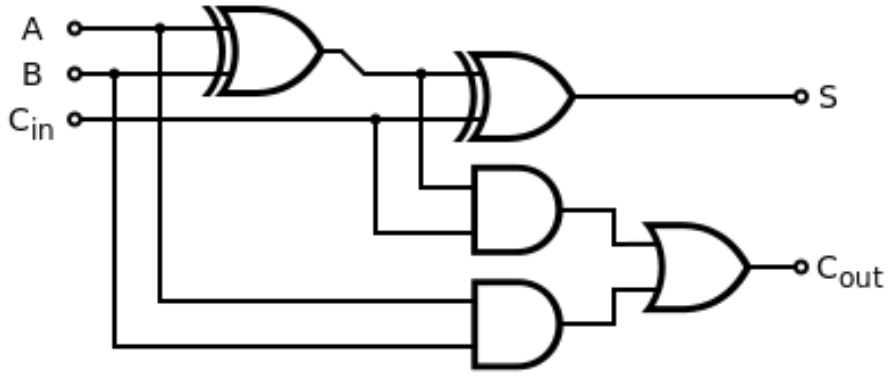
Logique combinatoire et séquentielle
EN102

Département Electronique 1ère année — Semestre 5

Fascicule de TD et TP



Contents

1 Organisation des enseignements **4**
1.1 Objectifs, déroulement et évaluation 4
1.1.1 Objectifs 4
1.1.2 Déroulement 4
1.1.3 Evaluation 5
1.2 Contenu pédagogique 5
1.2.1 Le cours 5
1.2.2 Les TD d'applications du cours 5
1.2.3 Les TD sur machine 5
1.2.4 Les Travaux Pratiques 5

2 Travaux Dirigés d'application du cours **6**
2.1 Représentation des nombres (TD1) 6
2.1.1 **Exercice 1 : Conversions de nombres entiers positifs** 6
2.1.2 **Exercice 2 : Conversions de nombre décimaux positifs** 7
2.1.3 **Exercice 3 : Numérisation d'une grandeur** 7
2.1.4 **Exercice 4 : Addition binaire** 7
2.1.5 **Exercice 5 : Soustraction binaire** 7
2.2 Fonctions logiques et algèbre de Boole (TD2) 7
2.2.1 **Exercice 6 : Table de vérité et formes canoniques** 8
2.2.2 **Exercice 7 : Formes canoniques** 8
2.2.3 **Exercice 8 : Algèbre de Boole** 8
2.2.4 **Exercice 9 : Algèbre de Boole** 8
2.2.5 **Exercice 10 : Tableaux de Karnaugh** 9
2.3 Les circuits combinatoires (TD3) 9
2.3.1 **Exercice 11 : Synthèse d'un circuit combinatoire** 9
2.3.2 **Exercice 12 : Comparateur N bits** 10
2.3.3 **Exercice 13 : Tri parallèle** 10
2.3.4 **Exercice 14 : Conception d'un circuit logique de vote** 11
2.3.5 **Exercice 15 : Conception d'un transcodeur binaire naturel /Gray** 11
2.3.6 **Exercice 16 : Conception d'un multiplieur** 11
2.3.7 **Exercice 17 : Synthèse avec des multiplexeurs** 11
2.4 Les circuits séquentiels (TD4) 12
2.4.1 **Exercice 18 : Bascule D et circuit synchrone** 12
2.4.2 **Exercice 19 : Détection de front** 13
2.4.3 **Exercice 20 : Look Up Table (LUT) à base de bascules D** 14
2.4.4 **Exercice 21 : Contrôleur d'afficheur 7 segments** 16
2.4.5 **Exercice 22 : Digicode** 17

2.4.6 **Exercice 23 : Analyse d'un Linear Feedback Shift Register (LFSR)** 17

3 Travaux Dirigés sur Machine **19**
3.1 Description, simulation et synthèse d'un Full Adder 19
3.1.1 Conception de l'architecture 19
3.1.2 Simulation de l'architecture 20
3.1.3 Synthèse automatique de l'architecture 22
3.1.4 Pour aller plus loin 23
3.2 L' additionneur 4 bits 23
3.2.1 Conception de l'architecture 23
3.2.2 Implémentation 24
3.3 Le compteur modulo 24
3.3.1 Conception de l'architecture 24
3.3.2 Implémentation 25
3.4 Machine à état finis (FSM) 25
3.4.1 Conception de l'architecture 25
3.4.2 Implémentation 26

4 Travaux Pratiques **27**
4.1 Compteur d'impulsions 27

1. Organisation des enseignements

1.1 Objectifs, déroulement et évaluation

1.1.1 Objectifs

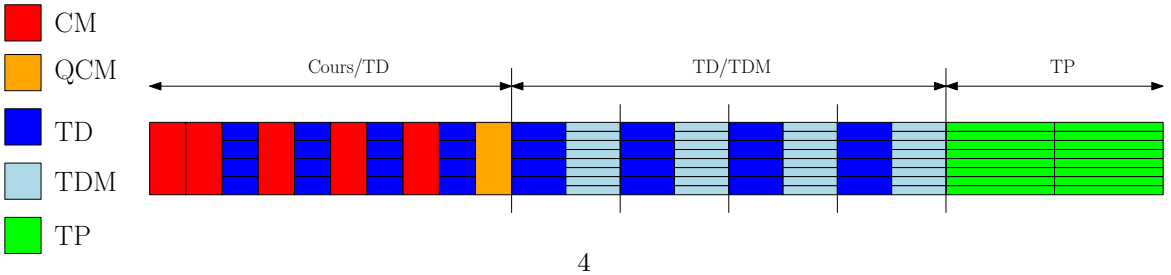
Voici quelques objectifs du module EN102:

- Maîtriser la représentation de l'information dans les bases binaires, octales et hexadécimales.
- Maîtriser l'arithmétique binaire élémentaire : complément à 2, addition, soustraction, multiplication.
- Concevoir des circuits numériques intégrant à la fois de la logique combinatoire et de la logique séquentielle synchrone.
- Analyser un circuit numérique et en déduire son comportement dans le temps (chronogramme).
- Décrire le comportement d'architectures de circuits numériques en utilisant le langage VHDL.
- Simuler le comportement d'architectures de circuits numériques.
- Synthétiser une architecture à partir de sa description comportementale.
- ...

1.1.2 Déroulement

Les enseignements se déroulent en plusieurs phases.

- 5 Cours (5 x 1h20 = 6h40)
- 4 TD d'application du cours (4 x 1h20 = 5h20)
- 1 QCM d'1h20 (algèbre de Boole et arithmétique binaire)
- 4 séances de TD/TDM (4 x 4h00 = 16h00)
- 2 TP (2 x 4h = 8h)



1.1.3 Evaluation

L'évaluation inclut

- Un QCM d'1h20 portant uniquement sur les parties 1 et 2 du cours, TD1 et TD2.
- Un examen écrit de 2h portant sur l'ensemble des cours, TD, TDM et TP.

1.2 Contenu pédagogique

1.2.1 Le cours

Le cours a pour objectif de donner les fondements théoriques et quelques clés sur les aspects pratiques afin d'aborder les TD et les TP de manière sereine.

- Partie 1 : Représentation de l'information + arithmétique binaire
- Partie 2 : Algèbre de Boole
- Partie 3 : Circuits combinatoires
- Partie 4 : Circuits séquentiels
- QCM : 1h20 portant uniquement sur les parties 1 et 2.

1.2.2 Les TD d'applications du cours

Les TD d'applications du cours permettent au travers d'exercices et d'exemples d'appronfondir les notions vues en cours.

1.2.3 Les TD sur machine

Les TD et TDM ont pour objectif de s'initier à la conception de circuits numériques en VHDL. Les 4 séances sont guidées par l'enseignant et permettent d'aborder les principes fondamentaux de la conception de circuits numériques synchrones grâce au langage VHDL.
NB: Les TDM commencent par une initiation à l'utilisation des outils Design Vision pour la synthèse de circuit et Modelsim pour la simulation logique.

1.2.4 Les Travaux Pratiques

Le module se termine par deux TP de 4h qui s'inscrivent dans la continuité des TDM. On propose de concevoir un système de comptage d'impulsions. Cependant, contrairement aux TDM, la conception est moins guidée. Une plus large part d'autonomie est laissée dans les choix de conception.

2. Travaux Dirigés d'application du cours

2.1 Représentation des nombres (TD1)

Ce TD permet d'approfondir les notions vues en cours dans la partie 1. L'ensemble des questions ne pourra pas être traité en séance avec votre enseignant. C'est à vous de terminer ces exercices en autonomie afin de préparer au mieux le QCM.

2.1.1 Exercice 1 : Conversions de nombres entiers positifs

Effectuez les conversions suivantes :

$(1030)_{10} = (\quad)_2$

$(1030)_{10} = (\quad)_8$

$(1030)_{10} = (\quad)_{16}$

$(10101)_2 = (\quad)_{10}$

$(10101)_2 = (\quad)_{16}$

$(10101)_2 = (\quad)_8$

$(A03F)_{16} = (\quad)_{10}$

$(A03F)_{16} = (\quad)_2$

$(A03F)_{16} = (\quad)_8$

$(7610)_8 = (\quad)_{10}$

$(7610)_8 = (\quad)_2$

$(7610)_8 = (\quad)_8$

2.1.2 Exercice 2 : Conversions de nombre décimaux positifs

On note (b, n, m) le format de représentation d'un nombre en base b avec n chiffres pour la partie entière et m chiffres pour la partie décimale.
NB: On se restreint au cas des nombres non-signés.
Effectuez les conversions suivantes :

$(10001001)_{(2,7,1)} = (\quad)_{10}$

$(10001001)_{(2,6,2)} = (\quad)_{10}$

$(10001001)_{(2,5,3)} = (\quad)_{10}$

$(10001001)_{(2,4,4)} = (\quad)_{10}$

$(10001001)_{(2,3,5)} = (\quad)_{10}$

$(21,25)_{(10,2,2)} = (\quad)_{(2,5,3)}$

$(21,20)_{(10,2,2)} = (\quad)_{(2,5,5)}$

2.1.3 Exercice 3 : Numérisation d'une grandeur

Nous désirons numériser une mesure de longueur entre 0 et 15 cm avec une précision supérieure ou égale à 0.1 mm.

1. Quel est le nombre de bits nécessaires ?
2. Quelle est la précision obtenue finalement ?
3. Donner la relation entre la longueur en mm et le code binaire.
4. Quelle est la longueur décimale qui correspond au nombre 72C en hexadécimale ?

2.1.4 Exercice 4 : Addition binaire

Soit l'opération en base 2 suivante : 01100 + 11110. En supposant que les deux nombres sont des nombres signés, que vaut le résultat en base 10 ?
Même question si l'on suppose que ce sont des nombres non-signés.

2.1.5 Exercice 5 : Soustraction binaire

Soit l'opération en base 2 suivante : 10001 - 11010. En supposant que les deux nombres sont des nombres signés, que vaut le résultat en base 10 ?
Même question si l'on suppose que ce sont des nombres non-signés.

2.2 Fonctions logiques et algèbre de Boole (TD2)

Ce TD permet d'approfondir les notions vues en cours dans la partie 2. L'ensemble des questions ne pourra pas être traité en séance avec votre enseignant. C'est à vous de terminer ces exercices

en autonomie afin de préparer au mieux le QCM.

2.2.1 Exercice 6 : Table de vérité et formes canoniques

Établir la table de vérité de la fonction suivante, puis l'écrire sous les deux formes canoniques :

1. $F_0 = X + Y.Z + \overline{Y}.\overline{Z}.T$
2. $F_1 = X.Y + Y.Z + X.Z$
3. $F_2 = X + Y.Z + \overline{Y}.\overline{Z}.T$
4. $F_3 = (X + Y)(\overline{X} + Y + Z)$
5. $F_4 = (\overline{X} + \overline{Y} + Z)(X + \overline{Y} + Z)(X + \overline{Y} + \overline{Z})(X + Y + \overline{Z})(X + Y + Z)$

2.2.2 Exercice 7 : Formes canoniques

Écrire 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
3. f(A,B,C) = 1 si et seulement si exactement une des variables A, B, C prend la valeur 1

Mettre ces fonctions sous la seconde forme canonique.

2.2.3 Exercice 8 : Algèbre de Boole

Démontrer **algébriquement** les relations suivantes :

1. $AB + ACD + \overline{B}D = AB + \overline{B}D$
2. $(\overline{A} + B)(A + C)(B + C) = (\overline{A} + B)(A + C)$
3. $\overline{(A + B)(\overline{A} + C)} = (A + \overline{B})(\overline{A} + \overline{C})$

2.2.4 Exercice 9 : Algèbre de Boole

Simplifier algébriquement les fonctions suivantes :

1. $F_1 = (X + Y)(\overline{X} + Y)$
2. $F_2 = \overline{X}.\overline{Y} + X.Y + \overline{X}.Y$
3. $F_3 = X.Y + \overline{Z} + Z.(\overline{X} + \overline{Y})$
4. $F_4 = (X + Y + Z)(\overline{X} + Y + Z) + X.Y + Y.Z$

2.2.5 Exercice 10 : Tableaux de Karnaugh

Simplifier, par la méthode des diagrammes de Karnaugh, les fonctions booléennes suivantes. Donner le résultat sous la forme d'une somme de produits et sous la forme d'un produit de sommes.

1. $F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.\overline{B}.C + \overline{A}.B.\overline{C} + A.B.\overline{C} + A.\overline{B}.\overline{C} + A.\overline{B}.C$
2. $F(A,B,C) = \overline{A}.\overline{B}.\overline{C} + \overline{A}.B.C + A.\overline{B}.C + A.B.\overline{C}$
3. $F(A,B,C) = \overline{A}.\overline{B}.C + A.\overline{B}.\overline{C} + A.B.\overline{C}$
4. $F(A,B,C) = (A+B+C)(A+\overline{B}+\overline{C})(\overline{A}+\overline{B}+C)(\overline{A}+\overline{B}+\overline{C})(\overline{A}+B+C)$
5. $F(A,B,C,D) = \overline{A}.B.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + \overline{A}.B.C.D + \overline{A}.B.C.\overline{D}$
6. $F(A,B,C,D) = \overline{A}.\overline{B}.\overline{C}.D + \overline{A}.\overline{B}.C.D + A.\overline{B}.\overline{C}.D + A.\overline{B}.C.D$
7. $F(A,B,C,D) = \overline{A}.B.\overline{C}.\overline{D} + A.\overline{B}.\overline{C}.\overline{D} + \overline{A}.\overline{B}.\overline{C}.D + A.B.\overline{C}.D + \overline{A}.B.C.D + A.\overline{B}.C.D + \overline{A}.\overline{B}.C.\overline{D} + A.B.C.\overline{D}$
8. $F(A,B,C,D) = (A+B+C+D)(A+\overline{B}+C+D)(A+\overline{B}+C+\overline{D})(\overline{A}+\overline{B}+\overline{C}+\overline{D})(A+\overline{B}+\overline{C}+D)(\overline{A}+B+\overline{C}+D)$
9. $F(A,B,C,D) = A.\overline{B}.\overline{C}.\overline{D} + \overline{A}.B.\overline{C}.D + A.\overline{B}.C.D + A.\overline{B}.\overline{C}.D + \overline{A}.B.C.D + \overline{A}.B.C.\overline{D} + A.B.C.\overline{D}$

2.3 Les circuits combinatoires (TD3)

2.3.1 Exercice 11 : Synthèse d'un circuit combinatoire

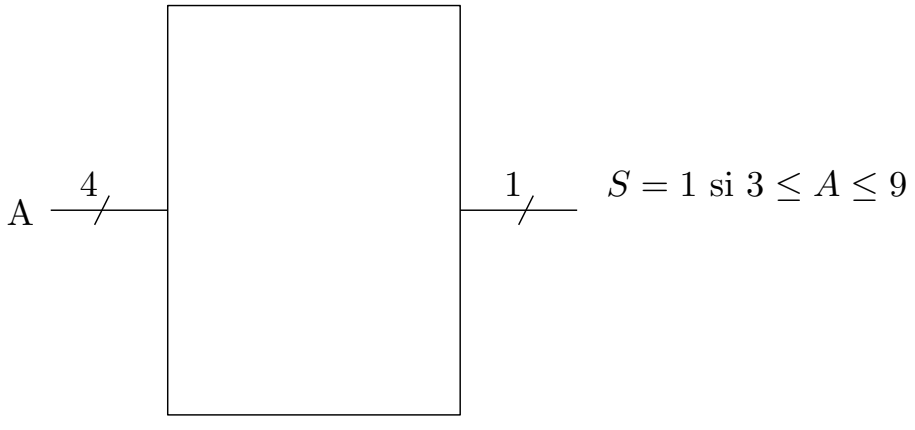


Figure 2.1: Circuit combinatoire.

On désire réaliser un système logique tel que la sortie S soit à 1 lorsque le nombre A de 4 bits présent en entrée est strictement compris entre 3 et 9.

Question 1 Donner la table de vérité de S.

Question 2 En déduire les 2 formes canoniques de S.

Question 3 Etablir le schéma logique le moins complexe possible.

2.3.2 Exercice 12 : Comparateur N bits

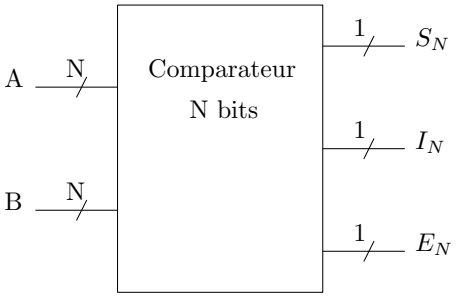


Figure 2.2: Comparateur N bits.

On souhaite réaliser un comparateur de deux mots de N bits dont l'interface est donnée sur la Figure 2.2. Les 2 mots d'entrée sont $A = (A_{N-1}A_{N-2}...A_1A_0)$ et $B = (B_{N-1}B_{N-2}...B_1B_0)$.
Les sorties du comparateurs sont :

- $S_N = 1$ ssi $A > B$
- $I_N = 1$ ssi $A < B$
- $E_N = 1$ ssi $A = B$

Question 1 Dans un premier temps, on souhaite réaliser un comparateur élémentaire de deux mots de 1 bit (N = 1). Etablissez les équations des sorties S_0, I_0, et E_0 en fonction des entrées A_0 et B_0.

Question 2 Dessinez le schéma de ce comparateur à partir d'opérateurs élémentaires (INV, NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

Question 3 On souhaite maintenant étendre l'amplitude du comparateur à deux mots de 2 bits. Après avoir établi les équations de S_1 , I_1, et E_1 en fonction des bits A_0,A_1, B_0 et B_1, concevez le schéma de ce comparateur en associant des comparateurs élémentaires et un minimum d'opérateurs (NAND, AND, NOR, OR à 2, 3 ou 4 entrées).

Question 4 A partir des équations trouvées précédemment, établir les relations de récurrence ci-dessous :
$S_N = f(S_{N-1}, A_N, B_N)$
$I_N = g(I_{N-1}, A_N, B_N)$
$E_N = h(E_{N-1}, A_N, B_N)$

2.3.3 Exercice 13 : Tri parallèle

Question 1 En utilisant des comparateurs et des multiplexeurs, proposez un circuit qui permet de comparer deux valeurs et de sortir la valeur la plus grande et la valeur la plus petite (opérateur "compare and select").

Question 2 En utilisant uniquement des opérateurs CS, proposez un circuit qui permet de trier 4 nombres signés sur codés 4 bits. Le circuit aura ainsi 4 entrées A,B,C et D sur 4 bits et 4 sorties S,T,U et V sur 4 bits avec $S < T < U < V$.

2.3.4 Exercice 14 : Conception d'un circuit logique de vote

Une société a 4 actionnaires (A, B, C et D) dont le nombre d'actions est le suivant :

- A: 60
- B: 100
- C: 160
- D: 180

Nous désirons construire une machine permettant le vote automatique par boutons poussoirs lors des réunions. Pour chaque actionnaire, le poids du vote est proportionnel au nombre d'actions qu'il possède. Une résolution est votée (V = 1) si la somme des actions correspondantes au vote OUI représente au moins la moitié des actions plus une. Faire la synthèse (table de vérité, simplification par un tableau de Karnaugh et logigramme) du dispositif de vote. Nous prendrons comme convention : vote OUI de A correspond à A = 1.
V=1 si le nombre de votes OUI est supérieur ou égal à 251 (moyenne 250 actions)

2.3.5 Exercice 15 : Conception d'un transcodeur binaire naturel /Gray

Réaliser deux circuits à base de portes XOR correspondant aux deux transcodeurs à quatre entrées binaires : Code binaire → Code de Gray Code de Gray → Code binaire

2.3.6 Exercice 16 : Conception d'un multiplieur

Considérons deux nombres entiers positifs sur 3 bits $A = (A_2A_1A_0)$ et $B = (B_2B_1B_0)$.

Question 1 Supposons que A="101" et B="110", posez la multiplication.

Question 2 Proposez une architecture à base d'additioneurs binaires et de portes AND.

2.3.7 Exercice 17 : Synthèse avec des multiplexeurs

Question 1 Combien de bits faut-il pour assurer l'adressage d'un MUX 2:1 ?

Question 2 Donnez la table de vérité d'un MUX 2:1.

Question 3 Donner l'expression logique de la sortie d'un MUX2:1 à partir de la table de vérité et en appliquant une simplification à l'aide d'un tableau de Karnaugh.

Question 4 En déduire un logigramme avec les portes logiques de base.

A	B	S
0	0	0
0	1	1
0	0	1
0	1	1

Question 5 Faites la synthèse d'un MUX 4:1 de la fonction S définie par la table de vérité suivante :
Les 0 et 1 logiques seront respectivement modélisés par la masse (GND) et la tension d'alimentation du circuit (VDD).

Question 6 Faites la synthèse de la même fonction logique à l'aide de MUX 2:1.

2.4 Les circuits séquentiels (TD4)

2.4.1 Exercice 18 : Bascule D et circuit synchrone

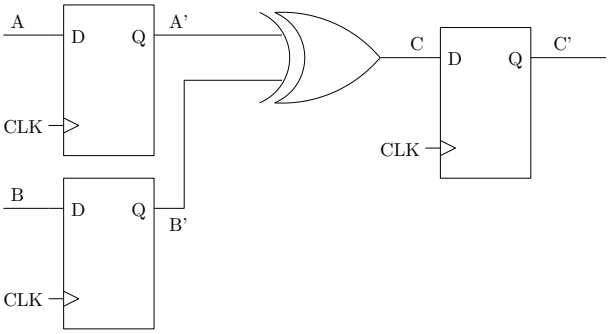


Figure 2.3: Circuit synchrone

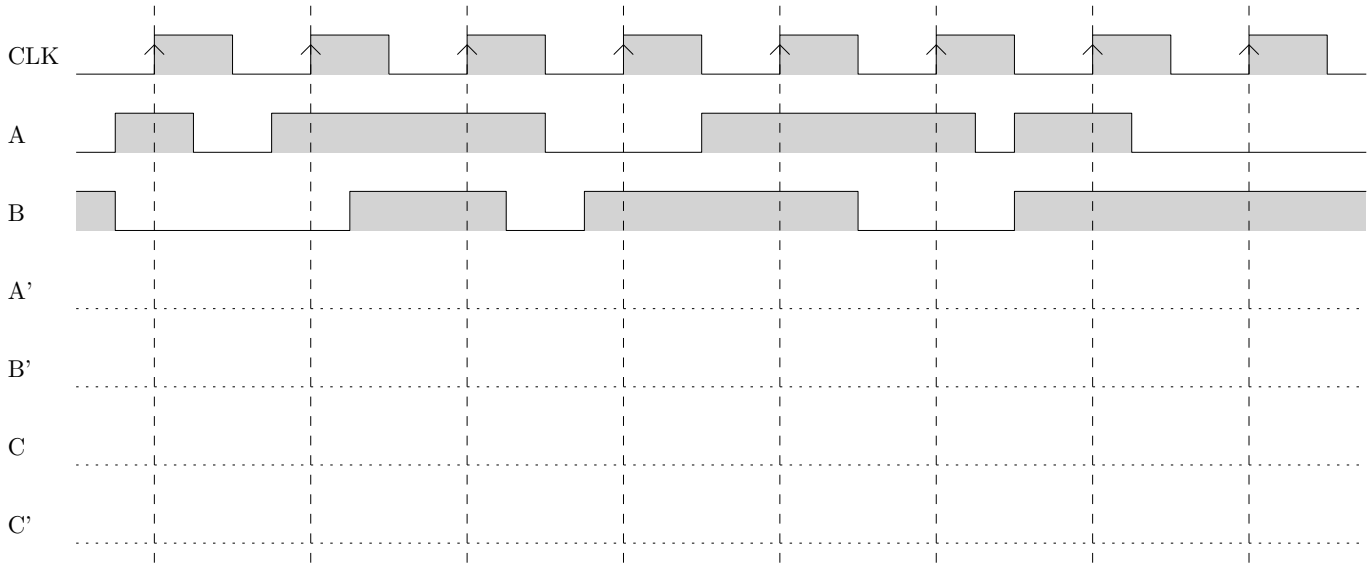


Figure 2.4: Chronogramme.

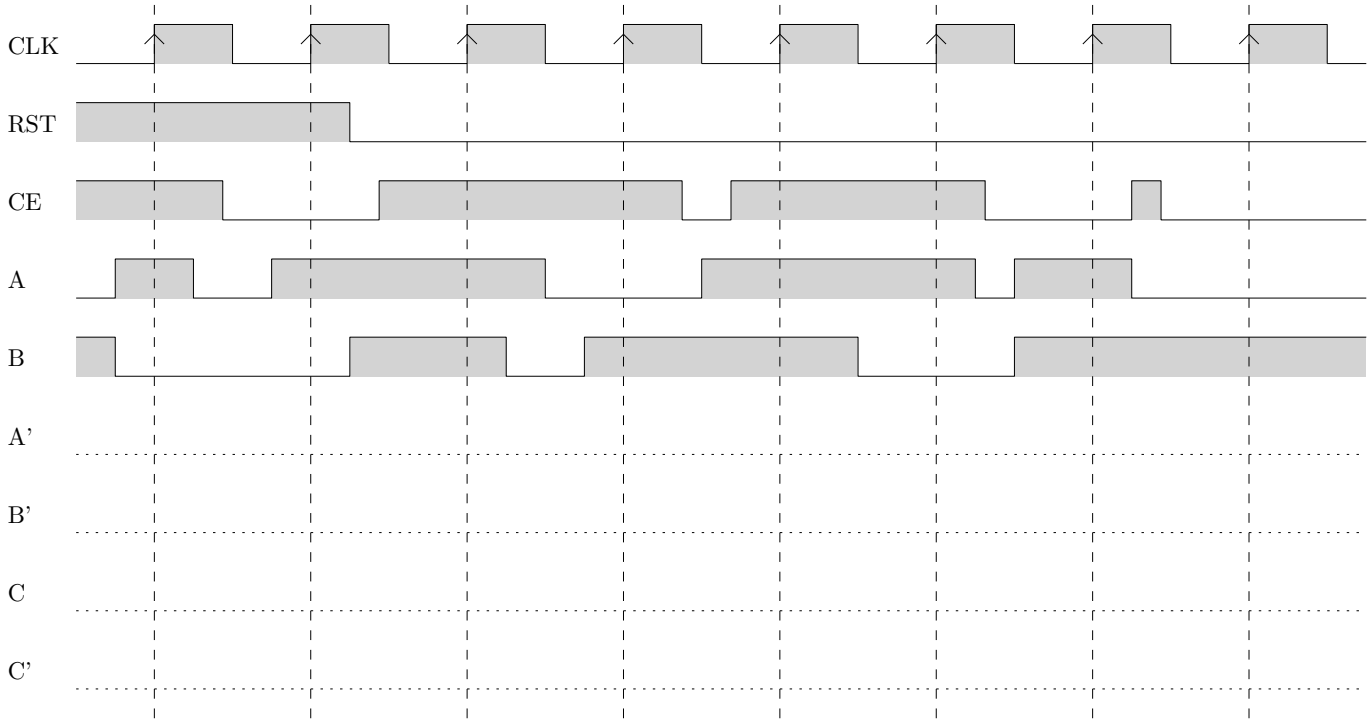


Figure 2.5: Chronogramme.

Dans cet exercice, on considère un temps de propagation $\tau_p << T_{clk}$ pour tous les opérateurs combinatoires et pour les bascules D.

Question 1 Soit le circuit de la Figure 2.3 constitué de bascule D Flip-Flop. Complétez le chronogramme de la Figure 2.4.

Question 2 On suppose maintenant que les bascules disposent d'une entrée RST de remise à zéro asynchrone active à 1 ainsi que d'une entrée d'activation CE active à 1. Complétez le chronogramme de la Figure 2.5.

Question 3 Quelle est l'utilité du mécanisme de remise à zéro asynchrone ?

Question 4 Quelle est l'utilité du mécanisme d'activation ?

2.4.2 Exercice 19 : Détection de front

Le circuit de la Figure 2.6 constitué de bascule D Flip-Flop et d'une porte logique. Ce circuit a pour but de détecter un front montant sur l'entrée A.

Question 1 Complétez le chronogramme de la Figure 2.7.

Question 2 En observant le signal S, déduisez-en le type d'événement détecté par ce circuit ?

Question 3 Déduisez-en l'équation logique de S en fonction de A' et A" ainsi que la porte

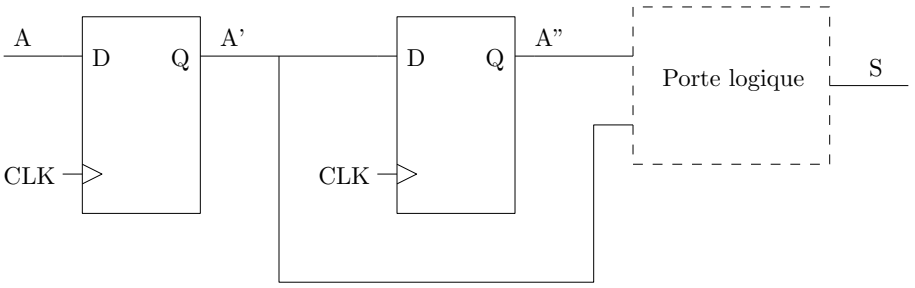


Figure 2.6: Circuit logique pour la détection de front.

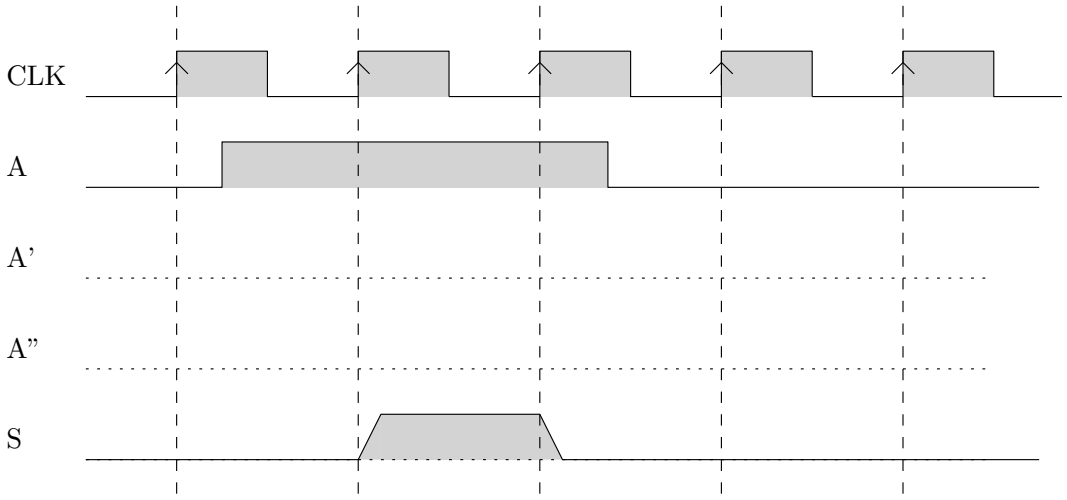


Figure 2.7: Chronogramme détection d'événement.

logique associée.

Question 4 Quelle modification faudrait-il faire pour détecter un front descendant sur A ?

Question 5 Même question pour un changement d'état quelconque ?

Question 6 En supposant que l'entrée A soit connectée à un bouton poussoir, quelle est l'utilité d'un tel circuit ? Proposez une application.

2.4.3 Exercice 20 : Look Up Table (LUT) à base de bascules D

Considérons le circuit de la Figure 2.8. Il est constitué d'un registre 4 bits dont le contenu est modifiable grâce à une entrée de chargement LOAD. Un étage de multiplexage permet ensuite d'aller lire un des bits du registre. On peut ainsi voir ce circuit comme une mémoire de capacité 4 bits ayant un adressage sur deux bits. NB: Il est possible de lire les élements de la mémoire indépendamment les uns des autres. Pour l'écriture il faut néanmoins écrire tout le contenu de la mémoire.
Nous allons voir que ce type de circuit peut être vu comme de la logique reconfigurable très utile dans les circuits FPGA.

Question 1 Suposons que le signal LOAD reste à 1 pendant quelques cycles d'horloges alors que l'entrée $D =$ "1000" (MSB à gauche). On suppose que le signal LOAD passe et reste à 0

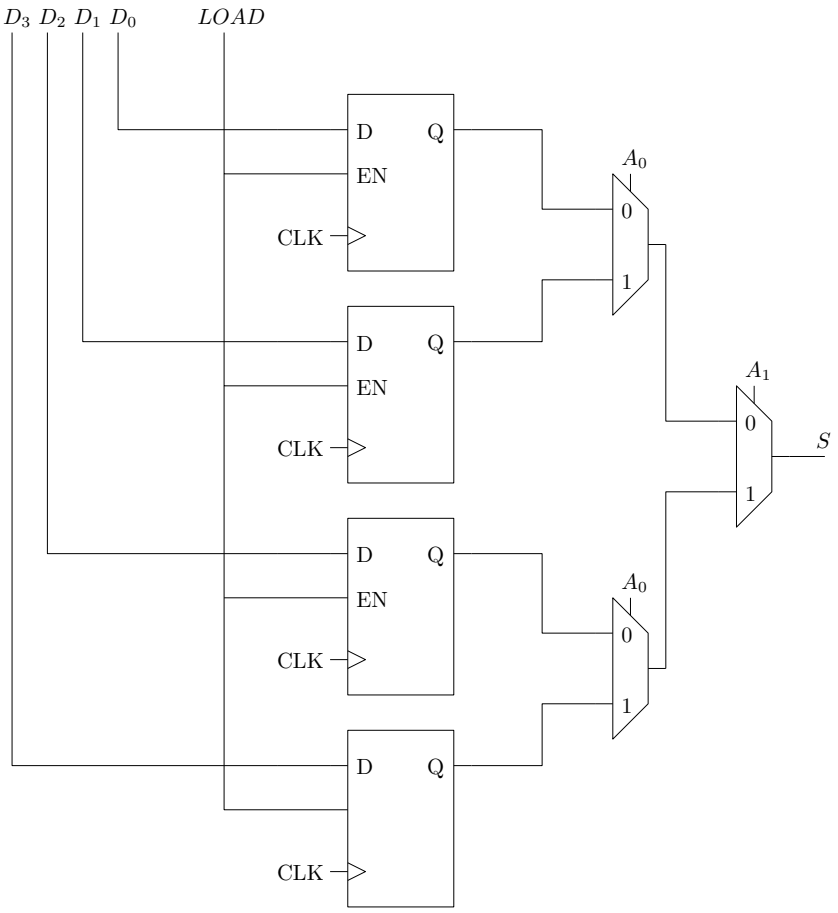


Figure 2.8: Look Up Table.

ensuite. Complétez la table de vérité suivante.

A_1	A_0	S
0	0	
0	1	
0	0	
0	1	

Question 2 Déduisez-en à quoi est équivalent le circuit dans ce cas précis ?

Question 3 Supposons que l'on charge la valeur $D = 1110$, que devient l'équivalence établit à la question précédente.

Question 4 Déterminez la valeur à charger pour obtenir l'équivalent des portes suivantes : NAND, NOR, XOR, NOT.

Question 5 Proposez une modification de l'architecture pour pouvoir modéliser n'importe qu'elle fonction logique à 3 variables.

2.4.4 Exercice 21 : Contrôleur d'afficheur 7 segments

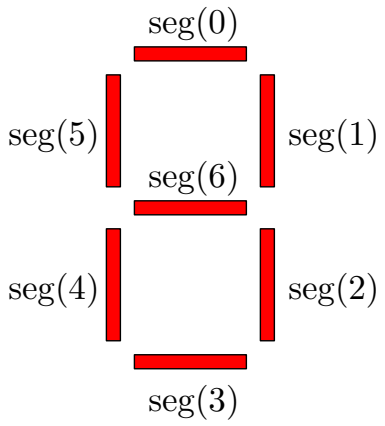


Figure 2.9: Afficheur 7 segments

Un afficheur 7 segments permet d'afficher un symbole quelconque en allumant ou éteignant les 7 segments lumineux qui le constituent. En supposant que chaque segment est allumé lorsqu'un 0 logique lui est appliqué, pour afficher le symbole '3' sur l'afficheur de la Figure 2.9, il faut appliquer la valeur seg="0110000".

Question 1 Quelle valeur faudrait-il appliquer pour afficher le chiffre 4 ?
Considérons que l'on souhaite afficher un nombre entier $0 \leq A \leq 9$ sur un afficheur 7 segments. Avant de pouvoir fournir à l'afficheur un vecteur binaire sur 7 bits qui code l'allumage de chaque segment, il faut d'abord transcoder la valeur de l'entier (codé en bianire naturel) en un vecteur "7 segments".

Question 2 A l'aide de tableaux d'une table de vérité et de tableaux de Karnaugh, déterminez les équations logiques d'un tel transcodeur.

Question 3 Supposons maintenant que nous disposons de 2 afficheurs à anodes communes. Si l'on souhaite afficher un chiffre sur l'afficheur AFF0, il faut positionner la valeur 7 segments sur le bus seg. Il faut d'autre part positionner AN(0)=1 et AN(1)=0 afin de sélectionner uniquement l'afficheur AFF0. Dans le cas ou l'on souhaiterait afficher deux chiffres distincts sur les deux afficheurs, il faut alors afficher séquentiellement les valeurs sur les afficheurs à une fréquence suffisemment élevée. De cette manière, la persistance rétinienne et le temps de réponse des afficheurs donnera l'impression d'un affichage simultanée des deux chiffres.

Question 4 Complétez le circuit de la Figure 2.10 afin qu'un tel mécanisme soit possible. Vous pouvez utiliser le transcodeur déjà réalisé ainsi que des fonctions numériques de base (registres, compteurs, multiplexeurs, etc...)

Question 5 Afin d'éviter un scintillement des segments normalement éteints, il est souvent préférable de synchroniser ce système à une fréquence pas trop élevée, par exemple 1kHz. Proposez une modification du contrôleur afin de réduire le rythme avec lequel les segments sont allumés à une fréquence proche de 1kHz.

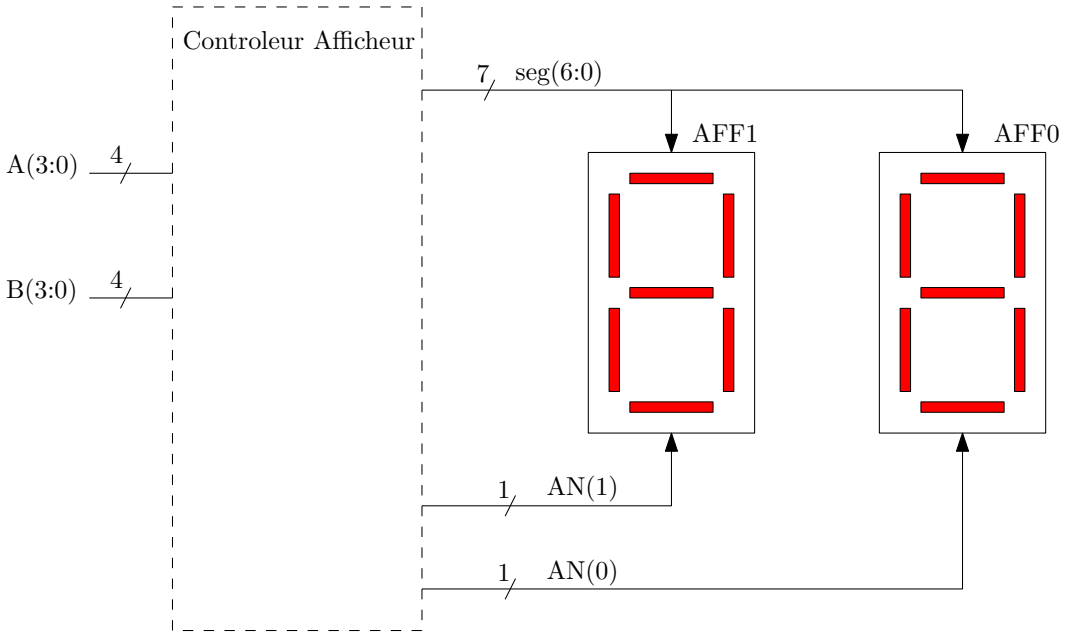


Figure 2.10: Double afficheur 7 segments

2.4.5 Exercice 22 : Digicode

On souhaite concevoir une machine d'état qui soit en mesure de détecter l'appui successif sur un clavier à 4 boutons poussoirs. Le système dispose ainsi de 4 entrées binaires A,B,C et D chacune associée à un bouton. Un appui sur un bouton génère une valeur à 1 sur l'entrée correspondante. Une sortie binaire S doit passer à 1 si et seulement si la séquence A,D,B,C est composée sur le clavier.

Question 1 Concevez la machine d'état qui permet d'implémenter ce comportement.

2.4.6 Exercice 23 : Analyse d'un Linear Feedback Shift Register (LFSR)

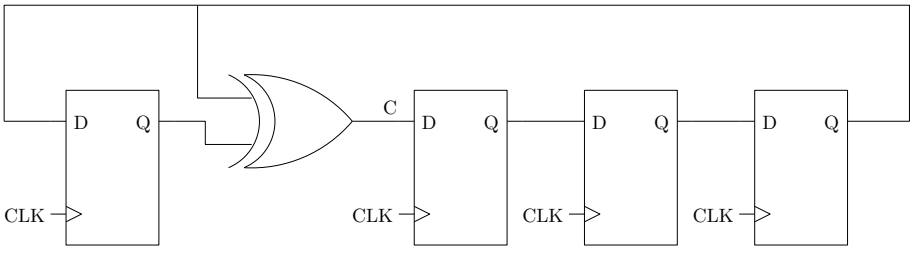


Figure 2.11: LFSR 1.

Question 1 Soit le circuit de la Figure 2.11. En supposant que les bascules D qui le composent sont initialisées à la valeur "0001", décrivez l'évolution du contenu des bascules au cours du temps.

Question 2 Quelle est la propriété de la séquence générée ?

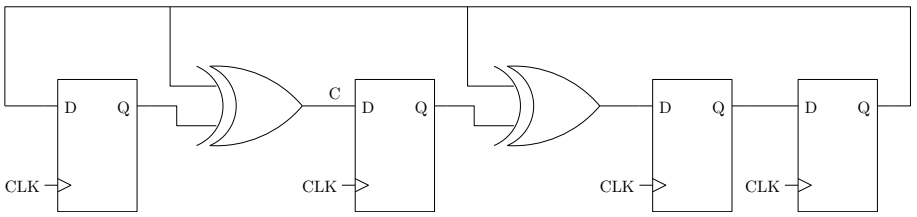


Figure 2.12: LFSR 2.

Question 3 Citez une ou plusieurs applications possible pour ce type de circuit.

Question 4 Sous les mêmes hypothèses, décrivez l'évolution du contenu des bascules au cours du temps pour le circuit de la Figure 2.12.

3. Travaux Dirigés sur Machine

3.1 Description, simulation et synthèse d'un Full Adder

3.1.1 Conception de l'architecture

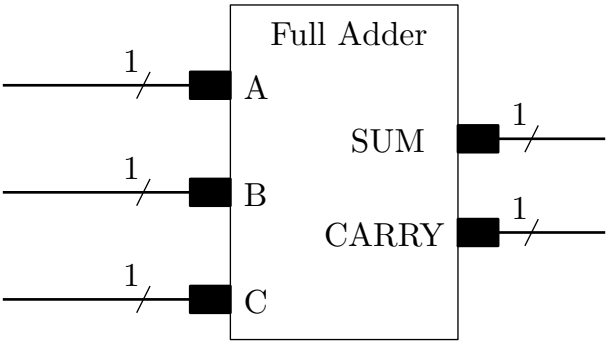


Figure 3.1: Interface du Full adder

Question 1 Ecrivez la définition de l'interface VHDL (entity) correspondant à la Figure 3.1.

Question 2 Donnez la table de vérité d'un Full Adder en respectant les notations de l'interface de la Figure 3.1.

Question 3 Dessinez le circuit logique permettant d'implémenter le Full Adder.

Question 4 Proposez une description VHDL de l'architecture du full adder en utilisant un process explicite

Question 5 Proposez une description VHDL de l'architecture du full adder en utilisant un process implicite

Question 6 A l'aide d'un chronogramme, proposez une séquence de vecteurs de tests permettant de tester ce full adder. Donnez également les valeurs attendues des sorties.

Question 7 Ecrivez le test bench associé à la séquence de test définie dans la question précédente.

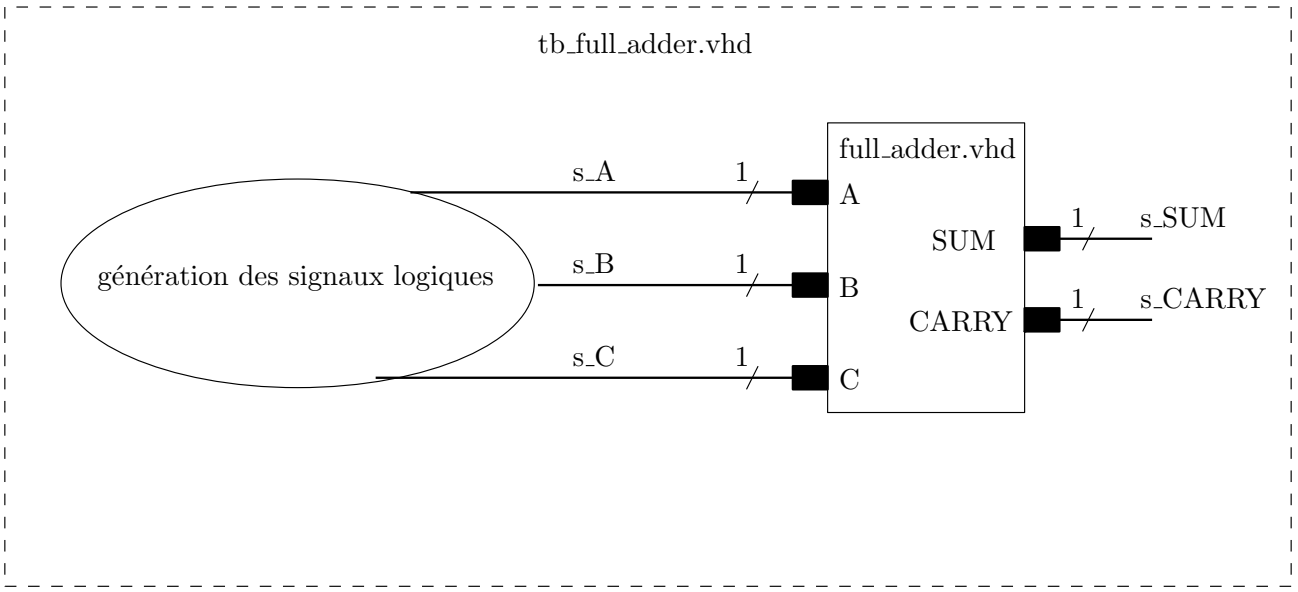


Figure 3.2: Principe du testbench

3.1.2 Simulation de l'architecture

Principe

La simulation comportementale d'une description VHDL s'effectue avant la synthèse logique. Cela permet de vérifier le fonctionnement/comportement de notre système, c'est à dire de s'assurer que le VHDL que l'on a écrit permet bien de décrire le comportement du système que l'on souhaite concevoir. Dans notre cas, il s'agit un full adder. Comme montré sur la Figure 3.2, pour faire une simulation comportementale, nous avons besoin d'un test bench `vhdl`. Un test bench est une description VHDL sans aucune entrée/sortie et dans laquelle on instancie le composant à tester (ici le full adder). Le principe général est ensuite de générer des signaux logiques qui sont envoyés en entrée du composant. La simulation consiste à observer la valeur logique des sorties et éventuellement la valeur des signaux internes du composant testé. Dans notre exemple, si on positionne les entrées A,B et C à 0,1 et 1 respectivement, on doit alors observer la valeur 0 sur SUM et 1 sur CARRY.

Simulation dans `Modelsim`

1. **Créez un repertoire** de travail
   - Téléchargez le contenu du template récupéré sur `cleroux.vvv.enseirb-matmeca.fr/EN102/TDM/TEMPLATE_TDL.zip`
   - Copiez ce fichier dans un repertoire que l'on nommera EN102
   - Décompressez le fichier TEMPLATE_TDL.zip
   - Renommez le repertoire en TDM1
2. **Lancez l'outil de simulation** `Modelsim`
   - Depuis l'interface graphique de Linux ou bien dans un terminal tapez la commande : `/usr/local/bin/launcher/mentor_modelsim_se_10.3d`

3. **Créez un nouveau projet** que l'on nommera `simu_full_adder` dans le répertoire `TDM1/simulation`
   - `File > New > Project`
   - `Browse > TDM1/simulation`
   - `Project Name : simu_full_adder`
4. **Créez un fichier VHDL** :
   - `Project > Add To Project > New File`
   - `Browse > TDM1/vhdl`
   - File name : `full_adder.vhd`
5. **Complétez la description VHDL du full adder** :
   - Dans l'onglet `Project`, double-cliquez sur le fichier `full_adder.vhd`
   - L'éditeur de texte s'ouvre, écrivez la description VHDL du full adder
     NB : Pour zoomer sur le fichier : appuyer simultanéement sur les touches CTRL et +
   - Sauvegardez votre fichier : CTRL + S
6. **Créez un test bench VHDL** :
   - `Project > Add To Project > New File`
   - `Browse > TDM1/vhdl`
   - File name : `tb_full_adder.vhd`
7. **Complétez la description VHDL du test bench** :
   - Dans l'onglet `Project`, double-cliquez sur le fichier `tb_full_adder.vhd`
   - L'éditeur de texte s'ouvre, écrivez la description VHDL du test bench
     NB : Pour zoomer sur le fichier : appuyer simultanéement sur les touches CTRL et +
   - Sauvegardez votre fichier : CTRL + S
8. **Faites une vérification synthaxique :**
   - `Compile > Compile All`
   - Si un message rouge apparait dans la fenêtre `Transcript`, double-cliquez sur le message, essayez de comprendre le message d'erreur et corriger votre description VHDL
   - Corrigez les erreurs jusqu'à ce que les deux fichiers n'en contiennent plus.
9. **Lancez la simulation :**
   - `Simulate > Start Simulation`
   - Dans l'onglet `Design`, sélectionnez le testbench : `tb_full_adder.vhd`
   - Décochez `Enable optimization`
   - Spécifiez une résolution temporelle d'1 ps `Resolution : ps`
   - Cliquez sur `OK`
10. **Vérifiez le comportement du full adder :**
    - Si la fenêtre `Wave` n'apparait pas : `View > Wave`

- Dans la fenêtre `sim`, dépliez le `tb_full_adder` et vérifiez que le full adder s'y trouve bien.
- Sélectionnez `tb_full_adder`, la liste des signaux du testbench est alors présente dans la fenêtre `Objects`.
- Faites glisser les signaux `s_A, s_B, s_C, s_SUM` et `s_CARRY` de la feneêtre `Objects` vers la fenêtre `Wave`.
- Relancer la simulation : `Simulate > Restart > OK`
- Simuler le comportement du système pour 100 ns : `Simulate > Run > Run 100`
- Vérifiez les valeurs logiques des sorties en fonction des entrées
- Sauvegardez la liste des signaux à simuler : `File > Save Format`

3.1.3 Synthèse automatique de l'architecture

Maintenant que nous avons vérifiez le comportement de notre description VHDL, nous allons synthétiser (= générer) le circuit qui permet d'implémenter ce comportement. Il s'agit là d'une première étape dans le processus de fabrication d'un circuit numérique. Pour cela, nous allons utiliser un outil de synthèse logique.

1. **Lancez l'outil Design Vision :**
   - Dans un terminal, placez-vous dans le répertoire de synthèse : `cd TDM1/synthesis`
   - Lancez l'outil en tapant la commande suivante :
     `/usr/local/bin/launcher/synopsys_design_vision_2013.03-SP5`
2. **Exécutez le script d'initialisation :**
   - Dans le terminal de `Design Vision`, tapez la commande :
     `source synopsys_dc.setup`
   - Une erreur apparait : `Error : Current design is not defined`, ignorez là.
3. **Vérifiez la synthaxe de votre description VHDL:**
   - Depuis le terminal de `Design Vision`, tapez la commande suivante :
     `analyze -f vhdl ../vhdl/full_adder.vhd`
   - Vérifiez qu'aucune erreur n'a été détectée par l'outil
4. **Faites une première synthèse de votre design :**
   - Depuis le terminal de `Design Vision`, tapez la commande suivante, en remplacant `MON_TOP_LEVEL` et `MON_ARCHITECTURE` par le nom de l'entité que vous souhaitez synthétiser et par le nom de son architecture :
     `elaborate MON_TOP_LEVEL -arch MON_ARCHITECTURE -lib WORK -update`
   - Vérifiez qu'aucune erreur n'a été détectée par l'outil
5. **Visualisez le résultat de cette première synthèse logique :**
   - Dans l'onglet `Hierarchy`, sélectionnez le `full_adder`, puis `Clique droit > Schematic View`
   - Une vue du top level apparait, double cliquez dessus pour voir l'intérieur

- Vérifiez que le circuit généré a le comportement d'un full adder.

6. **Deuxième synthèse logique et optimization :**
   - Lancez l'optimisation du circuit : tapez la commande:
     `compile_ultra`
   - Observez le résultat obtenu

3.1.4 Pour aller plus loin

Maintenant que vous avez un aperçu du flot de conception (simulation + synthèse). Nous allons l'appliquer à un autre système élémentaire, le comparateur 1 bit.
Celui-ci comporte 2 entrées A et B et les sorties S, I et E du comparateurs sont telles que :

- $S = 1$ ssi $A > B$
- $I = 1$ ssi $A < B$
- $E = 1$ ssi $A = B$

Créez un nouveau projet, puis décrivez, simulez et synthétisez ce comparateur à l'aide des outils `Modelsim` et `Design Vision`

3.2 L' additionneur 4 bits

3.2.1 Conception de l'architecture

Question 1 Soient les nombres A et B représentés en CA2 sur 4 bits : $(a_3a_2a_1a_0)$ et $(b_3b_2b_1b_0)$. Posez l'addition de ces deux nombres pour faire apparaître la somme $S = (s_3s_2s_1s_0)$ et les bits de retenues c_0, c_1, c_2 et c_3.

Question 2 Déduisez-en un circuit constitué de 4 Full adders qui permet d'implémenter un additionneur 4 bits dont l'interface est donnée sur la Figure 3.3

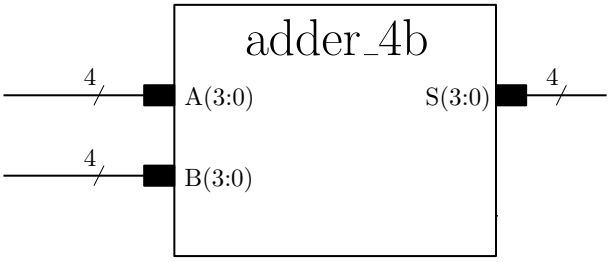


Figure 3.3: Interface de l'additionneur 4 bits

Question 3 Donnez l'interface VHDL (entity) de cet additionneur 4 bits.

Question 4 Proposez une architecture structurelle de l'additionneur 4 bits, en utilisant 4 full adder de la Figure 3.1.

Question 5 Proposez une architecture comportementale en utilisant la librairie std_numeric et un process explicite.

Question 6 Même question avec un process implicite

Question 7 Ecrivez un test bench qui permette de tester partiellement votre architecture.

Question 8 Modifiez votre circuit et la description VHDL comportementale pour que celui-ci détecte un débordement (*overflow*). Une nouvelle sortie, nommée OVF devra être positionnée à 1 si un tel événement se produit, à 0 sinon.

Question 9 On souhaite à présent concevoir un additionneur / soustracteur. Rappelez comment calculer la différence des deux nombres A et B.

Question 10 Ajoutez une entrée ADD_SUB à l'additionneur 4 bits. Si $ADD\_SUB = 0$, la sortie S doit donner la somme des deux valeurs, si $ADD\_SUB = 1$, la sortie S doit donner la différence $A - B$. Modifiez votre circuit et la description VHDL afin d'intégrer cette nouvelle fonctionnalité.

3.2.2 Implémentation

1. **Description structurelle**
   - A partir de la description structurelle d'un additionneur 4 bits (Question 4), appliquez le flot de conception du TDM1 (description, simulation et synthèse).
   - Discutez des avantages et inconvénients d'une telle description.
2. **Description comportementale**
   - Faites de même avec la description obtenue à Question 5.
   - Discutez des avantages et inconvénients d'une telle description.
3. **Ajout d'une détection d'Overflow**
   - Modifiez la description comportementale en ajoutant une détection d'overflow à votre additionneur (Question 8).
   - Effectuez la simulation et la synthèse de ce système
4. **Additionneur / soustracteur**
   - Créez un nouveau fichier VHDL `add_sub_4b.vhd` et concevez, testez et simulez un additionneur / soustracteur 4 bits (Question 10).

3.3 Le compteur modulo

3.3.1 Conception de l'architecture

Question 1 Rappelez la structure générale d'un compteur binaire synchrone.

Question 2 Faites la synthèse d'un compteur binaire modulo 6 synchrone.

Question 3 Donnez la description VHDL de ce compteur (entity + architecture) en utilisant une description comportementale.

Question 4 Modifiez ce compteur (circuit et description VHDL) pour lui ajouter une entrée de remise à zéro asynchrone et une entrée d'activation enable.

Question 5 Ajoutez à votre circuit une entrée logique UP_DOWN qui permet de compter ou décompter, puis modifez le compteur pour qu'il puisse à la fois compter et décompter.

Question 6 Transposez cette modification à votre description VHDL.

Question 7 Modifiez de nouveau le circuit pour lui ajouter une entrée d'initialisation synchrone. Expliquez pourquoi il est intéressant de disposer d'une telle fonctionalité alors que l'on a déjà une remise à zéro asynchrone.

Question 8 Transposez cette modification à votre description VHDL.

Question 9 Ajoutez une entrée de chargement parallèle à votre circuit et à la description VHDL.

3.3.2 Implémentation

1. **Implémentation d'un compteur**
   - Décrivez, simulez et synthétisez la description VHDL obtenue à la Question 3.
2. **Ajout d'entrées de contrôle**
   - Ajoutez une entrée de remise à zéro asynchrone (prioritaire) et une entrée d'activation enable
   - Simulez et synthétisez la description VHDL obtenue.
3. **Compteur / décompteur**
   - Ajoutez une entrée UP_DOWN qui permet de contrôler le comptage / décomptage.
   - Simulez et synthétisez la description VHDL obtenue.
4. **Chargement parallèle**
   - Ajoutez une entrée de chargement parallèle à votre description.
   - Simulez et synthétisez la description VHDL obtenue.

3.4 Machine à état finis (FSM)

3.4.1 Conception de l'architecture

La Figure 3.4 montre deux cartes électroniques C1 et C2 qui ont accès à un même bus B. Chacune des deux cartes demande sa connexion au bus en activant une entrée D qui est maintenue à 1 jusqu'à la fin de l'utilisation du bus. Lorsque le bus établit la connexion avec l'une des cartes, le signal P correspondant est activé. Ce signal reste à 1 pendant toute la durée de la liaison. La carte qui a utilisé le bus en dernier n'est pas prioritaire en cas de demande simultanée du bus.(à l'initialisation C1 est prioritaire) Sachant qu'une demande de connexion doit être servie le plus rapidement possible, réaliser un automate synchrone, dont les entrées sont D1 et D2 et les sorties P1 et P2, qui permet de gérer l'accès au bus.

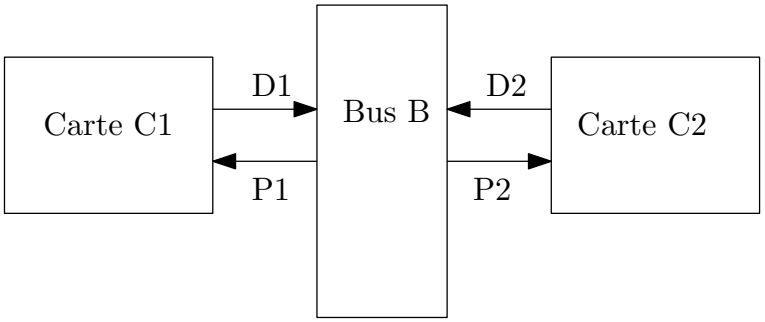


Figure 3.4: Bus partagé.

Question 1 Complétez les chronogrammes de la Figure 3.5.

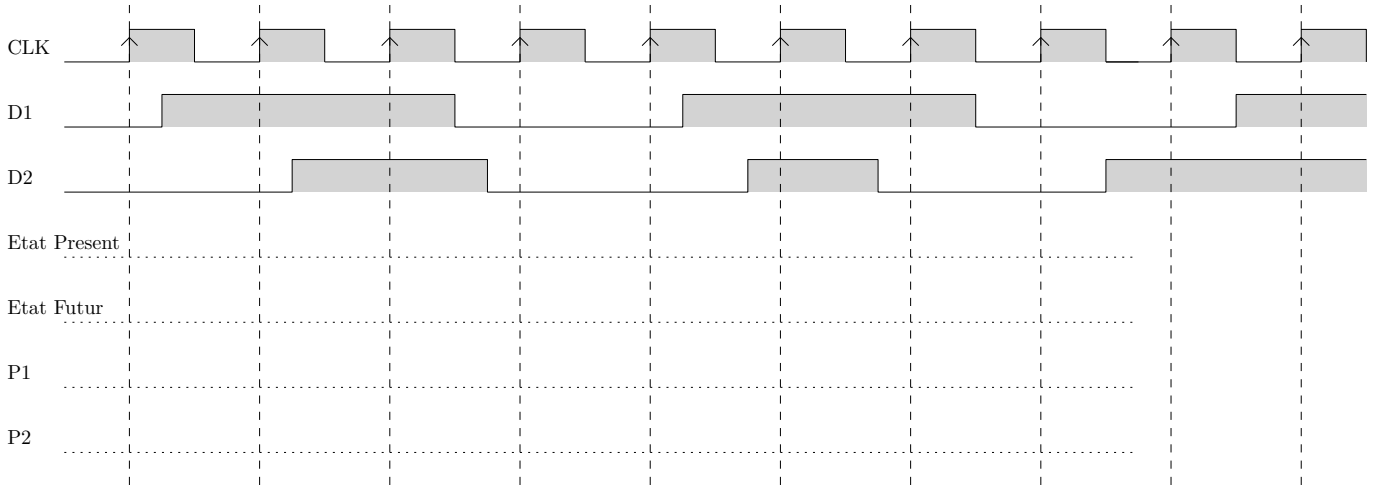


Figure 3.5: Chronogramme FSM.

Question 2 Etablir un graphe d'états permettant de décrire le comportement de l'automate.

Question 3 Faire la synthèse logique de l'automate à l'aide d'opérateurs logiques élémentaires.

Question 4 Donnez la description VHDL de cette machine d'états. Vous utliserez pour cela une description comportementale avec 3 process VHDL.

3.4.2 Implémentation

Décrivez, simulez et synthétisez la machine d'état obtenue à la Question 4.

4. Travaux Pratiques

4.1 Compteur d'impulsions

Nous souhaitons réaliser un système numérique capable de compter le nombre d'impulsions dans un train binaire série. On suppose que la fréquence de ce train binaire est bien inférieure à la fréquence du signal d'horloge qui cadence le système.

La Figure 4.1 montre un synoptique possible d'un tel système. Un premier étage, constitué d'une FSM permet de détecter les imulsions sur le train binaire d'entrée E. Dés qu'une impulsion est détectée, le signal D doit passer à 1 pendant un cycle d'horloge. Le deuxième étage est constitué d'un compteur modulo 256 qui indique le nombre d'impulsions comptées à un instant t. Un comparateur permet ensuite de comparer le nombre d'impulsion comptabilisée à une valeur d'entre VAL. Si le nombre d'impulsions comptées atteint la valeur VAL, alors une impulsion d'une durée de 10 cycles d'horloge doit être générée sur la sortie S à l'aide d'un timer. Une fois que la valeur VAL est atteinte, le système doit se réinitialiser de manière synchrone.

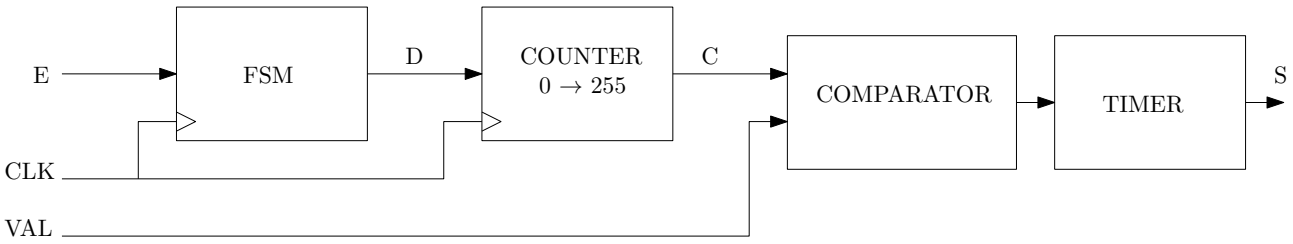


Figure 4.1: Détecteur d'impulsions.

Décrivez, simulez et synthétisez ce système.