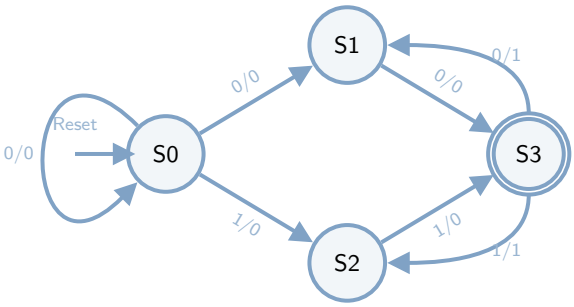
Introduction à l' Electronique

NUMÉRIQUE

Travaux Dirigés – 1ère année

Jean-Christophe Le Lann

Enoncés des TD



Année académique 2026-2027

Version 2.0

TD 1 – Électronique Numérique

Représentation des nombres et Logique combinatoire

Notes et Conseils

- Le premier objectif de ce TD est de manipuler différentes représentations des *nombres* et notamment de maîtriser les changements de base **sans calculatrice**.
- En dehors du TD, il est suggéré de vérifier que vous êtes suffisamment adroit pour les réaliser également *avec* calculatrice ou à l'aide de votre langage de programmation préféré.
- Le second objectif est de se familiariser avec l'algèbre de Boole.
- *Note : Tous les exercices ne seront pas nécessairement corrigés lors des séances de TD.*

Première partie

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 de cette partie du TD)*
2. Écrire les puissances 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 procédé similaire, trouver une valeur approchée de $\log_{10} 3$.
3. Calculer la valeur décimale de 10101011_2 en passant par les puissances de 2, puis en passant par les puissances de 16.
4. Écrire les puissances de 2^{-n} pour n variant de 1 à 4.

Remarque : Des nombres possédant une partie décimale sont transformés ici en de simples *entiers*, en supposant que la position de la virgule est fixée à l'avance. Cette représentation est appelée **virgule fixe** (*fixed-point*). Elle est très rapide à traiter par l'électronique, à coût matériel très faible, et est très utilisée en traitement du signal embarqué (DSP). À l'inverse, une représentation en **virgule flottante** est plus consommatrice de ressources (surface) et de temps, mais offre une plage dynamique et une précision beaucoup plus importantes (norme IEEE 754 traitée dans l'exercice 4).

2 Conversions entre bases

1. Convertir le nombre 39_{10} en binaire.
2. Convertir le nombre $(0,375)_{10}$ en binaire.
3. En déduire la représentation binaire du nombre $(39,125)_{10}$.
4. Convertir le nombre $(0,2)_{10}$ en binaire. Que remarquez-vous ?

5. Convertir $(345,158)_{10}$ en octal.
6. Convertir $(389)_{10}$ en hexadécimal.
7. Convertir 011001111011101_2 en hexadécimal.
8. Convertir $(10110001101011,1111)_2$ en octal.
9. Convertir $(10110001101011,111101)_2$ en hexadécimal.
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 en complément à 2

Rappels

Jusqu'ici, nous nous sommes intéressés aux nombres positifs. Il existe plusieurs manières de représenter un nombre négatif n. La manière la plus intuitive serait de dédier un bit (le plus à gauche) pour préciser le signe (ex : 0 pour +, 1 pour −). Malheureusement, cette représentation possède deux défauts majeurs : il existe deux représentations du zéro (+0 et −0) et 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 et logiques* (ALU) des ordinateurs.

Avec k bits, on peut représenter l'ensemble des nombres entiers dans l'intervalle :

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

Le complément à 2 d'un entier n est obtenu par le calcul de la quantité $2^k - n$. On peut 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 k 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, sa valeur décimale est :

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

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 formule (1) et vérifier le résultat en appliquant la formule (2).
3. Soustraire : $91_{10} - 108_{10}$ en passant par le complément à 2.
4. Écrire les nombres suivants en binaire sur 6 bits : $+24, +31, -24, -31$. Passez ensuite à 7 bits. Que remarquez-vous ?
5. Soustraire : $1100\ 0101\ 0100\ 0000_2 - 101000\ 1111\ 1111_2 - 110\ 1111\ 1110_2 - 10\ 1110_2$.

4 Norme IEEE 754

Rappel : norme IEEE 754 (simple précision, 32 bits) Un flottant est codé par trois champs : 1 bit de signe s, 8 bits d'exposant e et 23 bits de mantisse m. La valeur représentée est, pour la plupart des nombres :

$$V = (-1)^s \times 2^{e-127} \times (1+m)$$

où $m = \sum_{i=1}^{23} b_i 2^{-i}$ est la partie fractionnaire. L'exposant est représenté en excédent (biais) à 127. La mantisse est toujours normalisée avec un 1 implicite (gain de place), sauf pour les cas particuliers suivants :

- si $e = 255$ et $m = 0$, alors $V = \pm\infty$ (selon s) ;
- si $e = 255$ et $m \neq 0$, alors $V =$ `NaN` (Not a Number) ;
- si $e = 0$ et $m = 0$, alors $V = \pm 0$;
- si $e = 0$ et $m \neq 0$, il s'agit d'un nombre dénormalisé, dont la valeur est $V = (-1)^s \times 2^{-126} \times m$.

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

Deuxième partie
Algèbre de Boole et Logique combinatoire

Nous commençons par quelques manipulation élementaires autour de la minimisation d'expressions booléennes et leur mise en forme canonique, à l'aide de l'algèbre de Boole. Nous illustrons également la méthode des tableaux de Karnaugh qu'il faut voir comme une technique tabulée de minimisation astucieuse qu'il est intéressant de connaître pour de petits circuits. Nous poursuivons par la construction d'opérateurs matériels élémentaires : additionneurs, multiplexeurs, comparateurs.

5 Quelques formulations booléennes

Écrire sous la première forme canonique (somme de mintermes) 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.

6 Minimisation d'expression booléennes

Simplifier algébriquement les fonctions suivantes :

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

7 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.

7.1 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 (donc care X).

7.2 K-map à 4 variables

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

8 Équations logiques de circuits de base

8.1 Additionneur

1. Établir 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 s et la retenue sortante c_o. Dessiner le circuit.
2. Établir la table de vérité d'un *additionneur complet* pour deux opérandes 1 bit. Ce dernier tient désormais compte d'une retenue entrante c_i. Établir 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.
6. Peut-on imaginer d'anticiper le calcul de la retenue ? Si oui, comment ?

8.2 Multiplexeur

La fonction 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 1 bit). On parle de multiplexeur "2 vers 1".
2. Calculer l'équation d'un multiplexeur "2 vers 1" pour des entrées codées sur 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 $n = 2$.
4. Etablir la structure du multiplexeur à m entrées, à l'aide de multiplexeurs.
5. Calculer le nombre de couches logiques traversées pour un multiplexeur m, n ("m vers 1" et dont les données sont codées sur n bits).

8.3 Comparateur

Soit deux données a et b quelconques, encodées par deux variables booléennes générales $(a_{n-1}, \ldots, a_0) \in \mathcal{B}^n$ et $(b_{n-1}, \ldots, b_0) \in \mathcal{B}^n$.

1. Démontrer que le résultat booléen f de comparaison d'égalité entre les deux données vaut :
$$f(a,b) = \prod_{i=0}^{i=n} \overline{a_i \oplus b_i}$$
2. Dessiner la structure du comparateur matériel pour $n = 4$.

9 Formes canoniques

Prérequis : Codage binaire et indexation des combinaisons booléennes
Pour un système à n variables logiques ordonnées, il existe 2^n combinaisons possibles de valeurs. Chaque combinaison peut être interprétée comme un nombre binaire, ce qui permet de lui associer de manière unique un indice entier compris entre 0 et $2^n - 1$. Par exemple, pour trois variables ordonnées (x, y, z), la combinaison de vérités $(1, 0, 1)$ est associée au mot binaire 101_2, qui correspond à l'indice décimal 5 : ce dernier pourra être évoqué comme le monôme m_5. Ce formalisme d'indexation numérique standardise la désignation de n'importe quel état du système et s'applique de manière générale en logique combinatoire, notamment pour l'organisation des tables de vérité, le référencement des cases d'un tableau de Karnaugh, etc.

Formes canoniques et passage de l'une à l'autre
La **première forme canonique** (ou Forme Normale Disjonctive, FND) est une somme de produits appelés mintermes (m_i), où chaque m_i est le produit logique associé à la combinaison d'indice i pour laquelle la fonction vaut 1. À l'inverse, la **deuxième forme canonique** (ou Forme Normale Conjonctive, FNC) est un produit de sommes appelées maxterms (M_i), où chaque M_i est la somme logique associée à la combinaison d'indice i pour laquelle la fonction vaut 0. Le passage d'une forme à l'autre exploite directement la complémentarité des indices au sein de l'ensemble des 2^n combinaisons. La fonction inverse $\bar{f}$ étant constituée des mintermes dont les indices sont absents de f, l'application de la double négation $(f = \overline{\bar{f}})$ et des lois de De Morgan permet de transformer la somme de ces mintermes manquants en un produit de maxterms de mêmes indices :

$$\begin{aligned} f(x,y,z) &= m_1 + m_3 + m_5 \quad \text{(la fonction vaut 1 pour les indices 1, 3 et 5)} \\ \bar{f}(x,y,z) &= m_0 + m_2 + m_4 + m_6 + m_7 \quad \text{(indices restants parmi les } 2^3 \text{ possibles)} \\ f(x,y,z) = \overline{\bar{f}(x,y,z)} &= M_0 \cdot M_2 \cdot M_4 \cdot M_6 \cdot M_7 \quad \text{(par passage au dual via De Morgan)} \end{aligned}$$

Ainsi, commuter entre les deux formes canoniques revient simplement à inverser la nature de l'opérateur principal (somme ou produit) en basculant sur l'ensemble complémentaire des indices de l'univers des combinaisons. *Note : Ces formes ne sont pas simplifiées.*

Établir les deux formes canoniques des fonctions suivantes :

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

10 Vu-mètre

On se propose de réaliser un “contrôleur de 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 en fonction de la grandeur.

1. Schématiser le principe d’ensemble dans le cas où la grandeur à visualiser est codée sur 3 bits.
2. À partir du tableau de Karnaugh, trouver une réalisation possible.
3. On suppose que l’on dispose désormais d’un composant complexe de décodage (un décodeur 3 vers 8). Rappeler la table de vérité de ce circuit. Utiliser alors ce décodeur pour proposer une nouvelle réalisation.

11 Le jeu des cambrioleurs

Le principe du jeu [1] 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 d’indiquer le nombre de niveaux franchis correctement.
2. Sachant que l’on peut envoyer 1 million de codes par seconde, 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 des codes sur 5 bits.

Cet exercice ne nécessite pas de formules complexes, mais une réflexion architecturale.

1. Le jeu est inspiré du jeu "Dix de Chute" des années 70.

TD 2 – Électronique Numérique

Logique Séquentielle

Notes et Conseils

L'objectif de ce TD est la découverte des circuits *séquentiels*. Nous commençons par découvrir sa brique fondamentale : la *bascule D*. Elle permet de mémoriser des signaux à des instants discrets. A travers différents exercices (dont l'un de *reverse engineering*), nous cherchons à appréhender la manière de concevoir différents circuits séquentiels : ce temps discret nécessite de penser en terme de *suites numériques*, en raisonnant sur l'instant présent et l'instant passé.

Première partie

Bascule D et circuits séquentiels élémentaires

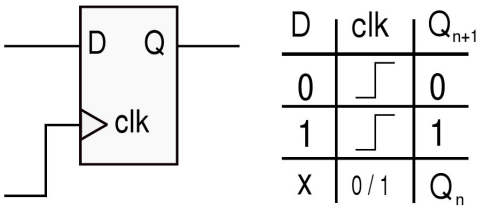
1 Découverte de la bascule D

Le paradigme dominant en conception de circuits numérique est le paradigme *synchrone* et *mono-horloge* : l'ingénieur organise sa réflexion et sa méthodologie de conception en reposant sur l'idée de la délivrance d'un *tempo* externe, parfait et régulier. Dans les faits, cet *tempo* est délivré par un signal carré d'horloge. Il existe certes des difficultés techniques à cette délivrance (*clock skew* notamment), mais le principe de *séparation des préoccupations* lui permet de les négliger.

Le circuit élémentaire sensible à l'horloge, et qui va permettre de concevoir des circuits séquentiels est la **bascule D**. C'est la *seule bascule qui va nous préoccuper ici*. C'est l'élément qui va nous permettre de distinguer *l'avant* de *l'après*. Dans votre esprit, il ne doit pas y avoir d'autre.

Toutefois, tout comme en Physique l'*atome* n'est finalement pas la brique *insécable* qu'on croyait, la bascule D possède bien évidemment une constitution interne : elle est souvent présentée comme le résultat d'un assemblage savant autour d'une autre bascule *asynchrone* cette fois, appelé *latch (verrou) RS*. En réalité, il existe beaucoup d'autres architectures VLSI (intégration à très grande échelle), qui utilisent des principes physiques différents pour mémoriser l'état. Nous prenons l'option pédagogique de nous concentrer uniquement sur la bascule D ici.

Le symbole de la bascule D, ainsi que sa table de vérité, sont rappelés ci-dessous.

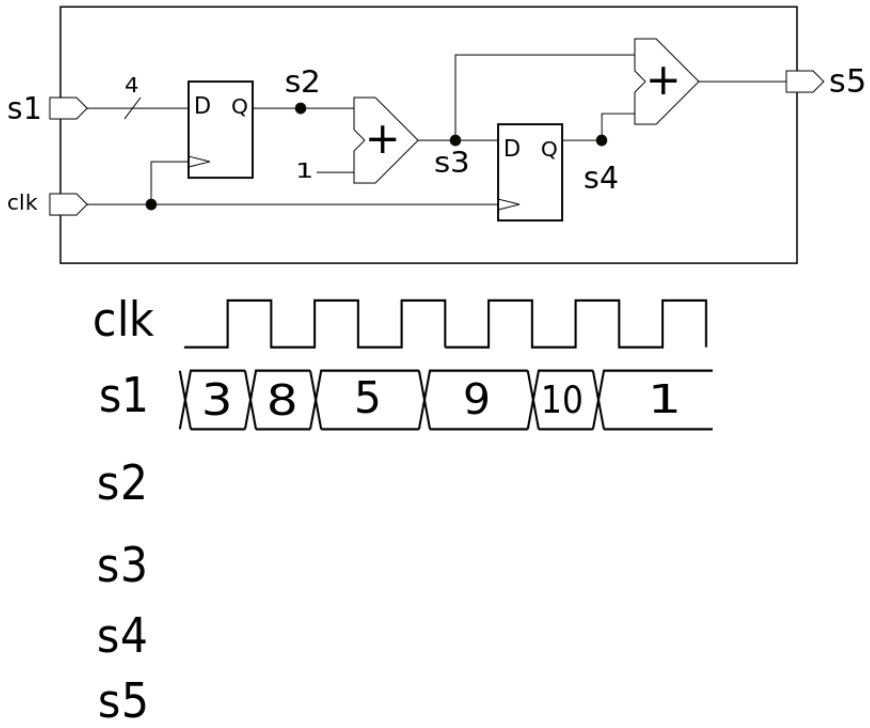


La bascule D a pour fonction de recopier l'entrée D sur sa sortie Q, lors d'un court temps d'échantillonnage : il s'agit généralement du **front montant** de l'horloge. Cette horloge est supposée périodique. Pour que l'échantillonnage se passe correctement *d'un point de vue physique*, le signal d'entrée doit respecter deux contraintes : il doit être parfaitement stable *un peu avant* et *un peu 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'horloge absorbe ces variations : c'est l'*hypothèse synchrone*. 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" : c'est un des nombreux mécanismes d'*abstraction* rencontrés en conception de circuits numériques.

Exercice : Compléter le chronogramme suivant où un signal S_1 sur 4 bits, extérieur au système et asynchrone, 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 $t_{clk \to Q}$;

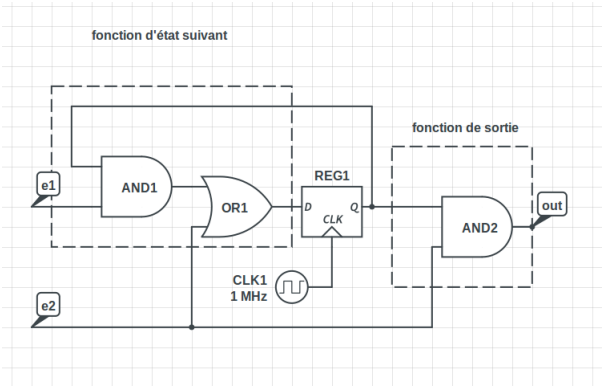
— les zones de transition des 4 bits se présentent ici comme des “X” ;

— 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 $\phi = \frac{1+\sqrt{5}}{2} \approx 1{,}618$. En effet, Euler a démontré que pour un n donné (un "step"), on a :

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

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".

Deuxième partie
Chemin de données ou *datapath*

Nous avons déjà abordé la notion de "chemin de données" dans le "jeu des cambrioleurs" (TD1) : il s'agissait uniquement d'acheminer, en combinatoire, une donnée d'une entrée vers une sortie, sans traitement spécifique. En général, les *datapaths* sont plus complexes et associent à la fois logique séquentielle et logique combinatoire. Ils constituent le coeur des architectures de calculs.

L'essentiel de leur puissance de calcul provient de deux notions fondamentales :

— **Le parallélisme** : il s'agit d'effectuer plusieurs opérations durant un 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.

5 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 dispositif 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 :

$$\ldots, 0, 0, 0, 0, 8, 9, 13, 4, 3, 2, 1, 0, 0, 0, \ldots$$

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

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

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 $xor(a, b)$.

TD 3 – Électronique Numérique

Automates ou Machines d'états finis

Notes et Conseils

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

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

Les automates peuvent être vus comme l'outil conceptuel équivalent aux équations différentielles du monde analogique et continu : ces automates permettent de décrire l'évolution temporelle d'un système. Ils sont donc fondamentaux, non seulement pour la conception en Electronique, mais plus généralement pour toute l'Informatique.

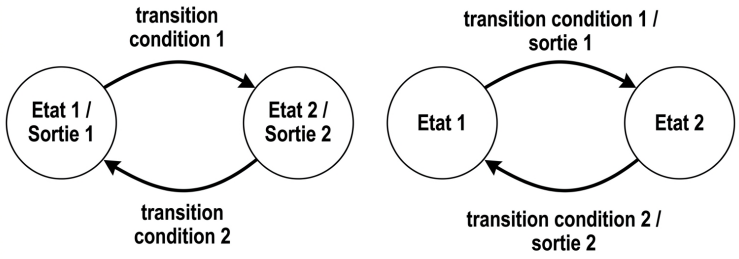
a. Ce nom quelque peu étrange signifie que le *nombre* d'états est *fini*. Il existe également des automates dont le nombre d'états est infini...

1 Diagramme états-transitions : Moore et Mealy

On rappelle que le diagramme *états-transitions* (ou *state transition diagram*, aussi appelé *diagramme à bulles*) est une représentation graphique abstraite du comportement d'un automate. Cette représentation 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 . 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).



(a) FSM de Moore et Mealy. Attention ! Les deux automates présentés ici ne sont ici pas équivalents en terme de comportement ! Voir exercice)

2 Causalité d'une FSM

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

— **Réactivité** (exhaustivité) : 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.

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

Une autre manière de résumer ces deux propriétés 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.

Questions

1. Observez la figure 1.Vérifier la consistance de la machine d'état suivante.
2. Est-ce un automate de Moore ou de Mealy ?

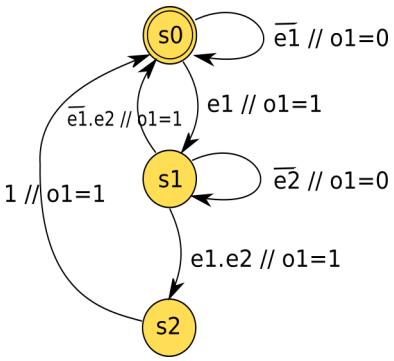


FIGURE 1 – FSM de l'exercice 1 et 2

3 Du diagramme états-transitions au circuit numérique.

Nous cherchons ici à réaliser l'électronique de l'automate de l'exercice précédent.

Notes et Conseils

La méthode manuelle consiste à décrire tout d'abord dans un tableau :
— L'évolution des états en fonction des entrées, en conservant leur nom *symbolique*.
— L'évolution des sorties en fonctions des états, et des entrées (cas de Mealy).

Après **choix d'un encodage binaire des états**, on connaît le nombre n de bascules D impliquées :
— On réécrit le tableau précédent pour chacune des bascules D_i.
— On déduit les équations $D_i = f(Q_0, Q_1, \ldots, Q_{n-1}, e_0, \ldots)$, où Q_j représente l'état courant de la bascule j et $\{e_k\}$ l'ensemble des entrées du circuit.

On rappelle qu'il existe un nombre infini de possibilité d'encodage des états. Cet encodage influe sur le nombre de bascules D nécessaires au stockage des états, mais également sur la

complexité de la logique combinatoire de la fonction de transitions et de la fonction de sortie (et donc de la rapidité du circuit).

On s'appuie sur les tableaux génériques suivants 1 et 2, qui permettent de procéder de manière systématique.

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 –ici sur 2 bits)

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

Questions

- Etablir le tableau d'évolution des états symboliques et de la sortie.
- Choisir l'encodage *dense* le plus naturel.
- Etablir le tableau d'évolution des états encodés et de la sortie.
- Etablir les équations de chacun des bits d'états.
- Etablir les équations de chacun des bits de sortie.
- Dessiner le circuit à l'aide portes logiques élémentaires.

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 connaît une certaine signature numérique (travail de filtrage d'un proxy, etc.).
- L'analyse de flux multimédia : 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 reconnaissance de séquence du génome, etc.

4.1 Détecteur de séquence `10` : Moore versus Mealy

Dans le cadre de la conception d'un sous-système de communication série, on souhaite réaliser un détecteur de séquence capable de repérer le motif binaire **"10"** sur un flux de données synchronisé par une horloge.

Le système possède :

- Une entrée binaire synchrone X (le bit courant),
- Une sortie binaire synchrone Z qui vaut **1** si et seulement si la séquence **"10"** vient d'être détectée sur l'entrée X, et **0** sinon.

On considère que les séquences peuvent se chevaucher (par exemple, dans la séquence `1 1 0`, le motif **"10"** est détecté sur les deux derniers bits).

1. **Automate de Mealy.** Concevoir un automate de type **Mealy** réalisant cette fonction. On donnera :

— la description des états et leur signification,
— la table des transitions (format : *état suivant / sortie*).

2. **Automate de Moore.** Concevoir un automate de type **Moore** ayant un comportement fonctionnel équivalent. On donnera :
   — la description des états, leur signification et la sortie associée,
   — la table des transitions.
3. **Comparaison temporelle.** Relever les valeurs de la sortie Z pour les deux automates sur la séquence d'entrée suivante, appliquée à chaque front montant de l'horloge :

   $$X = 0,\ 1,\ 0,\ 0,\ 1,\ 1,\ 0$$

   On supposera que les deux automates sont initialisés dans leur état de départ avant le premier front.
4. **Analyse.** Les deux automates sont-ils strictement équivalents ? Justifier en précisant la différence fondamentale entre les machines de Moore et de Mealy, et expliquer le phénomène observé à la question 3.

4.2 Détecteur de séquence `0110`

On considère le détecteur de séquence de la figure 2. 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 !

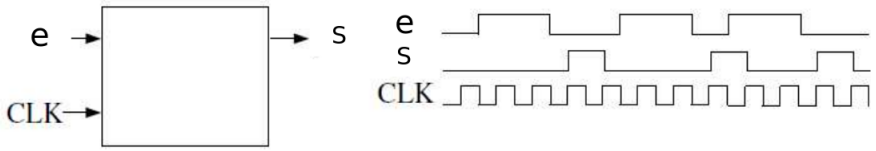


FIGURE 2 – Schéma de principe du détecteur de séquence `0110`. Chronogramme associé.

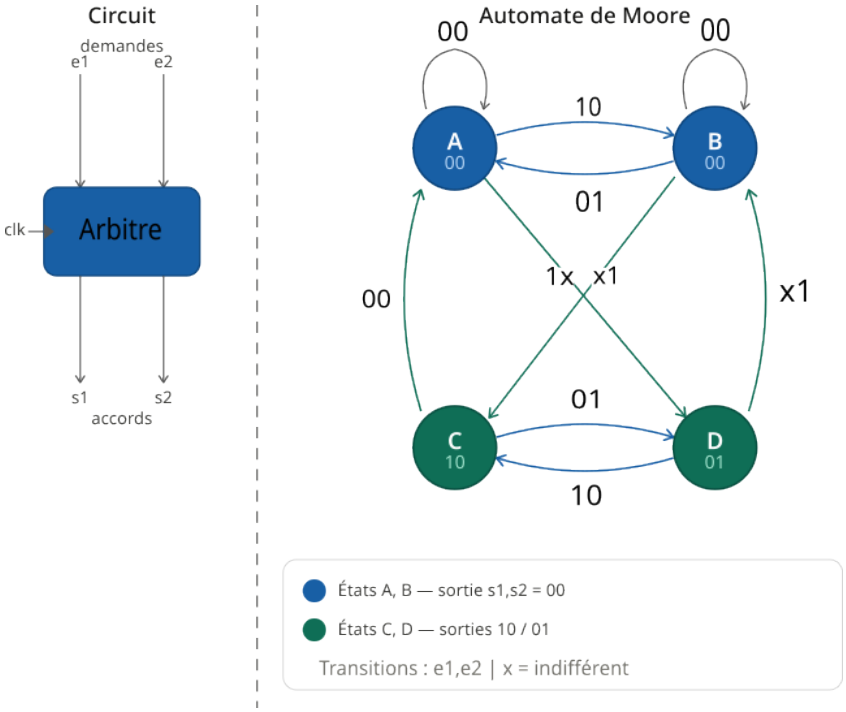
1. Proposer un diagramme états-transitions pour ce détecteur. On choisira une machine de Moore.
2. Proposer un encodage dense des états.
3. Réaliser le circuit.

5 Arbitre à priorité tournante (round-robin)

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.

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

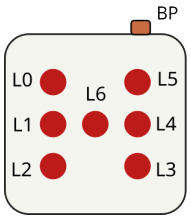
1. Proposer un encodage des états tel que chaque état soit codé sur 2 bits
2. Etablir la fonction de transition du système sous forme de table de vérité. Simplifier.
3. Etablir la fonction de sortie du système. Simplifier.
4. Dessiner le circuit correspondant. Combien de portes faut-il ?

6 Problème du dé électronique. Encodage des états spécifique.

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 sur 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.

On choisit d'utiliser un encodage des états particulier, *spécifique au problème* : la sortie de chaque bascule $Q_i, i \in 0..6$ pilote directement la led L_i correspondante. La fonction de sortie de l'automate est ainsi la *fonction identité*.

Question

1. Réaliser le tableau décrivant l'évolution de chaque bascule lorsque le dé est sollicité (bouton poussoir appuyé).
2. En déduire les équations constitutives du circuit correspondant.
3. Dessiner le circuit d'une ou deux de ces équations.
4. Rappeler succintement le schéma-blocs composé de 2 blocs : la logique d'état suivant et les bascules D (registres).
5. Choisir un autre encodage (par exemple one-hot) et réaliser à nouveau le circuit.

7 La magie de l'encodage "*one-hot*" ou "*un bit par état*"

Une machine distribue des canettes à 1€. Elle n'accepte que des pièces de 0,50€ et de 1€. Elle ne rend pas la monnaie.

On précise quelques éléments :
— **Entrées :** 'P50' (pièce de 50c) et 'P100' (pièce de 1€). Les deux entrées ne sont jamais actives en même temps.
— **Sorties :** 'DIST' (Distribuer la canette) et 'REND' (Rendre 0,50€ de monnaie - uniquement si le client a donné 1,50€).
— **Fonctionnement :**
  — Si somme totale atteint **1,00€** → 'DIST=1'.
  — Si somme totale atteint **1,50€** → 'DIST=1' ET 'REND=1' (car il a donné 1,50€ pour une canette à 1€).

Questions

1. Déssiner le diagramme états-transitions en conservant le nom des états symboliques.
2. Réaliser l'encodage *one-hot*.
3. Etablir les équations D_i de chacune des bascules mises en jeu.

8 Automate *serrure numérique*. Le problème des *spécifications*.

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**.

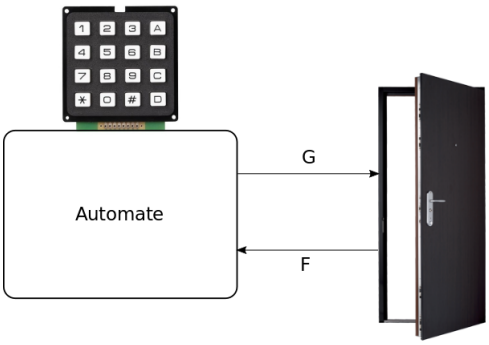


FIGURE 3 – Schéma de principe de la a serrure numérique.

Un capteur permet de connaître l'état de la porte : F indique son état. F vaut 1 si la porte est Fermée, et 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 $G = 0$.
À 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 dispositif.
2. Discuter de la robustesse de la solution, et de son réalisme. Proposer des alternatives.
3. Optionnel : construire le circuit complet.

TD 4 – Électronique Numérique

Initiation à VHDL

Notes et Conseils

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 pratiques :

- Nous travaillerons sous Linux.
- VHDL n'est pas associé à un éditeur particulier. Vous pouvez utiliser gEDIT, vim, Emacs, Atom, etc.
- La compilation se fera en ligne de commande Linux : `ghdl -a code.vhd`

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 :

- **std_logic_1164** : ce package définit entre autres les types les plus usités.
  - **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 **rising_edge** (et **falling_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.

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

— 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 :

```
library IEEE; -- appel a la bibliotheque compilee
use IEEE.std_logic_1164.all; -- on declare utiliser tous les elements du package.
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 sorties. Ces ports possèdent également un nom, une direction (**in** ou **out** dans notre cas) et un type.

```
entity MonCircuit is
port (
    reset_n : in std_logic;
    horloge : in std_logic;
    e1 : in std_logic;
    e2 : in std_logic;
    e3 : in std_logic_vector(3 downto 0);
    e4,e5 : in std_logic_vector(3 downto 0);
    o1 : out std_logic;
    o2 : out std_logic_vector(3 downto 0);
    o3 : out std_logic_vector(3 downto 0) -- pas de ';' pour le dernier port
); -- noter le ; ici
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

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

```
architecture nom_architecture of nom_entity is
    -- declarations diverses : signaux, types, fonctions,...
begin
    -- corps de l'architecture
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. À titre d'exemple, on présente ici quelques unes des possibilités du langage (restreintes à l'UE 1.2), à travers un exemple purement fictif :

```vhdl
architecture nom_architecture of MonCircuit is
    signal s1,s2,s3 : std_logic; -- declaration de 3 "fils" d'un bit chacun.
    signal v32 : std_logic_vector(31 downto 0); -- une "nappe" ou "bus" de 32 bits
    signal cout : std_logic;
begin
    -- *** assignations concurrentes ***
    s1 <= (e1 or not(e2)) xor e3; -- formule utilisant les ports d'I/O
    s2 <= s1 and e3;
    v32(1) <= s2; -- on connecte le signal s2 a un des fils de la nappe.
    o3 <= '1' & v32(2 downto 0); -- constitution du port de sortie o3.

    -- *** assignations conditionnelles ***
    s3 <= s1 when s2='0' else e1;

    -- instanciation d'entites :
    inst1 : entity work.half_adder(logic)
    port map(
        clk => horloge, -- repris de l'entite
        a => e1,
        b => s1,
        cout => cout, -- signal "local"
        sum => o1 -- connecte au port de sortie de l'entite
    );
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 pré-existant, puis le connecter à des fils d'entrées-sorties par des signaux. On peut instancier autant de composants qu'on le souhaite. La 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 permettront 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'architecture du banc de test, il inclut donc :

— le circuit VHDL à tester (souvent labellisé `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 séquentielle. 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 plateformes 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
```

À 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 faisant 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 multiple de composants peut se révéler fastidieuse : imaginez un additionneur 64 bits ! Il faudrait instancier 64 fois les additionneurs 1 bit ! 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 décrite 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 connaît les équations aussi bien que vous !

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity test is
end test;

architecture example of test is
    signal a,b,f : signed(7 downto 0);
begin
    f <= a + b;
end example;
```

TD 9 – Électronique Numérique

Conception d'un accélérateur matériel pour le calcul d'un polynôme

Notes et Conseils

On se propose ici de concevoir un accélérateur matériel, dédié au calcul de polynômes de rang 3. L'étude commence par une réflexion algorithmique, qui suggère des optimisations. Nous poursuivons la démarche par la manipulation abstraite de cet algorithme, à l'aide de *graphes*. L'accélérateur conçu par la suite se présente comme une microarchitecture, composée classiquement d'un automate de contrôle et d'un chemin de données. On se limitera à des valeurs de x entières, codées sur 32 bits.

1 Réflexion algorithmique

La conception d'un accélérateur matériel naît souvent de la création d'un algorithme aux performances théoriques intéressantes (ex : compression par Transformée en Cosinus Discret), ou par la découverte d'une reformulation intéressante (ex : multiplication de Karatsuba [1]). Nous allons ici, en miniature, suivre cette démarche.

On considère un polynôme d'ordre 3 :

$$P_3(x) = a_3x^3 + a_2x^2 + a_1x + a_0 \tag{1}$$

Questions

- **Question 1 :** Combien de multiplications et d'additions sont-elles nécessaires ?
- **Question 2 :** À l'aide du schéma de Hörner, normalement connu, montrer que l'on peut réduire le nombre de ces opérations.

2 Flot de données

L'étude du flot de données (ou DFG : *dataflow graph*) permet d'établir les dépendances du calcul. On représente généralement ces dépendances à l'aide d'un graphe orienté : chaque nœud représente une opération (ici arithmétique) ou une entrée ou une sortie. Chaque arc entre deux nœuds est étiqueté par le nom de la variable qui circule du nœud source au nœud destination. Dans notre cas, le graphe est acyclique. Il indique clairement l'enchaînement des opérations, mais il s'agit encore d'une vision abstraite du calcul, indépendante de choix d'architecture matérielle. On peut toutefois noter que ce DFG peut être vu comme une première tentative de réalisation purement combinatoire du circuit (mais son chemin critique est très certainement rédhibitoire).

Questions

Question 3 : Établir le graphe de flot de données issu du schéma de Hörner.

1. Nous vous invitons à vous renseigner sur ces deux curiosités mathématiques

3 Ordonnancement. Assignation et partage des ressources

La notion d'**ordonnancement** permet de découper le DFG en différentes étapes de calculs, appelées **control-step** (CSTEP). Prosaïquement, cela revient à dessiner différentes lignes qui représentent effectivement cette découpe du DFG.

Chacune des étapes s'exécute en un seul cycle d'horloge. Selon la durée physique du cycle d'horloge, il est toutefois possible de réaliser des découpes du DFG plus ou moins "longues" : plus le cycle est long, plus il est possible d'envisager de **chaîner** un ensemble d'opérations. À la fin du cycle, on devra veiller à échantillonner les données produites : formulé autrement, on peut dire que la découpe en CSTEPs détermine où devront se trouver les *registres intermédiaires* de calcul.

En général, les opérateurs de calculs et les registres de calcul représentent la majeure partie de la surface occupée sur silicium. Il est souvent souhaitable de les utiliser avec parcimonie et notamment de les partager dès que cela est possible. On réalise l'**assignation** des opérations aux opérateurs afin d'établir où se réaliseront effectivement les calculs du DFG abstrait. Bien évidemment, à la fin d'un CSTEP tous les opérateurs utilisés sont à nouveau disponibles pour le CSTEP suivant, ce qui donne des opportunités de partage de ressources entre opérations ordonnancées sur deux CSTEPs différents. Ces informations d'assignation peuvent être également annotées sur le DFG : à côté de l'opération abstraite du DFG, on peut indiquer sur quel opérateur matériel elle s'exécutera.

Avant même la phase d'assignation, le concepteur se donne par ailleurs un objectif en terme de nombre de ressources utilisées au total : c'est la phase d'**allocation de ressources**. Il peut se donner le droit d'utiliser tel nombre de multiplieurs, tel nombre d'additionneurs, tel nombre de registres, etc. Parmi les contraintes, on retrouve d'autres paramètres variés et en particulier la fréquence attendue du circuit. On se fixe ici une fréquence d'horloge de 100 MHz.

Questions

- **Question 4 :** On s'alloue 1 multiplieur et un additionneur. Le multiplieur a un temps de calcul de 7 ns et l'additionneur de 2 ns. Concernant les registres, on considère que leur temps de setup et hold sont très inférieurs, et non significatifs par rapport aux opérateurs. Proposer un ordonnancement qui vous semble judicieux. À combien de CSTEPs aboutissons-nous ?
- **Question 5 :** Réaliser l'assignation des ressources et notamment des registres.

4 Microarchitecture : automate de contrôle et datapath

Dès lors qu'un ordonnancement est choisi, il est alors aisé de décrire un automate et un chemin de donnée. L'automate permet de séquencer les différentes étapes. Il réalise en outre, pour chacune d'entre elles, les actions appropriées. À l'issue de l'assignation, ces actions reviennent à contrôler les multiplexeurs d'un chemin de données (datapath) : ces multiplexeurs servent soit à échantillonner une donnée à l'entrée d'un registre, soit à router une donnée vers un opérateur matériel.

Questions

- **Question 6 :** Dessiner l'automate de contrôle.
- **Question 7 :** Dessiner un chemin de données.
- **Question 8 :** Compléter l'automate pour indiquer les actions réalisées dans chaque état.

5 Dimensionnement des registres

Une difficulté récurrente lorsqu'on traduit un algorithme en microarchitecture réside dans le dimensionnement des signaux, c'est-à-dire la détermination du nombre de bits nécessaires à la représentation des nombres, sans perte d'information préjudiciable. On parle aussi de leur *dynamique*. Il y a perte d'information lors d'une troncature (élimination brutale de bits à un certain rang).

Questions

- **Question 9 :** Soient deux signaux s_n et $s_{n'}$ entiers codés respectivement sur n et n' bits. Déterminer le nombre de bits nécessaires à l'addition $s_n + s_{n'}$, sans perte d'information.
- **Question 10 :** Même question pour $s_n \times s_{n'}$.
- **Question 11 :** En déduire la dynamique des signaux apparaissant dans le datapath.

6 Codage VHDL

Le code VHDL de cet accélérateur est fourni sur Moodle, ainsi qu'un banc de test. La syntaxe de VHDL est complexe et nouvelle pour vous, mais vous êtes à même d'en comprendre les grandes lignes.

Questions

- **Question 12 :** Observez attentivement le code de l'accélérateur. Dessiner l'architecture générale du système. Discutez avec votre encadrant afin de bien vérifier que vous avez compris la correspondance avec vos schémas précédents.
- **Question 13 :** Observez attentivement le code du banc de test. Discutez avec votre encadrant afin de bien vérifier que vous avez compris son fonctionnement.
- **Question 14 :** La simulation ne sera pas réalisée en séance, mais une vidéo sera mise à disposition.