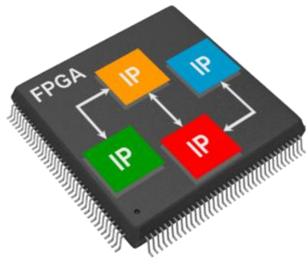


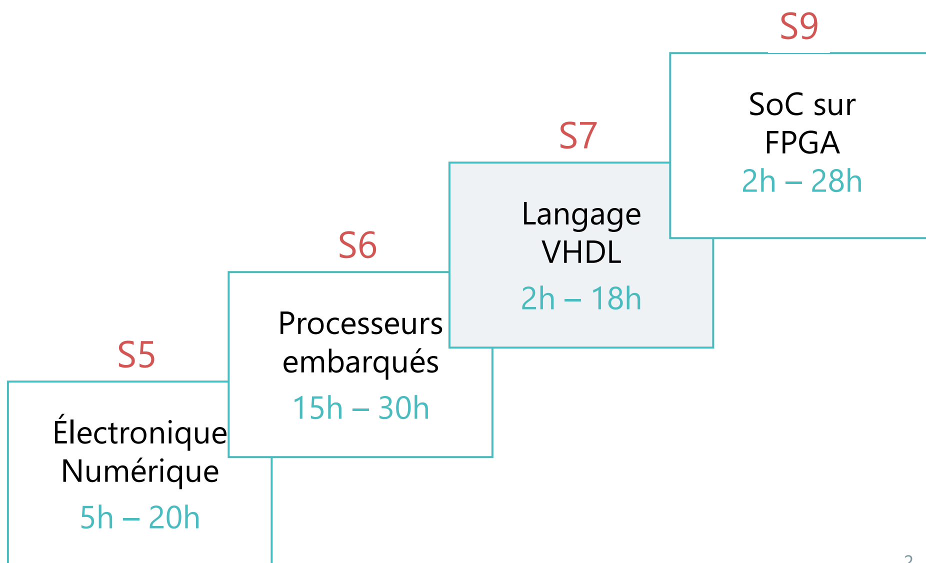
LANGAGE VHDL

Matthieu DENOUAL

L'École des INGÉNIEURS Scientifiques

1

1. CONTEXTE : MAQUETTE PÉDAGOGIQUE

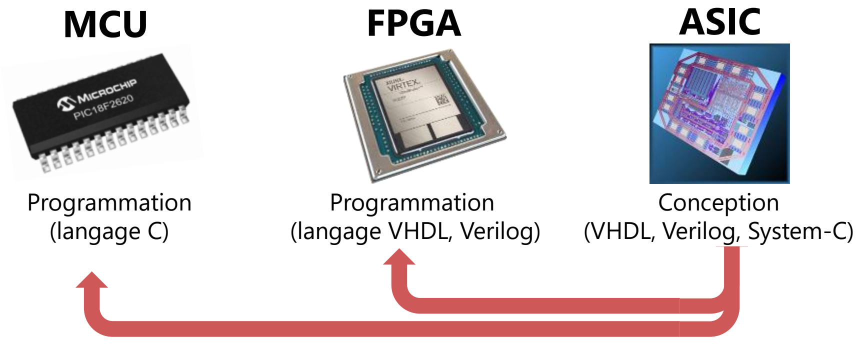


2

2

1. CONTEXTE : ÉLECTRONIQUE NUMÉRIQUE

Quelles solutions matérielles pour réaliser une fonction d'électronique numérique ?



VHDL (Verilog) : pour la programmation de FPGA

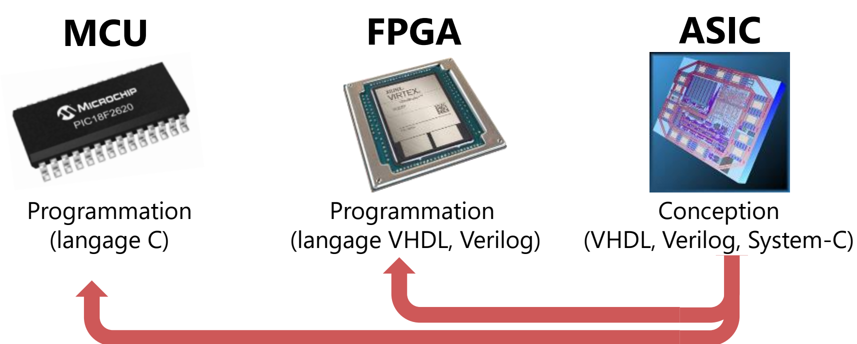
applications nécessitant de la vitesse de traitement,

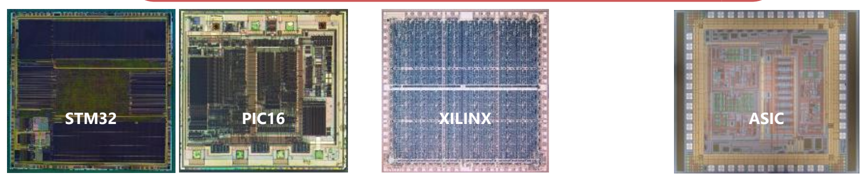
pour la conception de circuit.

3

3

1. CONTEXTE : ÉLECTRONIQUE NUMÉRIQUE





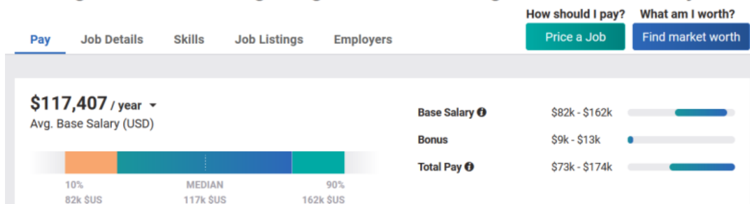
VHDL (Verilog) : pour la conception.

4

4

1. CONTEXTE : ÉLECTRONIQUE NUMÉRIQUE

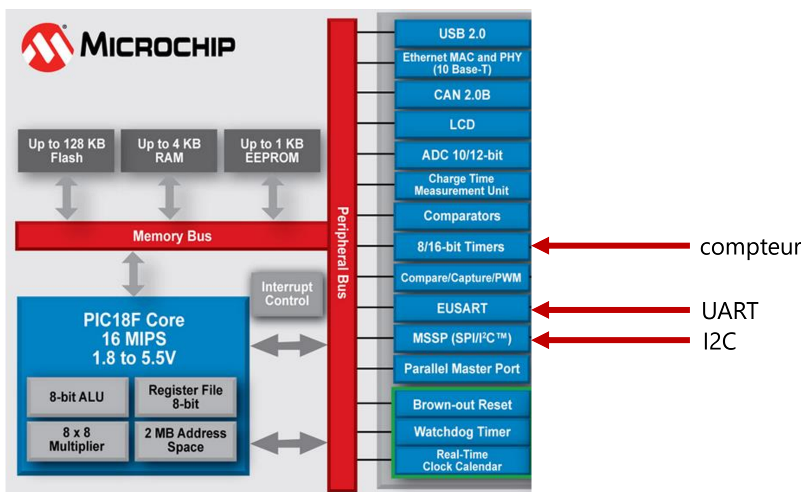
Les designers d'INTEL (les concepteurs des autres CPU et MCU) utilisent VHDL et Verilog pour la conception des microprocesseurs et microcontrôleurs.



5

2. LE MCU « PIC18F27K40 »

Lors des TP, vous développerez des périphériques comme ceux que vous avez pu utiliser en 1A dans les microcontrôleurs PIC.



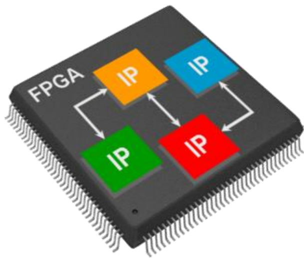
compteur

UART

I2C

7

7



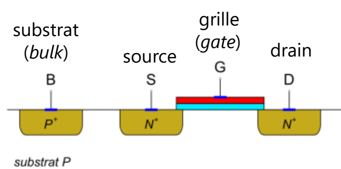
Le composant FPGA

Matthieu DENOUAL

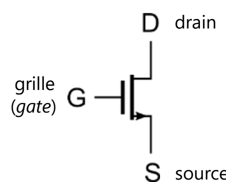
L'École des INGÉNIEURS Scientifiques

1

1.1. LE TRANSISTOR NMOS



Vue en coupe



Symbole électrique simplifié

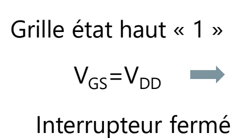
Symbole logique

- En logique, le transistor est utilisé en interrupteur commandé.
- L'interrupteur se situe entre le drain et la source.
- La commande le l'interrupteur se fait par la tension V_{GS} (entre Grille et Substrat).

Grille état haut « 1 »

$V_{GS}=V_{DD}$ ➡

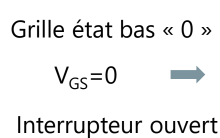
Interrupteur fermé



Grille état bas « 0 »

$V_{GS}=0$ ➡

Interrupteur ouvert

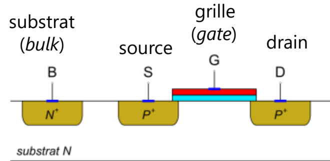


2

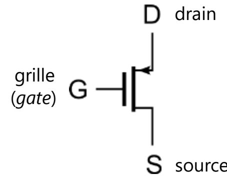
2

1.2. LE TRANSISTOR PMOS

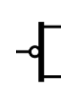
ENSI CAEN



Vue en coupe



Symbole électrique simplifié



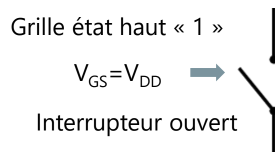
Symbole logique

- Comme son dual, le transistor PMOS est aussi utilisé en interrupteur commandé.
- Il y a une complémentarité entre les deux types, que ce soit au niveau de la structure ou au niveau du fonctionnement.
- La technologie CMOS (Complementary MOS) exploite cette complémentarité.

Grille état haut « 1 »

$V_{GS}=V_{DD}$ ➡

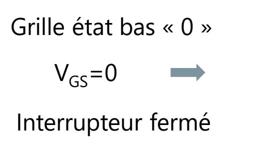
Interrupteur ouvert



Grille état bas « 0 »

$V_{GS}=0$ ➡

Interrupteur fermé

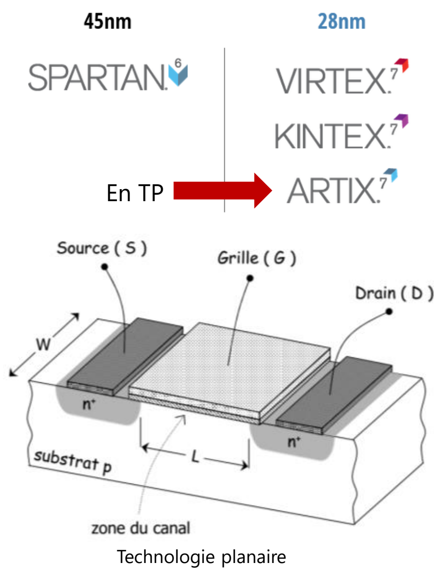


1.3. TECHNOLOGIE MICROÉLECTRONIQUE

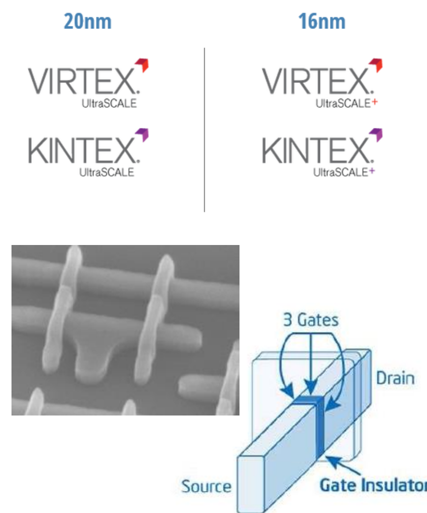
ENSI CAEN

45nm	28nm	20nm	16nm
SPARTAN 6	VIRTEX 7 KINTEX 7 ARTIX 7	VIRTEX UltraSCALE KINTEX UltraSCALE	VIRTEX UltraSCALE+ KINTEX UltraSCALE+

En TP ➡ ARTIX 7



Technologie planaire

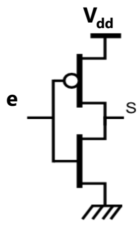


Technologie 3D (FinFET)

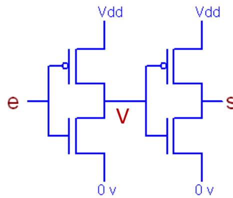
1.4. L'INVERSEUR

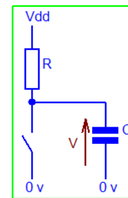
ENSI CAEN

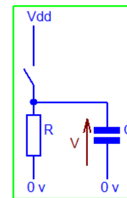
Illustration du temps de propagation quand e passe de 0 à 1



Structure

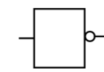


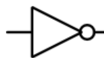




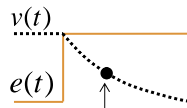
Double inversion

Schémas équivalents





Symbole



$$v(t) = V_{dd}\left(e^{-\frac{t}{RC}}\right)$$

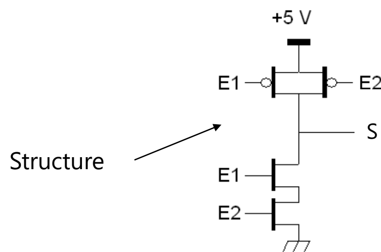
Changement d'état de s(t) à v(t)=$V_{dd}/2$

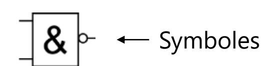
Il y a un temps de propagation dans tout circuit logique.
Sa connaissance permet d'adapter la fréquence de travail.

1.5. LA PORTE NAND

ENSI CAEN

Structure



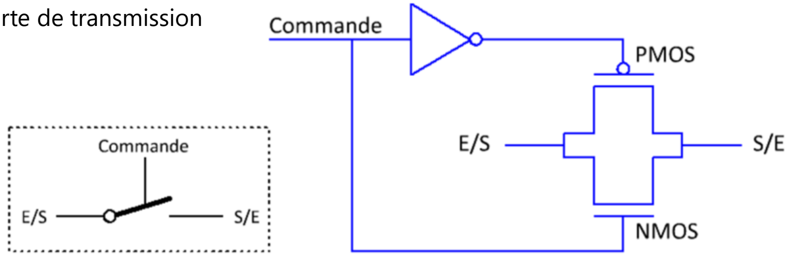


Symboles

- La porte NAND à 2 entrées, en technologie CMOS est constituée de 4 transistors.
- Toute fonction logique combinatoire peut être réalisée à partir de portes NAND. Ce qui en fait une porte universelle.
- La porte NAND est choisie comme unité de mesure pour évaluer la densité d'un circuit logique programmable.
- Exemple : un composant de densité 1000 portes peut embarquer une logique équivalente à 1000 portes NAND.

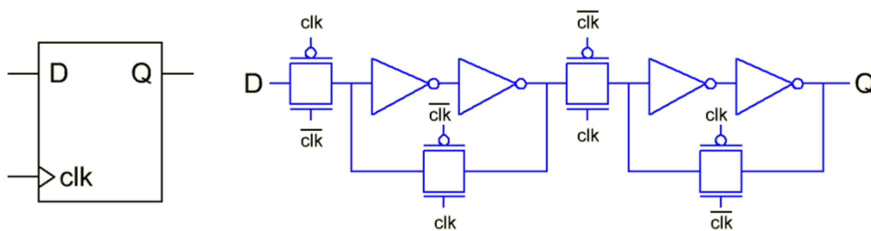
1.6. LA PORTE ANALOGIQUE

Porte de transmission

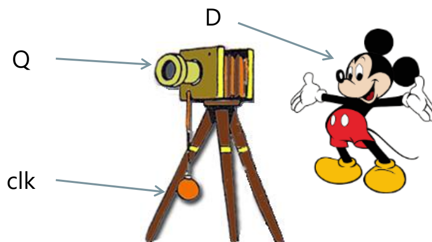


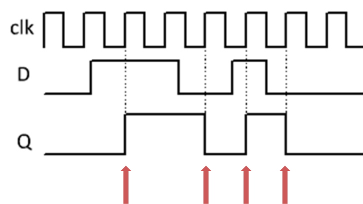
- Commande = 0L : les deux transistors présentent une impédance quasi infinie entre le drain et la source, le circuit se comporte comme un interrupteur ouvert
- Commande = 1L : les deux transistors en parallèle présentent une faible résistance R_{ON} équivalente entre le drain et la source, le circuit se comporte comme un interrupteur fermé

1.7. LA BASCULE D À FRONT

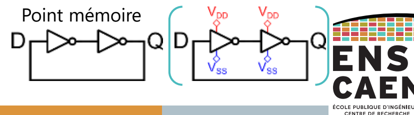


- La bascule D recopie l'entrée D sur la sortie Q sur front montant de clk.
- C'est un composant qui fige une situation comme le ferait un appareil photo.





1.7. LA BASCULE D À FRONT

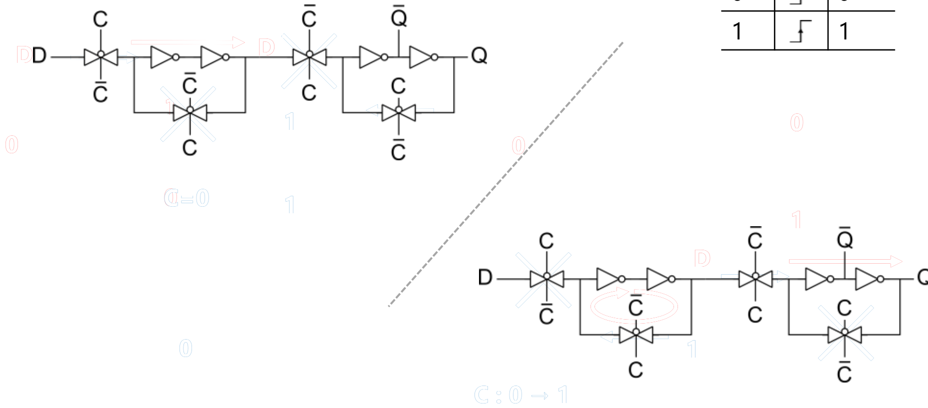


comment ça marche ?

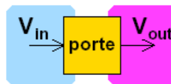
Structure interne possible de la bascule D à front.

Table de vérité de la bascule D à front

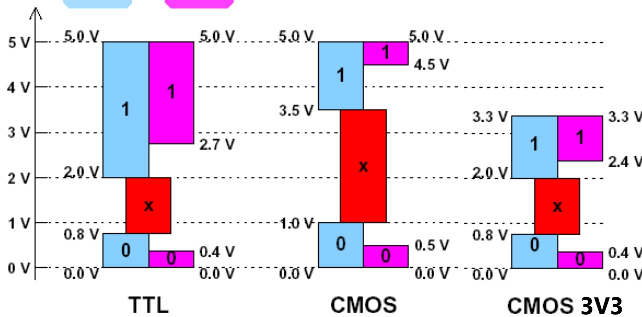
D	C	Q+
0	↑	0
1	↑	1



2.1. TENSIONS ET NIVEAUX LOGIQUES



$$P = k.f.V_{dd}^2$$



De nombreux standards

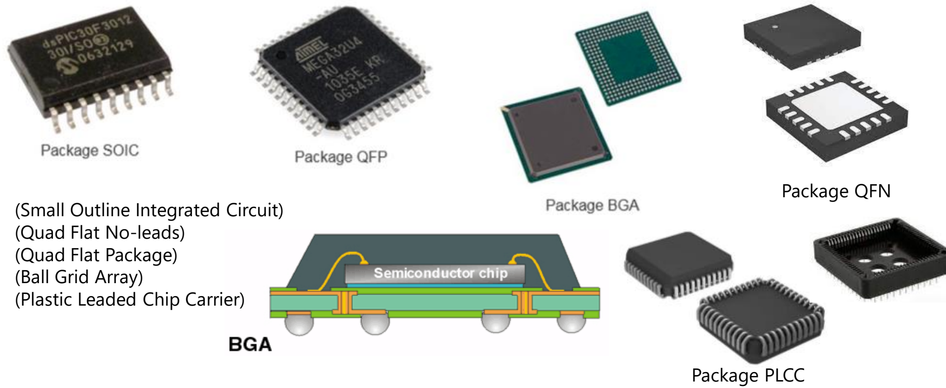
Exemple :
1.0 V +/- 0.1 V (NORMAL RANGE) AND 0.7 V - 1.1 V (WIDE RANGE) JESD8-14A.01

LVCMOS
(Low Voltage CMOS)

- Les fourchettes des niveaux logiques fixent les règles de dialogue entre les circuits tout en garantissant une immunité au bruit.
- Pour réduire la puissance consommée, on diminue la tension d'alimentation. Jusqu'à quelle valeur ?

Joint Electron Device Engineering Council
JEDEC Solid State Technology Association Document

2.2. LES BOITIERS BGA

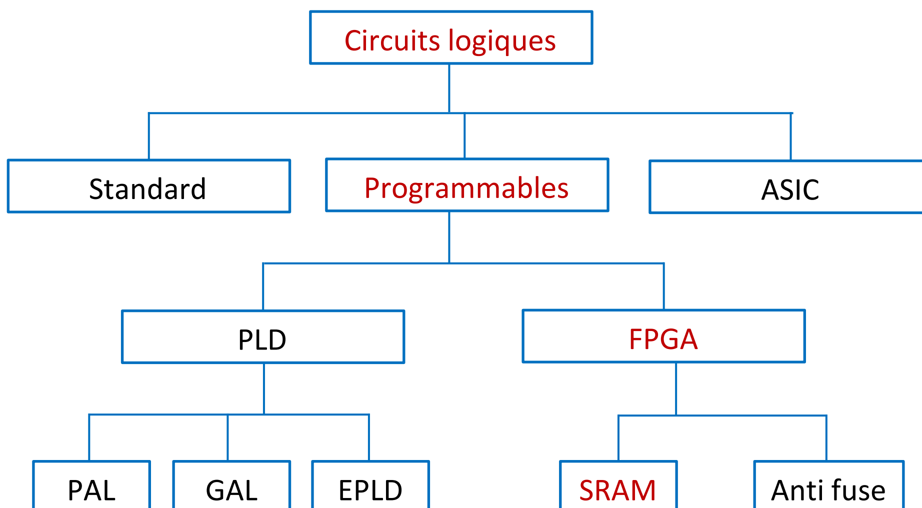


- Les boitiers de type BGA ont révolutionné la microélectronique en faisant exploser le nombre de broches > 1000.
- Le montage en surface de ces boitiers permet des économies non négligeables.
- La taille des composants se trouve réduite par la même occasion.

13

13

2.3. CLASSIFICATION DES CIRCUITS LOGIQUES



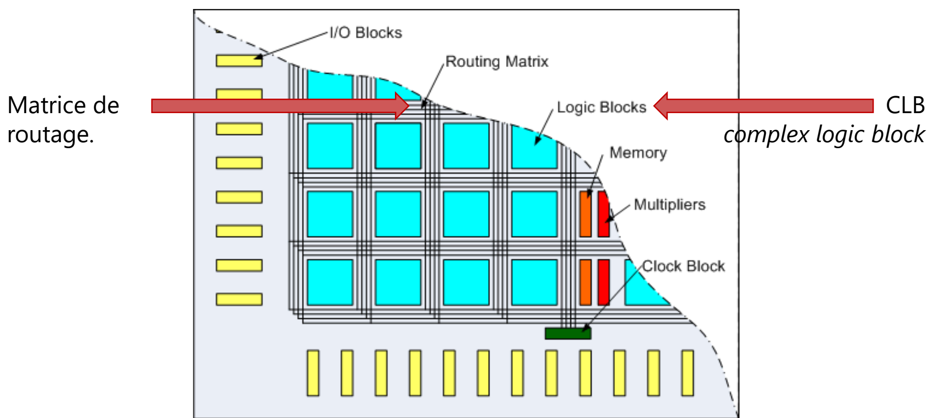
14

14

3.1. ARCHITECTURE DES FPGA

- Un circuit FPGA est une matrice symétrique comportant des ressources pour la logique, les entrées-sorties, les interconnexions et bien plus encore.

Matrice de routage.

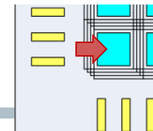


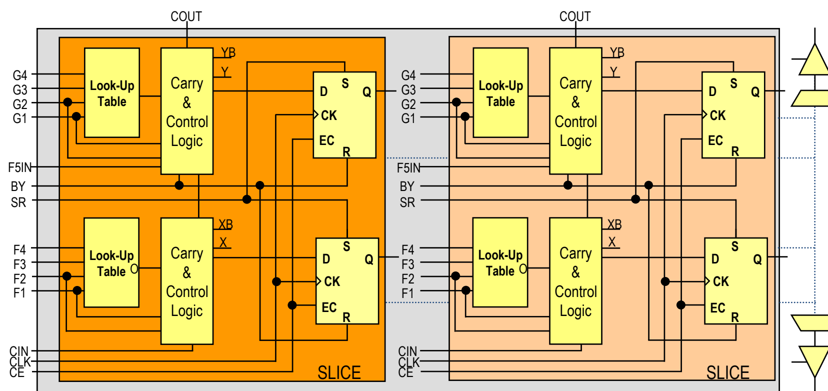
CLB
complex logic block

22

22

3.2. LE BLOC LOGIQUE CONFIGURABLE (CLB)



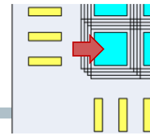


- Chaque CLB comporte 2 tranches (SLICE).
- Un slice contient 2 paires de (LUT + bascule + retenue).
- 2 buffers 3 états (BUFT) sont associés à chaque CLB.

Complex Logic Block

23

23

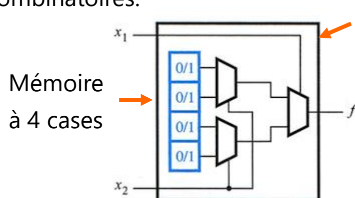


3.3. LOGIQUE COMBINATOIRE DANS LE FPGA

- Pour la réalisation des fonctions logiques combinatoires, les composants FPGA utilisent la technique de la table de consultation (LUT).
- Cette technique utilise des petites mémoires pour réaliser des fonctions combinatoires.

LUT à 2 entrées

Mémoire à 4 cases

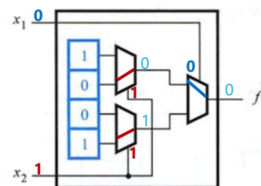


Remarque : si une LUT n'est pas utilisée pour de la logique, elle peut être utilisée comme une mémoire simple.

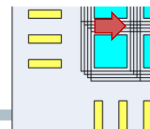
x_1	x_2	f_1
0	0	1
0	1	0
1	0	0
1	1	1

$f_1 = \bar{x}_1\bar{x}_2 + x_1x_2$

Fonction combinatoire à 2 entrées

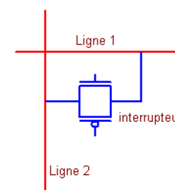


Réalisation dans une LUT à 4 adresses (2 bits d'adresse)

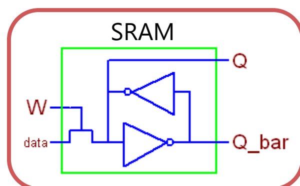


3.4. TECHNOLOGIE DE PROGRAMMATION

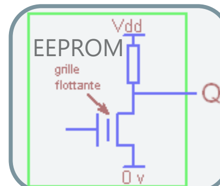
- Les composants logiques programmables comportent un ensemble de portes logiques préexistantes que l'on utilise ou non en les connectant ou non. La manière dont on les connecte entre elles crée les fonctions logiques souhaitées.
- La programmation d'un composant revient à fermer ou ouvrir l'interrupteur qui relie les 2 lignes qui se croisent.
- **Différentes technologies sont mises en œuvre pour commander ou réaliser l'interrupteur.**



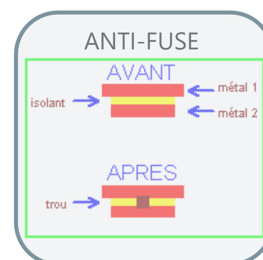
SRAM



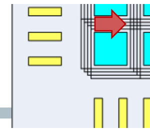
EEPROM



ANTI-FUSE

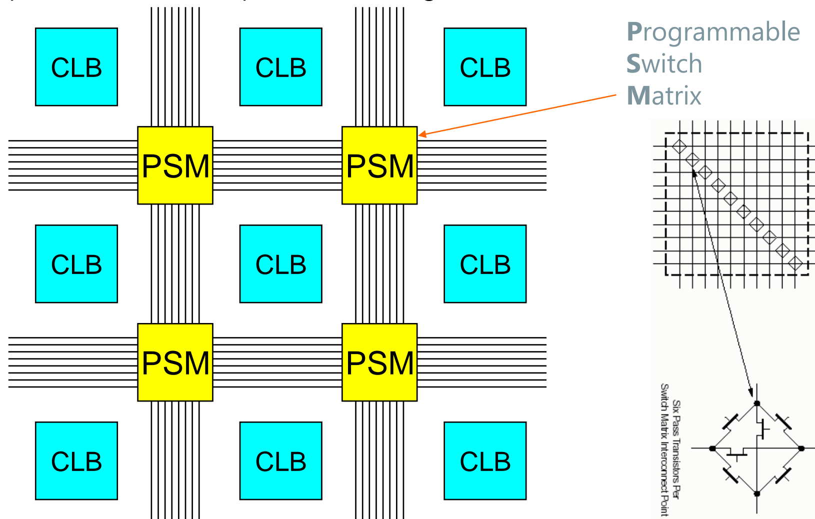


	Antifusible	EEPROM	SRAM
Facilité de programmation	**faible**	**Très importante**	**importante**
Densité	**faible**	**élevée**	**Très élevée**
Famille de circuit	**FPGA**	**PLD, CPLD**	**CPLD, FPGA**
Reprogrammable	**non**	**oui**	**oui**



3.5. LES MATRICES D'AIGUILLAGE

- Les ressources de routage sont abondantes dans un FPGA car le routage est un point clef dans la compilation d'un design.

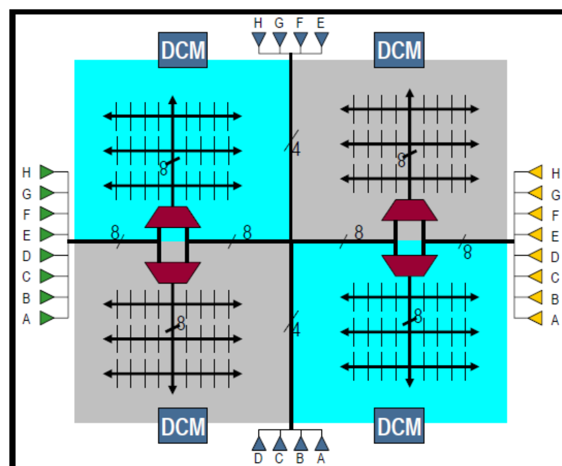


27

27

3.6. ROUTAGE DE L'HORLOGE

- Un réseau de routage est spécialement dédié à l'horloge pour lui permettre d'atteindre sans délai toutes les bascules du composant.

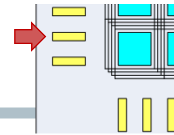


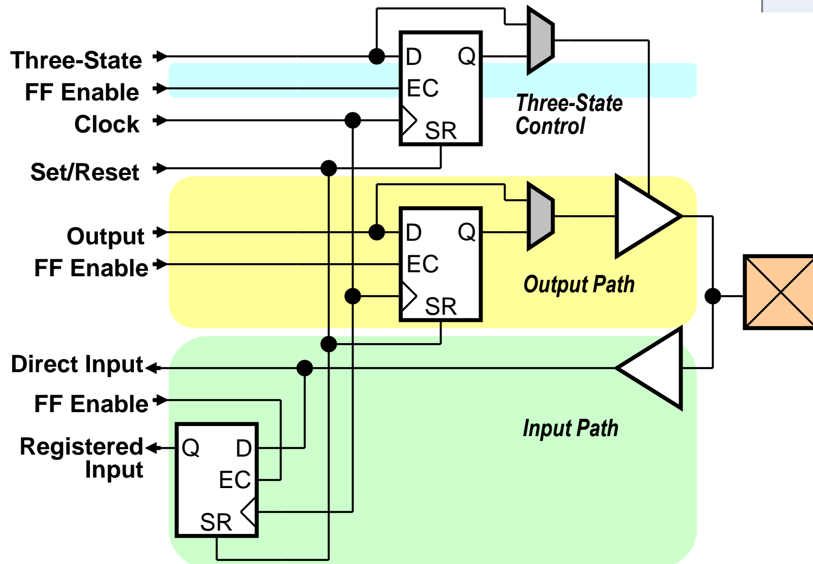
DCM : Digital Clock Manager

28

28

3.7. LE BLOC D'ENTRÉE SORTIE





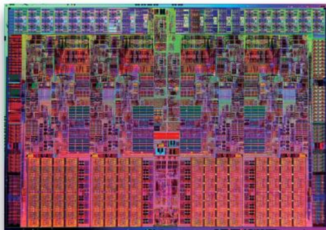
29

29

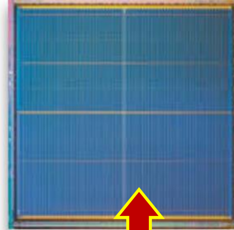
3.8. ARCHITECTURE SILICIUM DU FPGA

Architecture silicium des circuits

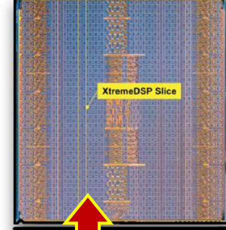
Intel Processor Core-i7



XILINX Virtex II

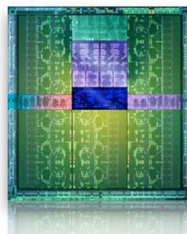


XILINX Virtex 4



On voit la structure régulière sous forme de matrice des FPGA par rapport aux structures processeurs.

NVIDIA K110 GPU



Intel Processor Quad-core Haswell

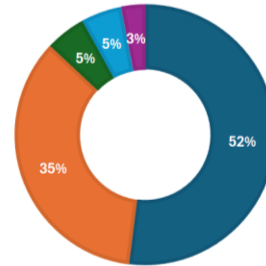
30

30

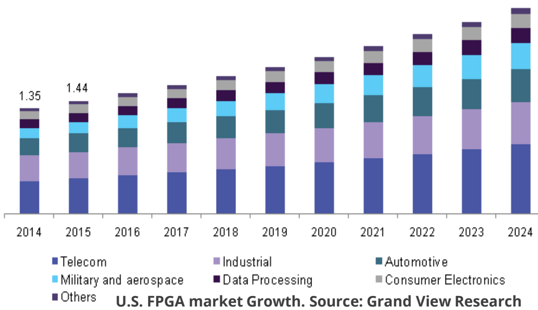
4.1. ACTEURS DU MARCHÉ DES FPGA

ENSI CAEN

Vendor	Market share 2020
AMD Xilinx	52
Intel (Altera)	35
Lattice	5
Microchip	5
Others	3



Source : the information network



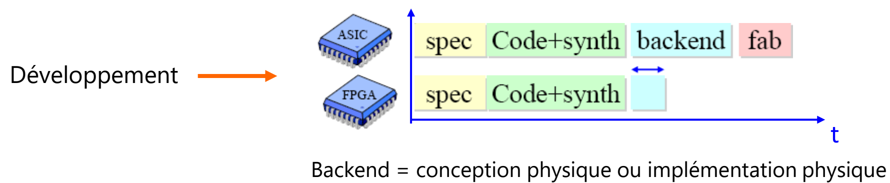
U.S. FPGA market Growth. Source: Grand View Research

31

4.2. CHOIX ENTRE FPGA/CPLD ET ASIC

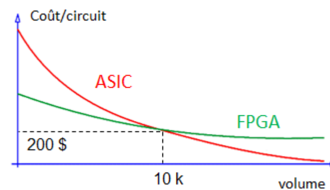
ENSI CAEN

Développement



Backend = conception physique ou implémentation physique

CRITERE	AVANTAGE
Temps de développement	FPGA
Coût petit volume	FPGA
Coût gros volume	ASIC
Performances	ASIC
Consommation	ASIC
Evolution, maintenance	FPGA



32

4.3. SECTEURS D’APPLICATION DES FPGA

Boosté par les développements véhicule autonome

Automotive Device Families

- XA Spartan-6 FPGAs
- XA Spartan-3A FPGAs
- XA Spartan-3A DSP FPGAs
- XA Spartan-3E FPGAs

Space-grade Device Families

- Space-grade Virtex-5QV FPGAs
- Space-grade Virtex-4QV FPGAs

Defense-grade Device Families

- Defense-grade Artix-7Q FPGAs
- Defense-grade Kintex-7Q FPGAs
- Defense-grade Virtex-7Q FPGAs
- Defense-grade Virtex-6Q FPGAs
- Defense-grade Spartan-6Q FPGAs
- Defense-grade Virtex-5Q FPGAs
- Defense-grade Virtex-4Q FPGAs

33

33

8.1. LES COMPOSANTS DE LA SÉRIE 7

	TP ARTIX 7	KINTEX 7	VIRTEX 7
Maximum Capability	Lowest Power and Cost	Industry’s Best Price/Performance	Industry’s Highest System Performance
Logic Cells	20K – 355K	70K – 480K	285K – 2,000K
Block RAM	12 Mb	34 Mb	65 Mb
DSP Slices	40 – 700	240 – 1,920	700 – 3,960
Peak DSP Perf.	504 GMACS	2,450 GMACs	5,053 GMACS
Transceivers	4	32	88
Transceiver Performance	3.75Gbps	6.6Gbps and 12.5Gbps	12.5Gbps, 13.1Gbps and 28Gbps
Memory Performance	1066Mbps	1866Mbps	1866Mbps
I/O Pins	450	500	1,200
I/O Voltages	3.3V and below	3.3V and below 1.8V and below	3.3V and below 1.8V and below

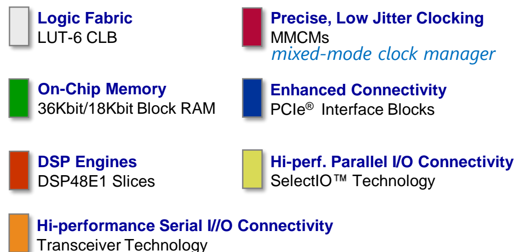
42

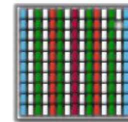
42

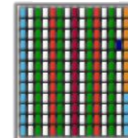
8.2. UNE ARCHITECTURE ALIGNÉE

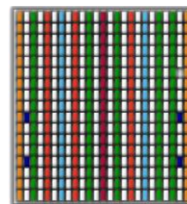
Common elements enable easy IP reuse for quick design portability across all 7 series families

- Design scalability from low-cost to high-performance
- Expanded eco-system support
- Quickest TTM *(Time To Market)*




Artix™-7 FPGA


Kintex™-7 FPGA



IP : Intellectual Property

43

43

8.3. LE BLOC LOGIQUE

Two side-by-side slices per CLB

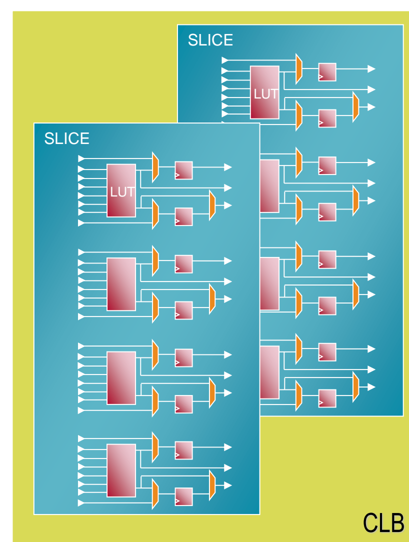
- Slice_M are memory-capable
- Slice_L are logic and carry only

Four 6-input LUTs per slice

- Consistent with previous architectures
- Single LUT in Slice_M can be a 32-bit shift register or 64 x 1 RAM

Two flip-flops per LUT

- Excellent for heavily pipelined designs



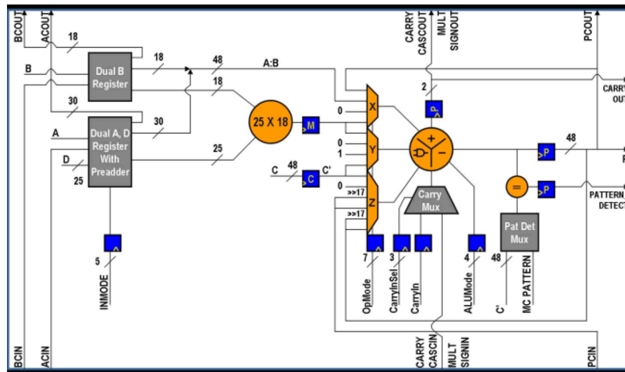
44

44

8.4. LE BLOC DSP

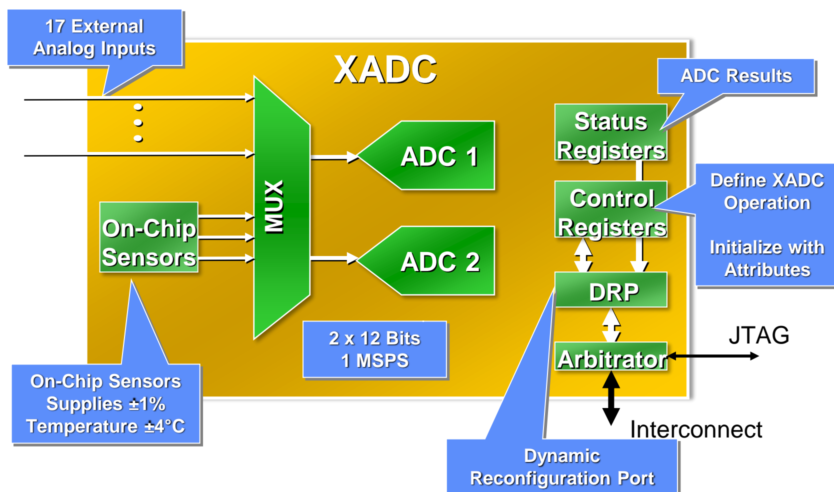
All 7 series FPGAs share the same DSP slice

- 25x18 multiplier
- 25-bit pre-adder
- Flexible pipeline
- Cascade in and out
- Carry in and out
- 96-bit MACC
- 48-bit ALU
- 17-bit shifter
- Dynamic operation (cycle by cycle)



45

8.5. LE CONVERTISSEUR ADC



46

8.6. RÉFÉRENCES DE LA FAMILLE 7

SPARTAN 7

XC	7	S	###	-1	FG	G	A	484	C
Commercial Xilinx	Generation	Family	Logic Cells In 1K units	Speed Grade -1 = Slowest -L1 = Low Power -2 = Mid	Package Type CP: Wire-bond (.5 mm) CS: Wire-bond (.8 mm) FG: Wire-bond (1 mm) FT: Wire-bond (1 mm)	G: RoHS 6/6	Package Designator	Package Pin Count	Temperature Grade (C, I, Q)

ARTIX 7

XC	7	A	###	-1	FB	G	484	C
Xilinx Commercial	Generation	Family	Logic Cells in 1K Units	Speed Grade -1 = Slowest -L1 = Low Power -L2 = Low Power -2 = Mid -3 = Highest	Package Type CP: Wire-bond (.5 mm) CS: Wire-bond (.8 mm) FB: Bare-Die Flip-Chip (1 mm) FF: Flip-Chip (1 mm) FG: Wire-bond (1 mm) FT: Wire-bond (1 mm) SB: Bare-Die Flip-Chip (.8 mm)	V: RoHS 6/6 G: RoHS 6/6 w/Exemption 15	Nominal Package Pin Count	Temperature Grade (C, E, I)

En TP : xc7a100tcsg324-3

KINTEX 7

XC	7	K	###	-1	FF	G	900	C
Xilinx Commercial	Generation	Family	Logic Cells in 1K Units	Speed Grade -1 = Slowest -L2 = Low Power -2 = Mid -3 = Highest	Package Type FB: Bare-Die Flip-Chip (1 mm) FF: Flip-Chip (1 mm)	V: RoHS 6/6 G: RoHS 6/6 w/Exemption 15	Nominal Package Pin Count	Temperature Grade (C, E, I)

VIRTEX 7

XC	7	V	###	-1	FF	G	1156	C
Xilinx Commercial	Generation	Family	Logic Cells in 1K Units	Speed Grade -1 = Slowest -2 = Mid -L2 = Low Power -3 = Highest	Package Type FF: Flip-Chip (1 mm) FH: Flip-Chip (1 mm) FL: Flip-Chip (1 mm) HC: Ceramic Flip-Chip (1 mm)	V: RoHS 6/6 G: RoHS 6/6 w/Exemption 15	Nominal Package Pin Count	Temperature Grade (C, E, I)

Notes:
-L1 is the ordering code for the lower power, -1L speed grade.
-L2 is the ordering code for the lower power, -2L speed grade.

C = Commercial (Tj = 0°C to +85°C) E = Extended (Tj = 0°C to +100°C) I = Industrial (Tj = –40°C to +100°C) Q = Expanded (Tj = –40°C to +125°C)

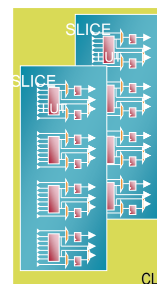
49

49

8.7. NOTION DE « LOGIC CELL »

	Artix®-7 FPGAs Optimized for Lowest Cost and Lowest Power Applications (1.0V, 0.95V, 0.9V)				
Part Number	XC7A15T	XC7A35T	XC7A50T	XC7A75T	XC7A100T
Logic Cells	16,640	33,280	52,160	75,520	101,440
Slices	2,600	5,200	8,150	11,800	15,850

- Une « logic cell » représente une LUT à 6 entrées. La logique de calcul de retenue est comptabilisée en appliquant un coefficient 1,6.
  - $1\ LC = 1.6\ LUT$
- Chaque LUT est associée à 2 bascules D.
- Prenons en exemple le composant XC7A100T



Pour Artix-7, chaque logic slice contient 4 LUT 6 entrées

$$nb\ de\ LUT = 4 * 15850 = 63400$$
$$nb\ de\ LC = 1.6 * 63400 = 101440$$

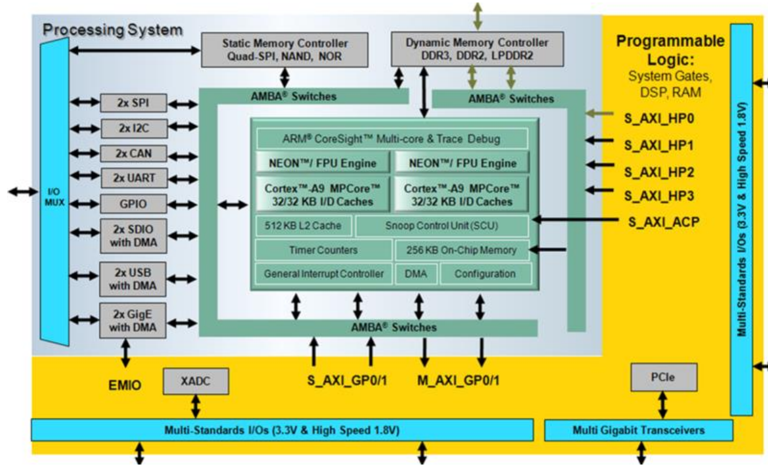
50

50

9. LE COMPOSANT ZYNQ 7000

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

- Il embarque un processeur ARM9 à 2 cœurs, des contrôleurs de différents types de mémoires, diverses interfaces et de la logique programmable.
- Ces 2 parties communiquent via un bus AXI. Circuit idéal pour les System On Chip.



10.1. DÉVELOPPEMENT

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

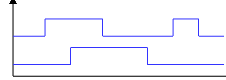
Design and implement a simple unit permitting to speed up encryption with RC5-similar cipher with fixed key set on 8031 microcontroller. Unlike in the experiment 5, this time your unit has to be able to perform an encryption algorithm by itself, executing 32 rounds…..

Specifications

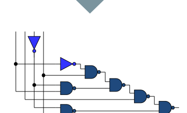
Description VHDL

```
Library IEEE;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity RC5_core is
    port(
        clock, reset, encr_decr: in std_logic;
        data_input: in std_logic_vector(31 downto 0);
        data_output: out std_logic_vector(31 downto 0);
        out_full: in std_logic;
        key_input: in std_logic_vector(31 downto 0);
        key_read: out std_logic;
    );
end AES_core;
```

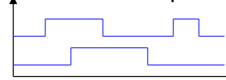
Simulation fonctionnelle



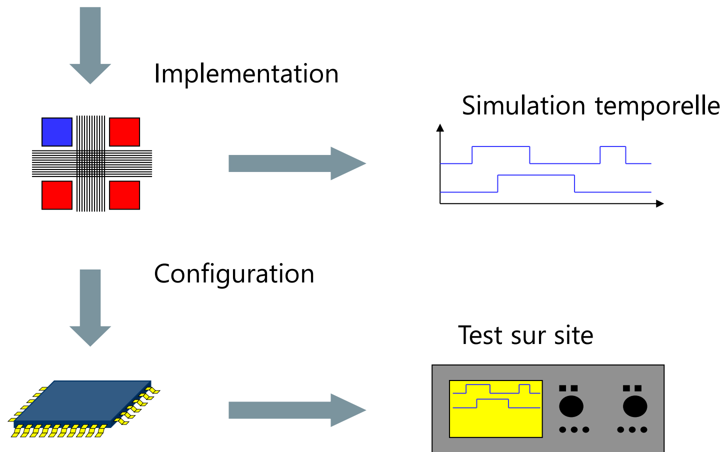
Synthése logique



Simulation après synthèse



DÉVELOPPEMENT-SUITE



10.1. DÉVELOPPEMENT

simulation : Pour vérifier que le code répond au cahier des charges (avant synthèse et implémentation car ces étapes prennent du temps).
Pour des applications critiques, il est souhaitable de simuler après synthèse et implémentation pour vérifier les chemins critiques et le respect des contraintes de temps.

analyse RTL : Structuration au niveau architecture registre (Register Transfer Level) **Quoi ?**

synthèse : Synthèse logique: associe des portes logiques pour réaliser la fonction **Comment ?**

implémentation : Détermine physiquement quelle porte est utilisée dans le FPGA pour chaque porte logique et réalise le routage associé. Placement & routage. **Où ?**

programmation : Programme le composant.

10.1. DÉVELOPPEMENT

analyse RTL

Structuration au niveau architecture registre (Register Transfer Level)

Fonctions élémentaires : registres, multiplexeurs, additionneurs, …

55

55

10.1. DÉVELOPPEMENT - SYNTHÈSE

synthèse

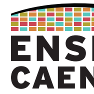
Synthèse logique: associe des portes logiques pour réaliser la fonction. Prend en compte les contraintes I/O Pin du fichier de contrainte.

Nombres/types de cellules utilisées

56

56

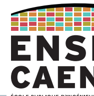
10.1. DÉVELOPPEMENT - IMPLÉMENTATION

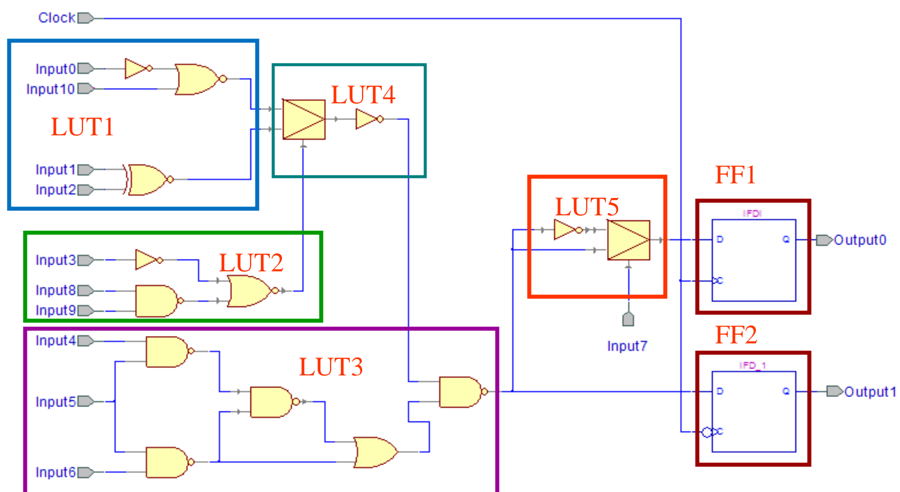


Détermine physiquement quelle porte est utilisée dans le FPGA pour chaque porte logique et réalise le routage associé.
Placement & routage

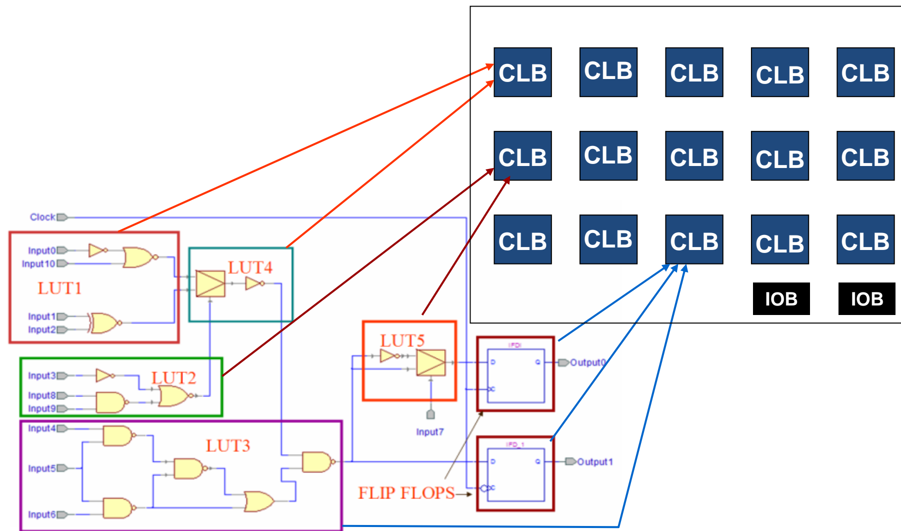
implémentation

10.2. IMPLÉMENTATION - MAPPING





IMPLÉMENTATION - PLACEMENT

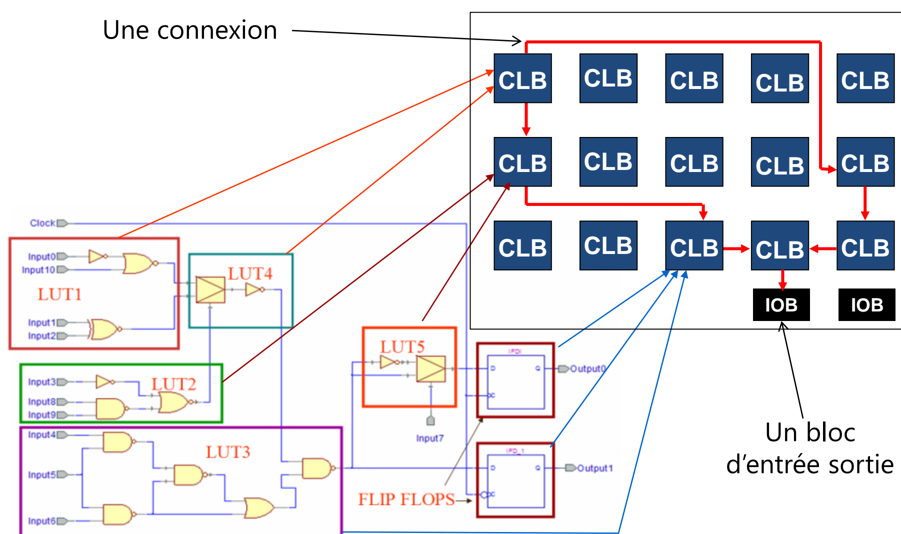


59

59

IMPLÉMENTATION - ROUTAGE

Une connexion



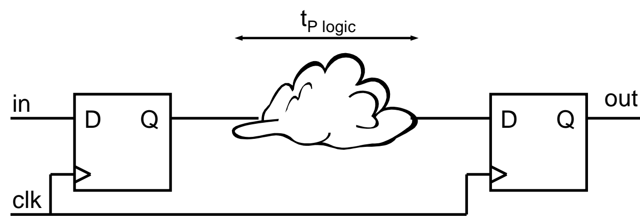
Un bloc
d'entrée sortie

60

60

10.3. LE CHEMIN CRITIQUE

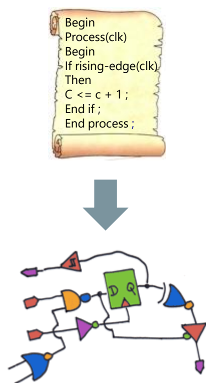
- Chemin critique = chemin reliant deux bascules et ayant le plus long délai.
- C'est de lui que dépendra la fréquence maximale de l'horloge.
- Période minimale d'horloge = délai du chemin critique.
- Celle-ci est calculée automatiquement par les outils de développement IDE.



$$t_{Critical} = t_{P\ FF} + t_{P\ logic} + t_{S\ FF}$$

61

61



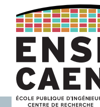
Le langage VHDL

Matthieu DENOUAL

L'École des INGÉNIEURS Scientifiques

1

1.1. PRÉSENTATION



- Vhsic Hardware Description Langage (Vhsic = Very High Speed Integrated Circuit) est un langage de description de matériel et non un langage software comme le C.
- Standardisé en 1987 par l 'IEEE (Institute Of Electrical and Electronics Engineers) sous la référence IEEE 1076-87. Une mise à jour importante a été faite en 1993 : IEEE 1076-93 est supportée par tous les outils. Dernière révision en 2008.
- Utilisé au début pour la modélisation et la simulation avant d'être adopté pour la synthèse logique. Toute la syntaxe n'est pas synthétisable !!!

simulation

- Permet de décrire un système avec un niveau d'abstraction élevé « algorithmique » ou un niveau proche du matériel « gate level ». Entre les deux se trouve le niveau RTL « Register Transfert Level ».

hiérarchique

- C'est le niveau RTL qui est utilisé pour la synthèse car il est moins lourd que le niveau « gate level » et il est indépendant de la cible. Le niveau « algorithmique » n'est pas forcément synthétisable.
- *70% de la syntaxe ADA : langage de programmation orienté objet de 1980 utilisé dans les systèmes temps réel et embarqués nécessitants fiabilité et sécurité.*

2

2

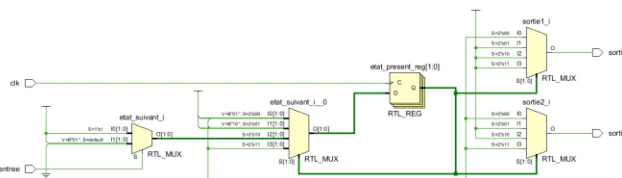
1.1. PRÉSENTATION

- Différences entre langage informatique (exemple langage C) et VHDL :

langage informatique	**VHDL**
Sous-programme (fonction, procédure) : Début durée Fin	Composant : Décrit une partie du matériel Hors du temps Permanent Concurrent → langage parallèle
Variables : temps de vie limité A = B; A = C;	Signaux : représentent une connexion entre composants A <= B; A <= C; *B et C connectés à A*
Derrière le code : *Mémoire programme, mémoire données, architecture matérielle figée*	*Derrière le code :* *Structure matérielle, connectique* **Langage informatique pour électronicien**

1.2. LA SYNTHÈSE LOGIQUE

```vhdl
entity MAEF3p is
    Port ( clk : in STD_LOGIC;
           entree : in STD_LOGIC;
           sortie1 : out STD_LOGIC;
           sortie2 : out STD_LOGIC);
end MAEF3p;

architecture Behavioral_3p of MAEF3p is
type type_etat is (etat0, etat1, etat2, etat3);
signal etat_present : type_etat;
signal etat_suivant : type_etat;
begin

PROC_SEQ : process(clk)
begin
    if rising_edge(clk) then
        etat_present <= etat_suivant;
    end if;
end process PROC_SEQ;

COMB_G : process(etat_present, entree)
begin
    case etat_present is
        when etat0 => etat_suivant <= etat1;
        when etat1 => etat_suivant <= etat2;
        when etat2 => if entree = '1' then
                          etat_suivant <= etat3;
                      else
                          etat_suivant <= etat1;
                      end if;
        when etat3 => etat_suivant <= etat0;
    end case;
end process COMB_G;

COMB_F : process(etat_present)
begin
    case etat_present is
        when etat0 => sortie1 <= '0'; sortie2 <= '0';
        when etat1 => sortie1 <= '1'; sortie2 <= '1';
        when etat2 => sortie1 <= '1'; sortie2 <= '1';
        when etat3 => sortie1 <= '1'; sortie2 <= '0';
    end case;
end process COMB_F;

end Behavioral_3p;
```



- La synthèse logique est l'opération qui consiste à traduire le code VHDL en fonctions logiques et bascules prêtes à être connectées dans le silicium.

1.3. SIMULATION / SYNTHÈSE

- On rencontre deux langages de description de matériel : VHDL (populaire en Europe, proche de Ada « pgm objet ») et Verilog (populaire aux US).
- Le langage VHDL permet de :
  1. Modéliser des circuits pour la simulation
  2. Décrire des applications pour circuits ASIC ou programmables (FPGA)

Modélisation pour la simulation	Description de système matériel
Norme IEEE 1076 La totalité de la norme peut être utilisée pour la modélisation	Norme IEEE 1076 Une partie seulement peut être utilisée pour la synthèse

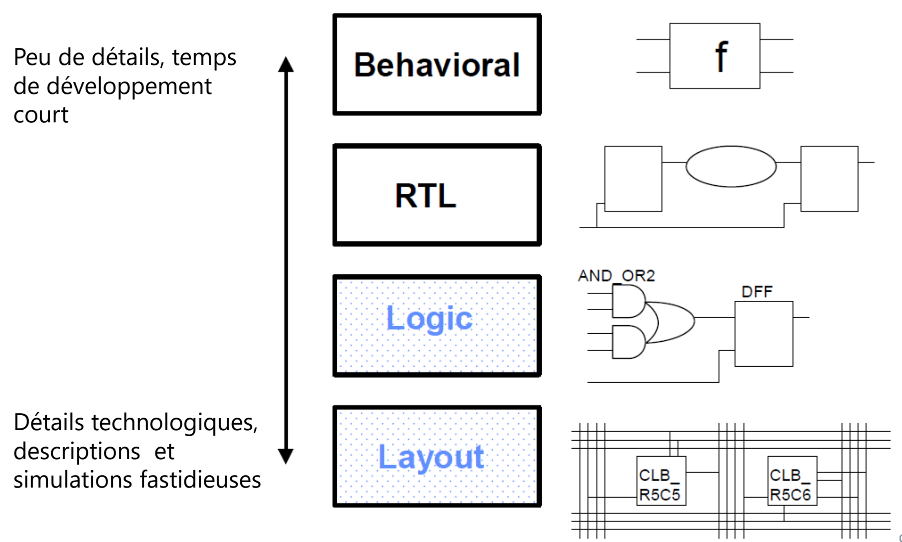
```
X <= '1' after 100 ns;
Wait for 200 ns;
Signal ent_s : std_logic := '0'
```

Les instructions after et wait à gauche ne peuvent pas être utilisées pour la synthèse

5

5

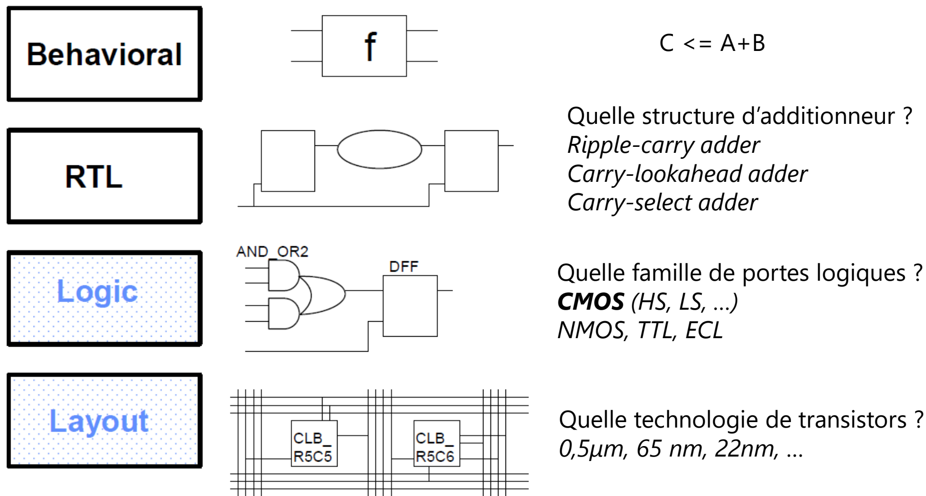
1.4. NIVEAUX D'ABSTRACTION



9

9

1.4. NIVEAUX D'ABSTRACTION



10

10

2.1. PREMIER EXEMPLE

VHDL : langage de description matériel
(HDL : Hardware Description Langage)

```
-------------------------------------------
-- Voici un exemple simple
-------------------------------------------
--  La déclaration des bibliothèques

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

-------------------------------------------
--          L'entité

entity portes is
    Port ( a : in std_logic;
           b : in std_logic;
           c : in std_logic;
           x : out std_logic;
           y : out std_logic;
           z : out std_logic);
end portes;
-------------------------------------------
--          L'architecture

architecture Behavioral of portes is
begin
     x <= a or b ;
     y <= b nand (not c) ;
     z <= a xor c ;
end Behavioral;
```

commentaire

Appel librairies

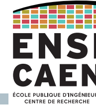
a, b, c → portes → x, y, z

entité

architecture

11

2.2. DEUXIÈME EXEMPLE




```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
--
--    commentaire
--
entity clk_div is
     generic (N : natural := 5);
     Port ( reset : in std_logic;
            clk : in std_logic;
            div_out : out std_logic);
end clk_div;

architecture Behavioral of clk_div is
     signal count_temp : std_logic_vector (N downto 0);

begin       -- autre commentaire

process (reset, clk)

begin
     if reset='0' then
          count_temp <= (others => '0');
     elsif (clk'event and clk='1') then
          count_temp <= count_temp + 1;
     end if;

end process;

div_out <= count_temp(N);

end Behavioral;
```

clk, reset → Clk_div → div_out

La nouveauté dans cet exemple est le bloc « process ».

Ce bloc est très pratique pour décrire des unités cadencées par une horloge.

On peut écrire plusieurs process, ils s'exécutent tous en parallèle.

12

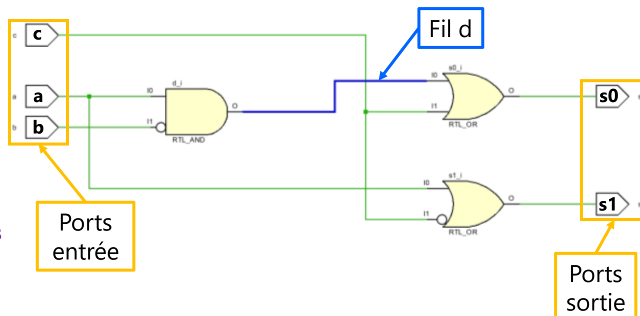
12

IMPORTANT, À RETENIR [1/2]

Le langage VHDL est un langage de description matériel, donc par essence proche du matériel.

La description de l'entité définit les ports d'entrée/sortie.
<u>Un signal correspond à un fil</u>.

```vhdl
entity ex_fils is
 Port ( a : in STD_LOGIC;
        b : in STD_LOGIC;
        c : in STD_LOGIC;
        s0 : out STD_LOGIC;
        s1 : out STD_LOGIC);
end ex_fils;

architecture Flow of ex_fils is
   signal d : STD_LOGIC;
begin
d <= a and (not b);
s0 <= d or c;
s1 <= a or (not c);
end Flow;
```



Ports entrée/sortie

Fil d   en sortie de fonction logique a./b

Nom des fils reprend le nom des ports.

13

13

IMPORTANT, À RETENIR [2/2]

Le langage VHDL est un langage de description matériel, donc par essence proche du matériel.

Les process (processus) permettent de définir des fonctions logiques qui se déroulent en parallèle (on parlera d'exécution concurrente).

```
entity ex_process is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           c : in STD_LOGIC;
           s0 : out STD_LOGIC;
           s1 : out STD_LOGIC);
end ex_process;

architecture Behavioral of ex_process is
begin
s0 <= a and (not b);
process(a)
    begin
        if a = '1' then
            s1 <= b;
        else
            s1 <= c;
        end if;
end process;
end Behavioral;
```

(processus implicite)

(processus explicite)

a, b, c → RTL_AND → s0 ; RTL_MUX → s1

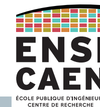
Ports entrée/sortie

implicite et explicite fonctionnent en parallèle.

14

14

PROCESSUS IMPLICITE OU EXPLICITE



```
architecture Behavioral of circuit is
   signal inter : std_logic ;

begin
   X <= (A or B) and C ;

   process (A,C)
      begin
         if (A='1' and C='0') then
            inter <= '1';
         else
            inter <= '0';
         end if;
   end process;

   Y <= inter;

end Behavioral;
```

Début et fin

Equation logique (processus implicite)

Bloc « process » (processus explicite)

Connexion (processus implicite)

L'ordre des processus n'a pas d'importance étant donné qu'ils s'exécutent de façon concurrente.

15

15

2.3. TROIS RÈGLES DE BASE

```vhdl
-----------------------------------------
-- Programme de démonstration
-----------------------------------------

library IEEE; -- librairie IEEE
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity majuscule is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           c : in  STD_LOGIC;
           x : out  STD_LOGIC;
           y : out  STD_LOGIC;
           z : out  STD_LOGIC);
end majuscule;

architecture Behavioral of majuscule is
   signal INTER : std_logic;

begin
   inter <= a or b;
   y <= not INTER;
   z <= a xor c;
end Behavioral;
```

Les commentaires commencent par un double tiret « -- » et se terminent à la fin de la ligne.

VHDL ne distingue pas les majuscules des minuscules :

inter et INTER désignent le même signal

Une instruction se termine par un « ; »
les espaces ne sont pas significatifs

2.4. LA LIBRAIRIE IEEE

- Le standard « IEEE.1164 » définit une librairie avec un certain nombre de packages dont certains sont indispensables pour tout programme VHDL.
- Les packages doivent être appelés avant la déclaration de l'entité.

```vhdl
library IEEE; -- librairie IEEE
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
```

- L'extension « .all » du nom de la librairie signifie qu'on intègre tout le contenu.

Standard Package

The Standard package predefines a number of types, subtypes, and functions.

```vhdl
package standard is
  type boolean is (false,true);
  type bit is ('0', '1');

  type severity_level is (note, warning, error, failure);
  type integer is range -2147483647 to 2147483647;
  type real is range -1.0E308 to 1.0E308;
  type time is range -2147483647 to 2147483647
```

```vhdl
type character is (
  nul, soh, stx, etx, eot, enq, ack, bel,
  bs,  ht,  lf,  vt,  ff,  cr,  so,  si,
  dle, dc1, dc2, dc3, dc4, nak, syn, etb,
  can, em,  sub, esc, fsp, gsp, rsp, usp,

  ' ', '!', '"', '#', '$', '%', '&', ''',
  '(', ')', '*', '+', ',', '-', '.', '/',
  '0', '1', '2', '3', '4', '5', '6', '7',
  '8', '9', ':', ';', '<', '=', '>', '?',
  '@', 'A', 'B', 'C', 'D', 'E', 'F', 'G',
  'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
  'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W',
```

Lien packages

2.4.1. LE PACKAGE STD_LOGIC_1164

- C'est ici qu'est défini le type « std_logic » qui est une extension du type « bit ».
  - 'U': uninitialized. This signal hasn't been set yet. ← Ex : RAM
  - 'X': unknown. Impossible to determine this value/result.
  - '0': logic 0
  - '1': logic 1
  - 'Z': High Impedance
  - 'W': Weak signal, can't tell if it should be 0 or 1.
  - 'L': Weak signal that should probably go to 0 ← Rappel à 0
  - 'H': Weak signal that should probably go to 1 ← Rappel à 1
  - '-': Don't care.
- Les opérations applicables à ce type sont : AND, NAND, OR, NOR, XOR, XNOR, NOT.
- Le type « std_logic_vector » est un tableau de « std_logic ».

```
architecture Behavioral of portes is
   signal s1, s2, s3 : std_logic_vector(3 downto 0);
begin
   s1(0) <= '1';
   s1(1) <= '0';
   s1(2) <= '0';
   s1(3) <= '1';
   s2 <= "1100";
   s3 <= s1 and s2;
```

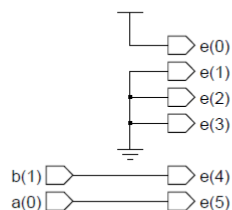

AGRÉGATS (RÉUNIONS D'ÉLÉMENTS)

```
a : in  std_logic_vector(2 downto 0);
b : in  std_logic_vector(2 downto 0);
c : out std_logic_vector(5 downto 0);
d : out std_logic_vector(5 downto 0);
e : out std_logic_vector(5 downto 0);
f : out std_logic_vector(2 downto 0);
```

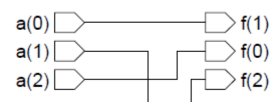
(MSB downto LSB)

leftmost in e — index

```
e <= (a(0), b(1), 0 => '1', others => '0');
```




```
(f(0), f(2), f(1)) <= a;
```



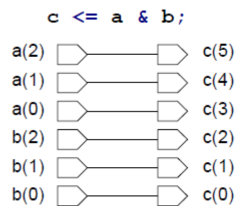
Rappel

MSB (Most Significant Bit) : bit de poids fort
LSB (Least Significant Bit) : bit de poids faible

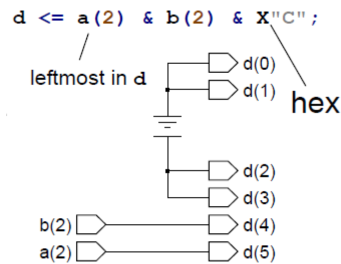
CONCATÉNATION (BOUT À BOUT)

```
a : in  std_logic_vector(2 downto 0);
b : in  std_logic_vector(2 downto 0);
c : out std_logic_vector(5 downto 0);
d : out std_logic_vector(5 downto 0);
e : out std_logic_vector(5 downto 0);
f : out std_logic_vector(2 downto 0);
```

```
c <= a & b;
```




```
d <= a(2) & b(2) & X"C";
```



LA FONCTION « RISING_EDGE »

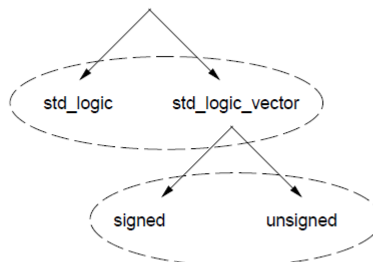
- Cette fonction, définie dans « std_logic_1164 », est très utile pour détecter les fronts montants d'une horloge.
- Elle vérifie bien que le signal part de '0' avant de passer à '1'.
- Il existe une fonction similaire qui teste les fronts descendants : « falling_edge »

```
process ( Clk, Reset )
begin
 if Reset = '1' then
    Q <= '0' ;
    elsif   rising_edge (Clk) then
    Q <= D ;
 end if ;
end process ;
```

```
package std_logic_1164  is
. . .
function rising_edge ( signal Clk : std_logic ) return boolean;
begin
   if ( Clk'event and Clk= '1' and Clk'last_ value = '0' ) then
     return true ;
        else
     return false ;
  end if  ;
end rising edge ;
. . .
```

2.4.2. LE PACKAGE « NUMERIC_STD »

- C'est ici que sont définis certains types pour représenter les entiers ainsi que les opérations arithmétiques.
  - The unsigned type
  - The signed type
  - The arithmetic functions: +, -, *
  - The comparison functions: <, <=, >, >=, =, /=
  - The shift functions: shl, shr
  - The conv_integer function
  - The conv_unsigned function
  - The conv_signed function
  - The conv_std_logic_vector function



- On y trouve aussi quelques fonctions de conversion de type comme celle-ci :.

```
conv_std_logic_vector(arg: unsigned, size: integer) return std_logic_vector;
```

```
architecture Behavioral of portes is
   signal b, c : std_logic_vector(3 downto 0);
   signal d : std_ulogic_vector(3 downto 0);

begin
   c <= b and d;                          -- FAUX
   c <= b and conv_std_logic_vector(d,4); -- BON
```

22

22

STD_LOGIC_UNSIGNED / STD_LOGIC_SIGNED

- Ces deux packages sont des extensions du package « numeric_std».
- Il ne faut appeler qu'un seul des deux à la fois !!!

STD_LOGIC_UNSIGNED :

- Dans ce package, les fonctions sont redéfinies pour traiter les nombres de type « std_logic_vector » comme des entiers non signés.

STD_LOGIC_SIGNED :

- Dans ce package, les fonctions sont redéfinies pour traiter les nombres de type « std_logic_vector » comme des entiers signés.
- Le **complément à 2** est utilisé pour la représentation des nombres négatifs.

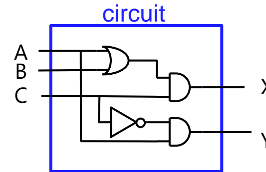
```
library IEEE; -- librairie IEEE
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;
-- or IEEE.std_logic_signed.all;
```

23

23

2.5. LE BLOC « ENTITY »

```
entity circuit is
    Port     ( A : in  STD_LOGIC;
               B : in  STD_LOGIC;
               C : in  STD_LOGIC;
               X : out  STD_LOGIC;
               Y : out  STD_LOGIC);
end circuit;
```



- L'entité donne une vue externe du circuit.
- Les signaux d'interface constituent dans la terminologie VHDL un « port ».
- Chaque signal doit posséder : un nom (choisi par l'utilisateur), un mode (in, out, inout) et un type (std_logic, std_logic_vector, integer, boolean, …).
- Dans le bloc entity, on peut déclarer un paramètre générique avec une valeur par défaut, si une architecture appelle cette entité elle pourra changer la valeur de N.

```
entity PARITE is
generic (N : integer := 8) ;
Port (ENTREE : in std_logic_vector(N-1 downto 0) ;
      P : out std_logic) ;
end PARITE ;
```

24

24

2.6. LE CHOIX DES NOMS

- Chaque élément manipulé par VHDL (signal, constante, bus, …) doit porter un nom. Celui-ci doit respecter les règles suivantes :
  1. Caractères admis : les 26 lettres de l'alphabet, les 10 chiffres décimaux et le caractère '_'.
  2. Le 1er caractère doit être une lettre.
  3. Le caractère '_' ne doit pas terminer un nom.
  4. Un nom ne doit pas être un mot réservé.
  5. La longueur d'un nom ne doit pas dépasser une ligne.

```
architecture Behavioral of portes is
   signal inter : std_logic;
   constant indice : integer := 12;
   signal sortie : std_ulogic_vector(3 downto 0);
```

Il faut respecter une règle simple : un nom doit permettre de deviner le type d'information représentée

Attention aux noms des fichiers, évitez les accents et caractères spéciaux → des problèmes observés lors des simulations.

25

25

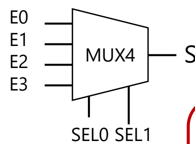
2.7. LE BLOC « ARCHITECTURE »

- Le bloc architecture décrit le système matériel à concevoir, cela peut être un système combinatoire ou séquentiel, simple ou complexe.

- Il existe trois façons de décrire un circuit électronique en VHDL :
  - Description de bas niveau : on écrit des équations logiques :
    Flow, flot
  - Description modulaire : il s'agit d'associer des blocs existants :
    Structural, structurel
  - Description comportementale : on décrit le comportement du circuit :
    Behavioral, comportemental

2.7. LE BLOC « ARCHITECTURE »

```
entity MUX4 is
    Port ( E0 : in STD_LOGIC;
           E1 : in STD_LOGIC;
           E2 : in STD_LOGIC;
           E3 : in STD_LOGIC;
           SEL0 : in STD_LOGIC;
           SEL1 : in STD_LOGIC;
           S : out STD_LOGIC);
end MUX4;
```



- Exemple multiplexeur 4 vers1 : MUX4

Flow, flot

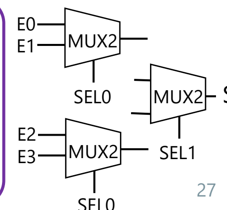
```
architecture Flow of MUX4 is
begin
    S <= ((not SEL0) and (not SEL1) and E0) or
         (SEL0 and (not SEL1) and E1) or
         ((not SEL0) and SEL1 and E2) or
         (SEL0 and SEL1 and E3);
end Flow;
```

Behavioral, comportemental

```
architecture Behavioral of MUX4 is
begin
process(SEL0,SEL1,E0,E1,E2,E3)
    begin
        if((SEL0= '0') and (SEL1='0')) then S<=E0;
        elsif((SEL0= '1') and (SEL1='0')) then S<=E1;
        elsif((SEL0= '0') and (SEL1='1')) then S<=E2;
        elsif((SEL0= '1') and (SEL1='1')) then S<=E3;
        end if;
    end process;
end Behavioral;
```

Structural, structurel

```
architecture Structural of MUX4 is
component MUX2
    Port ( E0, E1, SEL : in STD_LOGIC; S : out STD_LOGIC);
end component;
signal S_MUX2_a, S_MUX2_b :  STD_LOGIC;
begin
instance_MUX2_a : MUX2 port map(E0=>E0, E1=>E1, SEL=>SEL0, S=>S_MUX2_a);
instance_MUX2_b : MUX2 port map(E0=>E2, E1=>E3, SEL=>SEL0, S=>S_MUX2_b);
instance_MUX2_c : MUX2 port map(E0=>S_MUX2_a, E1=>S_MUX2_b, SEL=>SEL1, S=>S);
end Structural;
```

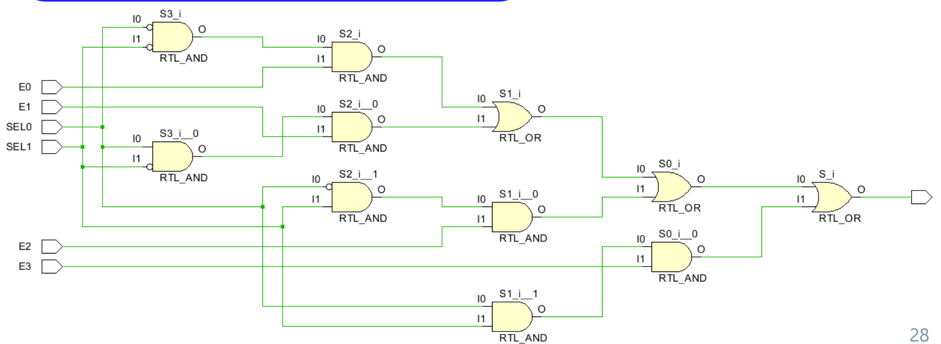


2.7. LE BLOC « ARCHITECTURE »

- Exemple multiplexeur 4 vers1 : MUX4

Flow, flot

```
architecture Flow of MUX4 is
begin
    S <= ((not SEL0) and (not SEL1) and E0) or
         (SEL0 and (not SEL1) and E1) or
         ((not SEL0) and SEL1 and E2) or
         (SEL0 and SEL1 and E3);
end Flow;
```



28

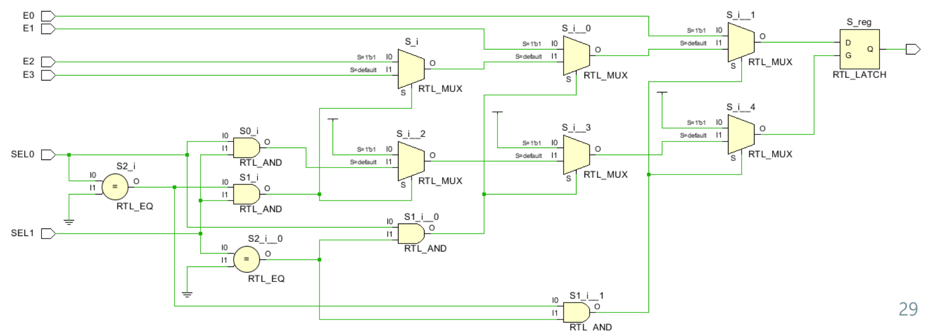
28

2.7. LE BLOC « ARCHITECTURE »

Exemple multiplexeur 4 vers1 : MUX4

```
architecture Behavioral of MUX4 is
begin
process(SEL0,SEL1,E0,E1,E2,E3)
    begin
        if((SEL0= '0') and (SEL1='0')) then S<=E0;
        elsif((SEL0= '1') and (SEL1='0')) then S<=E1;
        elsif((SEL0= '0') and (SEL1='1')) then S<=E2;
        elsif((SEL0= '1') and (SEL1='1')) then S<=E3;
        end if;
    end process;
end Behavioral;
```

Behavioral, comportemental

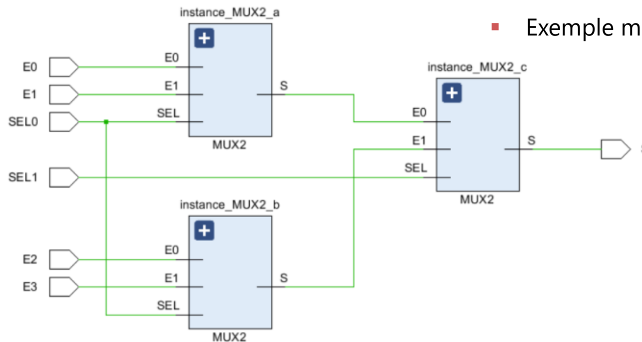


29

29

2.7. LE BLOC « ARCHITECTURE »

- Exemple multiplexeur 4 vers1 : MUX4



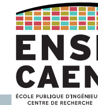
Structural, structurel

```
architecture Structural of MUX4 is
component MUX2
    Port ( E0, E1, SEL : in STD_LOGIC; S : out STD_LOGIC);
end component;
signal S_MUX2_a, S_MUX2_b :  STD_LOGIC;
begin
instance_MUX2_a : MUX2 port map(E0=>E0, E1=>E1, SEL=>SEL0, S=>S_MUX2_a);
instance_MUX2_b : MUX2 port map(E0=>E2, E1=>E3, SEL=>SEL0, S=>S_MUX2_b);
instance_MUX2_c : MUX2 port map(E0=>S_MUX2_a, E1=>S_MUX2_b, SEL=>SEL1, S=>S);
end Structural;
```

30

30

2.8. DÉCLARATIONS / INSTRUCTIONS



```
architecture Behavioral of controle is
    type etat_type is (zero, un, deux, trois, quatre, cinq);
    signal etat : etat_type := zero;
    signal etat_suiv : etat_type ;
    signal compteur : std_logic_vector(26 downto 0) :=(others => '0');

begin
process(clk)
begin
    if rising_edge(clk) then
        etat <= etat_suiv;
        if (etat = un) then
          compteur <= compteur + 1;
        end if;
        if (etat = zero) then
          compteur <= (others => '0');
        end if;
    end if;
end process;

process(etat, compteur)
begin
    case etat is
            when zero =>
                        valid <= '0';
                        verr <= '0';
                        raz <= '0';
                            etat_suiv <= un;
            when un =>
```

Avant <u>begin</u> : déclaration

- Types
- Signaux
- Variables
- composants

Après <u>begin</u> : code

- affectations
- process

31

31

3.1. SYNTAXE HORS « PROCESS »

- Affectation inconditionnelle

```
Y <= A or B ;
```

- Affectation conditionnelle

```
Y <= '1' when (SEL = "101") else '0' ;
```

```
Y <= A when sel="00" else
     B when sel="01" else
     C when sel="10" else
     '0';
```

- Affectation sélective

```
with SEL select
   Y <= A when "00",
        B when "01",
        C when "10",
        D when others;
```


3.2. SYNTAXE DANS « PROCESS »

- Affectation inconditionnelle

```
Y <= A or B ;
```

- Affectation conditionnelle

```
if (SEL = "101") then
   Y <= '1';
else
   Y <= '0';
end if;
```

Ce qui ressemble à du langage C correspond à des instructions à l'intérieur de process

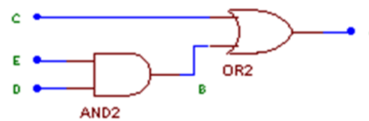
- Affectation sélective

```
case sel is
   when "00" =>    y <= '1';
   when "01" =>    y <= '0';
   when "10" =>    y <= '0';
   when others => y <= '1';
end case;
```

3.3. FONCTIONNEMENT CONCURRENT

- En dehors du PROCESS, toutes les instructions s'exécutent en parallèle. On parle de fonctionnement concurrent, c'est le principe des systèmes combinatoires.
- L'ordre des instructions n'a aucune importance.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity concurrent is
    Port ( C : in std_logic;
           D : in std_logic;
           E : in std_logic;
           A : out std_logic);
end concurrent;

architecture Behavioral of concurrent is
   signal B : std_logic;
begin
   A <= B OR C ;   -- solution
   B <= D AND E ;  --  n° 1

-- B <= D AND E ; -- solution
-- A <= B OR C ;  --  n° 2
end Behavioral;
```



Ces deux solutions donnent le même résultat après compilation, c'est-à-dire le schéma ci-dessus.

34

34

3.4. FONCTIONNEMENT SÉQUENTIEL

- Dans un « process », les instructions s 'exécutent de façon séquentielle.
- Le « process » s 'exécute à chaque changement d'état d 'un des signaux de la **liste de sensibilité**.
- La mise à jour des variables se fait au fur et à mesure que les instructions se déroulent.
- La mise à jour des signaux se fait à la fin du « process », après le « end ».
- Tous les « process » se déroulent en parallèle.
- L'ordre d'écriture des process n'a aucune importance.

liste de sensibilité

```
process (clk)
begin
   if rising_edge(clk) then
      count_s <= count_s + 1;
   end if;
end process;

count <= count_s;

end Behavioral;
```

Le signal « count_s » ne sera mis à jour qu'à la fin du « process », à la lecture du « end process »

35

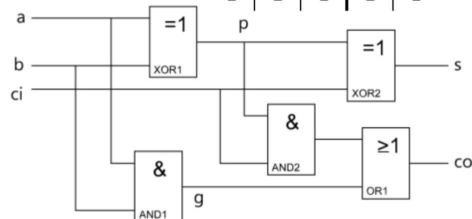
35

4.1. ADDITIONNEUR (1ÈRE FAÇON)

Description flot de données

ci	b	a	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

```
entity add2 is
    Port ( a : in  STD_LOGIC;
           b : in  STD_LOGIC;
           ci : in  STD_LOGIC;
           s : out  STD_LOGIC;
           co : out  STD_LOGIC);
end add2;

architecture   flot   of add2 is

   signal p, g : std_logic;

begin
   p <= a xor b;
   g <= a and b;

   s <= p xor ci;
   co <= g or (p and ci);

end   flot   ;
```



On décrit la structure matérielle niveau portes logiques de l'additionneur complet

36

36

4.2. ADDITIONNEUR (2ÈME FAÇON)

Description comportementale

```
architecture  Behavioral  of add is
   signal co_s : std_logic_vector (1 downto 0);
   signal ci_b_a : std_logic_vector (2 downto 0);
begin

   ci_b_a <= ci & b & a;
   co <= co_s (1);
   s <= co_s (0);

   co_s <= "00" when ci_b_a = "000" else
           "01" when ci_b_a = "001" else
           "01" when ci_b_a = "010" else
           "10" when ci_b_a = "011" else
           "01" when ci_b_a = "100" else
           "10" when ci_b_a = "101" else
           "10" when ci_b_a = "110" else
           "11" ;

end  Behavioral ;
```

ci	b	a	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

On écrit la table de vérité des sorties de l'additionneur, l'outil de synthèse trouvera la logique nécessaire

37

37

4.3. ADDITIONNEUR (3ÈME FAÇON)

Description comportementale

```
architecture Behavioral of add2 is
begin
   with ci & b & a select
      s <= '0' when "000",
           '1' when "001",
           '1' when "010",
           '0' when "011",
           '1' when "100",
           '0' when "101",
           '0' when "110",
           '1' when others;

   with ci & b & a select
      co <= '0' when "000",
           '0' when "001",
           '0' when "010",
           '1' when "011",
           '0' when "100",
           '1' when "101",
           '1' when "110",
           '1' when others;

end Behavioral ;
```

ci	b	a	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Autre façon d'écrire les tables de vérité des sorties, une table par sortie cette fois.

38

38

4.4. ADDITIONNEUR (4ÈME FAÇON)

Description comportementale

```
architecture Behavioral of add is
   signal co_s : std_logic_vector (1 downto 0);
   signal ci_b_a : std_logic_vector (2 downto 0);
begin

process (ci_b_a) -- process (all) IEEE 2008
begin
   case ci_b_a is
      when "000" =>  co_s <= "00";
      when "001" =>  co_s <= "01";
      when "010" =>  co_s <= "01";
      when "011" =>  co_s <= "10";
      when "100" =>  co_s <= "01";
      when "101" =>  co_s <= "10";
      when "110" =>  co_s <= "10";
      when others => co_s <= "11";
   end case;
end process;

ci_b_a <= ci & b & a;
co <= co_s (1);
s <= co_s (0);

end Behavioral ;
```

ci	b	a	co	s
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Encore une table de vérité écrite cette fois dans un process. Rappelez-vous ! La syntaxe n'est pas la même

39

39

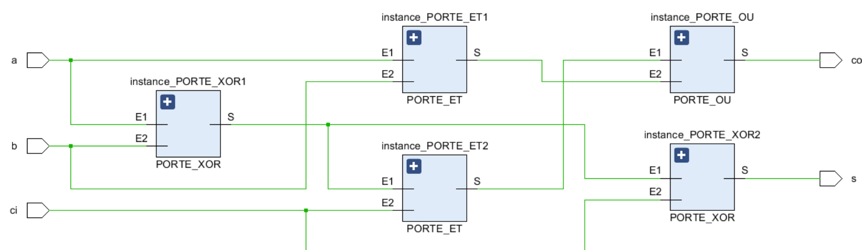
4.4. ADDITIONNEUR (5ÈME FAÇON)

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

Description structurelle

```
architecture Structural_ADDI of addiStruct is

component PORTE_ET
port(E1, E2 : in STD_LOGIC; S : out STD_LOGIC);
end component;

component PORTE_XOR
port(E1, E2 : in STD_LOGIC; S : out STD_LOGIC);
end component;

component PORTE_OU
port(E1, E2 : in STD_LOGIC; S : out STD_LOGIC);
end component;

signal p, g, i : STD_LOGIC;
```

```
begin
instance_PORTE_XOR1 : PORTE_XOR port map(E1=>a,E2=>b,S=>p);
instance_PORTE_XOR2 : PORTE_XOR port map(E1=>p,E2=>ci,S=>s);
instance_PORTE_ET1  : PORTE_ET  port map(E1=>a,E2=>b,S=>g);
instance_PORTE_ET2  : PORTE_ET  port map(E1=>p,E2=>ci,S=>i);
instance_PORTE_OU   : PORTE_OU  port map(E1=>i,E2=>g,S=>co);
end Structural_ADDI;
```



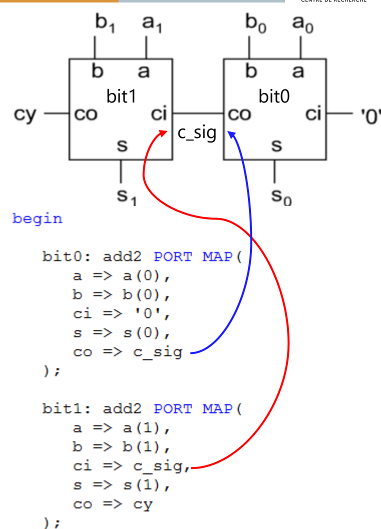
4.5. ADDITIONNEUR MOTS À 2 MOTS DE 2 BITS

ENSI CAEN
ÉCOLE PUBLIQUE D'INGÉNIEURS CENTRE DE RECHERCHE

Description structurelle

```
entity  addi4bits  is
    Port ( a : in  STD_LOGIC_VECTOR (1 downto 0);
           b : in  STD_LOGIC_VECTOR (1 downto 0);
           s : out  STD_LOGIC_VECTOR (1 downto 0);
           cy : out  STD_LOGIC);
end  addi4bits  ;

architecture  structural  of  addi4bits  is

   COMPONENT add2
   PORT(
      a : IN std_logic;
      b : IN std_logic;
      ci : IN std_logic;
      s : OUT std_logic;
      co : OUT std_logic
      );
   END COMPONENT;

   signal c_sig : std_logic;
```

Fil pour relier la retenue de sortie c0 de l'additionneur de poids faible à la retenue d'entrée ci de l'additionneur de poids fort.




```
begin

   bit0: add2 PORT MAP(
      a => a(0),
      b => b(0),
      ci => '0',
      s => s(0),
      co => c_sig
   );

   bit1: add2 PORT MAP(
      a => a(1),
      b => b(1),
      ci => c_sig,
      s => s(1),
      co => cy
   );

end  structural;
```

4.6. ADDITIONNEUR MOTS À N BITS – BEH.

- Voici un additionneur générique qui utilise l'opération arithmétique « + » du package « numeric_std ».

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity addi_N is
   generic (N : natural := 8);
    Port ( a : in  STD_LOGIC_VECTOR (N-1 downto 0);
           b : in  STD_LOGIC_VECTOR (N-1 downto 0);
           ci : in  STD_LOGIC;
           s : out  STD_LOGIC_VECTOR (N-1 downto 0);
           co : out  STD_LOGIC);
end addi_N;

architecture Behavioral of addi_N is
   signal sum : STD_LOGIC_VECTOR (N downto 0);

begin
   sum <= ('0' & a) + ('0' & b) + ci;
   s <= sum (N-1 downto 0);
   co <= sum(N);

end Behavioral;
```

Dans la pratique, constante N déclarée dans un package appelé pour rendre l'ensemble du code évolutif

& concaténation

43

4.6. ADDITIONNEUR MOTS À N BITS – STRUCT.

- Additionneur générique structurel à partir d'un additionneur complet 1 bit.

```
entity Full_Adder is
port(
  A,B,Ri : in std_logic;
   S,R : out std_logic);
end Full_Adder;

architecture flot of Full_Adder is
signal somme_int : std_logic ;
begin
  somme_int <= A xor B ;
  S <= somme_int xor Ri ;
  R <= (A and B) or (somme_int or Ri);
end flot;
```

Code de l'additionneur complet 1 bit.

```
entity additionneur_Nbits is
generic (N: integer := 8);
port (Av, Bv : in std_logic_vector(N downto 1);
        Ri : in std_logic;
        Sv : out std_logic_vector(N downto 1);
        R : out std_logic);
end additionneur_Nbits;

architecture structural of additionneur_Nbits is
component Full_Adder
port(A,B,Ri : in std_logic; S,R : out std_logic);
end component;

   signal Cv : std_logic_vector(1 to N+1);
begin
label_boucle_generation : for I in 1 to N generate
   instance : Full_Adder port map (Av(I), Bv(I), Cv(I), Sv(I), Cv(I+1));
   end generate;
   Cv(1) <= Ri;
   R <= Cv(N+1);
end structural;
```

Code de l'additionneur N bit utilisant l'instruction generate.

4.7. COMPTEUR À N BITS

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE.std_logic_unsigned.all;

entity compteur1 is
    generic (N : natural := 8);          <- Variable générique
    Port ( clk : in  STD_LOGIC;
           count : out  STD_LOGIC_VECTOR (N-1 downto 0));
end compteur1;

architecture Behavioral of compteur1 is
   signal count_s : STD_LOGIC_VECTOR (N-1 downto 0);   <- Signal intermédiaire

begin

process (clk)          <- Liste de sensibilité
begin
   if rising_edge(clk) then       <- Front montant
      count_s <= count_s + 1;
   end if;
end process;

count <= count_s;      <- Connexion de la sortie

end Behavioral;
```


45

4.8. COMPTEUR AVEC RESET ASYNCHRONE

```
entity compteur2 is
    generic (N : natural := 8);
    Port ( clk : in  STD_LOGIC;
           reset : in STD_LOGIC;
           count : out  STD_LOGIC_VECTOR (N-1 downto 0));
end compteur2;

architecture Behavioral of compteur2 is
   signal count_s : STD_LOGIC_VECTOR (N-1 downto 0);

begin

process (clk, reset)      <- Liste de sensibilité complète
begin
   if reset='1' then       <- Reset prioritaire = asynchrone
      count_s <= (others => '0');    <- Mise à '0' de tous les bits
   elsif rising_edge(clk) then
      count_s <= count_s + 1;
   end if;
end process;

count <= count_s;

end Behavioral;
```


46

4.9. COMPTEUR AVEC RESET SYNCHRONE

```
entity compteur3 is
    generic (N : natural := 8);
    Port ( clk : in  STD_LOGIC;
           reset : in STD_LOGIC;
           count : out  STD_LOGIC_VECTOR (N-1 downto 0));
end compteur3;

architecture Behavioral of compteur3 is
   signal count_s : STD_LOGIC_VECTOR (N-1 downto 0);

begin

process (clk)
begin
   if rising_edge(clk) then
      if reset='1' then
         count_s <= (others => '0');
      else
         count_s <= count_s + 1;
      end if;
   end if;
end process;

count <= count_s;
```

Liste de sensibilité réduite

Front d'horloge prioritaire = Reset synchrone

4.10. COMPTEUR AVEC VALIDATION

```
entity compteur4 is
    generic (N : natural := 8);
    Port ( clk : in  STD_LOGIC;
           reset : in STD_LOGIC;
           ce : in STD_LOGIC;
           count : out  STD_LOGIC_VECTOR (N-1 downto 0));
end compteur4;

architecture Behavioral of compteur4 is
   signal count_s : STD_LOGIC_VECTOR (N-1 downto 0);

begin

process (clk)
begin
   if rising_edge(clk) then
      if reset='1' then
         count_s <= (others => '0');
      elsif ce='1' then
         count_s <= count_s + 1;
      else
         count_s <= count_s;
      end if;
   end if;
end process;

count <= count_s;

end Behavioral;
```

« reset » prioritaire sur « ce »

« ce » valide le comptage

Pas de changement si ce = '0'

À PROPOS DU « RESET »

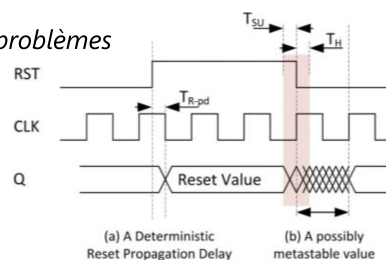
1ère règle : n'utiliser de signal « reset » que si cela est nécessaire !

- De toute façon, toutes les bascules sont remises à '0' à la mise sous tension.
- Les bénéfices sont les suivants :
    1. Moins de logique générée par le compilateur.
    2. Moins de problème au moment du routage.

À PROPOS DU « RESET »

Le « reset » asynchrone peut conduire à des problèmes

- Métastabilité des bascules (comportement non prédictible/non déterministe).
- Problèmes de fiabilité.



2ème règle : le signal « reset » doit être synchrone et local !

- La remise à zéro des bascules se fait au même moment sur front d'horloge comme les autres changements d'état.
- Pas de reset global pour éviter les effets des retards différents d'une bascule à l'autre.

4.11. DÉCODEUR 2 VERS 4

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity decode_2_4 is
    Port ( entree : in std_logic_vector(1 downto 0);
           sortie : out std_logic_vector(3 downto 0));
end decode_2_4;

architecture Behavioral of decode_2_4 is

begin

   sortie <= "0001" when entree = "00" else
             "0010" when entree = "01" else
             "0100" when entree = "10" else
             "1000" when entree = "11" else
             "ZZZZ";

end Behavioral;
```

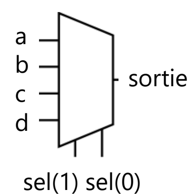
- Système combinatoire décrit hors « process » en utilisant la structure « when … else »

53

53

4.12. MULTIPLEXEUR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity multipexe is
    Port ( a : in std_logic;
           b : in std_logic;
           c : in std_logic;
           d : in std_logic;
           sel  : in std_logic_vector(1 downto 0);
           sortie : out std_logic);
end multipexe;

architecture Behavioral of multipexe is

begin
   sortie <= a when sel = "00" else
             b when sel = "01" else
             c when sel = "10" else
             d when sel = "11" else
             'X';
end Behavioral;
```



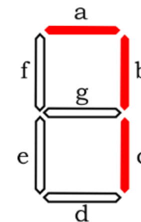
- Système combinatoire décrit hors « process » en utilisant la structure « when … else »

54

54

4.13. DÉCODEUR HEXA VERS 7 SEGMENTS

```
entity hex2led is
    Port ( HEX : in std_logic_vector(3 downto 0);
           LED : out std_logic_vector(6 downto 0));
end hex2led;

architecture Behavioral of hex2led is

begin
       with HEX SELect
   LED<= "1111001" when "0001",    --1
         "0100100" when "0010",    --2
         "0110000" when "0011",    --3
         "0011001" when "0100",    --4
         "0010010" when "0101",    --5
         "0000010" when "0110",    --6
         "1111000" when "0111",    --7
         "0000000" when "1000",    --8
         "0010000" when "1001",    --9
         "0001000" when "1010",    --A
         "0000011" when "1011",    --b
         "1000110" when "1100",    --C
         "0100001" when "1101",    --d
         "0000110" when "1110",    --E
         "0001110" when "1111",    --F
         "1000000" when others;    --0
 end Behavioral;
```



- Système combinatoire décrit hors « process » en utilisant la structure « with … select ». C'est une façon d'écrire la table de vérité.

4.14. ENCODEUR DE PRIORITÉ

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity encodeur is
    Port ( entree : in std_logic_vector(3 downto 0);
           sortie : out std_logic_vector(1 downto 0));
end encodeur;

architecture Behavioral of encodeur is
begin

   process (entree)
   begin
      if entree(3)='1' then sortie<="11";
      elsif entree(2)='1' then sortie<="10";
      elsif entree(1)='1' then sortie<="01";
      elsif entree(0)='1' then sortie<="00";
      else sortie<="XX";
      end if;
   end process;

end Behavioral;
```

- Système combinatoire décrit dans un « process » en utilisant la structure « if… then … else ». On peut décrire une fonction combinatoire dans un « process » !

4.15. CALCUL DU BIT DE PARITÉ

```
entity parite is
    Port ( mot : in std_logic_vector(7 downto 0);
           parite : out std_logic);
end parite;

architecture Behavioral of parite is
begin

   process (mot)
      variable x : std_logic;
   begin
      x := '0';
      for i in 0 to 7 loop
         if mot(i) = '1' then
            x := not x;
         end if;
      end loop;
         parite <= x;
   end process;

 end Behavioral;
```

Exemple d'utilisation de l'instruction for (uniquement dans un process)

- Système combinatoire décrit dans un « process » en utilisant la structure « for… loop ».

4.16. DIVISEUR D'HORLOGE PAR 2^N

```
entity clk_div is
   generic (N : natural := 5);

   Port ( clk : in  STD_LOGIC;
          reset : in  STD_LOGIC;
          div_out : out  STD_LOGIC);
end clk_div;

architecture Behavioral of clk_div is
   signal count_temp : std_logic_vector(N-1 downto 0);

begin

process (clk)
begin
   if rising_edge (clk) then
      if (reset = '1') then
         count_temp <= (others => '0');
      else
         count_temp <= count_temp + 1;
      end if;
   end if;
end process;

   div_out <= count_temp(N-1);
end Behavioral;
```

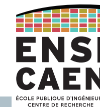
- La division d'une fréquence se fait toujours avec un compteur synchrone.

4.17.1. CONVERSION BINAIRE VERS BCD

- L'algorithme de conversion utilise le fait que tous les chiffres compris entre 0 et 4 produisent un résultat sur un chiffre lorsqu'ils sont multipliés par 2.
- Par contre tous les chiffres compris entre 5 et 9 donnent un résultat sur 2 chiffres lorsqu'ils sont multipliés par 2.

chiffre	Chiffre*2	dizaines	unités
0	0	0	0
1	2	0	2
2	4	0	4
3	6	0	6
4	8	0	8
5	10	1	0
6	12	1	2
7	14	1	4
8	16	1	6
9	18	1	8

4.17.2. APPLICATION DE L'ALGORITHME



$0 \leq nb \leq 4$

En décimal :

Nb	Nb*2
0	0
1	2
2	4
3	6
4	8

En BCD : un décalage à gauche

Nb	Nb*2
0000	0000
0001	0010
0010	0100
0011	0110
0100	1000

$5 \leq nb \leq 9$

Exemple en décimal : $2 \times 6 = 12$

En BCD : ajouter 3 et décaler à gauche

	diz	uni
6	0000	0110
+3	0000	0011
=	0000	1001
X2	0001	0010

4.17.3. ALGORITHME COMPLET

- Déclarer un registre de 20 bits : 4 + 4 + 4 + 8
- Les 8 bits de poids faible représentent le mot binaire à convertir
- Suivent ensuite 3 quartets dans l'ordre : les unités, les dizaines puis les centaines du résultat en BCD

Réaliser les étapes suivantes 8 fois

1. Si les unités sont ≥5 , ajouter 3. Faire la même chose pour les dizaines et les centaines
2. Décaler tout le registre d'un bit vers la gauche

Operation	Hundreds	Tens	Units	Binary	
HEX				F	F
Start				1 1 1 1	1 1 1 1

61

4.17.4A. CONVERSION D'UN MOTS À 8 BITS

					Mot binaire	
décalage	Opération	CENT	DIZ	UNI	7654	3210
	START	0000	0000	0000	1111	1111
1	SHIFT	0000	0000	0001	1111	1110
2	SHIFT	0000	0000	0011	1111	1100
3	SHIFT	0000	0000	0111	1111	1000
	+ 3	0000	0000	**1010**	1111	1000
4	SHIFT	0000	0001	0101	1111	0000
	+ 3	0000	0001	**1000**	1111	0000
5	SHIFT	0000	0011	0001	1110	0000
6	SHIFT	0000	0110	0011	1100	0000
	+ 3	0000	**1001**	0011	1100	0000
7	SHIFT	0001	0010	0111	1000	0000
	+ 3	0001	0010	**1010**	1000	0000
8	SHIFT	0010	0101	0101	0000	0000
		2	**5**	**5**		

62

4.17.4B. CONVERSION D'UN MOTS À 8 BITS

décalage	Opération	CENT	DIZ	UNI	Mot binaire 7654	3210
	START	0000	0000	0000	0000	1100
1	SHIFT	0000	0000	0000	0001	1000
2	SHIFT	0000	0000	0000	0011	0000
3	SHIFT	0000	0000	0000	0110	0000
4	SHIFT	0000	0000	0000	1100	0000
5	SHIFT	0000	0000	0001	1000	0000
6	SHIFT	0000	0000	0011	0000	0000
7	SHIFT	0000	0000	0110	0000	0000
	+3	0000	0000	**1001**	0000	0000
8	SHIFT	0000	0001	0010	0000	0000
		0	**1**	**2**		

63

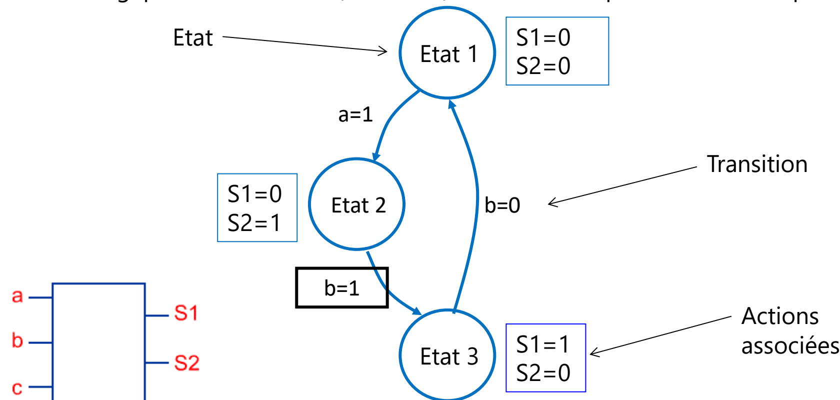
4.17.5. CODE VHDL (BINAIRE VERS BCD)

```
process(binaire)
    variable registre : std_logic_vector(19 downto 0);
begin
    registre := x"000" & binaire;       -- registre de travail

    for i in 1 to 8 loop         -- 8 bits à convertir
        if registre(11 downto 8) > 4 then    -- centaines
            registre(11 downto 8) := registre(11 downto 8) + 3;
        end if;
        if registre(15 downto 12) > 4 then  -- dizaines
            registre(15 downto 12) := registre(15 downto 12) + 3;
        end if;
        if registre(19 downto 16) > 4 then   -- unités
            registre(19 downto 16) := registre(19 downto 16) + 3;
        end if;
        registre := registre(18 downto 0) & '0';     -- décalage G
    end loop;

    cent <= registre(19 downto 16); -- sorties décimales
    diz <= registre(15 downto 12);
    uni <= registre(11 downto 8);
end process;
```


64

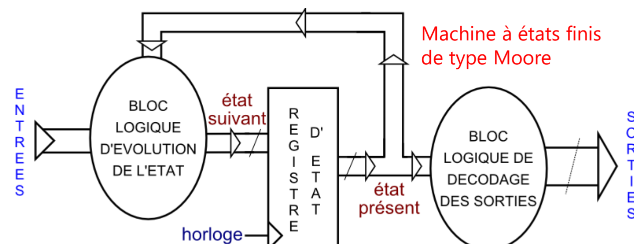
5.1. MACHINE D'ÉTATS

- Une machine d'états est un séquenceur qui décrit le fonctionnement d'un système séquentiel aussi complexe soit-il.
- On utilise des symboles : les bulles indiquent les états du système et les arcs orientés définissent les possibilités d'évolution. Le passage d'un état à un autre est régi par une condition (transition). Les sorties dépendent de l'état présent.



5.2. MACHINE À ÉTATS FINIS

Structure séquentielle pour réaliser des automates ou compteurs (le compteur est un cas d'automate simple). Une machine d'état est un opérateur séquentiel dont la sortie est fonction des entrées et de l'état précédent de la machine d'état. Elle utilise des bascules pour mémoriser l'état présent et des blocs de logique combinatoire pour générer les sorties et l'état futur. La machine d'état permet l'implémentation de diagrammes d'état décrivant des systèmes séquentiels.



Le type Moore garantit des **signaux logiques synchrones**. Il est privilégié dans les applications intégrées et cœur de calculateur pour la réalisation de séquenceur. Les sorties dépendent exclusivement de l'état

5.2. MACHINE À ÉTATS FINIS

Diagrammes d'états : représentation symbolique d'un système séquentiel. Il s'agit d'un outil graphique utilisé pour modéliser le comportement d'un système séquentiel en représentant ses différents états et les transitions entre ces états en fonction des entrées et des conditions. Il permet de décrire comment un système évolue au fil du temps, ce qui est essentiel pour la synthèse et la conception de circuits séquentiels comme les machines à états finis. Ce diagramme facilite la conception logique en offrant une vision claire des interactions entre les différents états du système.

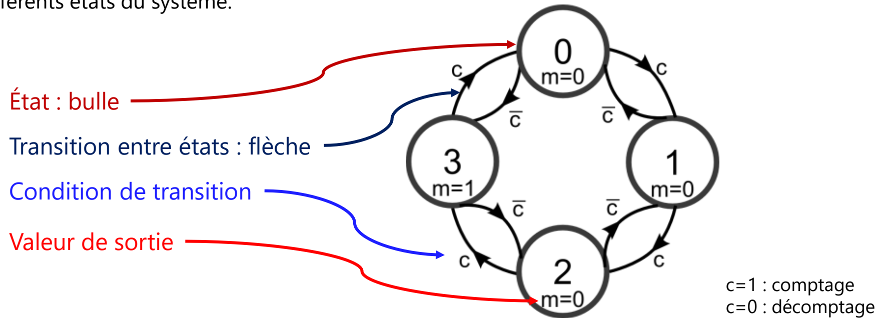


Diagramme d'état d'un compteur-décompteur
4 états avec sortie à 1 pour la valeur maximum

5.2. MACHINE À ÉTATS FINIS

Exemples de diagrammes d'états.

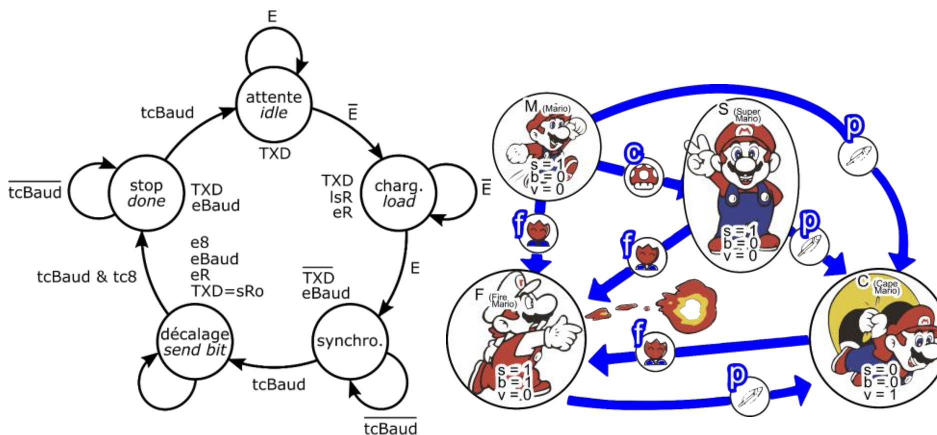
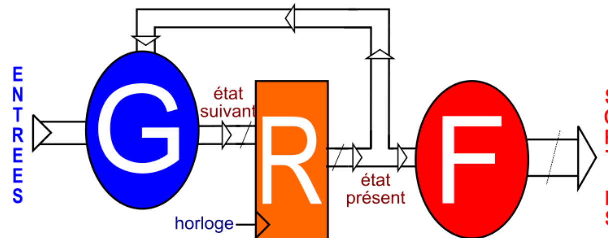


Diagramme d'état de la machine à états finis de contrôle de l'interface UART

Diagramme d'état : notion plus large que l'électronique numérique, adaptée aux systèmes séquentiels, à la programmation.

5.2. MAEF EN VHDL

- **4** façons de coder une machine d'état de type Moore
  - **3 processus** : 1 pour chaque bloc logique G, R et F
  - **2 processus** : (G+R) et F ou R et (G+F)
  - **1 processus** : G+R+F



G : bloc logique combinatoire d'évolution des états (prépare l'état suivant)

R : registre d'état (mémorise l'état)

F : bloc logique de décodage des sorties (évalue les sorties en fonction de l'état présent)

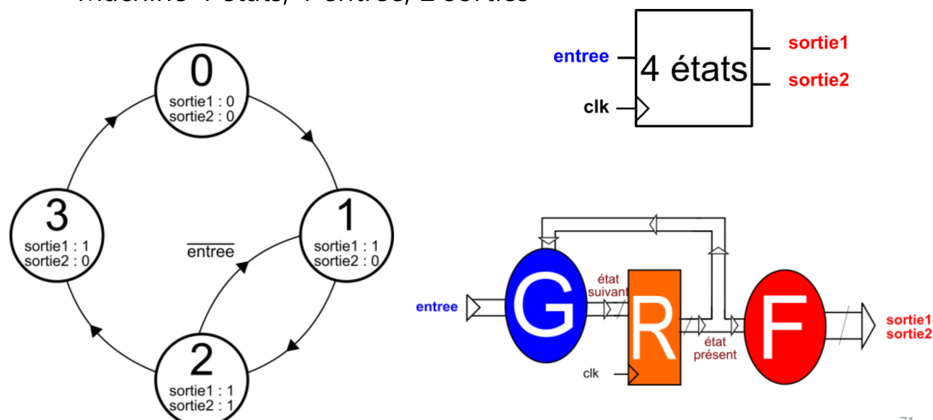
70

70

5.2. MAEF EN VHDL : SUR UN EXEMPLE

Diagramme d'état et schéma bloc de l'exemple

- Machine 4 états, 1 entrée, 2 sorties



71

71

5.2. MAEF EN VHDL : SUR UN EXEMPLE

```vhdl
entity MAEFxp is
    Port ( clk : in STD_LOGIC;
           entree : in STD_LOGIC;
           sortie1 : out STD_LOGIC;
           sortie2 : out STD_LOGIC);
end MAEFxp;
```

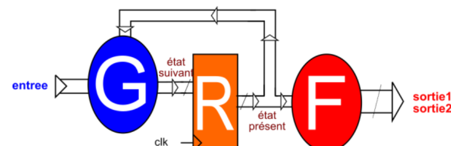

5.2. MAEF EN VHDL : SUR UN EXEMPLE

3 PROCESSUS (PROCESS)

```vhdl
architecture Behavioral_3p of MAEF3p is
type type_etat is (etat0, etat1, etat2, etat3);
signal etat_present : type_etat;
signal etat_suivant : type_etat;
begin

-- pour la synthèse du registre d'état
PROC_SEQ : process(clk)
begin
    if rising_edge(clk) then
    -- changement d'état sur front d'horloge
        etat_present <= etat_suivant;
    end if;
end process PROC_SEQ;

-- bloc logique combinatoire évolution des états
COMB_G : process(etat_present, entree)
begin
    case etat_present is
        when etat0 => etat_suivant <= etat1;
        when etat1 => etat_suivant <= etat2;
        when etat2 => if entree = '1' then
                          etat_suivant <= etat3;
                      else
                          etat_suivant <= etat1;
                      end if;
        when etat3 => etat_suivant <= etat0;
    end case;
end process COMB_G;
```

Définition d'un type état pour la clarté du code.
Dans ce cas, on a besoin de définir deux signaux du type état etat_present et etat_suivant

```vhdl
-- bloc logique combinatoire décodade des sorties
COMB_F : process(etat_present)
begin
    case etat_present is
        when etat0 => sortie1 <= '0'; sortie2 <= '0';
        when etat1 => sortie1 <= '1'; sortie2 <= '1';
        when etat2 => sortie1 <= '1'; sortie2 <= '1';
        when etat3 => sortie1 <= '1'; sortie2 <= '0';
    end case;
end process COMB_F;

end Behavioral_3p;
```



5.2. MAEF EN VHDL : SUR UN EXEMPLE

2 PROCESSUS (PROCESS) G+R ET F

```vhdl
-- 2 process Evolution+registre d'état et décodage sorties
architecture Behavioral_2p of MAEF2p is
type type_etat is (etat0, etat1, etat2, etat3);
signal etat : type_etat;
begin

-- pour la synthèse du registre d'état et du
-- bloc logique d'évolution des états
PROC_ETAT : process(clk)
begin
    if rising_edge(clk) then
       case etat is
         when etat0 => etat <= etat1;
         when etat1 => etat <= etat2;
         when etat2 => if entree = '1' then
                         etat <= etat3;
                       else
                         etat <= etat1;
                       end if;
         when etat3 => etat <= etat0;
     end case;
    end if;
end process PROC_ETAT;
```

Définition d'un type état pour la clarté du code.
Dans ce cas, un seul signal du type état.

```vhdl
-- bloc logique combinatoire décodade des sorties
COMB_F : process(etat)
begin
    case etat is
       when etat0 => sortie1 <= '0'; sortie2 <= '0';
       when etat1 => sortie1 <= '1'; sortie2 <= '1';
       when etat2 => sortie1 <= '1'; sortie2 <= '1';
       when etat3 => sortie1 <= '1'; sortie2 <= '0';
    end case;
end process COMB_F;

end Behavioral_2p;
```


74

5.2. MAEF EN VHDL : SUR UN EXEMPLE

2 PROCESSUS (PROCESS) R ET G+F

Définition d'un type état pour la clarté du code.
Dans ce cas, on a besoin de définir deux signaux du type état etat_present et etat_suivant

```vhdl
-- 2 process registre d'état et blocs logiques
-- (évolution états+décodage sorties)
architecture Behavioral_2pV2 of MAEF2pV2 is
type type_etat is (etat0, etat1, etat2, etat3);
signal etat_present : type_etat;
signal etat_suivant : type_etat;
begin

-- pour la synthèse du registre d'état
PROC_SEQ : process(clk)
begin
    if rising_edge(clk) then
    -- changement d'état sur front d'horloge
        etat_present <= etat_suivant;
    end if;
end process PROC_SEQ;
```



```vhdl
-- pour la synthèse des blocs logiques combinatoires
--(état suivant et décodage sorties)
COMB_F_G : process(etat_present, entree)
begin
    case etat_present is
       when etat0 => sortie1 <= '0'; sortie2 <= '0';
                     etat_suivant <= etat1;
       when etat1 => sortie1 <= '1'; sortie2 <= '1';
                     etat_suivant <= etat2;
       when etat2 => sortie1 <= '1'; sortie2 <= '1';
                     if entree = '1' then
                       etat_suivant <= etat3;
                     else
                       etat_suivant <= etat1;
                     end if;
       when etat3 => sortie1 <= '1'; sortie2 <= '0';
                     etat_suivant <= etat0;
    end case;
end process COMB_F_G;

end Behavioral_2pV2;
```

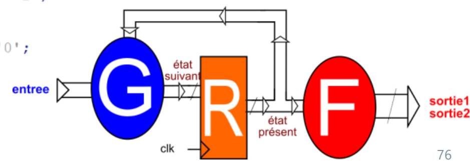
75

5.2. MAEF EN VHDL : SUR UN EXEMPLE

1 PROCESSUS (PROCESS) G+R+F

```
architecture Behavioral_1p of MAEF1p is -- 1 process
type type_etat is (etat0, etat1, etat2, etat3);
signal etat : type_etat;
begin

PROC_Unique : process(clk) -- process unique
begin
  if rising_edge(clk) then
    case etat is
        when etat0 => etat <= etat1;
                      sortie1 <= '1'; sortie2 <= '1';
        when etat1 => etat <= etat2;
                      sortie1 <= '1'; sortie2 <= '1';
        when etat2 => if entree = '1' then
                        etat <= etat3;
                        sortie1 <= '1'; sortie2 <= '0';
                      else
                        etat <= etat1;
                        sortie1 <= '1'; sortie2 <= '1';
                      end if;
        when etat3 => etat <= etat0;
                      sortie1 <= '0'; sortie2 <= '0';
    end case;
  end if;
end process PROC_Unique;

end Behavioral_1p;
```

Définition d'un type état pour la clarté du code.
Dans ce cas, un seul signal du type état.



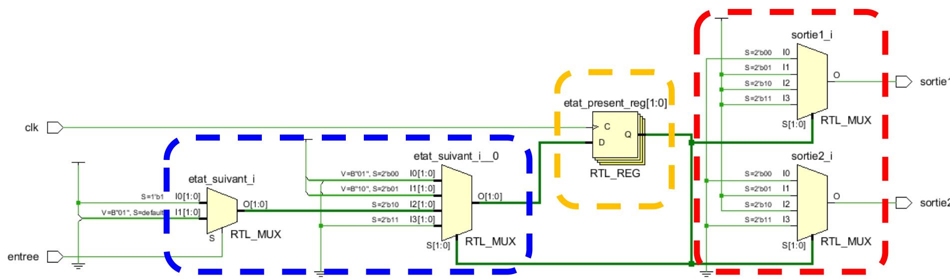
76

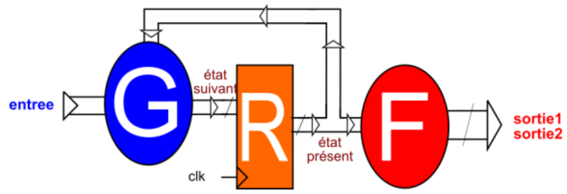
76

5.2. MAEF EN VHDL : SUR UN EXEMPLE

SYNTHÈSE RTL

- Les solutions 3 process et 2 process conduisent à la même structure.





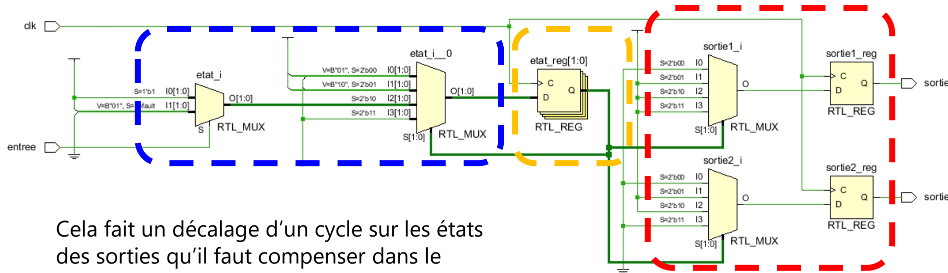
77

77

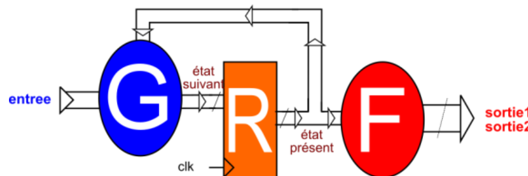
5.2. MAEF EN VHDL : SUR UN EXEMPLE

SYNTHÈSE RTL

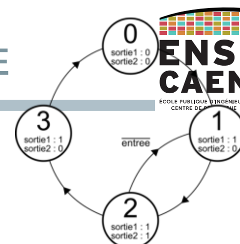
- La solution 1 process conduit à une structure avec des bascules sur les sorties.



Cela fait un décalage d'un cycle sur les états des sorties qu'il faut compenser dans le code.



5.2. MAEF EN VHDL : SUR UN EXEMPLE



SIMULATION

- Résultats de simulation équivalents

5.2. MAEF EN VHDL : SUR UN EXEMPLE

QUELLE SOLUTION CHOISIR ?

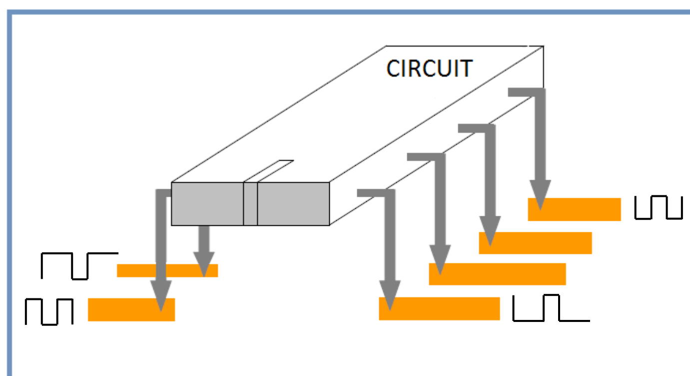
- Toutes les solutions sont fonctionnelles.

- Les électroniciens utilisent plutôt les solutions 3 ou 2 process. Solutions plus proches du matériel.
  - En phase d'apprentissage, privilégiez les solutions 3 ou 2 process. Code plus lisible, plus structuré.

- Des informaticiens plutôt une solution 1 process.
  - Attention à la gestion des sorties à cause des bascules dans ce cas.

6.1. PRINCIPE DU « TESTBENCH »

- Tester un circuit revient à lui imposer des signaux en entrée « stimuli » et à regarder comment évoluent les sorties. Si les réponses correspondent à ce que l'on attend, le test est bon, sinon il y a erreur.
- Le « testbench » est un programme VHDL qui réalise toutes ces opérations .



6.2. CIRCUIT COMBINATOIRE À TESTER

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity additionneur1 is
    Port ( a : in std_logic;
           b : in std_logic;
           retenue_e : in std_logic;
           somme : out std_logic;
           retenue_s : out std_logic);
end additionneur1;

architecture Behavioral of additionneur1 is

begin
   somme <= a xor b xor retenue_e;
   retenue_s <= (a and b)or
     (a and retenue_e)or(b and retenue_e);

end Behavioral;
```

- Nous avons affaire à un circuit combinatoire à 3 entrées et 2 sorties, il s'agit d'un additionneur complet.

6.3. TESTBENCH : DÉCLARATIONS

```
ENTITY additionneur1_testeur_vhd_tb IS
END additionneur1_testeur_vhd_tb;

ARCHITECTURE behavior OF additionneur1_testeur_vhd_tb IS

   COMPONENT additionneur1
   PORT(
      a : IN std_logic;
      b : IN std_logic;
      retenue_e : IN std_logic;
      somme : OUT std_logic;
      retenue_s : OUT std_logic
      );
   END COMPONENT;

   SIGNAL a :  std_logic;
   SIGNAL b :  std_logic;
   SIGNAL retenue_e :  std_logic;
   SIGNAL somme :  std_logic;
   SIGNAL retenue_s :  std_logic;

BEGIN
```

- En début de programme, on déclare le composant à tester ainsi que les différents signaux qui vont servir à connecter le composant.

6.4. TESTBENCH : SIMULATION

```
BEGIN

   uut: additionneur1 PORT MAP(
      a => a,
      b => b,
      retenue_e => retenue_e,
      somme => somme,
      retenue_s => retenue_s
   );

   tb : PROCESS
   BEGIN
      a<='0'; b<='0'; retenue_e<='0';
      wait for 100 ns;
      a<='1';
      wait for 100 ns;
      b<='1';
      wait for 200 ns;
      retenue_e<='1';
      wait for 100 ns;
      a<='0'; b<='0';
   END PROCESS;

END;
```

- La fin du programme sert à connecter le composant et lui imposer des « stimuli » en entrée pour savoir comment il réagit.

6.5. CIRCUIT SÉQUENTIEL À TESTER

```
entity diviseur is
    Port ( reset : in std_logic;
           clk : in std_logic;
           sortie : out std_logic);
end diviseur;

architecture Behavioral of diviseur is
   signal tempo : std_logic_vector(2 downto 0);
begin

process (reset, clk)
begin
   if (reset='0') then
      tempo <= "000";
   elsif (clk'event and clk='1') then
      tempo <= tempo + 1;
   end if;
end process;
sortie <= tempo(2);

end Behavioral;
```

- Il s'agit de simuler un diviseur de fréquence par 8 qui utilise un compteur binaire sur 3 bits.

6.6. TESTBENCH : DÉCLARATIONS

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY diviseur_testeur2_vhd_tb IS
END diviseur_testeur2_vhd_tb;

ARCHITECTURE behavior OF diviseur_testeur2_vhd_tb IS

   COMPONENT diviseur
   PORT(
      reset : IN std_logic;
      clk : IN std_logic;
      sortie : OUT std_logic
      );
   END COMPONENT;

   SIGNAL reset :  std_logic;
   SIGNAL clk :  std_logic;
   SIGNAL sortie :  std_logic;

BEGIN
```

- En début de programme, on déclare le composant à tester ainsi que les différents signaux qui vont servir à connecter le composant.

6.7. TESTBENCH : SIMULATION

```
BEGIN

   uut: diviseur PORT MAP(
      reset => reset,
      clk => clk,
      sortie => sortie  );

   horloge : PROCESS
   BEGIN
         clk <= '0';
         wait for 10 ns;
         clk <= '1';
         wait for 10 ns;
   END PROCESS;

   raz : PROCESS
      BEGIN
         reset <= '1';
         wait for 15 ns;
         reset <= '0';
         wait for 40 ns;
         reset <= '1';
         wait;
      END PROCESS;

END;
```

- La fin du programme sert à connecter le composant et lui imposer des « stimuli » en entrée pour savoir comment il réagit.

6.8. CONTRÔLE DES RÉSULTATS

```
-- Stimulus process
 stim_proc: process
begin
   -- hold reset state for 100 ns.
  reset <= '1';
     sig <= '0';
     wait for clk_period*5;
     reset <= '0';
     wait for 120ns;
     sig <= '1';
     wait for clk_period*10;

     assert sig_mef = '0'    -- teste si sig_mef = '0' sinon
     report "La valeur de sig_mef est fausse "  -- afficher ce message
     severity FAILURE;       -- et abandonner la simulation

     wait for clk_period*65535;
end process;
```

- Assert …
- Report …
- Severity …

88

6.9. TESTER DIFFÉRENTES CONFIGURATIONS

```
LIBRARY work;

CONFIGURATION Projet_cfg OF Projet IS
 FOR schematic

  FOR OTHERS:  add16
        USE ENTITY work.add16(beh_Ripple);
  END FOR;

  FOR OTHERS:  mult16
        USE ENTITY work.mult16(beh_Wallace);
  END FOR;

END FOR;

END Projet_cfg;
```

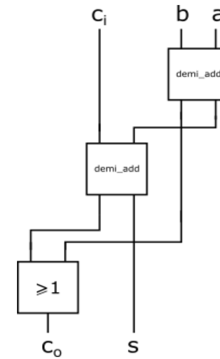
Déclaration dans un fichier de configuration séparé, ou bien dans le testbench.

Ici, nom de l'architecture que l'on choisit pour la simulation.

89

6.9. TESTER DIFFÉRENTES CONFIGURATIONS

```vhdl
entity ex_config_addi2 is
    Port (A, B, Ri : in STD_LOGIC; S, R : out STD_LOGIC);
end ex_config_addi2;

architecture Architectural_ADDI of ex_config_addi2 is
    component DEMI_ADDI
    port(A, B : in STD_LOGIC; S, R : out STD_LOGIC);
    end component;

    component PORTE_OU
    port(E1, E2 : in STD_LOGIC; Z : out STD_LOGIC);
    end component;

    signal S1, S2, S3 : STD_LOGIC;
-- spécification de configuration
    for instance_DEMI_ADD1 : DEMI_ADDI use entity work.DEMI_ADDI(Flow)
        port map(X=>A,Y=>B,Z=>S,W=>R);
    for instance_DEMI_ADD2 : DEMI_ADDI use entity work.DEMI_ADDI(Structurel)
        port map(X=>A,Y=>B,Z=>S,W=>R);
begin
    instance_DEMI_ADD1 : DEMI_ADDI port map(A=>A,B=>B,S=>S1,R=>S2);
    instance_DEMI_ADD2 : DEMI_ADDI port map(A=>S1,B=>Ri,S=>S,R=>S3);
    instance_PORTE_OU  : PORTE_OU  port map(E1=>S3,E2=>S2,Z=>R);
end Architectural_ADDI;
```



Pour le premier DEMI_ADDI architecture Flow, pour le second architecture Structurel

6.9. TESTER DIFFÉRENTES CONFIGURATIONS

Pour le premier DEMI_ADDI architecture Flow, pour le second architecture Structurel

Dans ce cas, pas d'intérêt particulier de mêler 2 architectures différentes. Il s'agit d'un exemple pour illustrer la syntaxe. Pour utiliser la même configuration pour les 2 modules DEMI_ADDI, on utilise l'instruction for all.

```vhdl
-- spécification de configuration
    for all : DEMI_ADDI use entity work.DEMI_ADDI(Flow)
        port map(X=>A,Y=>B,Z=>S,W=>R);
```