Micro architecture

Le 12 septembre 2017 , SVN-ID 423

Contents

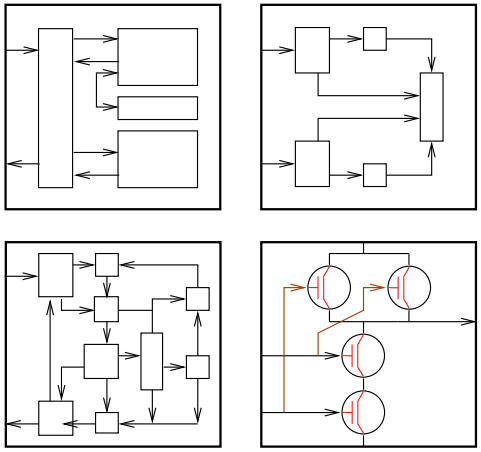
1 Circuits intégrés

1.1 Introduction

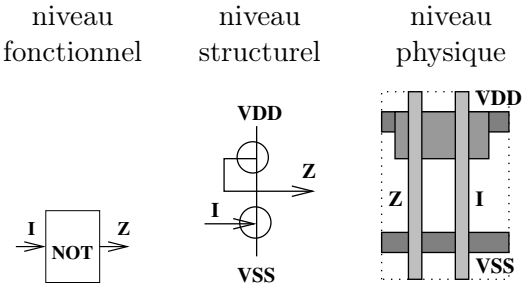
1.1.1 Les niveaux

Conception Hiérarchique

- CI: Interconnexion de blocs (processeur, RAM)
- blocs: Interconnexion de macro-cellules et de cellules
- Macro-cellules: Interconnexion de macro-cellules et de cellules
- Cellules: Interconnexion de transistors



Vues d'un objet



Les caractéristiques d'un objet sont: fonction, surface (coût de fabrication), caractéristiques temporelles (rapidité), puissance nécessaire aux entrées et disponible sur les sorties (connectique), consommation (coût d'utilisation).

1.1.2 Technologie

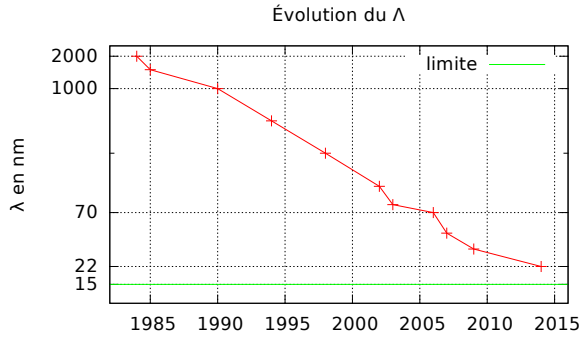
Quelques technologies

λ Mesure d'une technologie,la taille de la grille du plus petit transistor ($\sim$ sa hauteur).

Quelques propriétés Quand le λ diminue:

- l'intégration augmente,
- la rapidité augmente (jusqu'à un seuil),
- la consommation diminue.

Le λ a une limite physique.



Quelques familles technologiques

	bipolaire		CMOS	ASGA
	TTL	ECL		
intégration	−	0	++	−
rapidité*	0	+	−	++
consommation*	+	+	−**	?
puissance	+	−	−	−
coût & écologie	−	−	−	++

*: à $\lambda constant$

**: surtout au repos $\lambda constant$

CMOS Plus de recherche & développement

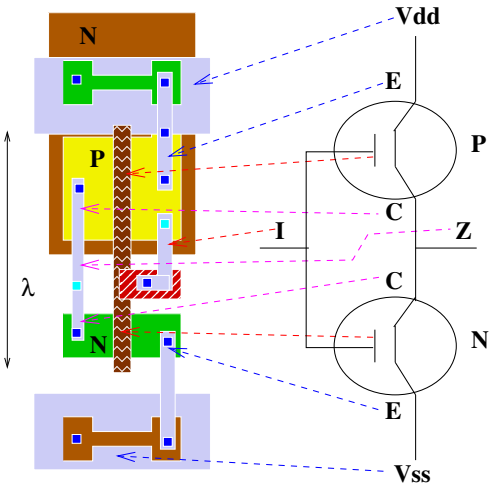
$\Longrightarrow$ λ proche de la limite.

$\Longrightarrow$ Vitesse très satisfaisante (quelques gH)

$\Longrightarrow$ Intégration inégalée

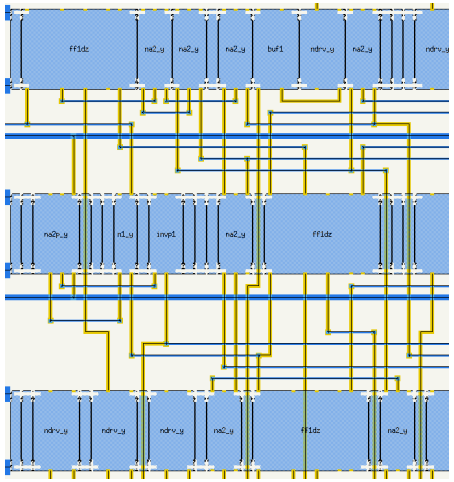
$\Longrightarrow$ Grande majorité des CI.

Cellule CMOS



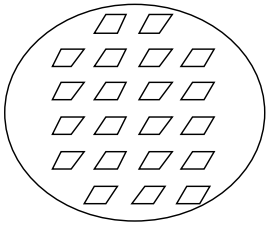
Inverseur CMOS ($\lambda = 1\,\mu m$) moins 2 rectangles de métal2 qui sortent I et Z en haut et en bas (il reste les contacts). Le CMOS ne consomme que lors des commutations.

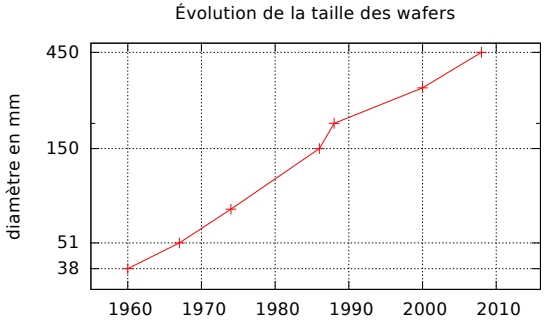
Assemblage de cellules



- placement des cellules
  - les cellules sont alignées dans des rangées,
  - les alimentations des cellules s'aboutent,
  - elles forment les rails d'alimentation.
- routage des cellules
  - horizontal/vertical 2 à N couches,
  - surface entre les rangées → surface perdue.
- routage et placement sont corrélés

Fabrication des CIs





Couches faites une à une

- oxydation (SiO_2, au four)
- dépôt de résine
- pose d'un masque (verre, argent)
- UV qui grille la résine où doit être déposée la couche
- révélation (accétone)

- attaque de la SiO_2 non protégé par HF
- nettoyage de la résine
- dépôt de la couche: passage au four (diffusion N (Phosphore) et P (Bore)), vaporisation sous vide (alu)

Durée 3/5 heures par couche

→ plusieurs jours.

Rendement de fabrication

$\rho = \frac{Nombredepucessansdfauts}{Nombredepucestotalsurunwafer}$

Les principales causes de défauts physiques: poussières, défauts cristallins, désalignements de masques.

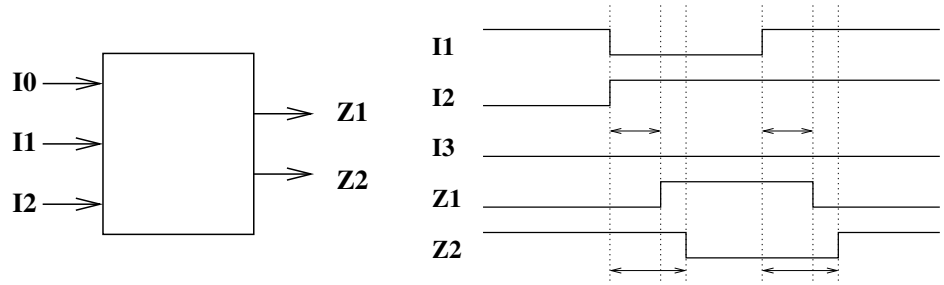
$\Longrightarrow$ Inclusion de logique dès la conception pour tester les CI.

catégories militaire, normale, dégradée ratée ($\rho = \lambda e^{kS}$, usuel 20%).

1.2 Cellules de base

1.2.1 Cellules combinatoires

Définition



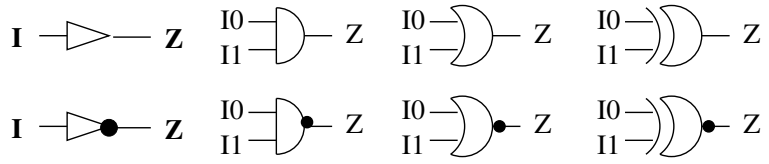
entrées stables ⇒ sorties stables: $Z_i = f_i(I_1, I_2, \ldots, I_N)$

Portes logiques

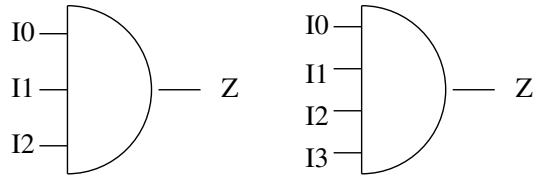
Fonction

Z = I, Z = not(I),
Z = I0 op I1 avec op: and, or, nand, nor, xor, nxor, ...

portes à 2 entrées

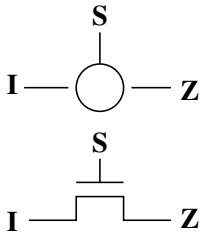


portes à N entrées



Trois état

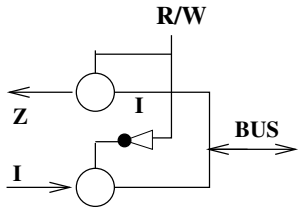
Fonction



Z := si (S) alors
I
sinon
HI
fsi

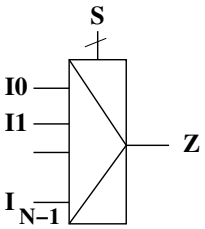
HI: Haute Impédence

exemple: bus bidirectionnel



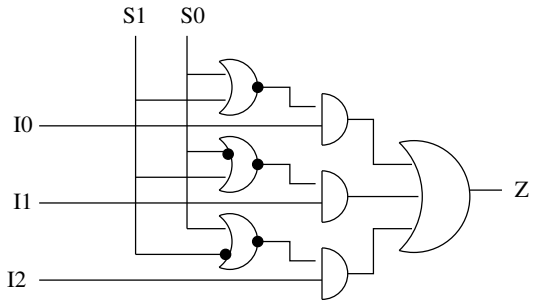
Multiplexeurs (N→1)

Fonction



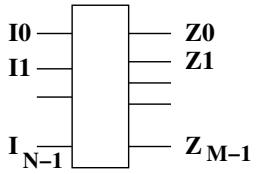
Z := cas (S)
0: I_0
1: I_1
...
i: I_i
...
N-1: I_{N-1}
fcas

Implémentation



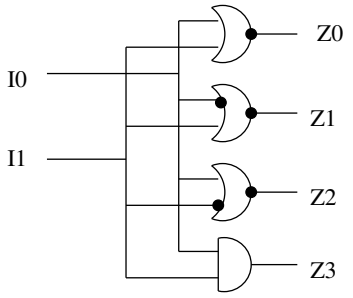
Décodeur ($N \rightarrow 2^N$)

Fonction Générateur de minterme, $N \rightarrow M = 2^N$, (2→4), (3→8)
Z_i := si (I=i) alors 1 sinon 0 fsi



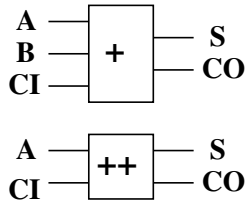
I_0	I_1	Z_3	Z_2	Z_1	Z_0
0	0	0	0	0	1
0	1	0	0	1	0
1	0	0	1	0	0
1	1	1	0	0	0

Implémentation



Additionneur, incrémenteur

Fonction

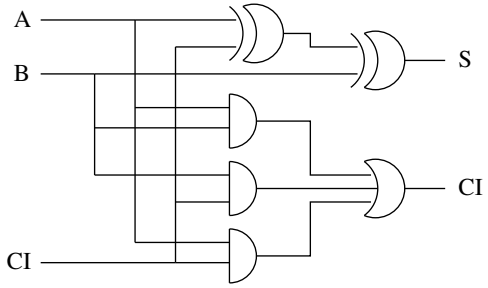


A	B	CI	CO	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Additionneur:
$S = A \oplus B \oplus CI$
$CO = A.B + A.CI + B.CI$

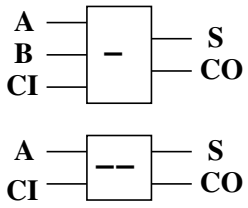
Incrémenteur:
$S = A \oplus CI$
$CO = A.CI$

Implémentation de ADD



Soustracteur, décrémenteur

Fonction

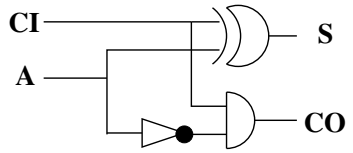


A	B	CI	CO	S
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Soustracteur:
$S = A \oplus B \oplus CI$
$CO = \overline{A}.B + \overline{A}.CI + B.CI$

Décrémenteur:
$S = A \oplus CI$
$CO = \overline{A}.CI$

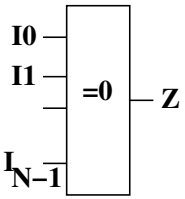
Implémentation de DECR



Comparateur

Égal à $\emptyset$

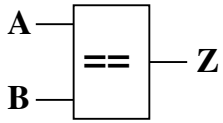
Fonction $N \leq 8$, $Z = 1 \Leftrightarrow \forall i, I_i = 0$



Implémentation

Égal

Fonction $Z = 1 \Leftrightarrow A = B$

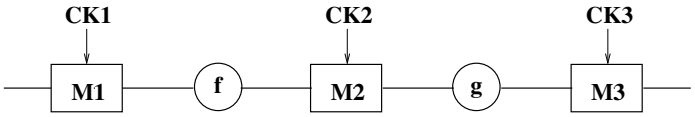


Implémentation

1.2.2 Cellules séquentielles

Définition

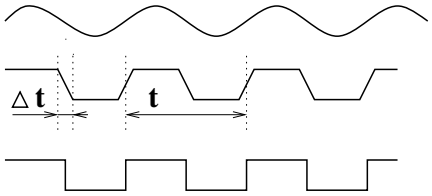
- Cellules de mémorisation (équivalentes aux variables des langages).
- Les mémorisation sont contrôlées par des signaux appelés horloge. Ils indiquent **quand** la mémorisation doit être faite. En effet, quand on fait des affectations, la séquence est importante:
  a=b+c; b=6; ≠ b=6; a=b+c
- Exemple:



temps	CK1	CK2	CK3	M1	M2	M3
t	H	H	H	x	y	z
$t+1$	L	H	H	a	y	z
$t+2$	H	L	H	a	f(a)	z
$t+3$	L	H	L	b	f(a)	g(f(a))

avec H: maintenir (hold) et L: charger (load)

- Les horloges doivent être des signaux francs



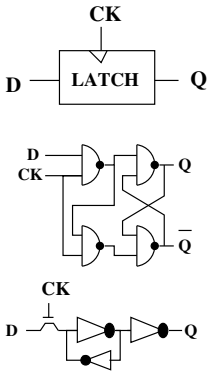
Il est déconseillé d'introduire des portes sur les horloges car

- Ca lisse les fronts ⟹ les cellules de mémorisation risque de disfonctionner.
- Ca désynchronise les mémorisations.

⟹ Une seule horloge controlant toutes les cellules séquentielles.

Latch (bascule à niveau)

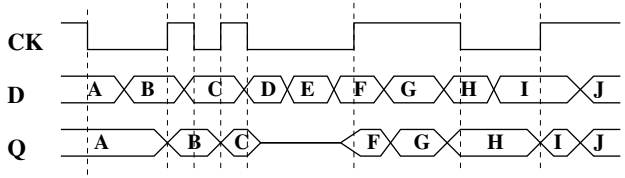
Fonction et implémentation



VI : bit

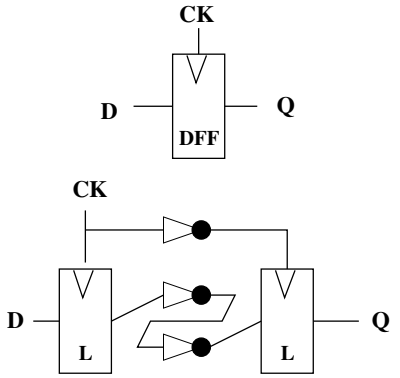
si CK=1 alors
  Q := D;
  VI := D;
sinon
  Q := VI
fsi
optionnel: clear, reset

Chronogramme



DFF (D Flip-Flop)

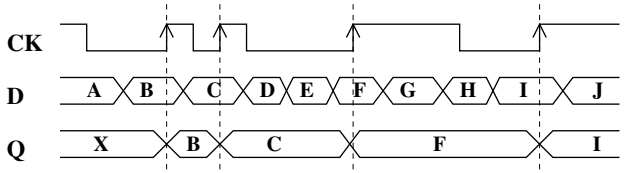
Fonction et implémentation



VI : bit

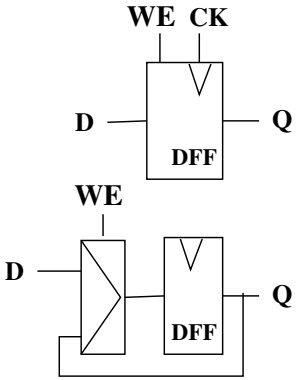
si CK ↑ alors
  Q := D;
  VI := D;
sinon
  Q := VI
fsi
optionnel: clear, reset

Chronogramme



DFF gardé

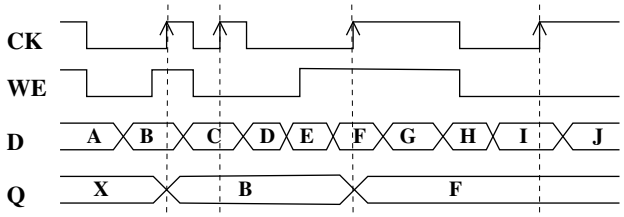
Fonction et implémentation



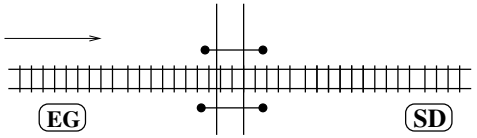
VI : bit

si CK ↑ et WE=1 alors
  Q := D;
  VI := D;
sinon
  Q := VI
fsi
optionnel: clear,
  reset

Chronogramme



1.2.3 Exemple



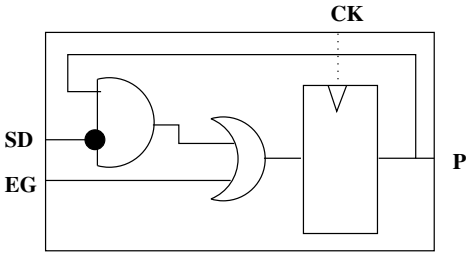
P=1 : train entre EG et SD
P' : valeur suivante de P

d'où P indique passage à niveau fermé

EG	SD	P	P'
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

P	EG SD			
	00	01	11	10
0			1	1
1	1		1	1

$P' = EG + \overline{SD}\,P$



1.3 Macro cellules

1.3.1 Principe

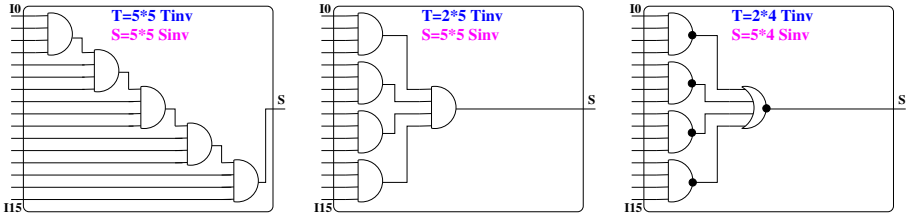
1. Les cellules de base sont limitées au niveau fonctionnel (pas de multiplieur, ...) dimension (pas de registre 32 bits, ...).

2. Tout construire avec des cellules de base trop fastidieux.

3. Une macro-cellule est un opérateur (il fournit une fonction). Elles combinent des cellules de base et/ou d'autres macro-cellules.

4. Les macro-cellules sont les blocs de base de la conception de CI.

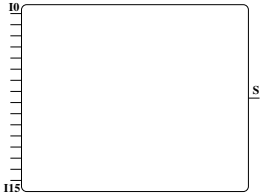
5. Caractéristiques d'une macro-cellule: temps de propagation (rapidité), surface.



1.3.2 Méthode de construction

1. Définir la vue externe de la boite: entrées/sortie, fonction.
   and16in: 16 entrées E_i d'un bit avec $i \in [0, 15]$, 1 sortie S d'un bit, la fonction est $S = \prod_{i=0}^{15} E_i$.

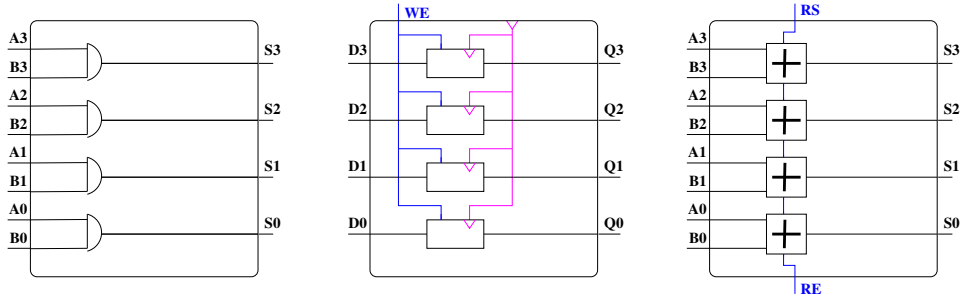
2. Dessiner la vue externe.



3. Dessiner l'intérieur.

1.3.3 Construction par tableau

La macro-cellule étend la fonctionnalité d'une cellule de base d'un bits à N bits (ou N bits, registre N bits, multiplexeur N bits, ...).

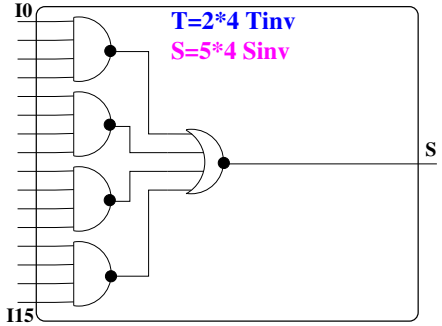


Dans le cas, où une entrée de la macro-cellule attaque une entrée de nombreuses cellules, il faut amplifier le signal.

1.3.4 Construction ad hoc

La fonction de la macro-cellule n'a pas d'équivalent en cellules de base ou la construction par tableau ne fonctionne pas. (or 16 entrées, multiplexeur N entrées, décodeur, multiplieur, décaleur).

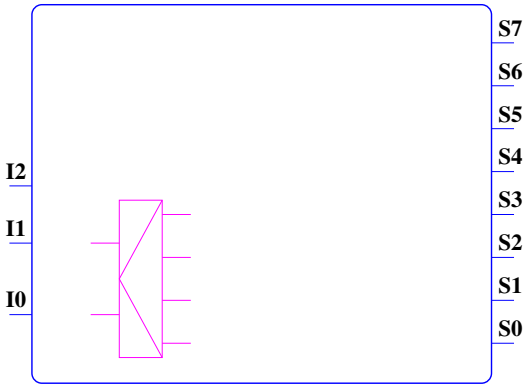
exemple: And 16 entrées



exercice: Décodeur 3x8
en utilisant:

1 décodeur 2x4

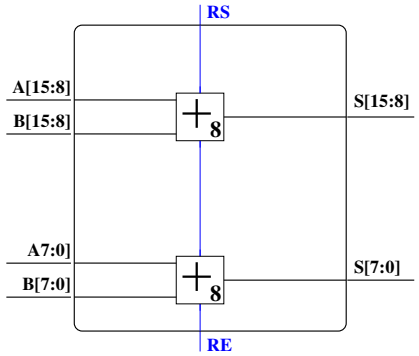
et un peu de glu.



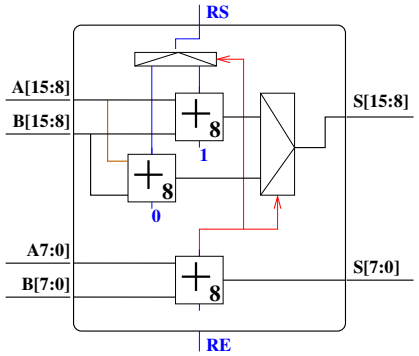
1.3.5 Recherche de performance temporelle

Procrastination

on veut tourner à 4 mH ($T = 250ns$) et on a une addition sur 16 bits.



Tp_1=25ns, Tp_8=200ns, Tp_{16}=400ns
$\Longrightarrow$ Addition en 2 cycles (500ns).

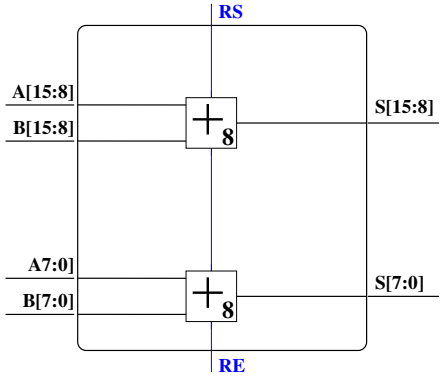


Casser la chaine longue en deux.
$Tp_{16} = Tp_8 + Tp_{mux} = 225ns$
$\Longrightarrow$ Addition en 1 cycle (250ns)

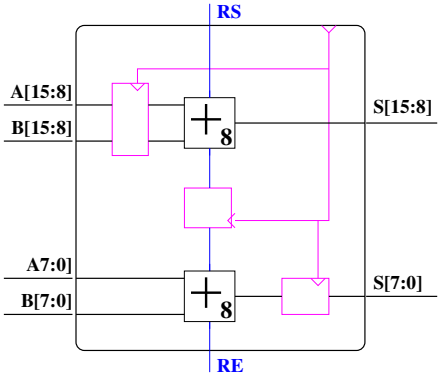
$\Longrightarrow$ Technique efficace mais coûte chère en surface.

Pipeline

on veut tourner à 4 mH ($T = 250ns$) et on a une addition sur 16 bits.



Tp_1=25ns, Tp_8=200ns, Tp_{16}=400ns
$\Longrightarrow$ 1 addition en 2 cycles (500ns).
$\Longrightarrow$ 3 additions en 6 cycles



Casser la chaine longue en deux.
$Tp_{16} = 2 \cdot Tp_8 = 2 \cdot 200ns$
$\Longrightarrow$ 1 addition en 2 cycles (500ns)
$\Longrightarrow$ 3 additions en 4 cycles

Résumé:

1. Une addition nécessite toujours 2 cycles.
2. N additions consécutives nécessite N+1 cycles soit:
   nombre additions par cycle= $\lim_{N\to\infty} \frac{N+1}{N} = 1$
3. Technique avec un coût en surface raisonnable.

1.4 Automates

1.4.1 Définition

Automate de Moore

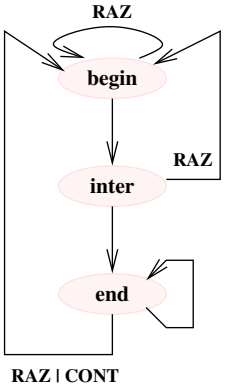
$\mathcal{Q}$:ensemble fini (états)
$\mathcal{E}$:ensemble fini (entrées)
$\mathcal{S}$:ensemble fini (sortie)
f:$\mathcal{Q}$ x $\mathcal{E} \rightarrow \mathcal{Q}$
g:$\mathcal{Q} \rightarrow \mathcal{S}$

Automate de Mealey

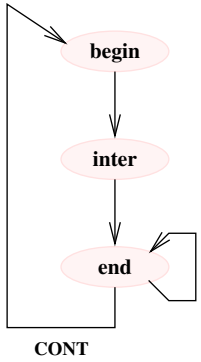
g : $\mathcal{Q}$ x $\mathcal{E} \rightarrow \mathcal{S}$

Exemple

Compteur par 2 avec RAZ et une entrée CONT qui contrôle le passage de 2 à 0



Reset synchrone



Reset asynchrone

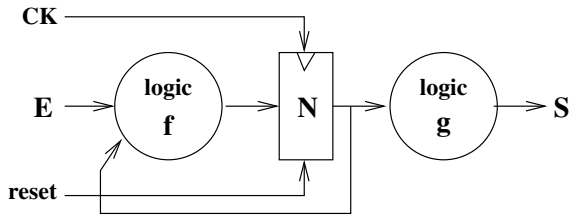
état	S_1S_0
begin	00
inter	01
end	10

1.4.2 Représentations usuelles

Voir figure 1.

1.4.3 Implémentation

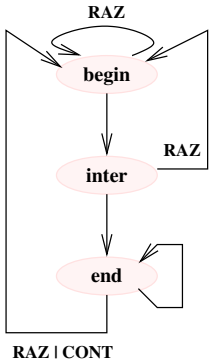
Méthode



1. codage des états
   - $2^N \geq \text{card}(\mathcal{Q})$: nombre de bits
   - simplifier g
   - simplifier f (1 bit change par transitions)
2. réalisation de f et g.
   - ce sont des fonctions logiques car les états sont codés.

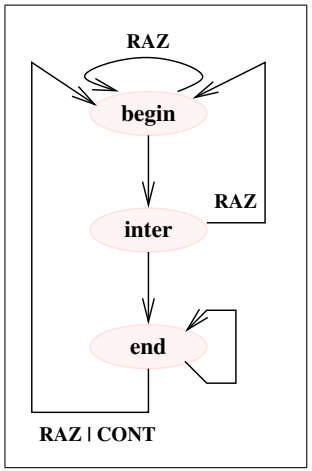
Exemple

Codage des états



3 états → 2 bits

état	Q_1	Q_0
begin	0	0
inter	0	1
end	1	0



état	S_1S_0
begin	00
inter	01
end	10

standard

```
state = 'begin'
pour toujours faire
 cas (state)
  'begin':
    out= '00'
    si RAZ=0 alors
      state= 'inter'
    fsi
  'inter':
    out= '01'
    si RAZ=0 alors
      state= 'end'
    sinon
      state= 'begin'
    fsi
  'inter':
    ...
 fin cas
fait
```

état	S_1S_0
begin	00
inter	01
end	10

algorithmique

état	entrées R/C		état suiv.	S
begin	1/∗	→	begin	00
begin	0/∗	→	inter	00
inter	1/∗	→	begin	01
inter	0/∗	→	end	01
end	1/∗	→	begin	10
end	0/0	→	end	10
end	0/1	→	begin	10

table

Figure 1: Représentations usuelles des autaomates.

Calcul de f

état	Q_1	Q_0	RAZ	CONT	D_1	D_0
begin	0	0	0	∗	0	1
begin	0	0	1	∗	0	0
inter	0	1	0	∗	1	0
inter	0	1	1	∗	0	0
end	1	0	0	0	1	0
end	1	0	0	1	0	0
end	1	0	1	∗	0	0

Q_1Q_0 R/C	00	01	11	10
00		1	∗	1
01		1	∗	1
11			∗	
10			∗	

$D_1 = Q_0\,\overline{RAZ} + Q_1\,\overline{RAZ}$

Q_1Q_0 R/C	00	01	11	10
00	1		∗	
01	1		∗	
11			∗	
10			∗	

$D_0 = \overline{Q_0}\,\overline{Q_1}\,\overline{RAZ}$

Calcul de g

Table des sorties

état	S_1S_0
begin	00
inter	01
end	10

Code des états

état	Q_1	Q_0
begin	0	0
inter	0	1
end	1	0

fonction g

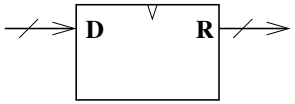
$S_1 = Q_1$
$S_0 = Q_0$

1.5 Processeurs

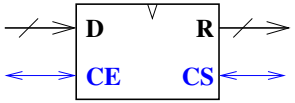
1.5.1 Définition

Un processeur est un bloc qui mange N données, déroule un algorithme puis produit M résultats et recommence indéfiniment.

Processeur synchrone Les nombres de données et de résultats sont fixes, l'algorithme se déroule en un nombre constant de cycles, le chronogramme des E/S est donc immuable (exemple: lecture de 3 données sur 3 cycles, 4 cycles de calcul, écriture de 2 résultats sur 2 cycles $\Longrightarrow$ 1 calcul tous les 9 cycles).

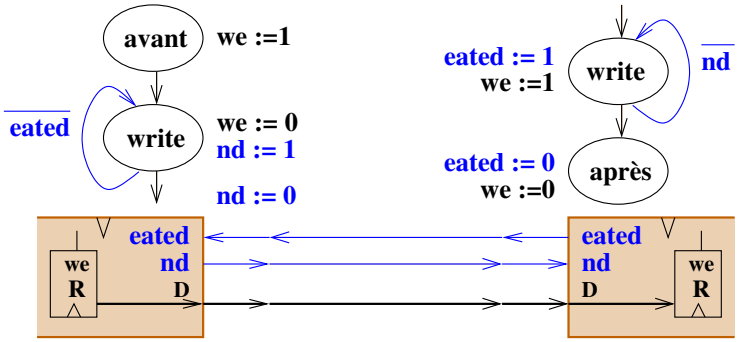


Processeur asynchrone Les nombres de données ou de résultats ne sont pas fixes ou l'algorithme se déroule en un nombre de cycles dépendant des données.
$\Longrightarrow$ ajout de signaux de synchronisation.



1.5.2 Protocole "poignée de main"

Le protocole de communication asynchrone le plus simple pour échanger une donnée est la poignée de main (handshake).



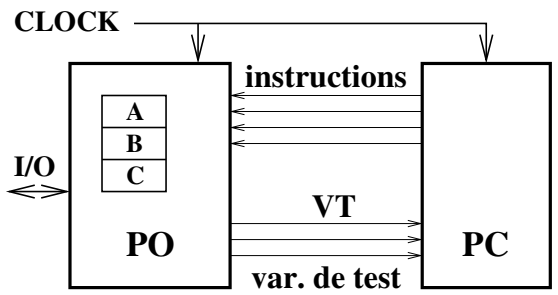
1.5.3 Fonctionnement

Voir la figure 2.

1.5.4 Étapes de conception

1. Spécifier l'interface du processeur et le protocole des E/S.
2. Spécifier l'algorithme du processeur.
3. Analyse de l'algorithme:
   - Définir les variables ($\Longrightarrow$ registres).
   - Lister les instructions
4. Réaliser la PO
5. Réaliser la PC

$\Longrightarrow$ méthode itérative.



PO: Partie **O**pérative

- Ensemble de variables
- Ensemble d'instructions
- algorithme:

```
pour tout CK faire
    cas (Inst)
        I0: a=a+b*c;
        I1: a=a+b;
            b=a+c;
            c=a;
        I2: b=a+c;
        ...
    fin_cas
fait
```

La PO exécute

PC: Partie **C**ontrôle

- Automate d'états (FSM)
- Etat $\Longrightarrow$ instructions
- algorithme:

```
etat= Initial
inst= f(etat)
pour tout CK faire
    etat= g(etat,VT)
    inst= f(etat)
fait
```

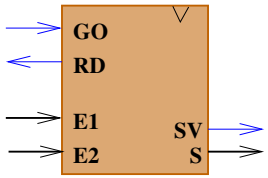
La PC ordonnance

Figure 2: Schéma général d'un processeur

1.5.5 Exemple PGCD

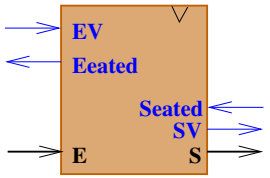
Interface et algorithme

Solution 1



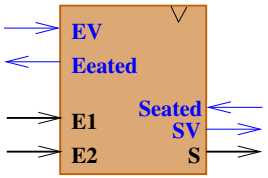

```
void pgcd()
 S_valid = 0
 répéter
  READY = 1
  attendre GO==1
  S_valid = 0
  lire A et B
  tant que A ≠ B faire
   if A < B alors
    B = B -A
   sinon
    B = B -A
   finsi
  fait
  ecrire A
  S_valid = 1
```

Solution 2




```
void pgcd()
 répéter
  lire A (handshake)
  lire B (handshake)
  tant que A ≠ B faire
   if A < B alors
    B = B - A
   sinon
    A = A - B
   finsi
  fait
  ecrire A (handshake)
```

Solution 3




```
void pgcd()
 répéter
  lire A & B (handshake)
  tant que A ≠ B faire
   if A < B alors
    B = B - A
   sinon
    A = A - B
   finsi
  fait
  ecrire A (handshake)
```

Analyse de l'algorithme

répéter
  lire A & B (handshake)
  tant que A ≠ B faire
    if A < B alors
      B = B - A
    sinon
      A = A - B
    finsi
  fait
  écrire A (handshake)

Variables

Besoin de 2 variables → RA RB

Instructions

RA ⟵ E1
RB ⟵ E2
S ⟵ RA
RA ⟵ RA - RB
RB ⟵ RB - RA
AeqB ⟵ RA = RB
AinfB ⟵ RA < RB

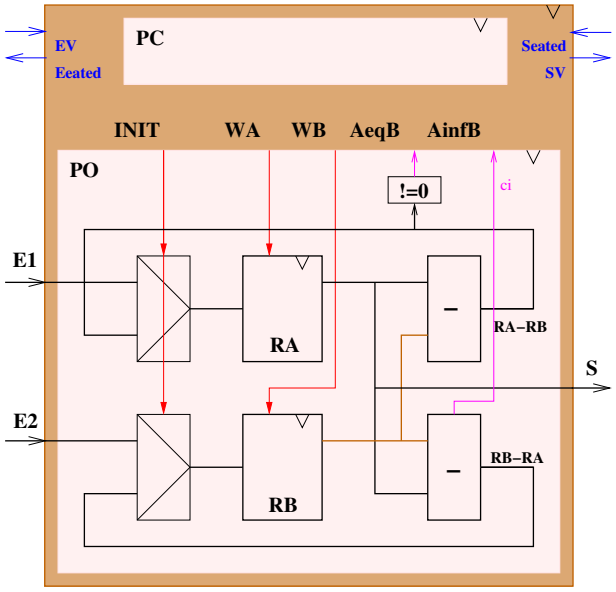
Réalisation de la PO

Variables

RA RB

Instructions

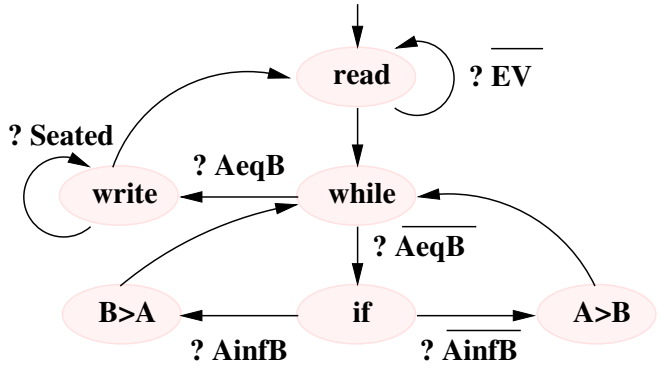
RA ⟵ E1
RB ⟵ E2
S ⟵ RA
RA ⟵ RA - RB
RB ⟵ RB - RA
AeqB ⟵ RA = RB
AinfB ⟵ RA < RB



Réalisation de la PC

Entrées de l'automate: EV Seated AeqB AinfB

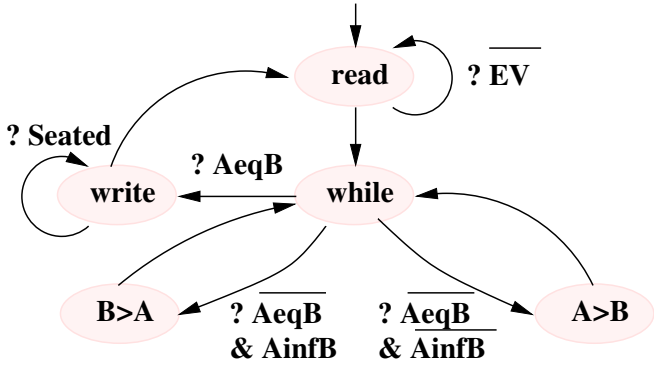
Sorties de l'automate: Eated SV INIT WA WB



état	Eea.	SV	INIT	WA	WB
read	1	0	1	1	1
while	0	0	*	0	0
if	0	0	*	0	0
A-B	0	0	0	1	0
B-A	0	0	0	0	1
écrit	0	1	*	0	0

⟹ 2 additionneurs, 3 cycles par itération.

Amélioration de la PC



état	Eea.	SV	INIT	WA	WB
read	1	0	1	1	1
while	0	0	*	0	0
A-B	0	0	0	1	0
B-A	0	0	0	0	1
écrit	0	1	*	0	0

$\Longrightarrow$ 2 additionneurs, 2 cycles par itération.
Attention: on ne peut pas aller directement de A>B à B>A et vice-versa.

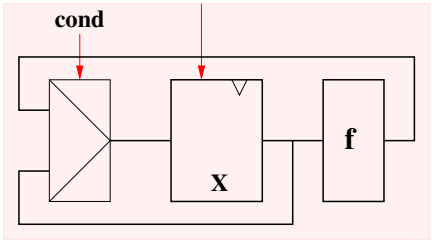
1.5.6 Câblage des si

Principe

Problème Une séquence "I1 si cond alors I2 sinon I3 fsi I4" nécessite au moins 2 états si le si est traité dans la PC.
$\Longrightarrow$ perte de performance.

Solution Faire exécuter le si par la PO, ainsi pour la PC on a la séquence "I1 I2-I3-câblé I4", elle peut éventuellement être exécutée en un cycle.

exemple "if cond then X = f(X) fsi" $\longrightarrow$ "X = cond ? f(X) : X"



Application au PGCD

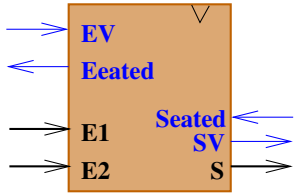
Interface et algorithme

```
void pgcd()
 répéter
 lire A & B (handshake)
 tant que A ≠ B faire
      if A < B alors
           B = B - A
      sinon
           A = A - B
      finsi
 fait
 écrire A (handshake)
```

```
void pgcd()
 répéter
 lire A & B (handshake)
 tant que A ≠ B faire
      B = B > A ? B-A : B
      A = A > B ? A-B : A
 fait
 écrire A (handshake)
```

Note: si A=B le corps de la boucle laisse A et B inchangés

Analyse de l'algorithme




```
lire A & B (handshake)
tant que A ≠ B faire
      B = B > A ? B-A : B
      A = A > B ? A-B : A
fait
écrire A (handshake)
```

Variables
Besoin de 2 variables → RA RB

Instructions

```
RA    ⟵ E1
RB    ⟵ E2
S     ⟵ RA
RA    ⟵ RB<RA ? RA-RB : RA
RB    ⟵ RA<RB ? RB-RA : RB
AeqB  ⟵ RA = RB
```

Partie opérative

Exercice

Partie contrôle

Exercice

1.6 Exercices

Exercice 1

Donnez le schéma au niveau transistor des portes NAND, NOR, OR et AND en technologie CMOS.

Exercice 2

Donnez en équivalent inverseur la surface et le temps de propagation des portes: NAND, NOR, AND et OR.

Exercice 3

Un processus de fabrication de CI donne une probabilité de 0.3 pour avoir au moins un défaut par cm^2.

1. Calculez la probabilité pour qu'un circuit de $1cm^2$ soit correct.
2. Calculez une borne maximale du taux de rendement au niveau d'un wafer de 100 cm^2 pour des CI de 5, 50, 500 mm^2.

Exercice 4

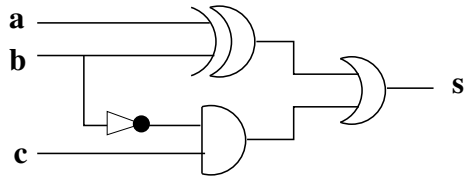
Faites le schéma en portes logiques de la fonction suivante:

a	b	c	f
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1

a	b	c	f
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

Exercice 5

Déterminez l'expression logique du schéma ci-dessous:



Exercice 6

Construisez un multiplexeur 4 entrées avec des trois-états et un décodeur.

Exercice 7

Complétez le chronogramme de la figure 3 avec les sorties Q d'un latch, d'un DFF et d'un DFF gardé.

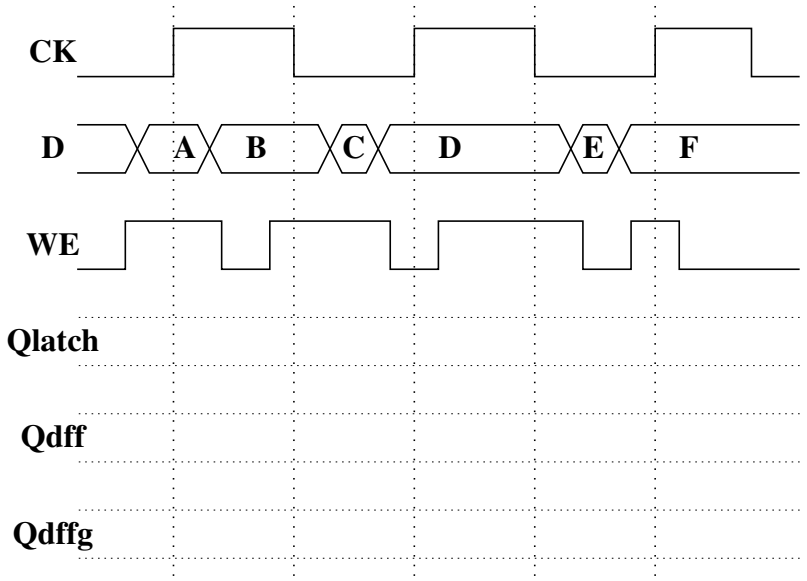


Figure 3:

Exercice 8

Implémentez la fonction
a.b $+\overline{a}$.c $+\overline{a}.\overline{b}.\overline{c}$
avec:

1. 1 multiplexeur à 8 entrées.
2. 1 multiplexeur à 4 entrées.

3. 1 multiplexeur à 2 entrées et 2 portes logiques.

Exercice 9

Réalisez un décodeur 4x16 iavec 2 décodeurs 2x4 et un peu de glu.

Exercice 10

Réalisation d'un additionneur-soustracteur avec:

a) Définissez la macro-cellule.

b) Construisez la avec un additionneur et un soustracteur.

c) Construisez la avec un additionneur et un un peu de glu.

Exercice 11

Définissez et construisez une ALU (Arithmetic and Logic Unit) dont les opérations sont: transparent, plus, moins, opposé, non, ou, et.

Exercice 12

Réalisation d'un multiplieur 4 bits.

a) Définissez la macro-cellule.

b) Construisez la.

c) Quelle est son temps de propagation en (additionneur 1 bit)?

d) Comment la pipeliner.

Exercice 13

Réalisez un automate qui compte modulo 6 (0, 1, 2, 3, 4, 5, 0, 1, ...) avec reset asynchrone.

Exercice 14

Construire un automate qui a une entrée E et une sortie S d'un bit et dont la fonctionnalité est: $S_i = 1$ si et seuleument si $E_{t-3} = 1 et E_{t-2} = 0 et E_{t-1} = 1$. Ainsi E=111010100..." donne S="000001010...".

Exercice 15

Construire un automate qui gère une pile de bits de taille 2 (on ne peut empiler que 2 bits) avec reset asynchrone. Donnez le nombre d'état d'une pile de 16 bits.

Exercice 16

Réalisez un processeur qui compte modulo N le nombre de cycle où son entrée E d'un bit est à 1 depuis le reset.

Exercice 17

Réalisez un processeur qui compte modulo N le nombre de fois où son entrée E d'un bit est passée de 1 à 0 depuis le reset.

Exercice 18

Construisez un processeur qui a une entrée E de N bits et une sortie S de N bits. Il lit 5 nombres sur E et écrit leur somme sur S.

1. Processeur asynchrone.

2. Processeur synchrone.

Exercice 19

Construisez un processeur qui a une entrée E de N bits et une sortie S de N bits. Il lit un premier nombre A sur E, puis il lit A nombres sur E et écrit leur somme sur S.

Exercice 20

Améliorez le PGCD vu en cours pour faire 2 itérations par cycle.

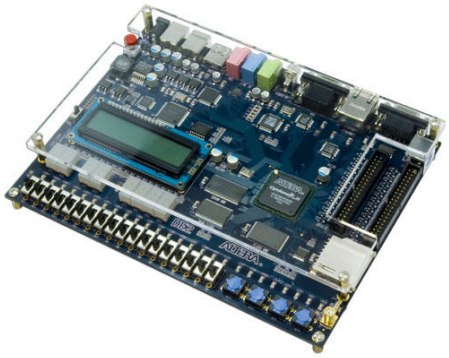
Exercice 21

Réalisez un processeur qui fait le PGCD de 2 nombres non signés de 8 bits en au plus 8 cycles.

2 FPGA

2.1 Carte FPGA et Quartus

2.1.1 DE2

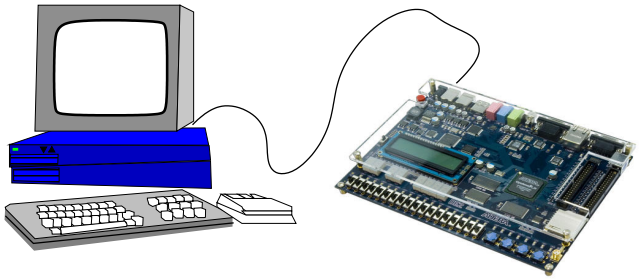


La carte FPGA (DE2 d'Altera) contient:

- 1 CI cyclone.
- des connecteurs: alimentation, chargement (usb), RS232.
- des horloges,
- des diodes, des 7 segments,
- des push-boutons, des switchs.

2.1.2 Quartus

C"est un logiciel tournant sur un PC permettant de décrire un CI et de le télécharger dans le FPGA.



Le logiciel comprend de nombreux formats de description de CI. Les deux principaux étant un éditeur schématique, le second étant le langage VHDL qui est un standard très utilisé dans la conception de CI.

2.1.3 VHDL

Signification: VHSIC Hardware Description Language
VHSIC: Projet de la defense US (**V**ery **H**igh **S**peed **I**ntegrated **C**ircuits)

1987 $1^{ère}$ version

Avantages complet, permet de décrire tous les CIs, solide,

Inconvénients: très complexe, verbeux, nombreuses descriptions équivalentes.

Utilisation:

- Simulation
- Synthèse

Le VHDL décrit dans ce cours est limité au VHDL synthétisable compris par Quartus.
$\Longrightarrow$ Comment traduire le schéma d'un processeur en VHDL.

2.2 VHDL: Format général d'une description

Un programme VHDL décrit un composant, un composant est une boite avec des valeurs qui arrivent en permanence[1] des ports d'entrée et qui écrit en

[1] on a flux de données sur chaque port et ces flux sont ordonnés dans le temps.

permanence des valeurs sur ses ports de sorties. Le format général d'une description est donnée ci-dessous:

LIBRARY,USE Ce sont des inclusions de librairies. Ici on inclut une librairie qui entre autre définit les types `STD_LOGIC` et `STD_LOGIC_VECTOR`.

ENTITY Elle définie l'interface du composant, c'est l'ensemble des ports externes, les port `IN` sont des ports d'entrée, les port `out` sont les ports de sorties. **Ces ports ne sont pas variables**, ce sont des flux de données, on les appelle des **signaux**,

ARCHITECTURE Ici on définit ce que fait le composant, trois possibilités de description sont possibles

- comportementale séquentielle (process).
- comportementale dataflow.
- structurelle (interconnexion de composants).

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;
USE IEEE.numeric_std.all;

ENTITY exemple IS
  PORT(
    port0  : IN  STD_LOGIC;
    port1  : IN
     STD_LOGIC_VECTOR
              (9 downto 0);
    port2  : OUT STD_LOGIC
  );
END exemple;

ARCHITECTURE mon_archi
  OF exemple IS
  -- declaration de signals
  -- declaration de composants
BEGIN
  -- process
  -- instructions DATA-FLOW
  -- instanciation de composant
END mon_archi;
```

2.3 VHDL: Types principaux

BOOLEAN Il peut prendre les true ou false

```
signal x :  boolean;
```

`STD_LOGIC` C'est un bit

```
signal x :  STD_LOGIC;
```

Il peut prendre les valeurs [2] `'0'`, `'1'` et `'-'`.

`STD_LOGIC_VECTOR` C'est un tableau de `STD_LOGIC`.

```
signal x :  STD_LOGIC_VECTOR(7 downto 0);
```

x est un tableau de 8 `STD_LOGIC` avec le poids fort à gauche. on peut ranger dans x des constantes de la forme `"00010001"` ou `"10010001"`.

[2] en réalité il y a encore d'autres valeurs

INTEGER C'est un entier sur 32 bits

```
signal x :  integer;
```

on peut ranger dans x les constantes 123, -16344.

Entier réduit (1) C'est un entier sur N bits, N étant inférieur à 32 et calculé automatiquement en fonction du range.

```
signal x :  integer range 0 to 22;
```

on peut ranger dans x les constantes 3 et 20 mais pas -123.

Entier réduit (2) C'est un entier sur N bits

```
signal x :  SIGNED(7 downto 0);
```

Énumération

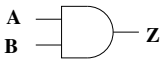
```
TYPE OPERATION IS (INIT, INCR, NOOP);
signal x :  OPERATION;
```

on peut ranger dans x les constantes `INIT`, `INCR` et `NOOP`.

2.4 VHDL: Composants combinatoires (data-flow)

2.4.1 Signal $\Longleftrightarrow$ fil

L'architecture contient l'instruction data-flow `z <= a AND b;`. Celle ci signifie que le signal z est un fil, il prend en permanence la valeur du "et logique" entre a et b. Il correspond donc au schéma:



Cette instruction peut s'exprimer de la manière mathématique suivante:

$$\forall t, z_t = a_t \wedge b_t$$

avec a_t, b_t et z_t les valeurs respectives des flux a, b et z au temps t,

Dessinez le schéma de cette architecture. Qu'en déduisez-vous?

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.ALL;

ENTITY ess IS
  PORT(
    a,b : IN  STD_LOGIC;
    z   : OUT STD_LOGIC
  );
END exemple;

ARCHITECTURE archi0 OF ess IS
BEGIN
  z <= a AND b;
END archi0;
```

```
ARCHITECTURE archi0 OF ess IS
BEGIN
  z <= a AND b;
  z <= a OR b;
END archi0;
```

Dessinez le schéma de cette architecture. Qu'en déduisez-vous?

```
ARCHITECTURE archi0 OF ess IS
BEGIN
  z <= a AND b;
  z <= a OR z;
END archi0;
```

Dessinez le schéma de cette architecture. Qu'est ce tmp?

```
ARCHITECTURE archi0 OF ess IS
signal tmp : STD_LOGIC;
BEGIN
  z   <= tmp AND NOT(a);
  tmp <= a AND b;
END archi0;
```

2.4.2 Table de vérité

Sur l'exemple ci-contre, est présentée une variante de l'instruction data-flow, le signal s prend en permanence la valeur `"111"` si le signal e vaut `"0000"`, la valeur `"011"` si le signal e vaut `"0001"`, la valeur `"110"` si le signal e vaut `"0010"`, ... et finalement pour les valeurs de e non énumérées la clause `OTHER` indique que le signal e vaut `"-0-"`. La clause `OTHER` est obligatoire si toutes les valeurs de e n'ont pas été énumérées.

Cette instruction implante les tables de vérité.

```
ARCHITECTURE exemple OF ess IS
  ...
  signal e : STD_LOGIC_VECTOR
                  (3 downto 0);
  signal s : STD_LOGIC_VECTOR
                  (2 downto 0);
  ...
BEGIN
  ...
  WITH e SELECT s <=
    "111" WHEN "0000",
    "011" WHEN "0001",
    "110" WHEN "0010",
    ...   WHEN ...
    "111" WHEN "1001",
    "-0-" WHEN OTHERS;
  ...
END archi0;
```

2.4.3 Multiplexeur

Sur l'exemple ci-contre, est présentée une autre variante de l'instruction data-flow. Le signal s prend en permanence la valeur `"111"` si la $1^{ère}$ condition est vraie, la valeur `"000"` si la $2^{ème}$ condition est vraie, la valeur `e0` si la $3^{ème}$ condition est vraie, et finalement si toutes les conditions précédentes sont fausses s prend la valeur du "ou logique" entre les signaux e0 et e1.

Les conditions doivent être mutuellement exclusives.

Si leur union (ou logique) n'est pas vraie, le dernier `ELSE` est obligatoire.

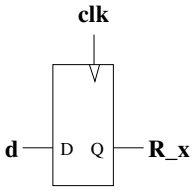
Cette instruction généralise les multiplexeurs (choisir 1 parmi N).

```
ARCHITECTURE exemple OF ess IS
  ...
e0 :  STD_LOGIC_VECTOR
            (3 DOWNTO 0);
e1 :  STD_LOGIC_VECTOR
            (3 DOWNTO 0);
s :  STD_LOGIC_VECTOR
            (2 DOWNTO 0);
  ...
BEGIN
  ...
s <=
  "111" WHEN (e0(0)<>e1(0))
         and (e0(1)<>e1(0))
         and (e0="000") ELSE
  "000" WHEN (e0(0)<>e1(0))
         and (e0(1)<>e1(0))
         and (e0="111") ELSE
  e0    WHEN (e0=e1) ELSE
  e0 OR e1 ;
  ...
END exemple;
```

2.5 VHDL: Composants séquentiels (process)

2.5.1 Registre simple

Un DFF sur front montant d'une horloge `clk` se décrit de la manière ci-contre.



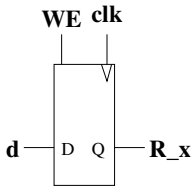
C'est le fait d'affecter le signal `R_x` dans un `PROCESS` qui le transforme en registre.

Le signal `R_x` peut toujours être lu partout, il correspond à la sortie du registre.

```
ARCHITECTURE exemple OF ess IS
  ...
  SIGNAL clk : STD_LOGIC;
  SIGNAL R_x : STD_LOGIC;
  SIGNAL d   : STD_LOGIC;
  ...
BEGIN
  ...
  PROCESS (clk)
  BEGIN
    if clk'EVENT AND clk = '1' then
       R_x <= d;
    END IF;
  END PROCESS;
  ...
END exemple;
```

2.5.2 Registre gardé

Un DFF sur front montant d'une horloge `clk` gardé par un signal `we` se décrit de la manière ci-contre.

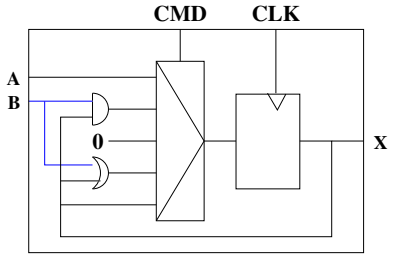


Le signal `R_x` est affecté dans un `PROCESS` c'est donc un point de mémorisation. Par contre, il n'est affecté que sur les fronts montant de `clk` seulement quand `we` est à 1.

```
...
SIGNAL we  : STD_LOGIC;
...
PROCESS (clk)
BEGIN
  if clk'EVENT AND clk = '1' then
    if we = '1' then
      R_x <= d;
    END IF;
  END IF;
END PROCESS;
```

2.5.3 Bloc registre

Il est possible de mixer dans un `PROCESS`, le DFF avec le multiplexeur qui le nourrit. Par exemple le compteur ci-dessous peut être traduit par le `PROCESS` ci-contre.



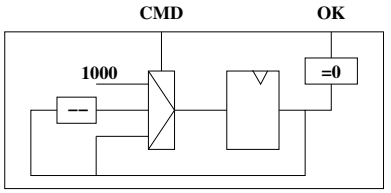
CMD: CLEAR INIT_A AND_B OR_B

```
ARCHITECTURE exemple OF ess IS
  ...
  SIGNAL clk,a,b : STD_LOGIC;
  ...
  TYPE   T_CMD_x IS (CLEAR,
            INIT_A, AND_B, OR_B) ;
  SIGNAL CMD_x : T_CMD_x;
  SIGNAL R_x : STD_LOGIC;
  ...
BEGIN
  ...
  PROCESS (clk)
  BEGIN
    if clk'EVENT AND clk = '1' then
      if CMD_x = CLEAR then
        R_x <= '0';
      elsif CMD_x = INIT_A then
        R_x <= a;
      elsif CMD_x = AND_B then
        R_x <= R_x AND  b;
      elsif CMD_x = OR_B then
        R_x <= R_x OR  b;
      else
        R_x <= R_x;
      END IF;
    END IF;
  END PROCESS;
  ...
END exemple;
```

2.6 VHDL: Composants usuels

2.6.1 Compteur

La description VHDL du compteur ci-dessous est donné ci-contre.



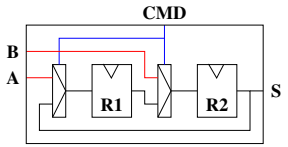
CMD: INIT, COUNT, NOOP

ATTENTION: cette description VHDL est fausse. Expliquez pourquoi et corrigez la.

```
ARCHITECTURE exemple OF ess IS
  ...
  SIGNAL clk : STD_LOGIC;
  ...
  TYPE   T_CMD_x IS (INIT, COUNT, NOOP) ;
  SIGNAL CMD_x : T_CMD_R;
  SIGNAL R_x : INTEGER
                  RANGE 0 to 1023;
  SIGNAL ok  : STD_LOGIC;
  ...
BEGIN
  ...
  PROCESS (clk)
  BEGIN
    if clk'EVENT AND clk = '1' then
      if CMD_x = INIT then
        R_x <= 1000;
      elsif CMD_x = COUNT then
        R_x <= R_x - 1;
      else
        R_x <= R_x;
      END IF;
      ok <= '1' when R_x = 0 else '0';
    END IF;
  END PROCESS;
  ...
END exemple;
```

2.6.2 Registres à décalage

Complètez la description VHDL ci-contre correspondant aux registres à décalage ci-dessous:

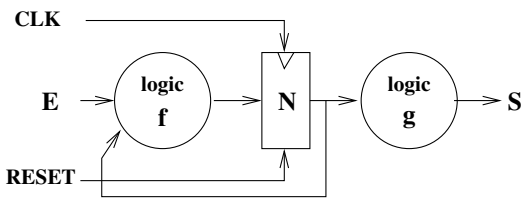


CMD: INIT, SHIFT, NOOP

```
ARCHITECTURE exemple OF ess IS
  ...
  SIGNAL A,B,S : INTEGER;
  ...
  TYPE   T_CMD_sh IS
         (INIT, SHIFT, NOOP);
  SIGNAL CMD_sh : T_CMD_sh;
  SIGNAL R_sh1,R_sh2 : INTEGER;
  ...
BEGIN
  ...
  PROCESS (clk) BEGIN
  if clk'EVENT AND clk = '1' then
    ...
  END IF; END PROCESS;
  ...
END exemple;
```

2.6.3 Automate d'état fini

L'implémentation matérielle d'un automate d'état fini de Moore est donné ci-contre.
Le gabarit VHDL d'un tel automate est donné ci-dessous.




```
ARCHITECTURE exemple OF ess IS
  ...
TYPE STATE_TYPE IS (etat0, etat1, etat2);
SIGNAL state   : STATE_TYPE;
SIGNAL e0,e1   : STD_LOGIC ;
SIGNAL s0,s1   : STD_LOGIC ;
  ...
BEGIN

  ...
-- fonction de transition de la PC (f)
PROCESS (clk)
BEGIN IF clk'EVENT AND clk = '1' THEN
  CASE state IS
    WHEN etat0 =>
      state <= etat1;

    WHEN etat1 =>
      IF e0='1' THEN
        state <= etat1;
      ELSE
        state <= etat2;
      END IF;

    WHEN etat2 =>
      IF e1='1' THEN
        state <= etat0;
      END IF;
  END CASE;
END IF; END PROCESS;

-- fonction de sortie de la PC (g)
WITH state SELECT s0   <=
  '0' WHEN    etat0,
  '1' WHEN    etat1,
  '0' WHEN    OTHERS;
WITH state SELECT s1   <=
  '1' WHEN    etat0,
  '0' WHEN    OTHERS;
END exemple;
```

2.7 Règles typographique de description d'un processeur

Le gabarit du VHDL décrivant un processeur est donné sur la figure 4. Les règles à appliquer sont données ci-dessous:

```
ARCHITECTURE exemple OF ess IS
-------------------------------------------------
-- connexions PC <--> PO
signal pc2po : STD_LOGIC;
signal po2pc : STD_LOGIC;
  ...
-------------------------------------------------
-- declaration pour la PC
TYPE STATE_TYPE IS (etat0, etat1, etat2);
SIGNAL state   : STATE_TYPE;

-------------------------------------------------
-- declaration pour la PO
  ...
BEGIN

-------------------------------------------------
-- Partie Controle
-------------------------------------------------

-- fonction de transition de la PC (f)
PROCESS (clk)
BEGIN
    IF clk'EVENT AND clk = '1' THEN
        CASE state IS
            WHEN etat0 =>
                state <= etat1;
            ...
        END CASE;
    END IF;
END PROCESS;

-- fonction de sortie de la PC (g)
WITH state SELECT s0   <=
    '0' WHEN    etat0,
    '1' WHEN    etat1,
    '0' WHEN    OTHERS;
WITH state SELECT s1   <=
    '1' WHEN    etat0,
    '0' WHEN    OTHERS;

-------------------------------------------------
-- Partie Operative
-------------------------------------------------

  ...
  ...
END exemple;
```

Figure 4: Gabarit d'un processeur.

Règle 1 Un fichier VHDL par processeur.

Règle 2 Bien séparer et marquer par des commentaires les parties opérative et de contrôle

Règle 3 Bien séparer et marquer par des commentaires les signaux de la partie contrôle, de la partie opérative et ceux servant d'interface entre les 2 parties.

Règle 4 Les signaux qui représentent des registres seront préfixés par `R_`.

Règle 5 Pour les blocs registre (registre + multiplexeur), on définira un type pour la commande et la commande sera de ce type. Par exemple si reg est le registre du bloc on aura:

```
TYPE T_CMD_reg is (...);
signal CMD_reg : T_CMD_reg;
signal R_reg : ...;
```

Règle 6 Il n'est pas interdit de commenter.

2.8 Exercices

Exercice 1

Donnez le schéma du processeur décrit en VHDL sur les figures 5 et 6.

Exercice 2

Traduire en VHDL la description du processeur donnée sur la figure 7.

Exercice 3

Le but de cet exercice est la conception d'un processeur qui lit le RX d'une ligne RS232 et génére des octets. Le protocole de la liaison RS232 est présenté dans l'annexe B et principalement à la section B.4 (page 34).

1. Spécifiez le processeur.
2. Donnez l'algorithme du processeur.
3. Implanter le processeur.
4. Donnez le VHDL de l'entity et de la partie opérative.

Exercice 4

Le but de cet exercice est la conception d'un processeur qui lit un octet et l'envoie bit par bit sur le TX d'une ligne RS232. Le protocole de la liaison RS232 est présenté dans l'annexe B et principalement à la section B.4 (page 34). On générera 2 stop bits.

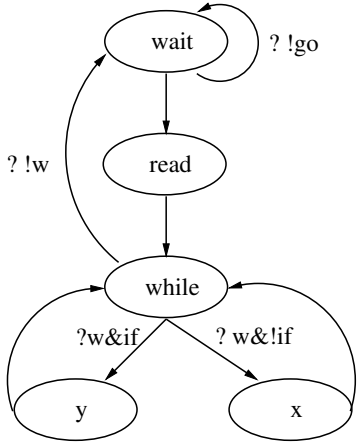
1. Spécifiez le processeur.
2. Donnez l'algorithme du processeur.
3. Implanter le processeur.
4. Donnez le VHDL de l'entity et de la partie opérative.

```
 1 LIBRARY IEEE;
 2 USE IEEE.std_logic_1164.ALL;
 3 use IEEE.numeric_std.all;
 5 ---------------------------------------------------------------------------
 6 -- Ce module additionne 2 nombres de 12 bits signes.
 7 -- Ses E/S sont les busin et busout.
 8 --
 9 -- Input:
10 --   busin_ctl (43 DOWNTO 40) : not used
11 --   busin_asrc(39 DOWNTO 32) : adresse emetteur (MYADR)
12 --   busin_ades(31 DOWNTO 24) : E_ADR
13 --   busin_data(23 DOWNTO 12) : operande B en complement a 2
14 --   busin_data(11 DOWNTO  0) : operande A en complement a 2
15 --
16 -- Output:
17 --   busout_ctl (43 DOWNTO 40) : "0000"
18 --   busout_asrc(39 DOWNTO 32) : MYADR
19 --   busout_ades(31 DOWNTO 24) : E_ADR
20 --   busout_data(23)           : V (overflow)
21 --   busout_data(22)           : C (retenue sortante)
22 --   busout_data(21)           : Z (resultat negatif)
23 --   busout_data(20)           : Z (resultat null)
24 --   busout_data(19 DOWNTO 12) : "00000000"
25 --   busout_data(11 DOWNTO  0) : resultat en complement a 2 (A+B)
26 ---------------------------------------------------------------------------
28 ENTITY plus12 IS
29  GENERIC(
30      MYADR : STD_LOGIC_VECTOR :=  "11111010" );  -- 250
31     PORT(
32         clk          : IN  STD_LOGIC;
33         -- interface busin
34         busin       : in  STD_LOGIC_VECTOR(43 DOWNTO 0);
35         busin_valid : in  STD_LOGIC;
36         busin_eated : out STD_LOGIC;
37         -- interface busout
38         busout       : OUT STD_LOGIC_VECTOR(43 DOWNTO 0);
39         busout_valid : OUT STD_LOGIC;
40         busout_eated : IN  STD_LOGIC);
41 END plus12;
44 ARCHITECTURE Montage OF plus12 IS
46     TYPE T_CMD_LoadNoop IS (LOAD, NOOP);
48     -- partie operative
49     -- le registre de transfert de busin vers busout
50     SIGNAL CMD_tft : T_CMD_LoadNoop;
51     SIGNAL R_tft   : STD_LOGIC_VECTOR(43 DOWNTO 0);
53     -- le registre resultat de A+B, ov
54     -- on etend R sur 13 bits pour avoir la retenue
55     SIGNAL CMD_res : T_CMD_LoadNoop;
56     SIGNAL R_res   : STD_LOGIC_VECTOR(12 DOWNTO 0);
58     -- les operandes A et B (1 bit de plus pour la retenue)
59     SIGNAL A,B     : STD_LOGIC_VECTOR (12 DOWNTO 0);  -- V1
60     -- bits de retenue et de somme de A+B             -- V1
61     SIGNAL r,s     : STD_LOGIC_VECTOR (12 DOWNTO 0);  -- V1
62     -- SIGNAL A,B   : SIGNED (12 DOWNTO 0);              -- V2
64     -- l' adresse destination
65     SIGNAL busin_ades    :  STD_LOGIC_VECTOR ( 7 DOWNTO 0);
67     -- message résulat
68     SIGNAL mess_resultat :  STD_LOGIC_VECTOR (43 DOWNTO 0);
70     -- partie controle
71     TYPE STATE_TYPE IS (ST_READ, ST_WRITE_TFT, ST_COMPUTE, ST_WRITE_SUM);
72     SIGNAL state : STATE_TYPE;
74 BEGIN
76 ---------------------------------------------------------------------------
77 --  Partie Opérative
78 ---------------------------------------------------------------------------
79     busin_ades          <= busin(31 DOWNTO 24) ;
80     a                   <= R_tft(23 DOWNTO 12) ; -- V1
81     b                   <= R_tft(11 DOWNTO  0) ; -- V1
82     -- a                   <= SIGNED (R_tft(23 DOWNTO 12)) ; -- V2
83     -- b                   <= SIGNED (R_tft(11 DOWNTO  0)) ; -- V2
85     mess_resultat(43 DOWNTO 40) <= "0000";
86     mess_resultat(39 DOWNTO 32) <= MYADR;
87     mess_resultat(31 DOWNTO 24) <= R_tft(39 DOWNTO 32);
88     mess_resultat(23)           <= -- overflow
89         '1' WHEN a(11)='1' AND b(23)='1' AND R_res(11)='0' ELSE -- N+N=P
90         '1' WHEN a(11)='0' AND b(23)='0' AND R_res(11)='1' ELSE -- P+P=N
91         '0' ;
92     mess_resultat(22)           <= R_res(12);  -- cout
93     mess_resultat(21)           <= R_res(11);  -- signe
94     mess_resultat(20)           <=   -- null
95         '1' WHEN R_res = "000000000000" ELSE '0';
96     mess_resultat(19 DOWNTO 12) <=   "00000000" ;
97     mess_resultat(11 DOWNTO 0)  <=   R_res(11 DOWNTO 0);
99     -- s,r <-- a + b;                                  -- V1
100    s <=  a XOR b XOR r;                               -- V1
101    r(0) <= '0';                                       -- V1
102    r(12 DOWNTO 1) <=                                  -- V1
103        ( a(11 DOWNTO 0) AND b(11 DOWNTO 0) ) OR       -- V1
104        ( a(11 DOWNTO 0) AND r(11 DOWNTO 0) ) OR       -- V1
105        ( r(11 DOWNTO 0) AND b(11 DOWNTO 0) );         -- V1
107    PROCESS (clk)
108    BEGIN IF clk'EVENT AND clk = '1' THEN
109        -- R_tft
110        if    ( CMD_tft = LOAD ) then
111            R_tft <= busin;
112        end if;
113        -- R_res
114        if    ( CMD_res = LOAD ) then
115            R_res(12 DOWNTO 0) <= s ;  -- V1
116            -- R_res(12 DOWNTO 0) <= STD_LOGIC_VECTOR(a + b) ;  -- V2
117        end if;
118    END IF; END PROCESS;
```

Figure 5: Description VHDL d'un processeur (partie 1)

```
120 ------------------------------------------------------------------------
121 -- Partie Controle
122 ------------------------------------------------------------------------
123 -- Inputs:  busin_valid busout_eated
124 -- Outputs: busin_eated busout_valid, CMD_res, CMD_tft, busout
125 ------------------------------------------------------------------------
127     -- fonction de transitition
128     PROCESS (clk)
129     BEGIN
130       IF clk'EVENT AND clk = '1' THEN
131           CASE state IS
132               WHEN ST_READ =>
133                   IF busin_valid  = '1' and busin_ades = MYADR THEN
134                       state <= ST_COMPUTE;
135                   ELSIF busin_valid  = '1' and busin_ades /= MYADR THEN
136                       state <= ST_WRITE_TFT;
137                   END IF;
139               WHEN ST_COMPUTE =>
140                   state  <= ST_WRITE_SUM;
142               WHEN ST_WRITE_SUM =>
143                   IF busout_eated = '1' THEN
144                       state  <= ST_READ;
145                   END IF;
147               WHEN ST_WRITE_TFT =>
148                   IF busout_eated = '1' THEN
149                       state  <= ST_READ;
150                   END IF;
151           END CASE;
152       END IF;
153     END PROCESS;
155     -- fonction de sortie
156     WITH state  SELECT busin_eated <=
157          '1'    WHEN   ST_READ,
158          '0'    WHEN   OTHERS;
160     WITH state  SELECT busout_valid <=
161         '1'     WHEN   ST_WRITE_TFT,
162         '1'     WHEN   ST_WRITE_SUM,
163         '0'     WHEN   OTHERS;
165     WITH state  SELECT CMD_res <=
166          LOAD   WHEN   ST_Compute,
167          NOOP   WHEN   OTHERS;
169     WITH state  SELECT CMD_tft <=
170          LOAD   WHEN   ST_READ,
171          NOOP   WHEN   OTHERS;
173     WITH state  SELECT busout <=
174         mess_resultat  WHEN   ST_WRITE_SUM,
175         R_tft          WHEN   OTHERS;
177 END Montage;
```

Figure 6: Description VHDL d'un processeur (partie 2)



état	init	wx	wy	ok
wait	*	0	0	1
read	1	1	1	0
while	*	0	0	0
x	0	1	0	0
y	0	0	1	0

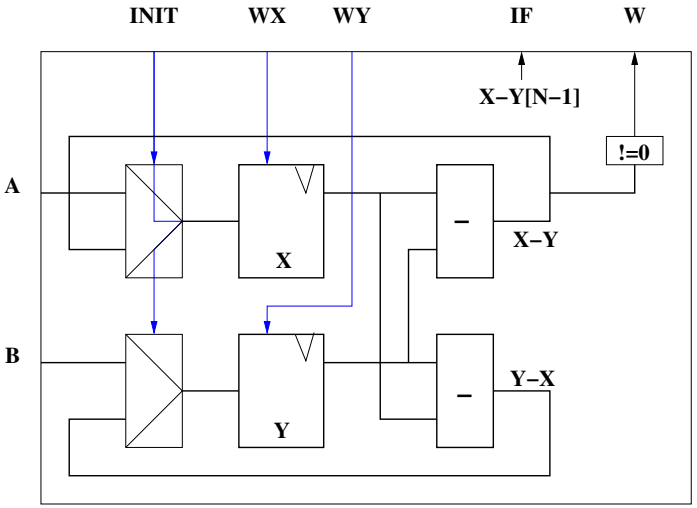


Figure 7: Description schématique d'un processeur

3 Travaux pratiques

3.1 Prise en main

Le but de cet exercice est de réaliser un montage dont la définition est la suivante:

entrées

- A: bouton pressoir
- B: un interrupteur
- C: un autre interrupteur

sorties

- S0: une diode
- S1: une autre diode

fonction

S0 est allumée {quand A est à 1 et (B et C ne sont pas simultanément à 1)} ou {quand A est à 0 et (B et C sont simultanément à 1)}.
S1 est allumée {quand A est à 0 et (B et C sont égaux)} {ou quand A est à 1 et (que B et C sont différents)}.

1. Écrire les fonctions logiques de S0 et de S1.

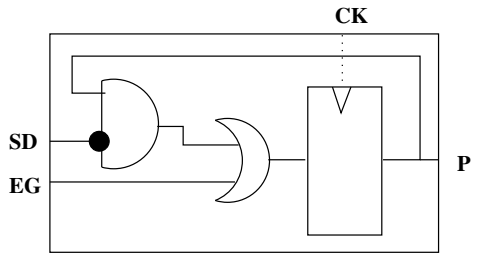
2. Réaliser le montage de S0 et S1 avec l'éditeur de schéma.

3. Réaliser le montage de $\overline{S0}$ et $\overline{S1}$ avec l'éditeur VHDL (Attention: le nom de base du fichier VHDL doit être différent du fichier schématique).

4. Modifier le montage précédent en y ajoutant une diode S2 qui sera allumée si les diodes S0 et S1 sont allumées.

3.2 Passage à niveau

Le but de cet exercice est de réaliser le montage correspondant au passage à niveau vu en cours page 8. Le schéma obtenu était le suivant:



entrées

- EG: le switch de gauche (entrée à gauche)
- SD: le switch de droite (sortie à droite)

sorties

- DG: la diode devant le bouton EG
- DD: la diode devant le bouton SD
- D3,D2,D1,D0: 4 diodes contigües entre DG et DD.

fonction

DG est allumée si EG est à 1 (présence d'un train).
DD est allumée si SD est à 1 (présence d'un train).
D3,D2,D1,D0 sont allumées quand un train est présent (passage à niveau fermé), et sont éteintes quand il n'y a pas de train (passage à niveau ouvert).

1. Réaliser le montage en VHDL.

2. En utilisant l'éditeur de schéma, et en instanciant deux composants, réaliser le montage pour une ligne à deux voies en sens contraire.

3.3 Décodeur 7 segments

Le but de cet exercice est de réaliser un montage dont la définition est la suivante:

entrées

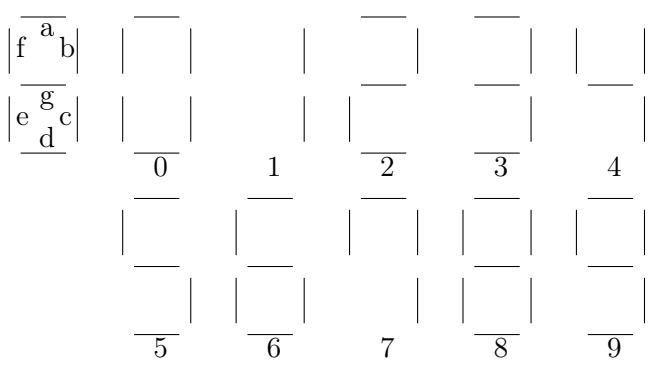
- e[3..0]: 4 interrupteurs contigus (de gauche à droite).

sorties

- err: 1 diode.
- s[6..0]: les 7 segments d'un afficheur.

fonction

err est allumée si la valeur binaire du vecteur e[3..0] est supérieure ou égale à la valeur décimale 10. err est éteinte dans le cas contraire.
Si err est allumée alors s[6..0] sont éteints, sinon s[6..0] sont allumés de manière à afficher le chiffre décimal représentant la valeur binaire du vecteur e[3..0] suivant la table donnée ci-dessous.



Montage 1 Réalisez en VHDL le montage et testez le.

Montage 2

1. Créez le composant du montage précédent.
2. Réalisez un montage instanciant 2 7-segments.
3. Testez le.

Montage 3 Réalisez un montage e[6..0] vers s[13..0] qui convertit le nombre binaire e[6..0] en décimal sur 2 chiffres ([0:99].

On pourra réaliser un petit programme (C ou autre langage) pour générer la table de conversion du binaire naturel vers 2 chiffres décimaux.

3.4 Automate

Le but de cet exercice est de réaliser un montage dont la définition est la suivante:

entrées A: 1 bouton.
sorties S: 1 diode
fonction

Lorsqu'on appuie sur le bouton la diode change d'état (Si elle est allumée, elle s'éteint. Si elle est éteinte, elle s'allume.). Lorsqu'on relâche le bouton, il ne se passe rien.

Exercice Donnez l'automate qui correspond à cette fonction.

Montage Réalisez le montage.

3.5 Diviseur de fréquence

L'objectif de ce montage est de réaliser des horloges. On visualisera ces fréquences sur des diodes. L'horloge de base de la carte est 50 MHz (appelée F dans la suite).

entrées
sorties S: 1 diode
fonction

La diode S clignote à N*F Hz. Elle enchaine des période éteinte allumée de $\frac{N}{2}$ cycles d'horloge.

Exercice Donnez le schéma qui correspond à cette fonction. Il contient:

- un compteur initialisé à N. Il est décrémenté à chaque coup d'horloge. Quand il atteint $\emptyset$, il est réinitialisé à N.
- S vaut en permanence $\emptyset$ si le compteur est inférieur à $\frac{N}{2}$ sinon 1.

Montage 1 Réalisez en VHDL le montage de ce compteur avec N égal à F.

Montage 2 Réalisez en VHDL un montage qui génère 3 horloges à 2 Hz, 1 Hz et 0,5 Hz.

3.6 RS232

1. Réalisez en VHDL le composant rs232out qui a été vu en TD. Un patron (rs232out.vhd.tpl) est disponible.

2. Faites le montage qui envoie au PC un octet dont la valeur sera entrée au moyen de 8 switches, le signal Ndata sera généré par un push-bouton, et le signal Busy sera visualisé par une diode.
   Testez votre montage avec les programmes C fournis.

3. Réalisez un montage avec l'éditeur schématique qui connecte les composants RS232in (fourni) et votre RS232out.
   Pour le tester, dans une fenêtre lancez un programme (fourni) qui lit le port série puis dans une autre fenêtre un programme (fourni aussi) qui écrit sur le port série.

3.7 BUSIA

1. Le composant terminateur dont le patron est disponible (terminateur.vhd.tpl) intègre deux fonctions:

   terminateur Il sort du bus vers l'extérieur tous les messages qui lui sont destinés (adresse 255).

   espion Il sort du bus tous les messages qui ont déjà fait un tour de bus. Ces messages ne sont pas mis à la poubelle mais envoyés aussi vers l'extérieur en vue de debug. En effet, le logiciel sur le PC affiche un message d'erreur quand il reçoit un tel message.

   La description précise de ce composant est donnée en commentaire dans le fichier patron.

   (a) Donnez l'algorithme de ce composant.

   (b) Donnez le schéma de sa partie opérative.

   (c) Donnez on automate de contrôle.

   (d) Complétez le fichier patron et compilez le.

2. Réalisez le BUSIA de base qui chaîne initiateur, plus12 et terminateur, qui connecte l'initiateur à rs232in, qui connecte le terminateur à terminateurSplit, et qui connecte le terminateurSplit à RS232out.

   On connectera le reset sur le push-button 0 et sur la diode verte entre les 7-segments. Quand le système tourne, la diode est éclairée.

   Testez le avec les programmes C fournis.

3. Pour la suite du projet vous devez insérer vos composants spécifique dans ce montage de base. Le composant "plus12" doit continuer à fonctionner tout le long du projet.

4 Projet

4.1 Objectif

Les objectifs du projet sont:

- De compléter le bus IA avec les composants: rs232out.vhdl et terminateurSplit.vhdl.
- D'y insérer le composant plus12.vhd fourni et de le tester avec les programmes C fournis également.
- D'ajouter sur le bus un ou plusieurs processeurs et de modifier les programmes C fournis pour qu'ils interagissent aussi avec ce ou ces composants.

Le ou les composants à ajouter sont ceux définis dans le chapitre suivant (sujet) ou éventuellement un sujet propre.
Dans ce cas, les contraintes sont:

- que le ou les composants soient ajoutés sur le bus,
- que ces composants interagissent avec le PC,
- que ces composants interagissent avec des E/S de la carte FPGA.
- que le sujet soit validé par l'enseignant.

4.2 Sujet

4.2.1 Titre

Réalisation d'une horloge multifonction.

4.2.2 Fonctions

Générateur de ticks Elle consiste à générer une horloge à 100 Hz. Cette horloge sera affichée sur une diode à 10 Hz.

Horloge Elle affiche l'heure H-MM-S sur 4 7-segments.

Générateur d'alarmes Avec les interrupteurs, on initialise une durée MM-SS sur 4 7-segments, puis on valide avec le bouton B4. Le générateur d'alarmes se met à décompter jusqu'à 00-00, qui se met à clignoter et envoie au PC un message ALARME. Pour pouvoir saisir une nouvelle valeur d'alarme, on appuie à nouveau sur B4.

A côté de ce mode manuel, il y a un mode logiciel, le PC envoie la durée de l'alarme.

Chronomètre On appuie sur le bouton B3, le temps se met à 0-00-0 (M-SS-D avec D $10^{-1}s$) sur 4 7-segments. Il se met à compter. On appuie sur le bouton B2, il s'arrête de compter et envoie un message au PC donnant la durée mesurée.

Sélecteur En fonction de 2 interrupteurs, il sélectionne la fonction affichée sur les 7-segments (horloge, alarme, chronomètre).

4.2.3 E/S du processeur

BUS-IA (entrée/sortie) Les échanges de messages avec le PC.

RUN (sortie) une diode.

B4 B3 B2 (entrée) 3 boutons pressoirs.

S4 S3 S2 S1 (sortie) 4 7-segments.

SEL1 SEL2 (entrée) 2 interrupteurs.

VAL[15:0 (entrée)] 16 interrupteurs.

4.2.4 Opérations du processeurs

Les commandes du processeur sont:

au reset Il fait clignoter RUN à $\sim$ 10 Hz. L'horloge se met à 0-00-0. Le générateur d'alarmes, dans l'état attente d'une demande d'alarme et affiche la valeur VAL. Le chronomètre affiche 0-00-0 et est prêt à chronométrer.

t-init(n) Génère un tick tous les n coups d'horloge.

t-check-ON() Demande au processeur d'envoyer au PC un message TICK1000 tous les 1000 ticks.

t-check-OFF() Demande au processeur d'arrêter d'envoyer des messages TICK1000.

h-init(h,m,s) Demande au processeur d'initialiser l'horloge à l'heure h-m-s.

a-init(m,s) Demande au générateur d'alarmes d'envoyer un message ALARME dans m minutes et s secondes.

Les réponses du processeur sont:

TICK1000() Ce message indique que 1000 ticks ont eu lieu depuis le message TICK1000 précédant. Il permet de calibrer le générateur de ticks (commande: t-init(n)).

ALARME() L'alarme demandée est arrivée.

TEMPS(m,s,d) Un temps a été mesuré par le chronométre, il vaut m minutes, s secondes et d 10^{-1} secondes.

4.3 Éléments à rendre

1. Un dossier décrivant l'application de manière non scolaire. Le plan type de ce dossier est donné en annexe C.

2. Une soutenance le dernier ou avant dernier jour du cours.

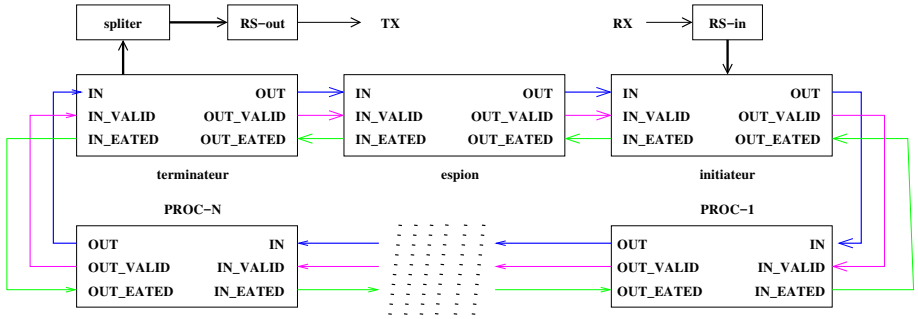
4.4 Éléments de notation

- Dossier (8 points à + ou - 2)
- Soutenance finale (12 points à + ou - 2)
  - Fonctionnement du composant plus12.
  - Fonctionnement de l'application.
  - Qualité du VHDL.
  - Niveau de l'application.
  - Qualité de l'interface utilisateur.

Tout développement en VHDL sauvage (hors des consignes) ne sera pas considéré.

Tout développement hors du bus ne sera pas considéré.

A BUS-IA



Le bus IA permet de chainer circulairement des processeurs et permet de propager des messages de 44 bits le long de la chaine comme le montre le schéma ci-dessus:

Format du message

bits 43 à 40 Ce sont les bits de contrôle du bus.
Les bits 43 à 41 sont réservé à un usage futur.
Le bit 40 est réservé au contrôleur du bus. Les processeurs doivent le mettre à $\emptyset$quand ils émettent un message. Le contrôleur détruit les messages dont ce bit est à 1 et transmet les messages dont ce bit est à $\emptyset$en le positionnant à 1.

bits 39 à 32 Ces 8 bits contiennent l'adresse du processeur qui a émis le message.

bits 31 à 24 Ces 8 bits contiennent l'adresse du processeur destinataire du message.

bits 23 à 0 Ces 24 bits sont les données du message.

Principe du bus IA

1. Un processeur qui reçoit de son prédécesseur sur le bus, un message (0,asrc,ades,data) ou (1,asrc,ades,data) où ades n'est pas son adresse, doit le retransmettre à son successeur sur le bus sans modification. Si l'adresse ades est son adresse ADR, il est libre d'envoyer à son successeur sur le bus $\emptyset$ou plusieurs messages (0,ADR,addr$_i$,data$_i$).
2. L'initiateur du bus lit sur la ligne RS232 4 octets (ades, data), en fait un message (0,255,ades,data) et l'émet sur le bus. Tant qu'il n'a pas de nouveau message, il transmet les messages arrivant de son prédécesseur sur le bus à son successeur sur le bus.
3. Le terminateur a l'adresse 255 et joue aussi le rôle de contrôleur. Lorsqu'il reçoit le message de son prédécesseur, (0,asrc,255,data) ou (1,asrc,ades,data), il écrit sur la ligne RS232 les 6 octets du message. Lorsqu'il reçoit de son prédécesseur, un message (0,asrc,ades,data) vec ades différent de 255, il le retransmet le message (1,asrc,ades,data) à son successeur.

Protocole

Le protocole de communication utilisé est la poignée de main.

Le protocole d'un émetteur sur le bus est: Placer le message sur OUT et 1 sur OUT_VALID, et les maintenir tant que OUT_EATED est égal à zéro.

Le protocole d'un récepteur sur le bus est de répéter tant que IN_VALID est égal à zéro: charger IN dans un registre en mettant 1 sur IN_EATED.

B RS232

B.1 Présentation

Le standard RS232 définit un protocole de communication série full-duplex point à point. Communication série signifie que tous les bits d'un mot sont transférés en séquence sur un seul fil électrique. Full-duplex signifie que le transfert peut se faire dans les deux sens en même temps (sur deux fils différents). Point à point signifie qu'il n'y a qu'un émetteur et qu'un seul récepteur.

Le standard de base autorise des transferts jusqu'à 20 Kbits/sec sur une distance de 16 mètres au maximum. Le standard a été par la suite étendu afin d'améliorer le débit et la distance.

Dans ce TP, vous utiliserez le protocole dans un cas particulier et nous ne fournirons donc que les informations dont vous aurez besoin.

B.2 Liste de signaux

Il existe deux brochages différents, le premier sur un connecteur DB-25, le second sur un connecteur DB-9.

DB-25	DB-9	Nom	Description
2	3	TX	Transmitted Data, TxD
3	2	RX	Received Data, RxD
4	7	RTS	Request To Send
5	8	CTS	Clear To Send
6	6	DSR	Data Set Ready
7	5	SG	Signal Ground
8	1	DCD	Data Carrier Detect
20	4	DTR	Data Terminal Ready
22	9	RI	Ring Indicator

RX entrée par laquelle transite les informations du correspondant vers l'ordinateur.

TX sortie par laquelle transite les données de l'ordinateur vers le correspondant.

GND masse de référence.

DTR sortie active à l'état haut qui permet à l'ordinateur de signaler au correspondant que le port série a été libéré et qu'il peut être utilisé s'il le souhaite.

DSR entrée active à l'état haut qui permet au correspondant de signaler qu'une donnée est prête.

RTS sortie active à l'état haut qui indique au correspondant que l'ordinateur veut lui transmettre des données.

CTS entrée active à l'état haut qui indique à l'ordinateur que le correspondant est prêt à recevoir des données.

RI entrée active à l'état haut qui permet à l'ordinateur de savoir qu'un correspondant veut initier une communication avec lui.

DCD entrée active à l'état haut qui signale à l'ordinateur qu'une liaison a été établie avec un correspondant.

En fait, dans sa forme la plus simple, une interface rs232 est réduit à 2 fils pour le transport des données (TX et RX croisés et autres entrées étant connectées à la masse), plus un fil de masse (GND). Dans ce cas on fait l'hypothèse que la ligne est toujours ouverte et que les interlocuteurs sont toujours prêts.

B.3 Paramètres de la communication

La liaison série est paramétrable en termes de : nombre de bits des mots échangés, fréquence de transfert, nombre de bits de synchronisation, et enfin présence et type de bit de parité pour la détection d'erreurs de communication. Le protocole fait l'hypothèse que les interlocuteurs se mettent d'accord sur ces paramètres AVANT de commencer la communication.

nombre de bits de données est compris en 5 et 8.

débit en bit par second (baud) est à choisir, pour chaque sens, parmi une liste prédéfinie (50, 75, 110, 134, 150, 200, 300, 600, 1200, 1800, 2400, 4800, 9600, 19200, 38400, 57600, 76800, 115200). Le débit définit la largeur d'une bit : à 200 baud, la largeur est de 5 ms.

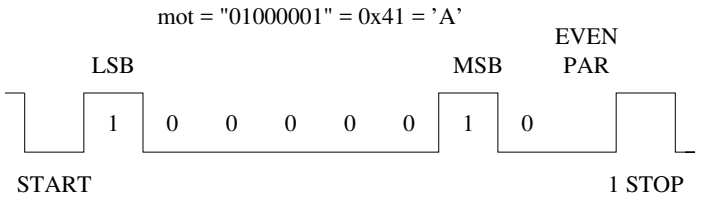
parité la parité est optionnelle et peut être paire ou impaire. Quand elle est paire, on compte le nombre de bit à 1 du mot envoyé et on complète par

0 ou 1 pour que le nombre totale de bit à 1 devienne pair. Pour la parité impaire, on cherche à obtenir un nombre de bit à 1 impair.

stop bits à 1 servant à la synchronisation ajoutés à la fin des mots au nombre de 1 ou 2.

B.4 Forme du signal

Quand rien ne circule sur la ligne, celle-ci est stable à l'état haut (1 logique). Le transfert d'un mot commence par 1 bit de start à 0, puis viennent les bits du mot en commençant par le poids faible et en finissant par le poids fort, puis on ajoute éventuellement le bit de parité, et enfin on termine par 1 ou 2 bit de stop à l'état haut. Le transfert suivant peut démarrer dès le cycle qui suit le dernier stop. Par exemple, pour le transfert d'un 'A', avec parité paire et un bit de stop.



C Dossier type

1 Introduction (max 1/2)

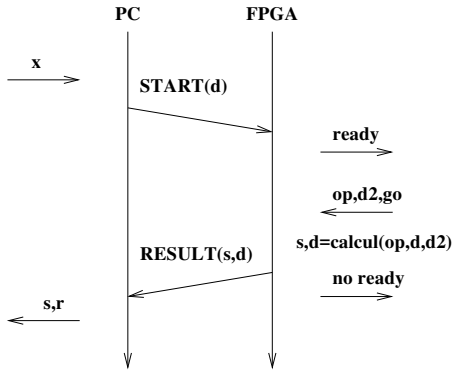
Donnez un nom à votre application et décrivez la de façon attractive et succincte.

2 Manuel utilisateur (1/2 à 1 page)

3 Présentation générale

3.1 Fonctionnement

Illustrez sous la forme d'un (éventuellement deux) MSC les actions de l'utilisateur, les sorties de l'application et les échanges entre le PC et la carte.



3.2 Format des messages

Spécifiez les types et les structures physiques des messages échangés entre le PC et la carte.

4 Description matérielle

4.1 Schéma général

Donnez le diagramme général des blocs au niveau du bus BUS-IA (voir figure 8).

N'oubliez d'indiquer pour chaque bloc son type.

4.2 Description des blocs

Donnez la vue externe (utilisateur) de chaque type de bloc: entrées, sorties, protocoles d'échange, ce qu'il fait, ...

5 Implémentation du bloc X

Choisissez un type de bloc et un seul, puis détaillez ce bloc X.

Pour cela on donnera son algorithme, son implémentation (schéma de la partie opérative, automate de la partie contrôle).

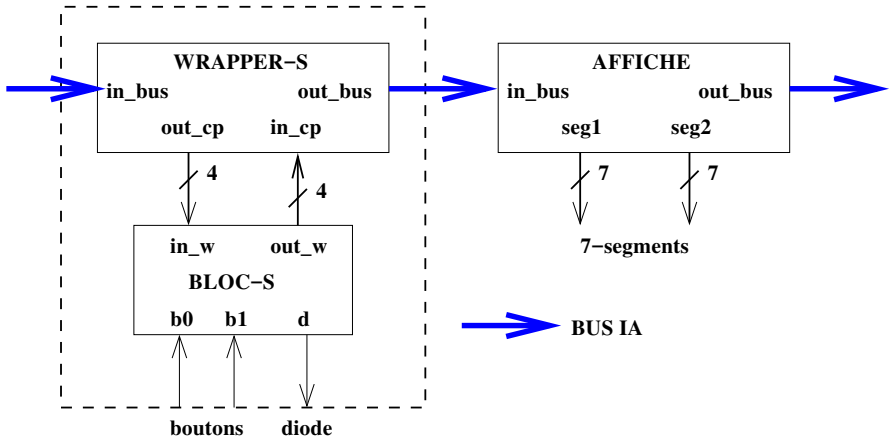


Figure 8: Bloc diagramme de l'application