Résumé électronique → digital

Asic

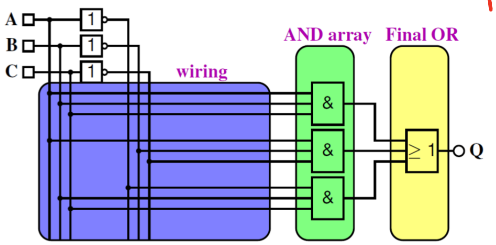
Application specific integrated circuit
- → billions of transistors
- → long design
- → cost a lot : 100k ... 5M
- ⇒ high volume markets

⇒ birth of programmable logic (also on Asic)

PAL (programmable array logic)

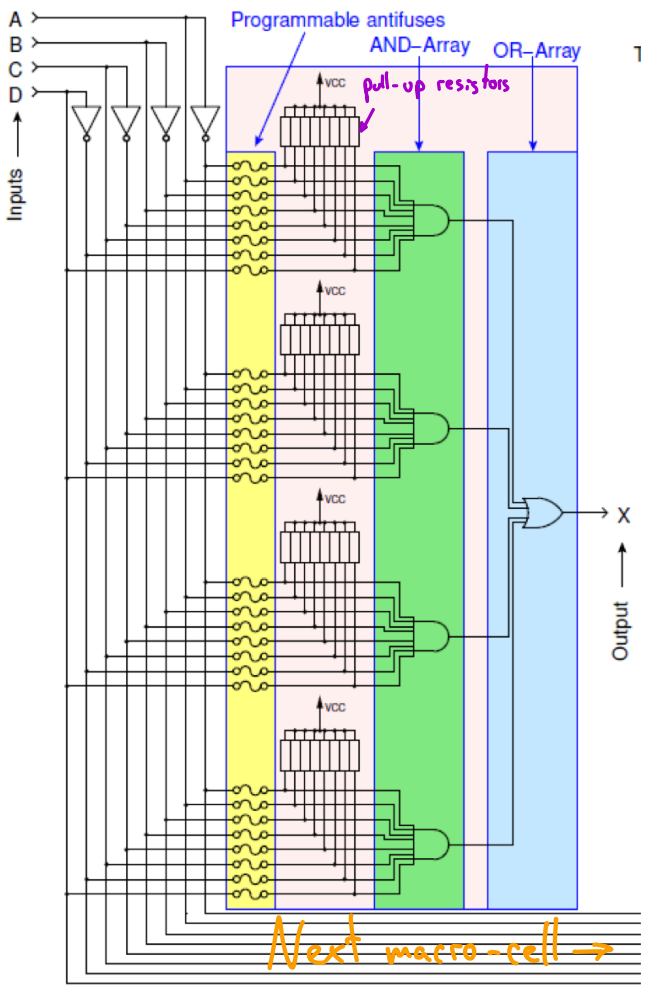
Based on canonical form

$Q = ABC + \bar{A}BC + A\bar{B}\bar{C}$ ← sum of product approach



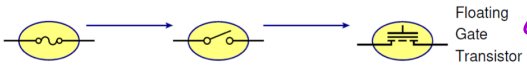
⊕ Fixed delay, low power, non volatile

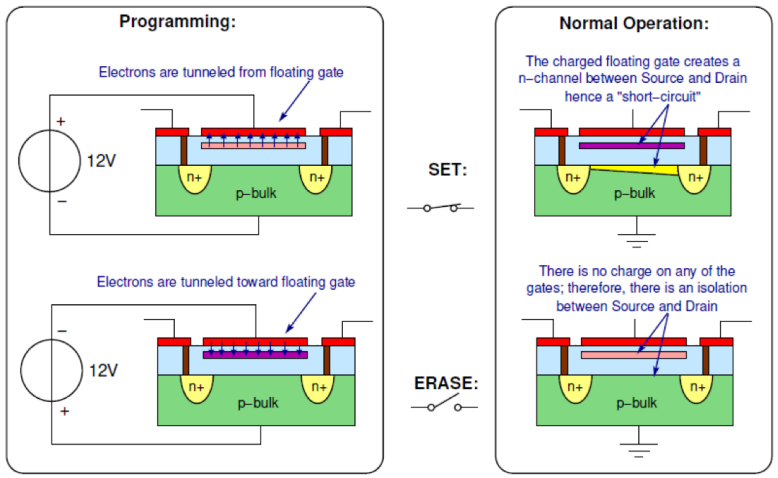
⊖ One time programmable (blown-fuse)

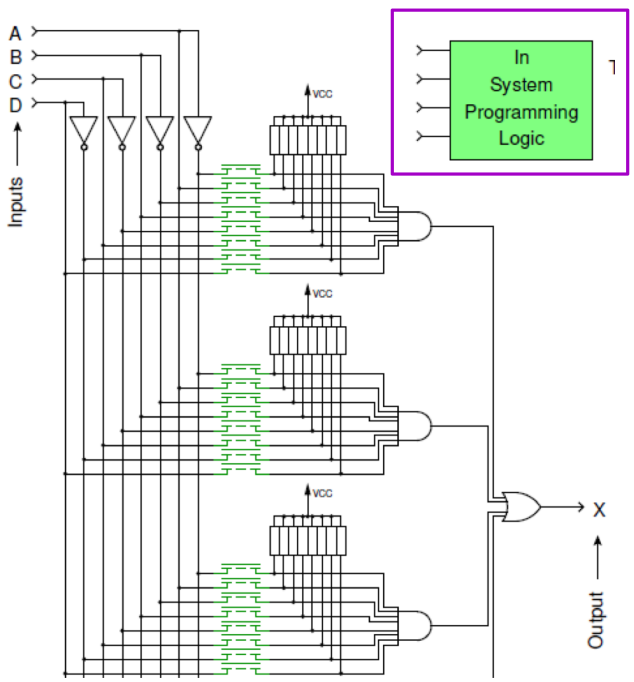


GAL (generic array logic)

Replace fuse with E^2 (Electrical erasable) switches

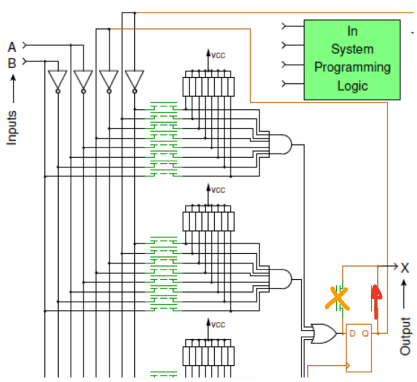






⊕ Program last 20 years
Reprogrammed 10k times

⊖ No sequential logic



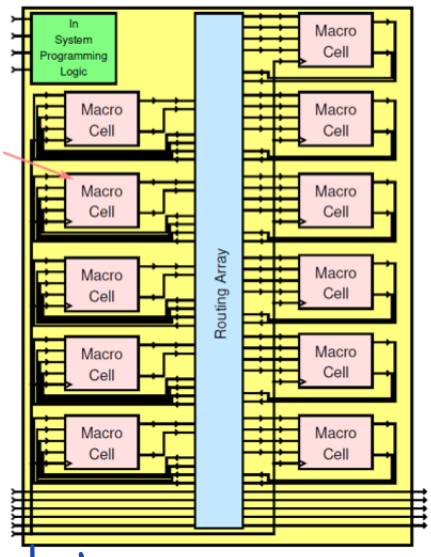
Add programmable flip-flop to add sequential ability

Sequential setting

CPLD (complex programmable logic device)

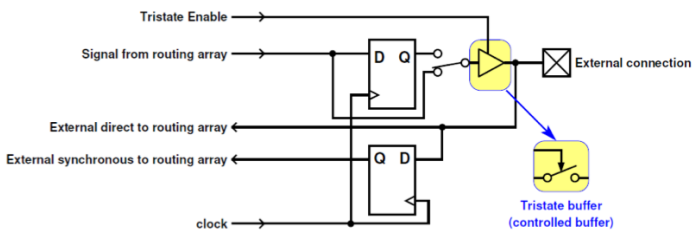
Sum of product approach

The routing array is a matrix of connections from macro-cells to macro-cells and outside. Each cross point has a E^2-switch.



I/O Block (IOB)

↳ connect to the outside



⊖ Hard to scale the routing array

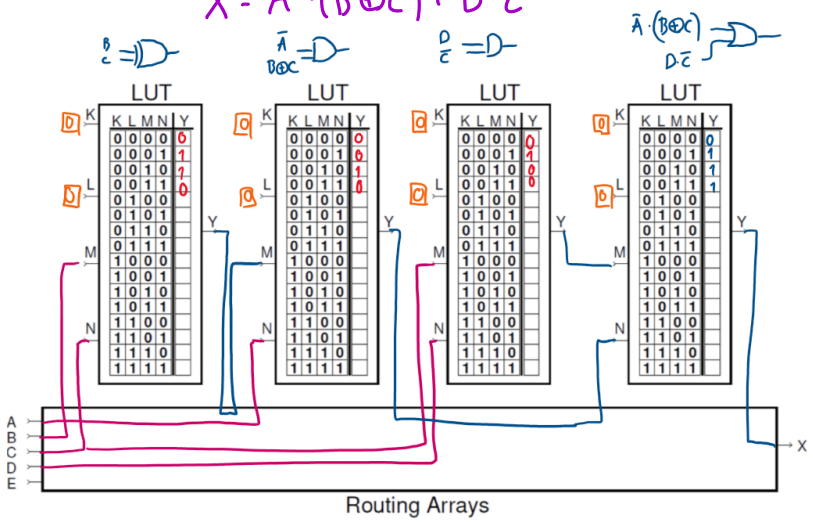
FPGA (Field programmable gate array)

D-Flipflop shift register + MUX = LUT

⊕ Very scalable, flexible architecture
very fast, mega parallelism

⊖ Volatile (lose power = reconfigure)
=> bit file

$X = \bar{A} \cdot (B \oplus C) + D \cdot \bar{C}$



Avec plus de variable on peut utiliser un LUT comme multiplexer.

VHDL Process (mainly simulation purpose)

δ-iterator is used to iterate until a stable signal is found

=> even-driven simulation → concept of "sensitivity"

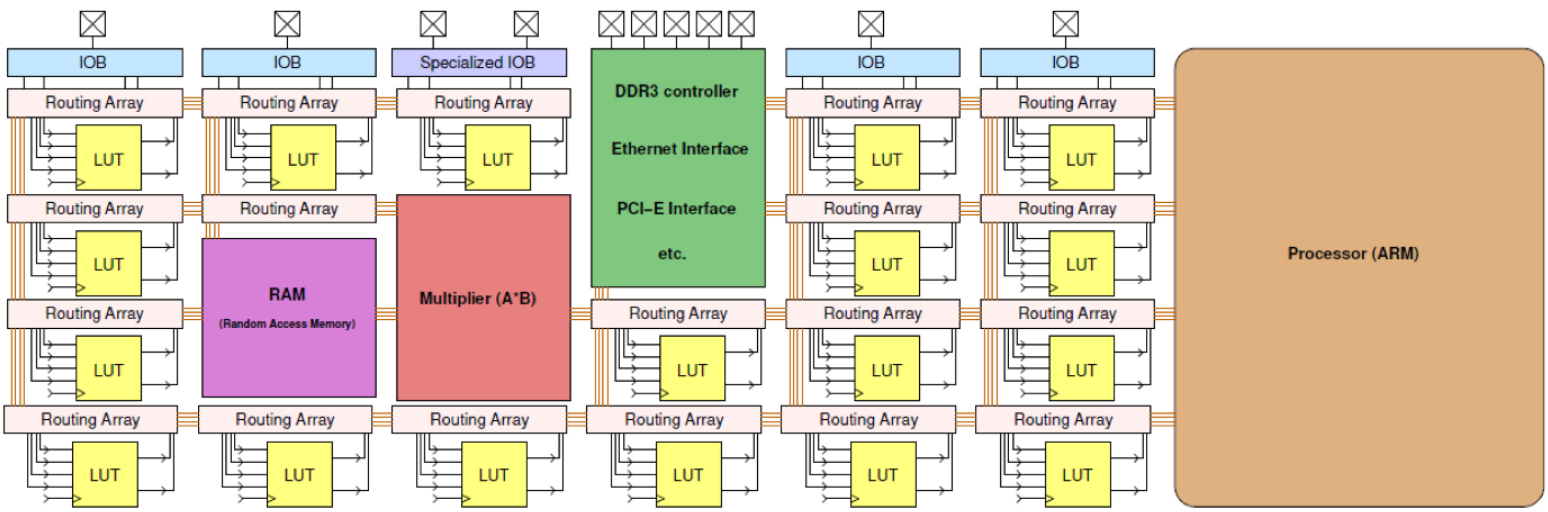
=> to reduce calculation we use explicit processes

Explicit process will only trigger on events of the signals in its sensitivity list.

Trigger by event ← fire event

↳ ① $Y = Y_\delta$
② $IN = IN_{event}$
③ Iterate until stable

VHDL
↓
Synthetizer
↓
gate level



Formulaire électronique numérique

Logique combinatoire

Quelques propriétés

$A \cdot A = A$ $\quad A \cdot \overline{A} = 0$

$A + A = A$ $\quad A + \overline{A} = 1$

$A \oplus A = 0$ $\quad A \oplus \overline{A} = 1$

$A \oplus 0 = A$ $\quad A \oplus 1 = \overline{A}$

$A \oplus 1 = \overline{A}$ $\quad A \oplus B = B \oplus A$

$y = 1 \longrightarrow$ minterms

$y = 0 \longrightarrow$ maxterms

$A \cdot (B \oplus C) = (A \cdot B) \oplus (A \cdot C)$

XOR $\quad A \oplus B = (A \cdot \overline{B}) + (\overline{A} \cdot B) = (A + B) \cdot (\overline{A + B})$

XNOR $\quad \overline{A \oplus B} = A \cdot B + \overline{A} \cdot \overline{B}$

$A \cdot (A + B) = A$ De Morgan

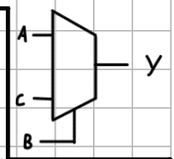
$A + (A \cdot B) = A$ $\quad \overline{A \cdot B} = \overline{A} + \overline{B}$

$\overline{A + B} = \overline{A} \cdot \overline{B}$

$\overline{A \oplus B} = \overline{A} \oplus B = A \oplus \overline{B}$

Multiplexeur

$Y = (A \cdot \overline{B}) + (C \cdot B)$



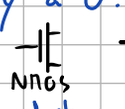
Portes en CMOS

Chaque entrée est connectée sur 1 NMOS et 1 PMOS

Trouver la fonction d'une porte :
- Chercher l'inverse de y et voir quand NMOS tire y à 0.

Dessiner une porte CMOS :
- PMOS sont en haut
- NMOS en bas

Si 2 PMOS en série alors il y a les 2 CMOS dual en série, mais parallèle au PMOS.



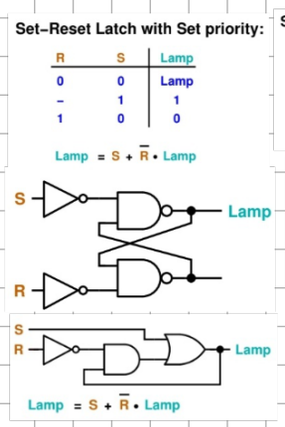
NMOS conduit si : gate = 1

PMOS conduit si : gate = 0

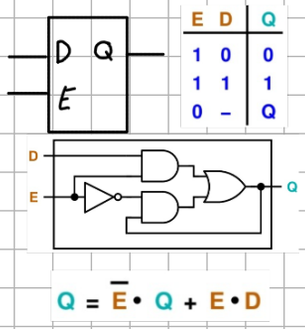
Logique séquentielle

Bascule asynchrone

Bascule set-reset



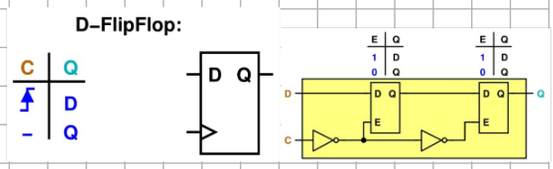
D-Latch



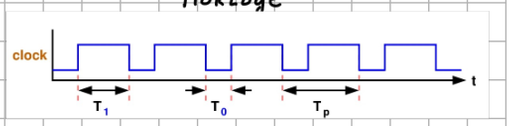
$Q = \overline{E} \cdot Q + E \cdot D$

Bascule synchrone

Sensible aux flancs (montant)



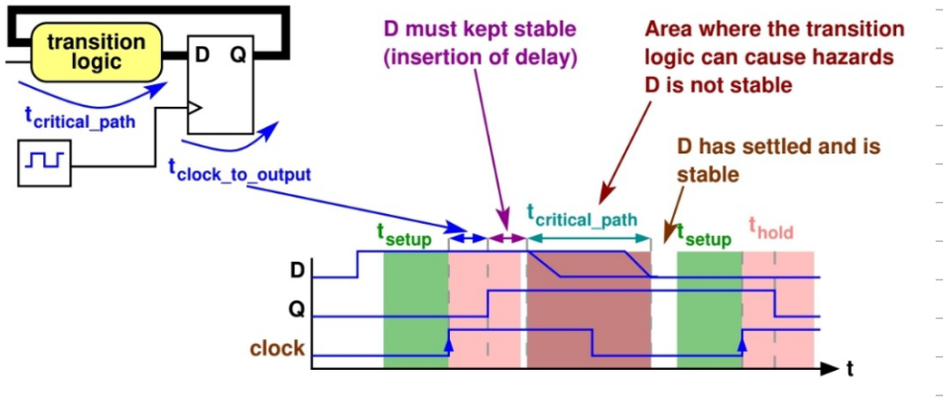
Horloge



$T_p = T_1 + T_0$ $\quad$ duty-cycle $= \frac{T_1}{T_p}$

Comportement non-idéal

- Chaque porte a un délais
- Pas de "Gated clock"



- Let's look at one positive edge.
- There are two time regions, the setup time t_{setup}, and the hold time t_{setup}; D must kept stable in this period, otherwise:
- The D-flipflop goes in metastability, and it settles at either **1** or **0**, independent of the value of **D**!

Systèmes de nombres

Signe-Magnitude :

MSB pour le signe, le reste décrit le nombre

Complément a 1

Positif : comme signe-Magn.

Négatif : inversion totale

⚠ ∃ +0 et -0

Complément a 2

Complément à 1

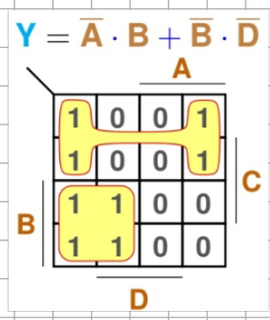
+1

Excess-N

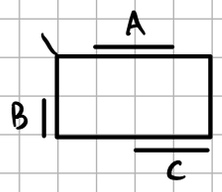
Nombre X_B

$X_{EN} = X_B + N$

Méthode de Karnaugh

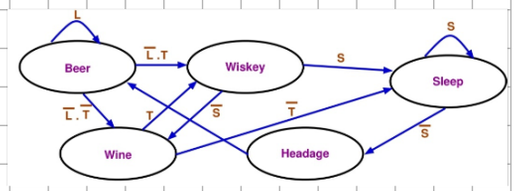


1. A partir d'une table de vérité à N variable, dessiner un diagramme avec 2^N cases et répartir les variables.
2. Remplir les cases selon la table de vérité.
3. Identifier les groupes valides
4. Sommer les expressions des groupes



Machine à états finis

1. <u>diagramme des états</u>
   - A. définir les états
   - B. définir les transitions et leurs conditions
     - Les conditions sont des combinaisons d'inputs
     - L'horloge n'est jamais une condition



2. <u>Complétude et consistance</u>
   - C. Vérifier la complétude et la consistance de chaque état
     - Comparer les expressions de transition avec la TT
   - D. Rendre les états complets et consistant si besoin

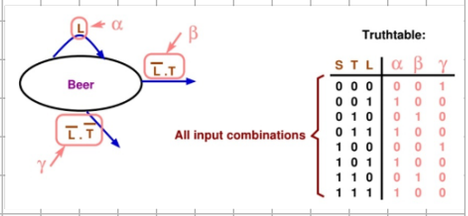
<u>Complétude</u>

Au moins une transition active possible

<u>Consistance</u>

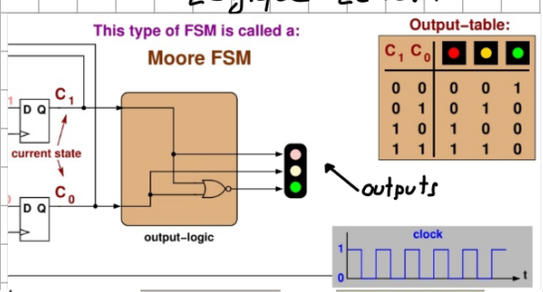
Au maximum une transition active à la fois

3. <u>Codage</u>
   - E. Coder X états sur N bits, $N \geq \left\lceil \frac{ln(x)}{ln(2)} \right\rceil$
     - Le codage influence le type de machine

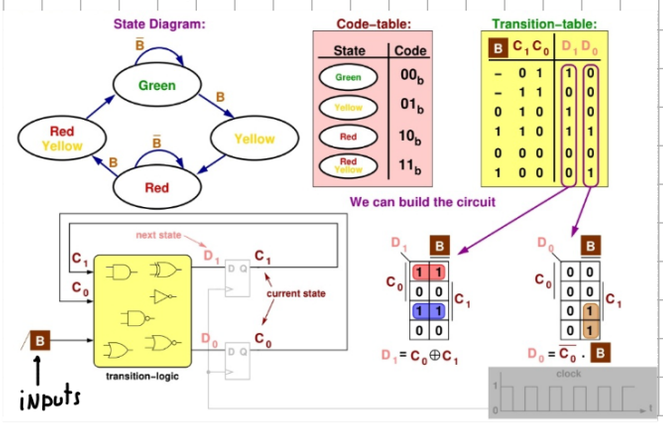


   - F. Traiter les états phantômes et le POR (si pas encore fait)
     - ⚠ Complétude et consistance
   - G. Créer les logiques de transition et de sortie

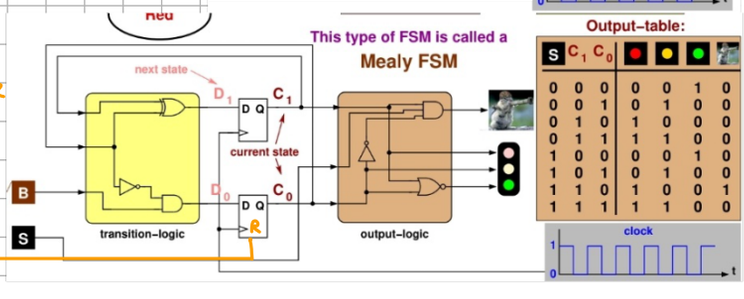
Logique de sortie



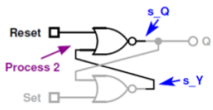
Logique de transition

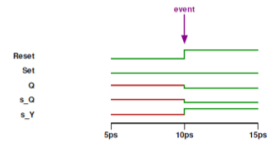


Circuit POR Asynchrone -définir état au reset



Simulation: δ-iterator





δ	s_Q$_\delta$
0	u
1	0
2	0

- At the event process 2 is triggered as it is sensitive to reset.
- At this moment two things happen, namely:
  - A sequential δ-iterator is initialized for all signals in the process left of the **<=** operator. The initial value is the one seen before the event.
  - Then the process is evaluated for the values seen at the event (new value).
- So, one δ-step is taken, determining the resulting value of the event.
- As s_Q$_{\delta=1}$ ≠ s_Q$_{\delta=0}$ a resulting event is "fired" triggering process 1 and 3.
- Another δ-step is taken. As now s_Q$_{\delta=2}$ = s_Q$_{\delta=1}$ a new stable situation is found, and s_Q is assigned the value of s_Q$_{\delta=2}$

Simulation: δ-iterator

Tr=*Triggered by an event*, Fi=*Fire an event*, St=*Stable*

δ	Q <= s_Q;				s_Q <= reset **NOR** s_Y;				s_Y <= set **NOR** s_Q;			
	Tr	Q$_\delta$	Fi	St	Tr	s_Q$_\delta$	Fi	St	Tr	s_Y$_\delta$	Fi	St
0		u		X	X	u				u		X
1				X		0	X					X
2	X					0		X	X			
3		0	X					X		1	X	
4		0		X	X					1		X
5				X		0		X				X

- The initial situation at the event for the whole system.
- Process 2 fires a resulting event.
- Process 1 and 3 get triggered by the resulting event.
- Process 2 gets triggered by the resulting event of process 3.
- A new stable situation is found, hence the last values assigned are the resulting values.

Explicite Process Body

- To understand the behavior of an explicit process we have to remember the δ-iterator.
- What happened in simulation:
  - An event triggers the simulator.
  - All output signals are replaced by their δ-representatives (e.g. Y$_\delta$).
  - All input signals are replaced by their δ_{event}-representatives (e.g. A$_{\delta e}$).
  - The simulator takes δ-iteration steps determining eventual new values for the δ-signal-representatives (e.g. Y$_\delta$).
  - This δ-iteration continues until a new *stable* situation is found.
  - All signals are assigned the value of their latest δ-representative (e.g. Y = Y$_\delta$).
- The sequential nature of this δ-evaluation is used in the body of an explicit process.

VHDL snippet:

```
example :PROCESS (A,B) IS
BEGIN
(1)  Z_δ <= NOT (A_δe);
(2)  Z_δ <= '0';
(3)  Z_δ <= NOT (A_δe) AND B_δe;
(4)  Z_δ <= A_δe NOR B_δe;
END PROCESS example;
```

sensitivity list

- To understand the behavior of the process we have to put it in the δ-iterator form.
- The moment we have an event on either A or B all input signals are transformed into their δ_{event}-representatives (e.g. A$_{\delta e}$, B$_{\delta e}$).
- On each δ-iteration the simulator will determine eventual new values of the δ-representatives (in this case Z$_\delta$).
- Within an explicit process this calculation is done by execution one line after the other (hence line (1), then (2), etc.).
- Only at the very end of the process the final value of the δ-representatives (in this case Z$_\delta$) is known for the current iteration.

VHDL snippet:

```
exercise :   PROCESS (A,B,C) IS
BEGIN
(1)  IF (A_δe = '1') THEN
(2)     X_δ <= B_δe;
(3)     Y_δ <= C_δe;
(4)  ELSE
(5)     X_δ <= C_δe;
(6)  END IF;
END PROCESS exercise;
```

- Let us review this exercise.
- We put it in the δ-iterator representation.
- We see that X$_\delta$ is always assigned a value. Either in line (2) when A$_{\delta e}$='1', or in line (5) when A$_{\delta e}$='0'. X represents, therefore, combinational logic.
- We see also that Y$_\delta$ is only assigned a value in line (3) when A$_{\delta e}$='1'. If A$_{\delta e}$='0', Y$_\delta$ is not assigned a new value meaning that Y **<=** Y$_{\delta e}$; the signal retains its "old" value, we have memory. In this case we have the behavior of a D-latch.

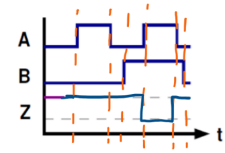
Explicit Processes

Implicit form:

VHDL snippet:

```
Z <= A NAND B;
```

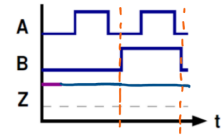
B	A	Z
0	0	1
0	1	1
1	0	1
1	1	0



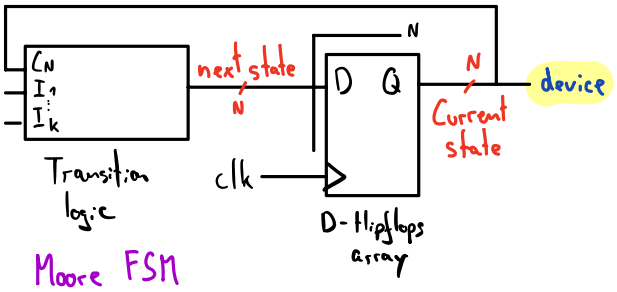
Explicit form:

VHDL snippet:

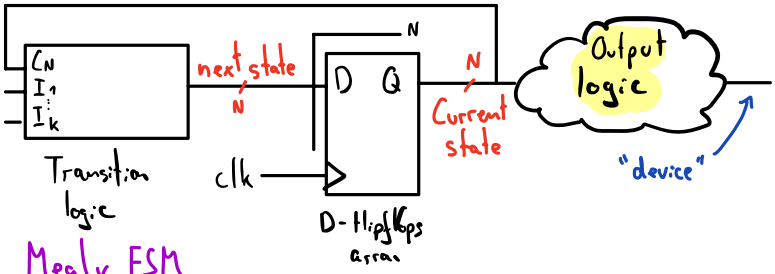
```
test : PROCESS ( B ) IS
BEGIN
    Z <= A NAND B;
END PROCESS test;
```



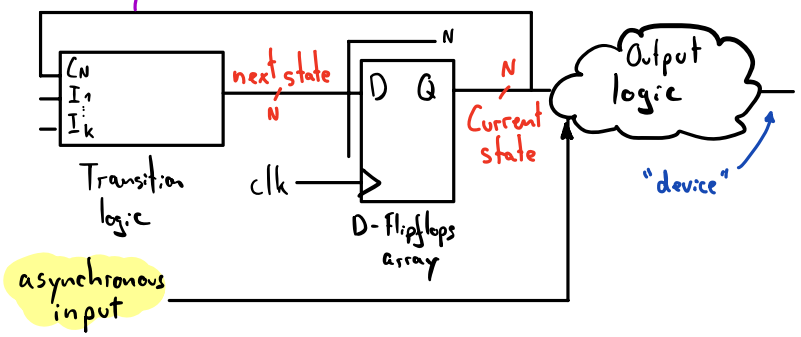
Medvedev FSM



Moore FSM



Mealy FSM



Moore FSM with datapath