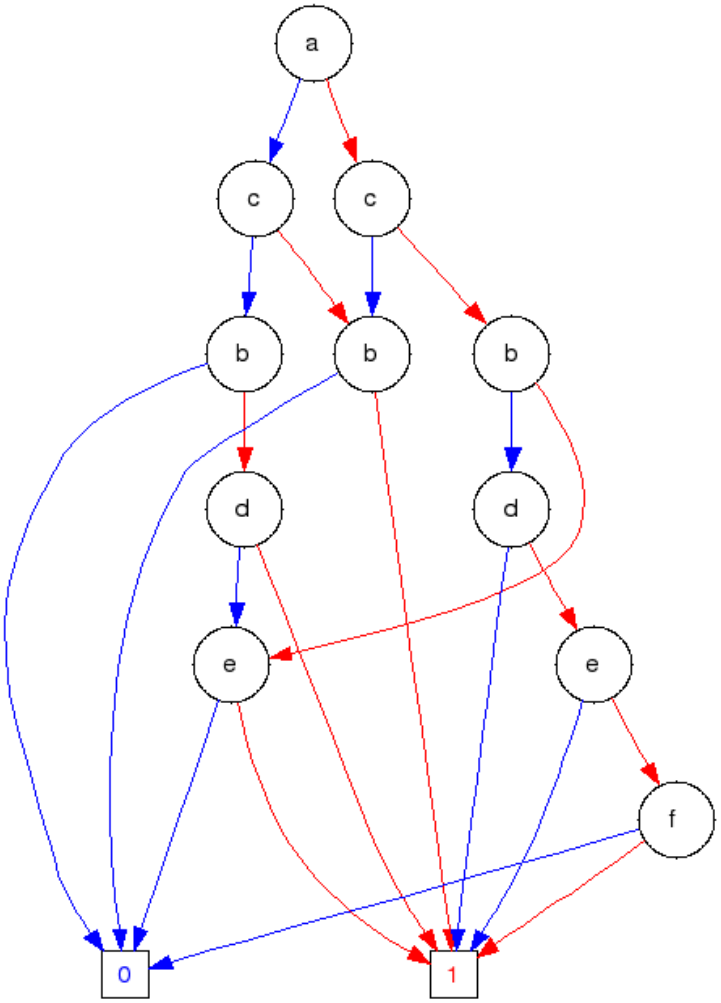


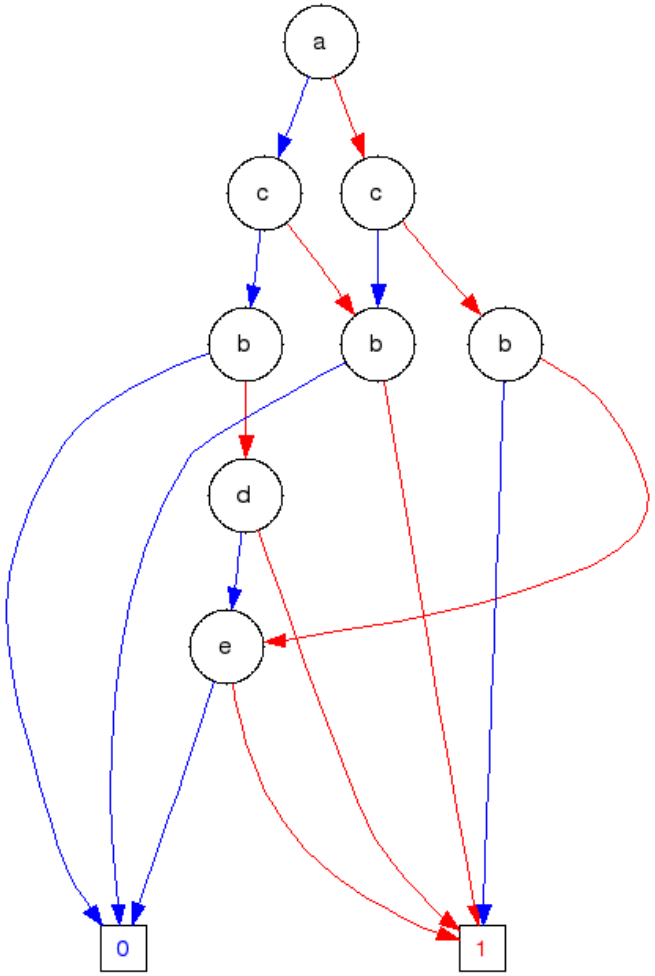
Source: Synopsys Inc

Choix de F_a

- Estimation du gain associé à F_a
  - gain(F_a) = gain(clk') - coût(circuit de dérivation)
  - gain(clk') = (proportion de 0 dans F_a) x #Registres(clk')
  - coût(circuit) = | BDD(F_a) |
- Problème: Trouver F_a dans l'intervalle [*A* 1] qui maximise gain(F_a)
- Cas extrêmes:
  - F_a = A, minimise commutation de clk', mais le BDD de A peut être grand ou profond
  - F_a = 1, surface nulle, mais clk' = clk, aucun intérêt
- Algorithme
  - Parcourir l'intervalle [*A* 1] en partant de A et en ajoutant des chemins à 1 au BDD de F_a
  - Choisir la meilleure fonction F_a

Exemple

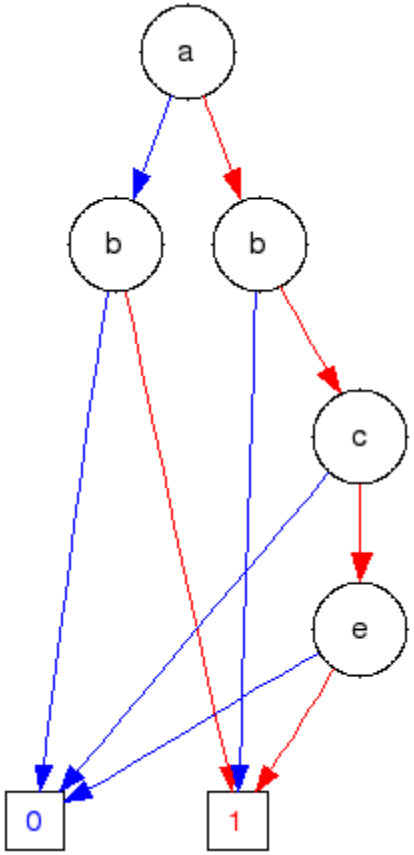




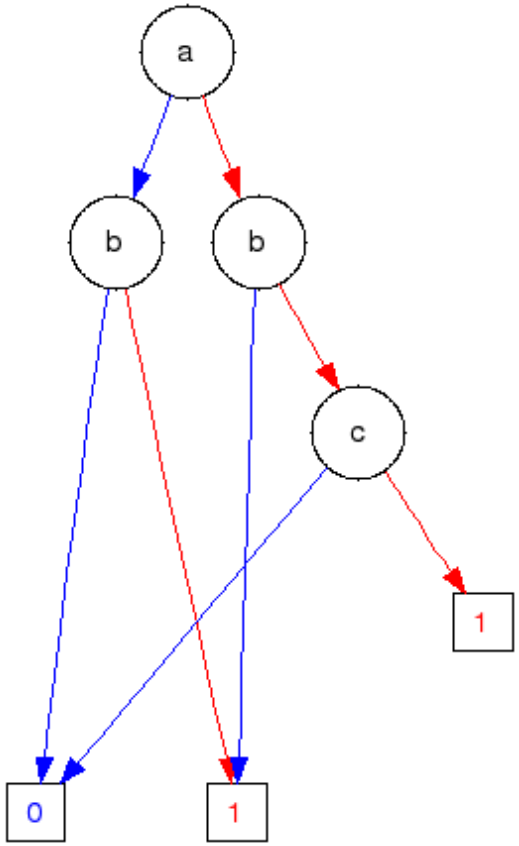
$|F_a| = 11$, $|Path(0)| = 31$

$|F_a| = 8$, $|Path(0)| = 30$

Exemple (suite)



$|F_a| = 5$, $|Path(0)| = 28$

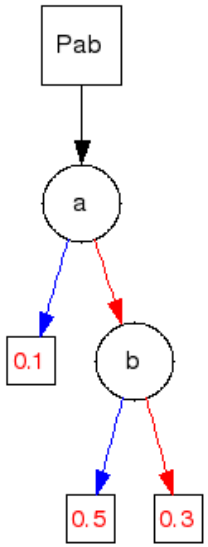


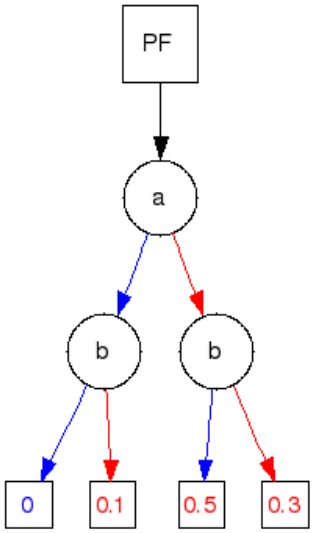
$|F_a| = 4$, $|Path(0)| = 24$

Utilisation de l'activité des signaux

- Définitions
  - Activité d'un signal A(x): {P(x_i = 0), P(x_i = 1)} à chaque cycle
  - A(X = [x_1 …x_n]) = {P(X = [0…0]), …, P(X = [1…1])}
  - A(f(X)) = {P(f(X) = 0), P(f(X) = 1)}, P(f(X) = 0) = $\sum_{f(X)=0} P(X)$
- Activité représentée par un ADD (BDD avec des feuilles à valeurs réelles)
- Exemple: f(a, b) = $\bar{a}\bar{b}$

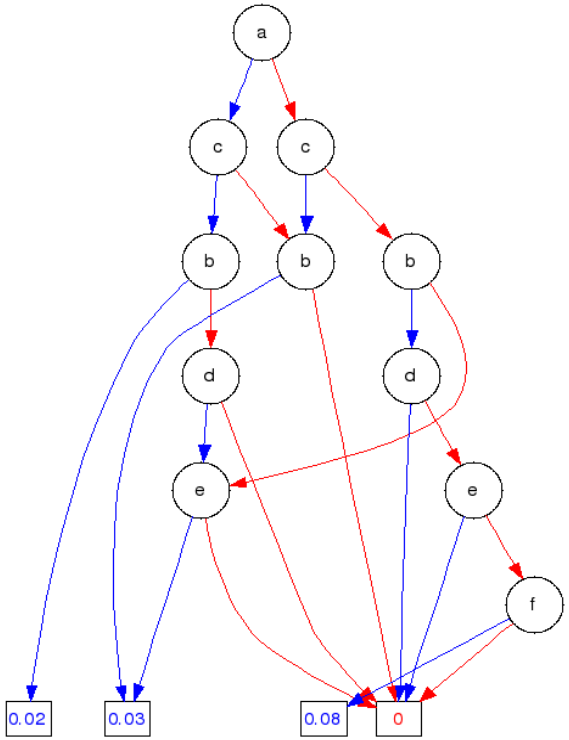
a	b	P_{ab}	f	$P_{f=0}$
0	0	0.1	1	0
0	1	0.1	0	0.1
1	0	0.5	0	0.5
1	1	0.3	0	0.3





Utilisation de l'activité des signaux (suite)

- Idée (Benini): utiliser les activités pour produire une meilleure solution
  - Activité de F_a représentée par son ADD
  - gain(F_a) calculé en utilisant P(F_a = 0) au lieu du nombre de chemins à 0 dans F_a



BDDs de F_a

$|F_a| = 11$, $|Path(0)| = 31$

P = 0.02 x 8 + 0.03 x 22 + 008 = 0.9

$|F_a| = 8$, $|Path(0)| = 30$, P = 0.82

$|F_a| = 5$, $|Path(0)| = 28$, P = 0.76

$|F_a| = 4$, $|Path(0)| = 24$, P = 0.64

Mise en œuvre

- On considère l'ensemble des registres du circuit
- Processus itératif
  - Choisir un partitionnement
    - Heuristiques (entrées communes, bus, existence d'une fonction F_a intéressante)
    - Raffinement d'un partitionnement précédent
  - Pour chaque partition
    - Calculer A
    - Choisir F_a
    - Synthétiser le circuit de dérivation de clk'
    - Connecter les registres de la partition à clk'

Conclusion

- Logiciels de synthèse et d'optimisation sont des outils indispensables pour la conception de circuits
- Ces logiciels utilisent intensivement les techniques de raisonnement Booléen
- Les BDDs sont supérieurs aux autres techniques pour explorer l'espace des fonctions Booléennes

Bibliographie

E.M. Sentovich et al., SIS: A System For Sequential Logic Synthesis, 1992

C. Yang and M. Ciesielski, BDS: A BDD-Based Logic Optimization System, 2002

L. Amarú et al., BDS-MAJ: A BDD-Based Logic Synthesis Tool Exploiting Majority Logic Decomposition, 2013

F. Mailhot and G. De Micheli, Algorithms For Technology Mapping Based On BDDs and Boolean Operations, 1993

T. Shiple and R. Hojati, Heuristic Minimization of BDDs Using Don't Cares, 1994

Y. Hong et al., Safe BDD Minimization Using Don't Cares, 1997

A.L. Oliveira, Exact Minimization of BDDs Using Implicit Techniques, 1998

R. Rudell, Dynamic Variable Ordering for Ordered BDDs, 1993

C. Berthet et al., New Ideas On Symbolic Manipulations Of Finite State Machines, 1990

O. Coudert et al., A New Viewpoint On Two-Level Logic Minimization, 1993

J. Warnock et al., IBM z13 Circuit Design And Methodology, 2015

D. Fussell, Lecture On CMOS Transitors And Gates, 2009

N. Ishiura et al., Minimization Of BDDs Based On Exchanges Of Variables, 1991

M. Fujita et al., On Variable Ordering Of BDDs For The Application Of Multi-Level Logic Synthesis, 1991

L. Benini, Automatic Synthesis Of Sequential Circuits For Low Power Dissipation Phd Thesis 1997

Bibliographie (suite)

P. Babighian et al., A Scalable Algorithm for RTL Insertion of Gated Clocks Based on ODCs Computation, 2005

A.P. Hurst, Sequential Optimization for Low Power Digital Design, PhD Thesis 2008

E. Arbel et al., Resurrecting Infeasible Clock-Gating Functions, 2009

H. Savoj et al., Sequential Equivalence Checking for Clock-Gated Circuits, 2014