Ensta Bretagne
2, rue François Verny
29806 Brest cedex
France
Tel +33 (0)2 98 34 88 00

Notes de cours

UE 1.2

July 2, 2020

Electronique Numérique
–travaux dirigés–

Jean-Christophe Le Lann
lelannje@ensta-bretagne.fr

TD1 – Electronique numérique

Représentation des nombres

1 Compter en base 2 et base 16

1. Compter de 0 à 15 en binaire, puis en hexadécimal. *(On conservera ces valeurs dans un tableau pour la durée du TD)*
2. Ecrire les puissance de 2^n pour n variant de 0 à 10. En déduire une valeur approchée de $\log_{10} 2$, puis $\log_{10} 5$. Avec un procédédé similaire, trouver une valeur approchée de $\log_{10} 3$
3. Calculer la valeur décimale de 10101011_2 en passant par les puissances de 2 et en passant par les puissances de 16
4. Ecrire les puissance de 2^{-n} pour n variant de 1 à 4

2 Conversions entre bases

1. Convertir le nombre 39 en binaire.
2. Convertir le nombre $(0.375)_{10}$ en binaire.
3. En déduire la représentation binaire du le nombre $(39.125)_{10}$
4. Convertir le nombre $(0.2)_{10}$ en binaire
5. Convertir $(345.158)_{10}$ en octal
6. Convertir $(389)_{10}$ en hexa
7. Convertir 011001111011101_2 en hexa
8. Convertir $(10110001101011.1111)_2$ en octal
9. Convertir $(10110001101011.111101)_2$ en hexa

10. Le nombre 101110001_2 est écrit en BCD (décimal codé binaire). Convertir le nombre décimal obtenu en binaire pur. Combien de bits sont alors nécessaires ? Conclure.

3 Nombres signés et non-signés sur k bits. Complément à 2. Addition et soustraction

Rappels

Jusqu'ici, nous nous sommes intéréssés aux nombres positifs. Il existe plusieurs manières de représenter un nombres négatif n. La manière la plus intuitive serait de dédier un bit (le plus à gauche) pour préciser le signe du nombre : par exemple on pourrait choisir 0 pour le signe + et 1 pour le signe −. Malheureusement, cette représentation possède deux défauts majeurs : il existe alors deux représentations du zéro et surtout, il faut modifier l'algorithme d'addition. Les architectes des ordinateurs ont retenu une autre représentation plus pratique : le **complément à 2**. C'est cette représentation qui régit le fonctionnement des *unités arithmétiques* des ordinateurs sur lesquelles nous reviendrons plus tard.

Avec k bits, on peut représenter l'ensemble des nombres

$$[-2^{k-1}, \ldots, 2^{k-1} - 1]$$

Le complément à 2 d'un entier n est obtenu par le calcul de la quantité $2^k - n$: on comprend ici que le terme "complément à 2" est un raccourci pour "complément à 2^k ". On peut également vérifier aisément que :

— Le complément à 2 du complément à 2 d'un entier n est n lui-même.
— Le complément à 2 de zéro est zéro (on néglige la retenue qui dépasse la taille fixée).

On doit retenir la formule de passage en complément à 2 d'un nombre négatif $-X$:

$$-X = \overline{X} + 1 \tag{1}$$

La barre figurant au dessus du X signifie "inversion de tous les bits" (complément à 1).

Si l'entier n est codé sur k bits $(s_{k-1} \ldots s_0)$ en complément à 2, on a alors :

$$n_{10} = -s_{k-1}2^{k-1} + \sum_{i=0}^{k-2} s_i 2^i \tag{2}$$

Exercices

1. Déterminer la plage des valeurs des entiers signés sur $k = 4$ bits. Représenter ces valeurs en binaire. Quelles sont les représentations des valeurs extrêmes ? de 0,1, et -1 ?

2. Quel est le complément à 2 du nombre 8, codé sur 6 bits ? Utiliser la formuler 1 et vérifier le résultat en appliquant la formule 2.
3. Soustraire : $91_{10} - 108_{10}$ en passant par le complément à 2
4. Ecrire les nombres suivants en binaire sur 6 bits : $+24, +31, -24, -31$. Passez ensuite à 7 bits. Que remarquez-vous ?
5. Soustraire 1100 0101 0100 0000 - 101000 1111 1111 - 110 1111 1110 - 10 1110

4 IEEE 754

Représenter le nombre 238,4375 dans sa représentation IEEE 754 (32 bits) en virgule flottante.

Electronique numérique
Logique combinatoire

Remarque : tous les exercices ne seront pas forcément traités en TD.

1 Equations logiques de circuits de base

1.1 Additionneur

1. Etablir la table de vérité d'un *demi-additionneur* logique, puis les deux équations logiques associées : ce circuit additionne simplement 2 entrées booléennes a et b (donc *sans retenue entrante*). Il calcule deux sorties : la somme et la retenue sortante. Dessiner le circuit.
2. Etablir la table de vérité d'un *additionneur complet* pour deux opérandes 1 bit. Ce dernier tient désormais compte d'une retenue entrante. Etablir les équations simplifiées des sorties, puis dessiner le circuit correspondant. Au préalable on démontrera que :

$$\overline{x \oplus y} = x \oplus \overline{y}$$

3. Montrer qu'un additionneur complet "1 bit" peut réutiliser le demi-additionneur de la question 1.
4. Dessiner un additionneur 4 bits en réutilisant l'additionneur 1 bit.
5. Quel est le chemin critique de ce dernier circuit ? On rappelle que le chemin critique est le parcours le plus long entre une entrée et une sortie combinatoire. On le mesurera en nombre de portes logiques élémentaires traversées.

1.2 Multiplexeur

La fonction (combinatoire) d'un multiplexeur est de sélectionner, grâce à un signal de contrôle, un signal d'entrée parmi plusieurs et de transmettre sa valeur sur une sortie unique.

1. Dessiner le symbole communément admis pour représenter un multiplexeur à 2 entrées binaires (chacune codée sur un 1 bit).
2. Calculer l'équation d'un multiplexeur à 2 entrées 1 bit.

3. Lorsque les données d'entrées sont codées sur $n > 1$ bits, chacun des bits du signal multiplexé suit évidemment un chemin similaire. Dessiner un tel multiplexeur à 2 entrées, pour $n = 2$.

2 Formes canoniques

Toute fonction booléenne F de plusieurs variables peut s'écrire sous deux formes spéciales, appelées *formes canoniques* : il ne s'agit ni plus ni moins que de l'expression de F sous forme de :

— (**première forme**) : somme de produits de chaque variable *ou de son complément*

— (**seconde forme**) : produit de sommes de chaque variable *ou de son complément*

Pour obtenir la première forme à partir de leur table de vérité, il suffit de sommer les produits pour lesquels la fonction vaut '1'. La seconde forme s'obtient de manière duale. Notons que ces formes ne sont pas simplifiées.

Etablir les deux formes canoniques des fonctions suivantes :

1. $F_1 = xy + yz + xz$
2. $F_2 = x + yz + \overline{y}.\overline{z}.t$

3 Quelques formulations booléennes

Ecrire sous la première forme canonique les fonctions définies par les propositions suivantes :

1. $F(a, b, c) = 1$ si et seulement si aucune des variables a, b, c ne prend la valeur 1.
2. $F(a, b, c) = 1$ si et seulement si au plus une des variables a, b, c prend la valeur 0.

4 Simplifications algébriques

Simplifier algébriquement les fonctions suivantes

1. $F = \overline{x}.\overline{y} + xy + \overline{x}y$
2. $F = (x + \overline{y})(x\overline{y} + z)z$

5 Simplifications par tableaux de Karnaugh

Simplifier, par la méthode des tableaux de Karnaugh, les fonctions booléennes suivantes. Dessiner le circuit correspondant et déterminer son chemin critique, en supposant que toutes les portes présentent un temps de propagation de 5 ns. Tous les problèmes ne seront pas forcément résolus en TE.

5.1 K-map à 3 variables

1. $F(a,b,c) = \overline{a}\overline{b}c + \overline{a}b\overline{c} + ab\overline{c}$
2. $F(a,b,c) = \overline{a}b\overline{c} + \overline{a}bc + ab\overline{c}$
3. $F(a,b,c) = \overline{a}\overline{b}c + a\overline{b}\overline{c} + ab\overline{c}$, sachant que la valeur de F pour les combinaisons $\overline{a}bc$ et abc est indifférente.

5.2 K-map à 4 variables

$F(a,b,c,d) = \overline{a}b\overline{c}\overline{d} + \overline{a}b\overline{c}d + \overline{a}bcd + \overline{a}bc\overline{d}$
$F(a,b,c,d) = \overline{a}\overline{b}\overline{c}d + \overline{a}\overline{b}cd + a\overline{b}\overline{c}d + a\overline{b}cd$

5.3 K-map à 5 variables

$F(a,b,c,d,e) = \overline{a}\overline{b}\overline{c}\overline{d}\overline{e} + \overline{a}\overline{b}c\overline{d}\overline{e} + \overline{a}b\overline{c}\overline{d}\overline{e} + \overline{a}bc\overline{d}\overline{e} + ab\overline{c}\overline{d}\overline{e} + ab\overline{c}d\overline{e} + abcd\overline{e} + abc\overline{d}\overline{e} + a\overline{b}\overline{c}\overline{d}\overline{e} + a\overline{b}c\overline{d}\overline{e} + \overline{a}b\overline{c}\overline{d}e + \overline{a}bc\overline{d}e + ab\overline{c}\overline{d}e + ab\overline{c}de + abcde + abc\overline{d}e$

6 Différentes utilisations du OU exclusif

1. Rappeler la table de vérité du OU exclusif, ainsi que son symbole (américain)
2. Dessiner la fonction OU-exclusif à partir de portes logiques ET, OU et NOT
3. Le OU-exclusif possède de nombreuses propriétés intéressantes. Compléter les énoncé suivants :
   (a) Sa sortie vaut 1 si exactement une des entrées vaut 1 :
   $$x \oplus y = ?? + ??$$
   (b) On peut le considérer comme un comparateur d'égalité pour 1 bit :
   $$x \oplus y = \text{SI } (x == y) \text{ ALORS } ? \text{ SINON } ?$$
   (c) On peut le considérer comme un inverseur conditionnel :
   $$x \oplus y = \text{SI } (?) \text{ ALORS } ? \text{ SINON } ?$$
   (d) On peut enfin le considérer comme un additionneur modulo 2.

7 Le jeu des cambrioleurs

Le principe du jeu est le suivant : un jeton est introduit par le joueur en haut du dispositif. Le but est de faire circuler ce jeton dans le dispositif, jusqu'à ce qu'il apparaisse en sortie. dans l'intervalle, le jeton peut rester coincé à différents endroits (ou "niveaux") : seul un code secret correctement introduit à cet endroit permet au jeton de continuer son parcours. Sinon, l'utilisateur voit le code d'erreur 0xDEAD apparaître en sortie. Chaque code secret, associé au niveau i est codé sur 4 bits.

1. Imaginer un circuit combinatoire qui simule ce jeu. On utilisera pour cela différents circuits connus : multiplexeurs et comparateurs. Enrichir votre circuit avec d'autres circuits combinatoires afin de le démarrer et d'indiquer le nombre de niveaux franchis correctement.
2. Sachant que l'on peut envoyer 1 million de codes par secondes, combien de temps faudrait-il, au pire cas, pour tester toutes les combinaisons possibles, dans le cas où le nombre de niveaux est 9 ? Refaites le calcul pour 5 bits.

8 Vu-mètre

On se propose de réaliser un "vu-mètre" : c'est un dispositif qui permet de visualiser une grandeur numérique (un volume sonore par exemple) au moyen d'une barrette lumineuse qui s'étire plus ou moins un fonction de la grandeur.

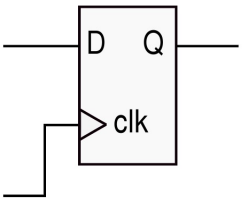
1. Schématiser le principe d'ensemble dans le cas où la grandeur à visualiser est codée sur 3 bits.
2. A partir du tableau de Karnaugh, trouver une réalisation possible.
3. On supposer que l'on dispose désormais d'un composant complexe de décodage (un décodeur). Rappeler la table de vérité de ce circuit. Utiliser alors ce décodeur pour proposer une nouvelle réalisation.

Electronique numérique
Logique séquentielle

Tous les exercices ne seront pas forcément résolus en TE.

1 Découverte de la bascule D

Le symbole de la bascule D, ainsi que sa table de vérité sont rappelés ici.

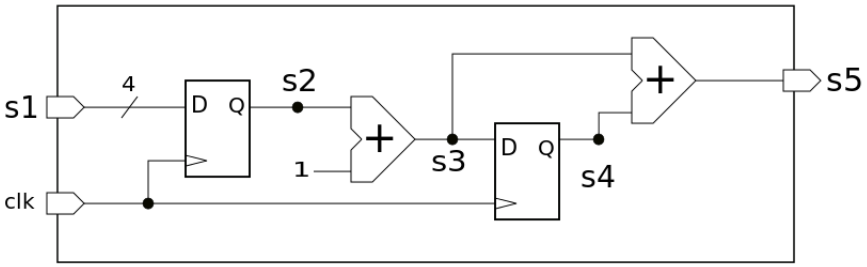


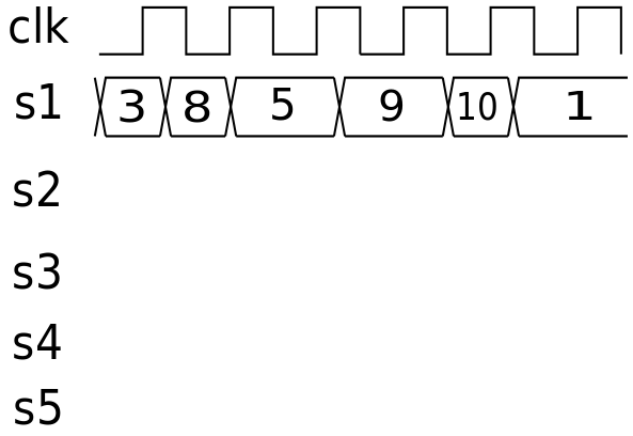
D	clk	Q_{n+1}
0	⎍	0
1	⎍	1
X	0 / 1	Q_n

La bascule D a pour fonction de recopier l'entrée D sur sa sortie Q, lors d'un court temps d'échantillonnage : il s'agit généralement du front montant de l'horloge. Cette horloge est supposée périodique. Pour que l'échantillonnage se passe correctement, le signal d'entrée doit respecter deux contraintes : il doit être parfaitement stable *avant* et *après* le front montant. Ces instants s'appellent temps de *setup* et temps de *hold*. La donnée est électriquement recopiée après un instant également très court appelé "clock to Q". Tous ces temps sont très inférieurs à la période d'horloge. **La bascule D permet ainsi de travailler sur un signal stable, pendant toute une période** : le temps est désormais **discrétisé**. Il n'y a plus à se soucier des temps de propagation caractéristiques – et parfois fluctuants– des différentes portes logiques réalisant les calculs ! Il suffit d'attendre une période d'horloge suffisamment longue pour observer un résultat combinatoire correct, échantillonné dans une bascule. L'écoulement du temps ne se fait que par

'quantum' de temps : celui des 'tops' (ou 'ticks') de cette horloge périodique. Il est même possible d'abstraire le fonctionnement du circuit encore plus en supposant que seuls ces instants d'échantillonnage sont significatifs et que les calculs s'effectuent instantanément sur ces 'tops' (on parle de calcul en "temps zéro"), et que rien ne se passe entre ces 'tops' ! Mais cela reste 'une vue de l'esprit'.

Compléter le chronogramme suivant où un signal S_1 sur 4 bits, extérieur au système, est fourni comme entrée (sous sa forme entière).

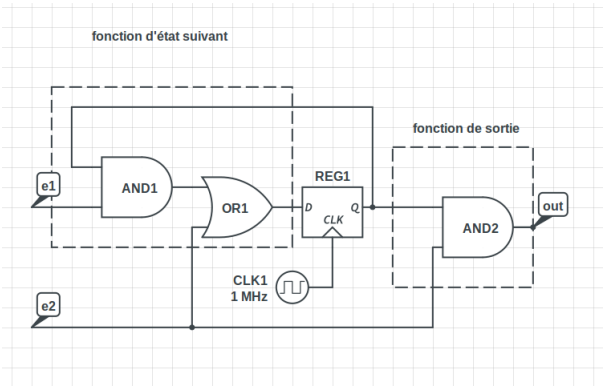




On précise ici que :
— nous ne tenons pas compte de la notion de temps de *setup* et temps de *hold*, ni du temps $t_{clk-2-Q}$
— les zones de transistion des 4 bits se présentent ici comme des 'X'.
— pour les signaux internes au système, on ne représentera pas ces transitions, supposées idéales : seuls des rectangles suffiront.
— on suppose que les sorties Q sont initialisées à 0.

2 Du circuit au chronogramme

Soit le circuit logique suivant.



Sachant que la bascule D est initialisée à 0, trouver le chronogramme de la sortie *out* correspondante, lorsque les valeurs d'entrée sont (respectivement, à chaque coup d'horloge) :

```
e1 : 0 1 0 1 1 0 1
e2 : 1 1 1 0 1 0 0
```

3 Du chronogramme au circuit

Nous proposons ici un exercice plus rare, uniquement par jeu : il s'agit de retrouver le circuit qui a généré les chronogrammes suivants. Il y a probablement plusieurs solutions ! Soyez perspicaces ! Les signaux e_i représentent des entrées, les signaux s_i des sorties, et les signaux n_i des signaux internes.

```
e1 : 0 0 1 1 0 0 0 0 1 1 0 0 1 0 0 0 0
e2 : 0 0 0 0 1 0 0 1 0 1 1 0 1 1 0 0 0
n1 : 0 0 1 1 1 0 0 1 1 1 1 0 1 1 0 0 0
s1 : 0 0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0
s2 : 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1 1
s3 : ? 1 1 0 1 1 1 1 0 1 1 1 1 0 1 1 1
```

4 Suite de Fibonacci en circuit

La suite de Fibonacci est bien connue : $u_{n+2} = u_{n+1} + u_n$, où $u_1 = 1$ et $u_0 = 0$. Elle date de 1202 et décrit la croissance d'une population de lapins, à partir d'un seul couple. Pour la petite histoire, cette suite a par ailleurs

un lien étonnant avec le nombre d'or $\phi = \frac{1+\sqrt{5}}{2} \approx 1.628$. En effet, Euler a démontré que pour un n donné (un "step"), on a :

$$u_n = \frac{1}{\sqrt{5}}(\phi^n - \phi'^n) \text{ , où } \phi' = -\frac{1}{\phi}$$

1. Dessiner le circuit numérique émettant la suite de Fibonacci. Un changement d'indice simple, au préalable, est conseillé.
2. Chaque bascule D possède un reset actif haut, et un set actif bas. Compléter le schéma précédent, de manière à initialiser correctement les bascules. On dispose d'un bouton poussoir d'initialisation qui fournit un signal de *reset* actif *haut* : lorsqu'on appuie sur ce bouton pour initialiser le circuit, ce signal vaut '1'.

Electronique numérique
Chemins de données ou *datapath*

Tous les exercices ne seront pas forcément résolus en TE.

1 Notion de chemin de données ou *datapath*

Nous avons déjà abordé le notion de "chemins de données" dans le "jeu des cambrioleurs" : il s'agissait uniquement d'acheminer, en combinatoire, une donnée d'une entrée vers une sortie, sans traitement spécifique. En général, les "datapath" sont plus complexes et associent à la fois logique séquentielle et logique combinatoire. Les datapaths permettent d'effectuer des calculs rapides. L'essentiel de leur puissance de calcul provient de deux notions fondamentales :

- Le **parallélisme** : il s'agit d'effectuer plusieurs opérations durant 1 seul cycle d'horloge, grâce à l'utilisation de plusieurs opérateurs matériels. Pour rappel, dans un processeur classique (et comme dans les langages dits "séquentiels"), les opérations se font en séquence, les unes à la suite des autres. Ici, à l'inverse, nous sommes en mesure de construire des calculateurs qui effectuent plusieurs opérations simultanément.
- Le **pipeline** : il s'agit d'effectuer des opérations "à la chaîne", sur des données, chaque *étage* du pipeline ayant une action spécifique sur ces données. Nous allons d'abord chercher à illustrer le rôle de la bascule D dans la structure d'un pipeline élémentaire : le registre à décalage.

2 Pipeline, Registres à décalage et Applications

Le registre à décalage le plus simple consiste à relier la sortie d'une première bascule D à l'entrée d'une seconde bascule D. Ceci se généralise à un nombre L de bascules ainsi enchaînées les unes aux autres (sans rebouclage). Une donnée e qui se présente au cycle n mettra L cycles (ou "coups d'horloge") à parvenir à la sortie s. Sa fonction est double :

1. Retarder la donnée a_0 d'un temps discret L : elle se présente sur l'entrée e au temps t et ressort sur la sortie s du dispostif au temps $t + L$.

2. Autoriser une donnée a_1 à se présenter sur l'entrée e au temps $t+1$, en même temps que a_0 est échantillonnée par la seconde bascule, etc.

Exercice 1 : registre à décalage simple Dessiner un registre à décalage de taille 4. Nommer correctement ses signaux et écrire l'équation de la sortie $s(t)$.

Exercice 2 : registre à décalage commandé Modifier le registre à décalage précédent de manière à le commander par un signal 'push', qui autorise les données à s'acheminer progressivement vers la sortie lorsque ce signal vaut '1'.

Exercice 3 : moyenne mobile Soit un flux séquentiel de données (boursier, médical, etc), dont les données successives arrivent au rythme de 100 Mhz. La moyenne mobile (ou moyenne glissante, ou *sliding average* en anglais) d'un tel flux continu permet de lisser une moyenne au cours du temps.

1. Calculer rapidement la moyenne mobile, sur 4 échantillons, du flux suivant :
$$..., 0, 0, 0, 0, 8, 9, 13, 4, 3, 2, 1, 0, 0, 0, ...$$
2. Concevoir un circuit numérique qui réalise le calcul d'une moyenne mobile sur 4 échantillons de nombres entiers, supposés non signés. Proposer une seconde solution. Laquelle est meilleure ? On s'autorise à utiliser la notion d'additionneur, mais pas de diviseur.
3. Combien de bits sont nécessaires pour calculer la somme de 4 données codées sur 8 bits ? Combien pour la moyenne mobile ?
4. Combien de bits sont nécessaires pour calculer la somme de n données codées sur m bits ?
5. Sachant que le temps de traversée de l'additionneur est 4 ns, quelle est la fréquence atteignable du circuit ? Est-ce compatible avec la fréquence d'arrivée des données ?

Exercice 4 : look-up table de FPGA Les FPGA sont des circuits réguliers qui ont l'étonnante propriété d'être reprogrammables à volonté. On parle de "circuits reconfigurables". Les FPGA sont constitués d'un assemblage de briques élémentaires appelées BLE (Basic Logic Element), interconnectées en réseau. Leur topologie détaillée est présentée dans le polycopié du cours. Un BLE se compose notamment d'une LUT (look-up table) à n entrées : c'est une mémoire qui permet d'enregistrer les valeurs attendues de n'*importe quelle fonction booléenne à n entrées*. C'est l'équivalent

matériel d'une petite table de vérité, qui se présente sous la forme d'un registre à décalage *avec 'enable'* (voir exercice précédent où on a appelé ce signal 'push') [1] à 2^n bascules : le flux qui entre dans ce registre à décalage s'appelle un *bitstream*. Il agit comme une programmation ou *configuration* de la LUT. Chaque sortie Q_i des bascules de la LUT est connectée à une des 2^n entrées d'un multiplexeur. Les signaux de contrôle du multiplexeur sont les n entrées, notées ici $\{a, b, c, \ldots, \alpha_{n-1}\}$.

1. Dessiner une LUT 2.
2. On suppose que le bitstream est entré à l'aide d'un signal de contrôle auxiliaire 'push' (cf exercice 2) de la manière suivante : 1000, où le bit le plus à droite représente la première donnée. Calculer la sortie du multiplexeur pour les combinaisons des entrées a, b.
3. Quelle est la fonction réalisée par le dispositif ?
4. Conclure en proposant un bitstream pour la fonction $xor(a, b)$.

3 Exo 5 : Pipeline et datapath

On cherche à concevoir un calculateur qui réalise l'opération arithmétique complexe $y = a * b * c * d$ en un seul cycle d'une horloge à 100 Mhz. Les données a, b, c, d sont codées sur 8 bits et stockées au préalable dans des bascules D (appelées registres). Le résultat y est également stocké dans un registre.

1. Sur combien de bits est-il nécessaire de stocker y ?
2. Dessiner une première architecture, purement combinatoire.
3. Sachant que le temps de traversée d'un multiplieur est de 4 ns, quelle est la fréquence maximum de fonctionnement du calculateur ? Respectez vous le cahier des charges ?
4. Catastophe ! Le temps de traversée du multiplieur est finalement revu à la hausse : 6 ns. Pouvez-vous corriger le circuit ? Respectez vous encore le cahier des charges ?

1. ...pour les besoins de notre exercice. Les LUTs des FPGAs sont réalisées au niveau transistors.

Electronique numérique
Automates (1)

Tous les exercices ne seront pas forcément résolus en TE.

Ce TD aborde plusieurs points relatifs aux *automates*, aussi appelés *machines d'états finis*[1] *finite state machines* (FSM).

— Diagrammes états transitions : machines de Mealy et de Moore.
— Causalité des machines d'états finis.
— Conception d'un automate matériel à partir d'un diagramme états-transitions.

1 Diagramme états-transitions. Moore et Mealy

On rappelle que le diagramme à bulle (ou "state transition diagram" ou diagramme "états-transistions") est une représentation graphique pratique des automates. Elle consiste à décrire l'enchaînement des transitions d'un état à l'autre par une flèche, les états étant représentés par des cercles ou "bulles".

On distingue deux grands types de FSM (voir figure 1)[2]. Dans le cas des automates de Moore, les sorties ne dépendent que des états, alors que dans le cas des automates de Mealy, les sorties dépendent à la fois des états et des entrées. Sur les transitions, on ajoute les *conditions* booléennes $c_i(s)$ qui autorisent à passer d'un état à un autre, ainsi que d'éventuelles sorties (généralement après une ou deux barres verticales) dans le cas des automates de Mealy. Lorsqu'on a affaire à un automate de Moore, les sorties sont annotées en conséquence dans les bulles (ou juste à côté !).

1. Ce nom quelque peu étrange signifie que le *nombre* d'états est *fini*. Il existe également des automates dont le nombre d'états est infini...

2. Attention ! Les deux automates représentés l'un à côté de l'autre ne sont pas équivalents en terme de fonctionnement. Il existe des algorithmes pour transformer Mealy en Moore (et réciproquement), de manière à obtenir des comportements identiques.... Voir par exemple `https://fr.wikipedia.org/wiki/Machine_de_Mealy`)

(a) label 1 (b) label 2

FIGURE 1 – FSM de Moore et Mealy

2 Causalité d'une FSM

Pour que qu'une FSM soit considérée comme *causale*, correcte ou *consistante*, elle doit vérifier les deux propriétés suivantes :

— **Réactivité** (Exhaustivité des conditions) : il doit exister une condition vraie sur au moins une des transitions sortant d'un état. .

$$\forall s \in S : \sum_i c_i(s) = 1$$

— **Déterminisme** (Exclusivité) : deux conditions sur les transitions sortant d'un état ne peuvent être vraies en même temps.

$$\forall s \in S, \forall (i,j) \text{ avec } i \neq j, c_i(s).c_j(s) = 0$$

Une autre manière de résumer ces deux propriété est qu'il existe – à tout instant discret – *un, et un seul, état suivant.*

Notons qu'un état peut avoir comme état suivant lui-même : dans ce cas, le système décrit ne change pas d'état.

Observez la figure 2.

— Vérifier la consistance de la machine d'état suivante.

— Est-ce un automate de Moore ou de Mealy ?

2.1 Equations logiques de l'automate

On cherche les équations logiques de la fonction "état suivant", ainsi que de la fonction de sortie de l'automate précédent. Ces deux ensembles nous permettrons de réaliser concrètement le circuit. On suggère ici de procéder en 2 étapes. La première consiste à conserver le nom *symbolique* des états tandis que la seconde fait apparaître l'*encodage* de ces états. Dans la FSM à réaliser, en choisissant un encodage *one-hot*, trois bits seraient nécessaires pour encoder les états. On va se limiter toutefois ici à un encodage *dense* où 2 bits seuls sont nécessaires. On s'appuie sur les tableaux génériques suivant 1 et 2, qui permettent de procéder de manière systématique.

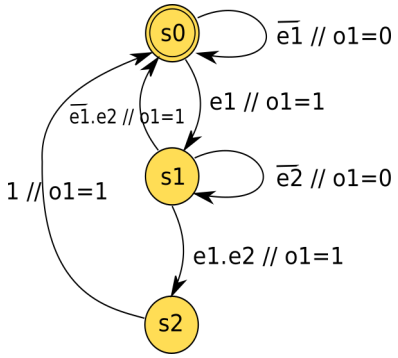


FIGURE 2 – FSM de l'exercice 1 et 2

TABLE 1 – Encodage symbolique (nom des états préservés)

état courant	entrée 1	...	entrée n	état suivant	sortie 1	...	sortie n

TABLE 2 – Encodage binaire (états encodés)

Q1	Q0	entrée 1	...	entrée n	D1	D0	sortie 1	...	sortie n

3 Application

Soit le diagramme états-transitions de la figure 3. On suppose que les entrées A et B proviennent de deux capteurs. Les sorties sont S1,S2 et S3.

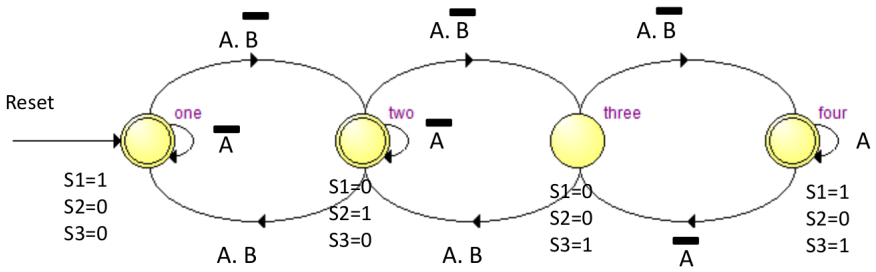


FIGURE 3 – Diagramme états-transitions (à modifier).

1. Cette machine est-elle causale ? Si non, proposer une modification afin de la rendre causale.

2. Proposer un encodage "one-hot" pour cette FSM.
3. Etablir les équations de la fonction de transition. On rappelle que ces équations déterminent les entrées D des bascules d'état de l'automate.
4. Etablir les équations de sortie.
5. Dessiner la structure globale du circuit, en faisant apparaître, sous forme de nuage logique, les fonctions précédentes.
6. Tenter de dessiner le circuit complet.

4 Automate "détecteur de séquence"

Il est fréquent de devoir détecter une séquence particulière dans un flux de données. Les automates se prêtent très bien à ce genre de détection. On peut citer :

— L'analyse de paquets réseaux, détection de paquets interdits (entrants ou sortants) dont on connait une certaine signature numérique. C'est le travail de filtrage d'un "proxy", etc
— L'analyse de flux multimedia : par exemple "start codes" de séquences, dans un film (stocké en numérique, dans un format de compression).
— L'analyse génomique : il existe des accélérateurs FPGA dédiés à la reconnaisance de séquence du génôme, etc.
— etc.

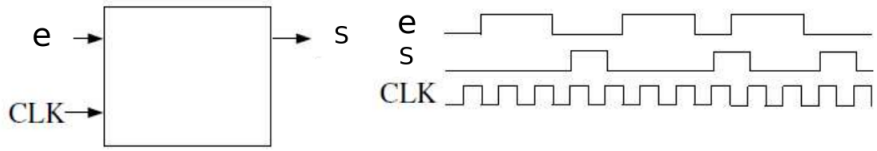


FIGURE 4 – Schéma de principe du détecteur de séquence. Chronogramme associé.

On considère le détecteur de séquence de la figure 5. Son rôle est de détecter les séquences "0-1-1-0" sur son unique entrée x. Lorsque cette séquence est détectée, il émet un '1' sur sa sortie y. Un chronogramme est fourni à titre d'exemple. Notez que dans le flux "0-1-1-0-1-1-0", la séquence apparaît deux fois !

1. Proposer un diagramme états-transitions pour ce détecteur. On choisira une machine de Moore.
2. Proposer un encodage dense des états.
3. Réaliser le circuit.

5 Automate "serrure numérique"

L'accès à un local est protégé par une serrure codée associée à un automatisme commandant la gâche électrique de la porte. La combinaison secrète est : A,D,C.

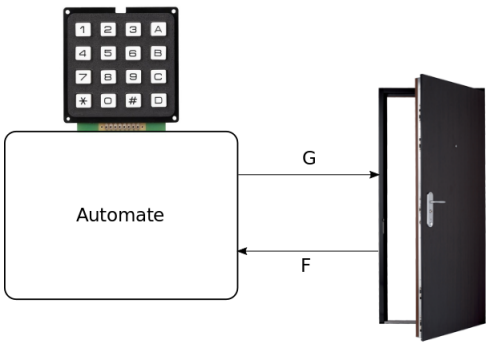


FIGURE 5 – Schéma de principe de la serrure numérique.

Un capteur permet de connaître l'état de la porte : F indique son état. F vaut à 1 si la porte est fermée à 0 si elle est ouverte. G commande l'ouverture de la porte : si G=1 la porte peut être ouverte. Lorsque la porte se referme, on a G=0. A l' initialisation la porte est fermée, le contact de porte F=1. On précise qu'on ne peut appuyer que sur un seul bouton à la fois.

1. Proposer un automate de contrôle du dispostif.
2. Discuter de la robustesse de la solution, et de son réalisme. Proposer des alternatives.
3. Optionel : construire le circuit complet.

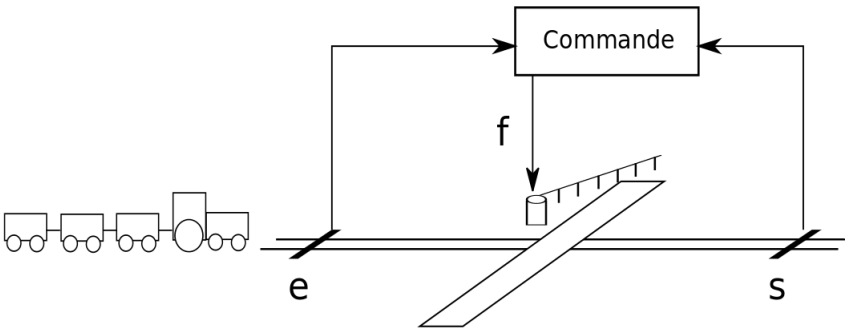
Electronique numérique

Machines d'états finis (2)

Tous les exercices ne seront pas forcément résolus en TE.

1 Problème de modélisation séquentielle

Le circuit suivant doit commander un passage à niveau.



Deux capteurs e (entrée) et s (sortie) signalent l'entrée et la sortie d'un train ; les signaux e et s durent exactement 1 cycle pour le passage d'un train ; il peut y avoir au maximum 2 trains entre les points e et s ; un train peut entrer alors que simultanément un autre sort ; le passage à niveau doit être fermé s'il y a au moins un train entre e et s.

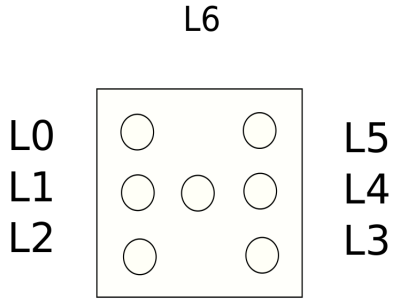
Questions

1. Donner un diagramme d'état ("à bulles") du circuit de commande.

2. Réaliser ce circuit à l'aide de bascules D.

2 Problème du dé électronique

On doit réaliser une boîte équipée d'un bouton poussoir, de 7 diodes lumineuses (LEDs) d'un circuit logique (à définir) et d'une pile. Ce dispositif doit simuler un dé, lancé aléatoirement : il affiche entre 1 et 6.



Chaque fois que l'utilisateur appuie sur le bouton-poussoir toutes les leds s'allument puis, dès que l'opérateur relâche le bouton, les LEDs se fixent à l'une des configurations ci-dessous.

A l'appui sur le bouton poussoir toutes les configurations sont affichées en séquence à la cadence d'une horloge rapide. L'utilisateur ne peut pas distinguer les combinaisons, du fait de la persistence rétinienne. Seule une illumination moyenne lui apparait.

Question

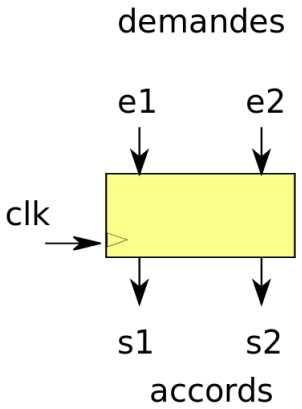
— Réaliser ce système, de manière à ce que la fonction de sortie de l'automate soit l'identité. Pour cela choisir un encodage des états approprié.
— Dessiner un schéma-blocs composé de 2 blocs : la logique d'état suivant et les bascules D (registres).
— Choisir un autre encodage (par exemple one-hot) et réaliser à nouveau le circuit.

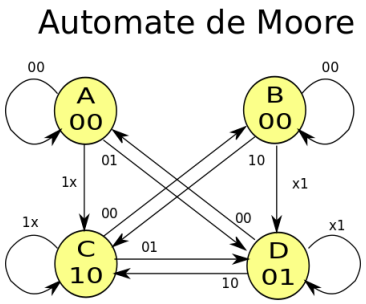
3 Du diagramme à bulles au circuit. Exemple de l'arbitre à priorité tournante (round-robin)

On rappelle que le diagramme à bulle est une représentation, qui se veut intuitive, du comportement séquentiel d'un automate d'états finis (FSM) : chaque état est représenté par un cercle (bulle) duquel peuvent partir différentes transitions (ou arcs) vers d'autres états possibles. Les transitions sont étiquetées par la condition booléenne amenant à l'état suivant.

On désire réaliser un circuit d'arbitrage entre deux clients susceptibles d'utiliser une ressource qui n'admet qu'un seul occupant à la fois (par exemple deux enfants qui veulent jouer au vélo, et il n'y a qu'un vélo. Le cas se présente également dans les ordinateurs, où différents périphériques peuvent vouloir accéder à un ensemble de fils –bus central– en même temps). Le but du circuit est d'accorder correctement la ressource à un seul client à la fois.

Les clients manifestent respectivement sur e_1 et e_2 leurs désirs d'utiliser la ressource : $e_i = 1$ si le client i désire la ressource, $e_i = 0$ s'il ne la désire pas.





dans les états : valeurs de s1,s2

sur les transitions : e1,e2
x : indifférent

Le circuit génère sur s_1 et s_2 l'attribution de la ressource à chaque client : $s_i = 1$ si la ressource est attribuée au client i, $s_i = 0$ sinon.

Lorsque la ressource est attribuée à un client, elle le demeure tant que le client la désire.

Lorsque la ressource est libre, elle est attribuée au premier qui la demande ; en cas de demandes simultanées, elle est attribuée selon une certaine priorité, mais pour éviter les injustices, la priorité entre les clients change à chaque attribution : le client qui obtient la ressource devient le moins prioriatire.

L'analyse du problème conduit à l'automate suivant : il y a quatre états, deux états A et B dans lesquels la ressource est libre, et deux états C et D dans lesquels la ressource est occupée.

Questions

1. Proposer un encodage des états tel que chaque état soit codé sur 2 bits
2. Etablir la fonction de transition du système sous forme de table de vérité. Simplifier.
3. Etablir la fonction de sortie du système. Simplifier.
4. Dessiner le circuit correspondant. Combien de portes faut-il ?

Electronique numérique
Initiation à VHDL (1/3)

Le but de ce TD est une première découverte du langage VHDL, et notamment de sa syntaxe. Nous aborderons également des concepts importants, comme la notion de design hiérarchique, à travers l'exemple de l'additionneur binaire. Nous donnerons également un exemple de banc de test ("*testbench*") qui sont le moyen de simuler nos circuits, en créant un laboratoire virtuel, grâce au langage VHDL.

Remarques :

- *Nous travaillerons sous Linux.*
- *VHDL n'est pas associé à un éditeur particulier. Vous pouvez utiliser gEDIT, vim, Emacs, Atom etc.*
- *Pour accéder aux outils Linux de l'Electronique, tapez "module load electronique" dans votre terminal.*
- *La compilation se fera en ligne de commande Linux.*

1 Notion de bibliothèques (library)

La plupart des descriptions VHDL s'appuient sur des **package**s préexistants, fournis par l'organisme IEEE qui est en charge de la standardisation du langage [1]

Ces packages correspondent à des *espaces de nommage* où l'on peut regrouper différents éléments :

- création de nouveaux types de données (**type)**
- déclaration de constantes importantes (**constant)**
- déclaration de fonctions (**function)** et procédures (**procedure)**

Ces packages permettent d'être réutilisés par différents circuits. Pour utiliser les packages, il est au préalable de les compiler dans une bibliothèque (**library**). Dans notre cas, nous n'allons pas créer de nouveaux packages, mais réutiliser ceux fournis par l'IEEE. Il s'agit essentiellement de 2 packages :

- $\boxed{std\_logic\_1164}$: ce package définit entre autre les types les plus usités.

1. L'IEEE (Institute of Electrical and Electronics Engineers) est la plus grande organisation mondiale de professionnels dans le domaine scientifique et technique.

- **std_logic** : ce type correspond au booléen ('0' et '1') , mais également au 'U' (undefined), 'X'(conflit), 'Z' (haute impédance), et d'autres valeurs possibles d'un signal.
- **std_logic_vector** : ce type correspond à un ensemble de signaux, que l'on cherche à manipuler comme un vecteur.
- Différentes fonctions de conversions
- La fonction **r**ising_edge (et **f**alling_edge) qui nous seront utiles pour la description des bascules D.

— *numeric_std* définit quant à lui :

- les types **signed** et **unsigned** permettant de décrire des nombres, avec un nombre de bits arbitraires.
- des fonctions permettant de réaliser directement les opérations arithmétiques (+,-,*, etc), sur ces types **signed** et **unsigned**.
- des fonctions de conversions permettant d'utiliser facilement ces types.

On retrouve fréquemment l'inclusion de ces bibliothèques de la manière suivante :

```
1 library IEEE; --appel a la bibliotheque compilee
2 use IEEE.std_logic_1164.all ; -- on declare utiliser tous les elements du package.
3 use IEEE.numeric_std.all;
```

2 Notion d'entité

L'**entity** d'un circuit modélise son enveloppe extérieure : on y retrouve le nom du circuit, ainsi que la liste des ports d'entrées et de sortie. Ces ports possèdent également un nom, une direction (**in** ou **out** dans notre cas) et un type.

```
1  entity MonCircuit is
2    port (
3      reset_n : in std_logic;
4      horloge : in std_logic;
5      e1 : in std_logic;
6      e2 : in std_logic;
7      e3 : in std_logic_vector(3 downto 0);
8      e4,e5 : in std_logic_vector(3 downto 0);
9      o1 : out std_logic;
10     o2 : out std_logic_vector(3 downto 0);
11     o3 : out std_logic_vector(3 downto 0) --pas de ';' pour le dernier port
12   ); --noter le ; ici
13 end MonCircuit; -- le nom du circuit est rappele ici.
```

Questions :

1. Dessiner l'enveloppe externe du composant décrit par cette entité.

2. Dessiner l'entité d'un demi-additionneur, puis le code VHDL de cette entité.

3 Notion d'architecture

A l'inverse d'une entité, l'**architecture** permet de décrire l'*intérieur* d'un composant. Une **architecture** s'organise syntaxiquement de la manière suivante :

```
1  architecture nom_architecture of nom_entity is
2  --declarations diverses : signaux, types, fonctions,...
3  begin --
4   -- corps de l'architecture
5  end nom_architecture;
```

Le corps de l'architecture contient un ensemble hétérogène d'éléments de description :

— Assignations concurrentes, éventuellement conditionnelles.

— Processus exécutés de manière séquentielle par le simulateur.

— Appels à des sous-composants.

Ces éléments fonctionnant en parallèle, l'ordre dans lequel ils apparaissent au sein de l'architecture n'a pas d'importance.

A titre d'exemple , on présente ici quelques unes des possibilités du langage (restreints à l'UE 1.2), à travers un exemple purement fictif :

```
1  architecture nom_architecture of MonCircuit is
2    signal s1,s2,s3 : std_logic; --declaration de 3 "fils" d'un bit chacun.
3    signal v32 : std_logic_vector(31 downto 0); -- une "nappe" ou "bus" de 32 bits
4    signal cout : std_logic;
5  begin
6
7    -- *** assignations concurrentes ***
8    s1 <= (e1 or not(e2)) xor e3; -- formule utilisant les ports d'I/O
9    s2 <= s1 and e3;
10   v32(1) <= s2; -- on connecte le signal s2 a un des fils de la nappe.
11   o3 <= '1' & v32(2 downto 0); constitution du port de sortie o3.
12
13   -- *** assignations conditionnelles ***
14   s3 <= s1 when s2='0' else e1;
15
16   -- instanciation d'entites :
17   inst1 : entity work.half_adder(logic)
18     port map(
19       clk => horloge,--repris de l'entite
20       a => e1,
21       b => s1,
22       cout => cout, -- signal "local"
23       sum => o1 -- connecte au port de sortie de l'entite
```

```
24      );
25  end nom_architecture;
```

Questions :

4. Rappeler la constitution interne du demi-additionneur.
5. Coder l'architecture de ce demi-additionneur.

4 Décrire un design hiérarchique : cas de l'additionneur

L'exemple d'architecture précédent présentait la notion d'instanciation d'entité : il s'agit de faire appel à des circuits décrits par ailleurs (par exemple dans un autre fichier) pour décrire l'architecture du circuit courant. Il faut voir le procédé comme le fait de prendre sur une étagère un composant préexistant, puis le connecter à des fils d'entrées-sorties par des signaux. On peut instancier autant de composants qu'on le souhaite. Le connexion peut se faire de la manière présentée dans l'exemple précédent : on indique où se connecte le signal en faisant référence au port dit "formel" sur lequel il vient se connecter.

Questions :

6. Dessinez la constitution interne d'un additionneur 1 bit complet, à partir du demi-additionneur.
7. Coder en VHDL l'architecture de cet additionneur 1 bit.
8. Coder en VHDL l'entité et l'architecture d'un additionneur 8 bits.

5 Notion de banc de test

Afin de simuler le circuit précédent (additionneur 8 bits), nous devons créer un banc de test virtuel. Ce banc de test permet de simuler le comportement d'instruments comme des générateurs de signaux qui fourniront des stimuli ou à l'inverse des sondes, oscilloscopes et loggers etc, qui nous permettrons de vérifier que le circuit conçu fonctionne comme on l'espérait. L'écriture de ces bancs de tests permet d'utiliser toute la puissance du langage VHDL : il n'est pas nécessaire que ces instruments soient décrits de la même manière que le circuit que l'on cherche à tester. On dit que ces bancs de tests ne sont pas forcément "synthétisables". C'est ce banc de test qui constituera notre simulation. Concernant l'archictecture du banc de test, il inclut donc :

— le circuit VHDL à tester (souvent labelisé 'DUT' pour "design under test")
— les générateurs de stimuli virtuels : générateurs d'horloge, de reset ou flux de données plus complexes, etc.

— les instruments de mesure virtuels.

Un banc de test pour le circuit d'addition est fourni sous Moodle.

Questions

10. Le testbench présente également une entité. Localisez cette entité. En quoi est-elle particulière ? Comment peut-on l'expliquer ?
11. Un générateur d'horloge est contenu dans le banc de test. Combien de lignes sont nécessaires ? Dessinez le chronogramme de cette horloge.
12. Pour information, le simulateur est dit "à événements discrets". Il fonctionne en exécutant le code de l'instant courant, qui provoque une avalanche d'événements futurs à planifier. Si toutefois, plus aucun événement n'est à traiter, le simulateur va s'arrêter (notion de *famine*). Il est possible de provoquer cette famine afin de sortir proprement d'une simulation. Comment notre banc de test provoque t-il concrètement cet arrêt ?
13. Les autres stimuli sont ici générés dans un *processus* : ce dernier s'exécute de manière séquentiel. Il existe des instructions qui permettent d"'étaler" dans le temps ces actions séquentielles. Dessiner le chronogramme des stimuli.

6 Simulation sous GHDL

L'outil GHDL est un simulateur VHDL open source et gratuit. Il est disponible sur plateforme Linux, MacOS et Windows. Il a été développé par un jeune ingénieur français : Tristan Gingold. C'est un véritable tour de force ! Cocorico !

Pour **analyser** la syntaxe d'un fichier VHDL, il suffit de taper, en ligne de commande :

```
ghdl -a MonFichier.vhd
```

A chaque fichier VHDL va correspondre un fichier binaire, généré par cette commande. C'est exactement ce qui se passe avec le langage C (la même technologie de backend GCC est d'ailleurs utilisée par GHDL).

Il faut ensuite réaliser l'exécutable de simulation, agglomération des fichiers précédents et d'un procédé spécifique de simulation (GRT) : c'est l'*édition de liens* ou **élaboration**. Ne pas oublier d'*analyser* le testbench également !

```
ghdl -a FichierBancDeTest.vhd
ghdl -e NomEntiteBancDeTest
```

Remarque : dans la dernière commande, noter que ce n'est pas le nom du fichier qui est passé en argument, mais le nom de l'entité, écrite dans le fichier. VHDL ne force pas un nommage identique entre le fichier et l'entité qu'il contient.

Enfin, pour *simuler* l'ensemble, on lance le simulateur avec (la lettre R faire référence à "RUN")

```
ghdl -r FichierBancDeTest --wave=test.ghw
```

Un fichier de traces de simulation (waveforms) est alors enregistré sous test.ghw. Pour le visualiser, lancer :

```
gtkwave test.ghw
```

L'ensemble de ces commandes bash peut être regroupé dans un script, auquel on donne les droits en exécution (chmod +x).

Analyser le résultat. Faites preuve d'initiative pour afficher les signaux voulus !

Questions

14. Votre additionneur 8 bits fonctionne-t-il selon vous ?
15. Observez les derniers stimuli du banc de test, ainsi que le chronogramme sous Gtkwave. L'additionneur fonctionne-t-il également pour les nombres signés ?

7 Pour aller plus loin...

Instructions *generate* L'instanciation mutliple de composant peut se révéler fastidieuse : imaginez un additionneur 64 bits ! Il faudrait instancier 64 fois les additionneurs 1 bits ! Fort heureusement, VHDL fournit un moyen de décrire ce procédé répétitif à l'aide d'une construction du langage : l'instruction *generate*. Votre enseignant vous montrera cela. Ces *generate* peuvent s'appuyer également sur des paramètres liés à l'entité, qui permettent de décrire des composants à "géométrie variable". Ainsi, notre entité pourrait être décrire pour un nombre n de bits passé en paramètre **generic**...Mais on peut faire encore plus simple...

Opérateurs du package *numeric_std* L'exercice que nous avons réalisé ici nous a permis de prendre en main le langage, à travers le cas de l'additionneur 8 bits. Cet additionneur a été construit de toute pièce à partir des équations logiques, que nous connaissions. Cet exercice reste toutefois "académique", car les packages IEEE nous fournissent directement le moyen de réaliser l'addition et autres opérations arithmétiques en utilisant les signes

"+", "-" etc...fort heureusement ! Ces opérations seront également parfaitement synthétisées en portes logiques : le synthétiseur connait les équations aussi bien que vous !

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity test is
6  end test;
7
8  architecture example of test is
9    signal a,b,f : signed(7 downto 0);
10 begin
11   f <= a + b;
12 end example;
```

Electronique numérique

Initiation à VHDL (2/3)

Circuits séquentiels. Automates.

Le TD précédent nous a permis de prendre en main VHDL, à travers des descriptions structurelles (instanciation de composants), ainsi que des équations logiques (assignations concurrentes). Nous poursuivons ici cette découverte du langage : nous allons notamment décrire des bascule D à l'aide du langage, et ainsi construire notre premier circuit séquentiel (un datapath simple). Nous allons également découvrir deux manières de décrire des automates en VHDL :

- En transposant directement les équations logiques des automates.
- Puis en reposant sur les capacités de synthèse du langage à un niveau plus élevé : le niveau RTL.

1 Décrire des bascules D

Nous avons vu précédemment que VHDL nous offre le moyen direct de décrire des équations logiques à l'aide d'assignations concurrentes. La partie droite d'une assignation concurrente peut faire appel à des expressions à deux opérandes (expressions dites "binaires"...) comme le **and**,**or**,**not**,**nor** etc...Ces opérations font partie intégrante du langage : ce sont des mots clés de VHDL. La correspondance avec nos portes logiques est donc immédiate. Malheureusement, ce n'est pas le cas pour la bascule D : il n'existe pas de mot clé pour décrire une telle bascule, de manière directe. Pour décrire les bascules D, il est obligatoire de recourir à des **process**, au sein d'une **architecture**.

```
1 entity a_circuit is
2  port(
3     reset_n : in std_logic;
4     clk : in std_logic;
5     din : in std_logic;
6     qout : out std_logic
7  );
8 end entity;
9
```

```
10 architecture example of a_circuit is
11 begin
12
13   -- ...
14
15   bascule_d: process(reset_n,clk)
16   begin
17     if reset_n='0' then
18       qout <= '0';
19     elsif rising_edge(clk) then
20       qout <= din;
21     end if;
22   end process;
23
24   -- etc
25   -- autres codes...
26
27 end example;
```

L'exemple précédent décrit une (seule) bascule D. On retiendra le gabarit de code qui conduit à créer (inférer est le terme exact) la bascule D. Quelques remarques s'imposent :

- **Toute assignation concurrente sous le contrôle d'un front d'horloge génère une (ou plusieurs) bascule(s) D.**
- Le nommage de l'entrée D et de la sortie Q n'est pas obligatoire : on peut utiliser n'importe quels noms de signaux (pourvu qu'ils soient déclarés). Dans le jargon, on parlera alors de signaux "clockés".
- on peut utiliser n'importe quel *type* de signaux : **std_logic**, **std_logic_vector(63 downto 0)**, **signed(31 downto 0)**, **unsigned(7 downto 0)**, mais également des types créés par l'utilisateur.
- Le caractère prioritaire du reset asynchrone (prioritaire par rapport à l'horloge) est ici manifeste dans le code (if ... elsif...).
- Le front montant se nomme ... **rising_edge**.
- On peut coder plusieurs bascules D, à l'aide d'un seul processus, ou alors créer plusieurs processus. C'est au choix.

2 Descriptions de chemin de données séquentiel

Une relecture de l'énoncé du TD précédent rappelle que les opérateurs arithmétiques traditionnels sont crées automatiquement grâce aux symboles "+","-" etc, dès lors que les données manipulées sont typées en **signed** (par exemple **signed(7 downto 0)**) ou **unsigned(7 downto 0)**.

Questions

1. Coder en VHDL au moins deux architectures numériques qui réalisent un compteur, qui compte de 0 à 255, en respectant l'entité donnée

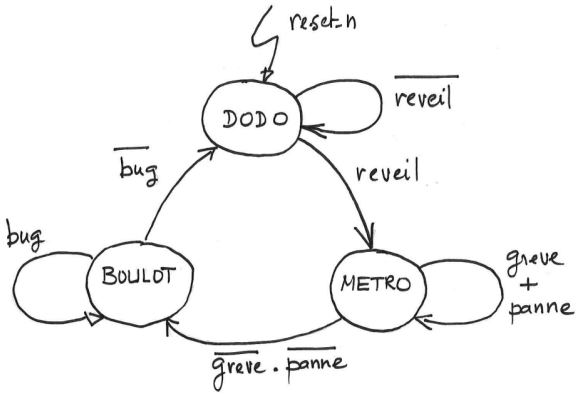


FIGURE 1 – Diagramme états-transitions de l'automate "I love Paris"

ci-dessous. On note que les opérandes sont codés en **unsigned(7 downto 0)** (octets non-signés) ; On prendra soin de dessiner le circuit avant de coder l'architecture.

2. Tester le circuit à l'aide du testbench fourni.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity compteur is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      value : out unsigned(7 downto 0)
10   );
11 end compteur;
```

3 Descriptions d'automates au niveau logique

Soit un automate bien connu des informaticiens parisiens, sur la figure 1.

Questions

1. Vérifier que, dans le cas d'un encodage one-hot, les équations d'état suivant sont bien données par :
$$\begin{cases} D_0 &= Q_0.\overline{reveil} + Q_2.\overline{bug} \\ D_1 &= Q_0.reveil + Q_1.(greve + panne) \\ D_2 &= Q_1.\overline{greve}.\overline{panne} + Q_2.bug \end{cases}$$
2. Coder l'automate "I_love_Paris" en VHDL, au niveau logique, en prenant soin de bien coder les bascules D à l'aide de processus. Respectez

l'entité fournie ici. On utilisera une assignation conditionnelle (when) pour coder le signal de sortie "up_and_running".

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity i_love_paris is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      reveil : in std_logic;
10     panne : in std_logic;
11     greve : in std_logic;
12     bug : in std_logic;
13     up_and_running : out std_logic
14   );
15 end i_love_paris;
```

3. Utilisez le testbench donné sous Moodle pour tester votre automate. Observez le résultat.

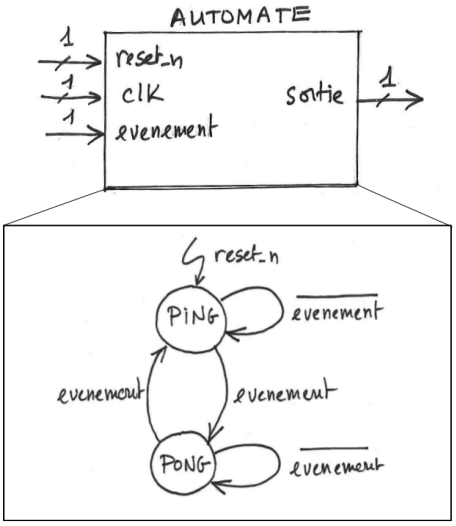
4 Descriptions d'automates au niveau RTL

Ce niveau d'abstraction RTL est le plus couramment utilisé, car il permet de s'affranchir des détails des équations logiques, et décrire des automates (et *micro-architectures*) de manière plus naturelle. Le niveau RTL repose sur la notion d'*inférence matérielle* : l'Electronicien doit connaître certains *motifs de conception*[1], afin de permettre à l'outil de synthèse d'établir automatiquement les équations logiques sous-jacentes. Le cas des automates d'états finis est instructif en ce sens. Voici un exemple de codage VHDL, qui décrit un automate, sans expliciter les équations logiques sous-jacentes, ni l'encodage des états. Pour information, le synthétiseur peut choisir de lui-même un encodage qui maximise la fréquence de fonctionnement du circuit (encodage one-hot généralement), ou tout autre type de contraintes imposées par le concepteur.

Exemple d'automate de niveau RTL Nous donnons ici à titre d'exemple un diagramme états-transitions, ainsi que son code VHDL de niveau RTL. Notez que nous avons séparé le processus ici appelé "next_state_p", qui code la fonction de transition (calcul de l'état suivant), et la sortie de l'automate.

1. "Design patterns", en anglais.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3  use ieee.numeric_std.all;
4
5  entity automate is
6    port(
7      reset_n : in std_logic;
8      clk : in std_logic;
9      evenement : in std_logic;
10     sortie : out std_logic
11   );
12 end automate;
13
14 architecture rtl of automate is
15
16   type state_type is (PING,PONG);
17   signal state, next_state : state_type;
18
19 begin
20   process(reset_n,clk)
21   begin
22     if reset_n='0' then
23       state <= PING;
24     elsif rising_edge(clk) then
25       state <= next_state;
26     end if;
27   end process;
28
29   -- logique d'etat suivant
30   next_state_p: process(state,evenement)
31   begin
32     next_state <= state;--default
33     case state is
34       when PING =>
35        if evenement='1' then
36          next_state <= PONG;
37        end if;
38       when PONG =>
39        if evenement='1' then
40          next_state <= PING;
41        end if;
42       when others =>
43         null;
44     end case;
45   end process;
46
47   -- logique de sortie
48   sortie <='1' when state=PONG else '0';
49
50 end rtl;
```



Questions

1. Prenez le temps de comprendre le codage utilisé ici.
2. Coder l'automate "I_love_Paris" en VHDL, au niveau RTL, en vous inspirant de l'exemple donné.
3. Utilisez le même testbench que précédemment pour tester votre automate. Modifier uniquement le nom de l'architecture associée à l'entité instanciée.