POLYTECHNIQUE
MONTRÉAL

INF1500
Logique des systèmes numériques

Laboratoire 3

Soumis par:
Mohamad Awad - 2034584
Lourhmati Oussama - 2081643
Groupe 05

Le 5 novembre 2020

Introduction

Le système de ce laboratoire est un circuit intégrant trois modules : un convertisseur de code binaire en code Gray, un module “ Crypto” et un module diviseur qui fait une division par 4 et place le quotient sur deux bits et le reste sur deux bits. Un multiplexeur sélectionne entre les sorties de ces trois modules pour qu’elle soit la sortie du système global. Nous détaillons dans ce rapport , les étapes de réalisation de chaque module, et validons leur fonctionnement par des simulations.

1- Description du système

Convertisseur Binaire vers Gray:

Table de vérité : BIN_GRAY

Code décimal	Code hexadécimal	Entrée (code binaire)				Sortie (code Gray)			
		E3	E2	E1	E0	S3	S2	S1	S0
0	0	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	1
2	2	0	0	1	0	0	0	1	1
3	3	0	0	1	1	0	0	1	0
4	4	0	1	0	0	0	1	1	0
5	5	0	1	0	1	0	1	1	1
6	6	0	1	1	0	0	1	0	1
7	7	0	1	1	1	0	1	0	0
8	8	1	0	0	0	1	1	0	0
9	9	1	0	0	1	1	1	0	1
10	A	1	0	1	0	1	1	1	1
11	B	1	0	1	1	1	1	1	0
12	C	1	1	0	0	1	0	1	0
13	D	1	1	0	1	1	0	1	1
14	E	1	1	1	0	1	0	0	1
15	F	1	1	1	1	1	0	0	0

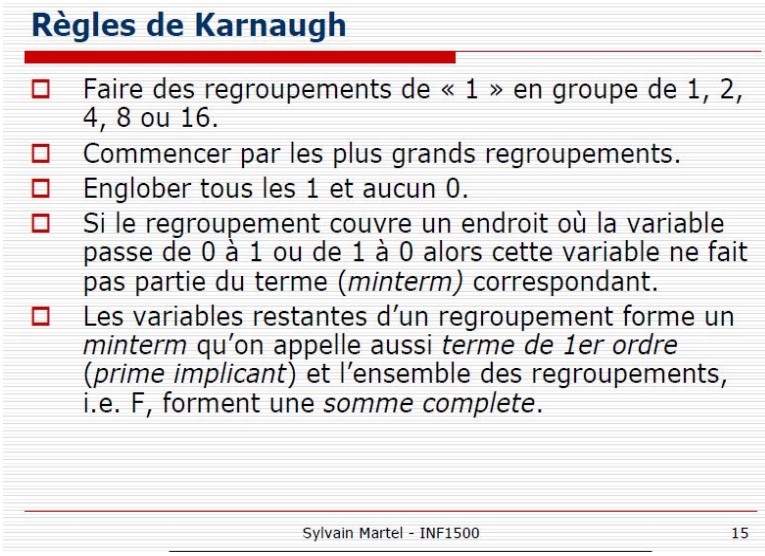
Le Code Gray est un code où à chaque incrémentation d'une unité, un seul bit change. Il est utilisé pour éviter les états transitoires indésirables: différence entre les délais de transitions des bits. La table de vérité BIN_GRAY a été remplie suivant la formule suivante:

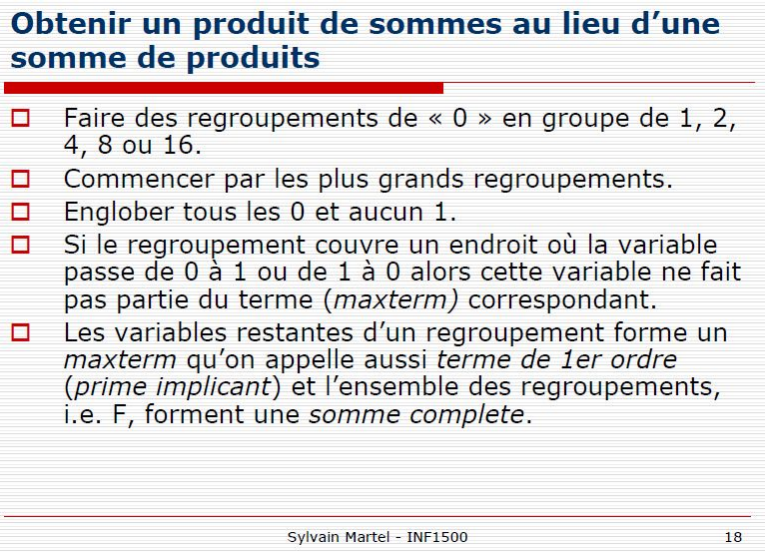
(N⊕2N) / 2 ***Référence:*** ***Code Gray***

Où N est le nombre en code binaire à 4 bits. 2N est le double de N, obtenu par un shift à gauche de 1 bit ; ⊕ est l'opération ou_exclusif **XOR.** Enfin, **/2** est un shift à droite de 1 bit. L'opération **XOR** est faite bit par bit: **le bit i de N ⊕ le bit i de 2N** . Notons que **0 ⊕ 0=0 ; 0 ⊕ 1 = 1 ⊕ 0 = 1 et 1 ⊕ 1=0.**

Exemples: (0101 ⊕ 1010)/2 = 1111/2 = 0111. (01111 ⊕ 11110)/2 = 10001/2 = 1000.

Tables de Karnaugh:





Référence: Sylvain Martel- INF1500- Cours 4.

Les tables de Karnaugh donnent des équations simplifiées de chacun des bits de sortie. Le Code Gray est utilisé dans ces tables et les règles pour obtenir les équations sont les suivantes :

Expression de la sortie S0

		E1 E0			
		00	01	11	10
E3 E2	00	0	1	0	1
	01	0	1	0	1
	11	0	1	0	1
	10	0	1	0	1

S0 = (E1' • E0) + (E1 • E0')
= E1⊕ E0

Expression de la sortie S1

		E1 E0			
		00	01	11	10
E3 E2	00	0	0	1	1
	01	1	1	0	0
	11	1	1	0	0
	10	0	0		11

S1 = (E1'•E2)+(E2' • E1)=
E1⊕E2

Expression de la sortie S2

		E1 E0			
		00	01	11	10
E3 E2	00	0	0	0	0
	01	1	1	1	1
	11	0	0	0	0
	10	1	1	1	1

S2 = (E3+E2)•(E3'+E2')
=E3•E2'+ E3'•E2 = E3⊕ E2

Expression de la sortie S3

		E1 E0			
		00	01	11	10
E3 E2	00	0	0	0	0
	01	0	0	0	0
	11	1	1	1	1
	10	1	1	1	1

S3 = E3

Les équations nous permettent alors de construire le module convertisseur, chaque bit de sortie a sa propre équation et donc une différente architecture:

- **S0= E0 ⊕ E1 : une porte XOR à deux entrées , reçoit E0 et E1 et la sortie est liée au bit de sortie S0**
- **S1= E1 ⊕ E2 : une porte XOR à deux entrées , reçoit E1 et E2 et la sortie est liée au bit de sortie S1**
- **S2= E2 ⊕ E3 : une porte XOR à deux entrées , reçoit E1 et E0 et la sortie est liée au bit de sortie S2**
- **S3 : le bit d'entrée E3 est lié directement au bit de sortie S3.**
- Donc, pour le module entier, on aura besoin d'un total de trois portes XOR à deux entrées chacune, un bus d'entrée et un bus de sortie. En fait, les équations ont bien expliqué la formule que nous citons dans la page 3: ***(N⊕2N) / 2 .***

si N est representé par ***0_E3_E2_E1_E0*** alors 2N , obtenu par un shift à gauche de 1 bit est représenté par ***E3_E2_E1_E0_0*** . l'opération **XOR** est appliquée bit par bit et on aura un shift à droite de 1 bit pour obtenir le code Gray représenté par ***E3⊕0_E3⊕E2_E2⊕E1_E1⊕E0 -> S3_S2_S1_S0*****.** Notons que: **E3⊕0= E3'•0 + E3•0' = 0 +E3•1 =E3.**

La figure 1.0 introduit le block design du module convertisseur BIN_GRAY.

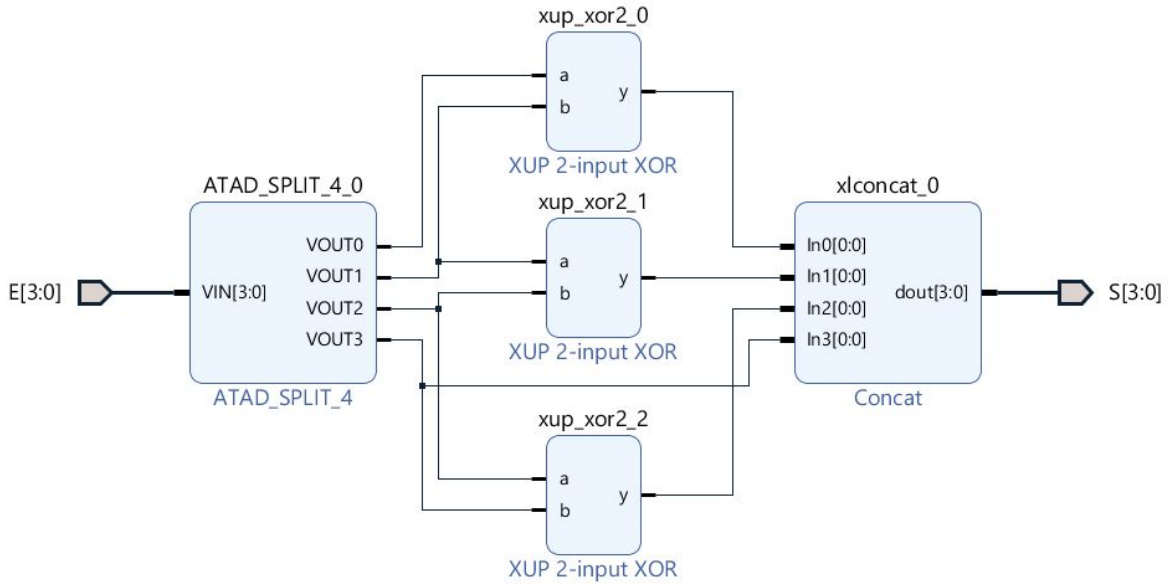


Figure 1.0: Module BIN_GRAY

Division:

Table de vérité: La table de vérité du module diviseur, place le reste de la division par 4 dans les bit S3 et S2 et le quotient dans les bits S1 et S0.

Entrée					Sortie		
A3	**A2**	**A1**	**A0**	**S3**	**S2**	**S1**	**S0**
0	0	0	0	0	0	0	0
0	0	0	1	0	1	0	0
0	0	1	0	1	0	0	0
0	0	1	1	1	1	0	0
0	1	0	0	0	0	0	1
0	1	0	1	0	1	0	1
0	1	1	0	1	0	0	1
0	1	1	1	1	1	1	0
1	0	0	0	0	0	0	1
1	0	0	1	0	1	1	0
1	0	1	0	1	0	1	0
1	0	1	1	1	1	1	0
1	1	0	0	0	0	1	1
1	1	0	1	0	1	1	1
1	1	1	0	1	0	1	1
1	1	1	1	1	1	1	1

Pour ce diviseur, nous n'avons pas besoin de faire les tables de Karnaugh. Il est clair qu'il s'agit d'une simple permutation des positions : les bits d'entrées [3],[2],[1],[0] deviennent pour la sortie: [1], [0], [3], [2] respectivement. Dans le bus de sortie: **S[3]=A[1], S[2]=A[0], S[1]=A[3], S[0]=A[2].** La figure 1.2 introduit le module diviseur DIV_4_REST.

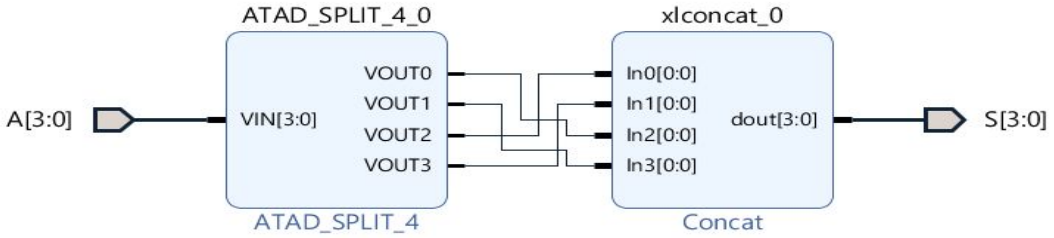


Figure 1.2: Module DIV_4_REST

Crypto:

Table de vérité: La table de vérité du Crypto nous a été donnée, les cases en couleurs contiennent des “X : don’t-care” , on a choisi les valeurs des “X” de manière à simplifier les équations obtenues dans les tables de Karnaugh.

Entrée					Sortie			
A3	A2	A1	A0		S3	S2	S1	S0
0	0	0	0		1	0	1	1
0	0	0	1		0	1	0	1
0	0	1	0		1	0	0	1
0	0	1	1		0	0	1	1
0	1	0	0		0	0	0	0
0	1	0	1		1	1	1	0
0	1	1	0		1	1	1	0
0	1	1	1		1	0	0	0
1	0	0	0		1	0	0	0
1	0	0	1		0	1	1	0
1	0	1	0		1	0	1	0
1	0	1	1		1	0	0	0
1	1	0	0		0	0	1	1
1	1	0	1		1	1	0	1
1	1	1	0		0	0	0	0
1	1	1	1		0	0	1	0

Expression de la sortie S0

		A1 A0			
		00	01	11	10
A3A2	00	1	1	1	1
	01	0	0	0	0
	11	1	1	0	0
	10	0	0	0	0

S0 = (A3'•A2')+(A3•A2•A1')

Expression de la sortie S1

		A1 A0			
		00	01	11	10
A3A2	00	1	0	1	0
	01	0	1	0	1
	11	1	0	1	0
	10	0	1	0	1

S1= (A3'A2'A1'A0') + (A3'A2'A1A0) + (A3'A2A1'A0) + (A3'A2A1A0') + (A3A2'A1'A0) + (A3A2'A1A0') + (A3A2A1'A0') + (A3A2A1A0)

Expression de la sortie S2

		A1 A0			
		00	01	11	10
A3A2	00	0	1	0	0
	01	0	1	0	1
	11	0	1	0	0
	10	0	1	0	0

S2= (A1'•A0) + (A3'•A2•A1•A0')

Expression de la sortie S3

		A1 A0			
		00	01	11	10
A3A2	00	1	0	0	1
	01	0	1	1	1
	11	0	1	0	0
	10	1	0	1	1

S3 = (A3+A2+A0') • (A0+A1+A2') • (A0'+A1+A2) • (A3'+A2'+A1')

On pourrait encore écrire l'équation de S1 d'une façon plus organisée, basé sur le fait que **: ☆X•Y + X•Z = X•(Y+Z)☆**

S1 = (A3'A2'A1'A0')+(A3'A2'A1A0) + (A3'A2A1'A0)+(A3'A2A1A0') + (A3A2'A1'A0) + (A3A2'A1A0') + (A3A2A1'A0') + (A3A2A1A0)

= A3'A2'•(A1'A0'+A1A0) + A3'A2•(A1'A0+A1A0') + A3A2'•(A1'A0+A1A0') + A3A2•(A1'A0'+A1A0) = (A1'A0'+A1A0)•(A3'A2'+A3A2) + (A1'A0+A1A0')•(A3'A2+A3A2').

On sait que : **X ⊕ Y = XY'+YX' et que (X⊕ Y)' = XY+X'Y'.** On va appliquer ces deux expressions ici: **S1= (A1 ⊕A0)•(A3 ⊕ A2) + (A1 ⊕ A0)'•(A3 ⊕ A2)'.**

Les équations nous permettent de construire notre module "Crypto", chaque bit de sortie a sa propre équation, donc une propre architecture. Puisque chaque bit est inversé au moins dans une équation, nous incluons quatres inverseurs correspondants aux bits d'entrée dans notre circuit.

S0 = (A3'•A2')+(A3•A2•A1'):

- **Une porte AND à deux entrées reçoit A3 inversé et A2 inversé.**
- **Une porte AND à trois entrées reçoit A1 inversé et A2 et A3 .**
- **Une porte OR à deux entrées reçoit les sorties des deux portes AND et sa sortie est liée à S0.**

S1= (A1 XOR A0)•(A3 XOR A2) + (A1 XNOR A0)•(A3 XNOR A2) :

- **Une porte XOR à deux entrées reçoit A1 et A0, une autre porte XOR à deux entrée reçoit A2 et A3.**
- **Les sorties des deux portes XOR entrent ensemble dans une porte AND à deux entrées.**
- **Une porte XNOR à deux entrées reçoit A1 et A0 et une autre porte XNOR à deux entrées reçoit A3 et A2.**
- **Les sorties des deux portes XNOR entrent ensemble dans une nouvelle porte AND à deux entrées.**
- **Les sorties des deux portes AND entrent ensemble dans une porte OR à deux entrées . La sortie de la porte OR est liée à S1.**

S2= (A1'•A0) + (A3'•A2•A1•A0'):

- **Une porte AND à deux entrées reçoit A1 inversé et A0**
- **Une porte AND à quatre entrées reçoit A3 inversé , A2, A1 et A0**
- **Une Porte Or à deux entrées reçoit les sorties des deux portes AND. La sortie de la porte Or est liée à S2**

S3 = (A3+A2+A0') • (A0+A1+A2') • (A0'+A1+A2) • (A3'+A2'+A1')

- **Une porte Or à trois entrées reçoit A3, A2 et A0 inversé .**
- **Une porte Or à trois entrées reçoit A1, A0 et A2 inversé .**
- **Une porte Or à trois entrées reçoit A1, A2 et A0 inversé.**
- **Une porte Or à trois entrées reçoit A3 inversé , A2 inversé et A1 inversé**
- **Une porte AND à quatre entrées reçoit les sorties de toutes les portes OR. La sortie de la porte AND est liée à S3**

X	Y	Z		X.Y+X.Z	X.(Y+Z)
0	0	0		0	0
0	0	1		0	0
0	1	0		0	0
0	1	1		0	0
1	0	0		0	0
1	0	1		1	1
1	1	0		1	1
1	1	1		1	1

☆X•Y + X•Z = X•(Y+Z)☆ Voici une démonstration de l'égalité utilisée dans ce qui précède.

La figure 1.1 introduit le module “Crypto”

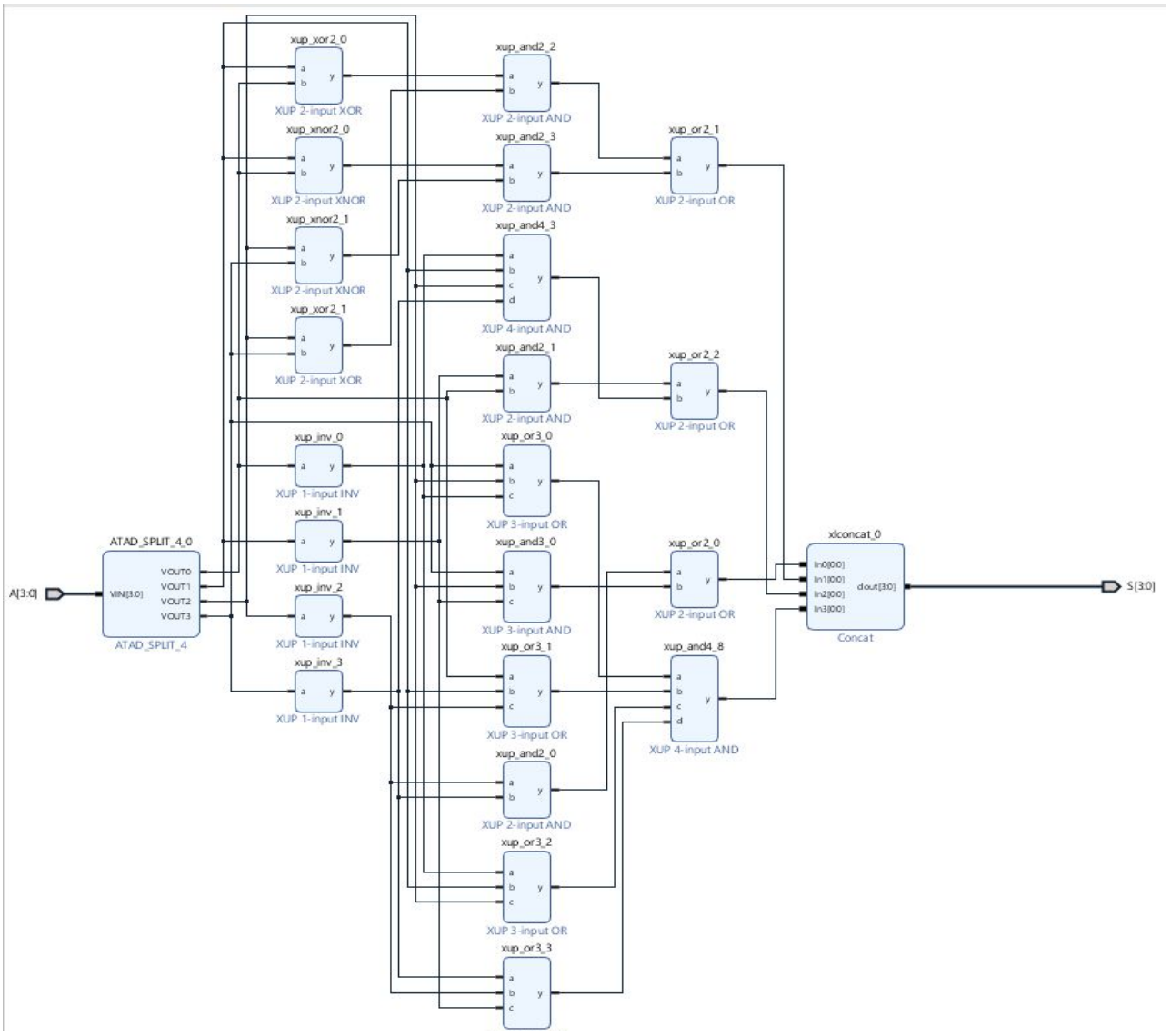


Figure 1.1: Module CRYPTO

Multiplexeur:

Table de vérité: voici la table de vérité indice du multiplexeur, celui-ci choisit , en fonction des signaux de sélection SEL , une de trois de ses entrées, et l'envoie à la sortie . Donc il va choisir, par le même scénario , entre les bits A B C, pour l'indice $i \in \{0,1,2,3\}$.

A[i]	B[i]	C[i]	SEL[0]	SEL[1]	S[i]
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	1	0	0
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	0	1	0
1	1	0	1	0	1
1	1	1	1	1	0

Comme on a 5 variables pour cette table de vérité , on a donc besoin une table de karnaugh à 5 variables pour donner l'équation de S[i]. Pour cela , on procède par une représentation sous forme de deux tables "voisines" de 4 variables chacun.

A=0		BC			
		00	01	11	10
SEL [1] SEL [0]	00	0	0	0	0
	01	0	0	1	1
	11	0	0	0	0
	10	0	1	1	0

A=1		BC			
		00	01	11	10
SEL 1 SEL 0	00	1	1	1	1
	01	0	0	1	1
	11	0	0	0	0
	10	0	1	1	0

Les mêmes règles sont suivies pour une table de karnaugh avec moins de variables, en effet, ces deux tables sont voisines. Donc si une case a les même B_C_SEL[1]_SEL[0] dans la première table qu'une autre case dans la deuxième table, ces deux cases sont voisines. Deux regroupements qui couvrent les mêmes emplacements des cases sont des regroupements voisins, or ils forment un seul regroupement plus grand.

Les regroupements sont les plus grands possible en puissance de 2, nous encadrons dans la table de karnaugh à 5 variables, les regroupements convenables des mintermes, chacun identifié par une couleur.

Si un regroupement est complètement sur une seule table, alors la variable A fait partie du minterm correspondant, car elle ne change pas sur toute la table.

Nous obtenons l'équation suivante:

S[i] = SEL[0]'•SEL[1]'•A[i] + SEL[0]•SEL[1]'•B[i] + SEL[0]'•SEL[1]•C[i]

- **Puisque SEL[0] et SEL[1] sont inversés au moins une fois dans l'équation on introduit deux inverseurs dans notre circuit**
- **Une porte AND à trois entrées reçoit SEL[0] inversé SEL[1] inversé et A[i].**
- **Une porte AND à trois entrées reçoit SEL[0] , SEL[1] inversé et B[i].**
- **Une porte AND à trois entrées reçoit SEL[0] inversé ,SEL[1] et C[i].**
- **Une porte OR à trois entrées reçoit les sorties des trois portes AND. la sortie de OR est liée à la bit de sortie [i] du multiplexeur.**

Le même scénario est fait quatre fois, une fois pour chaque bit S[i]. Pour cela, on construit une entité que nous appelons **MUX_INDEX**, qui prend 3 bits simples A B C et un bus de deux bits SEL comme entrées, et qui fait sortir un bit simple S, et qui a l'architecture dictée ci-dessus. La figure 1.3.0 introduit le block design du **MUX_INDEX**.

Le module multiplexeur **MUX3_1** est constitué donc de 4 **MUX_INDEX**, chacun prenant les bits de son indice. la figure 1.3.1 introduit le block design du **MUX3_1.**

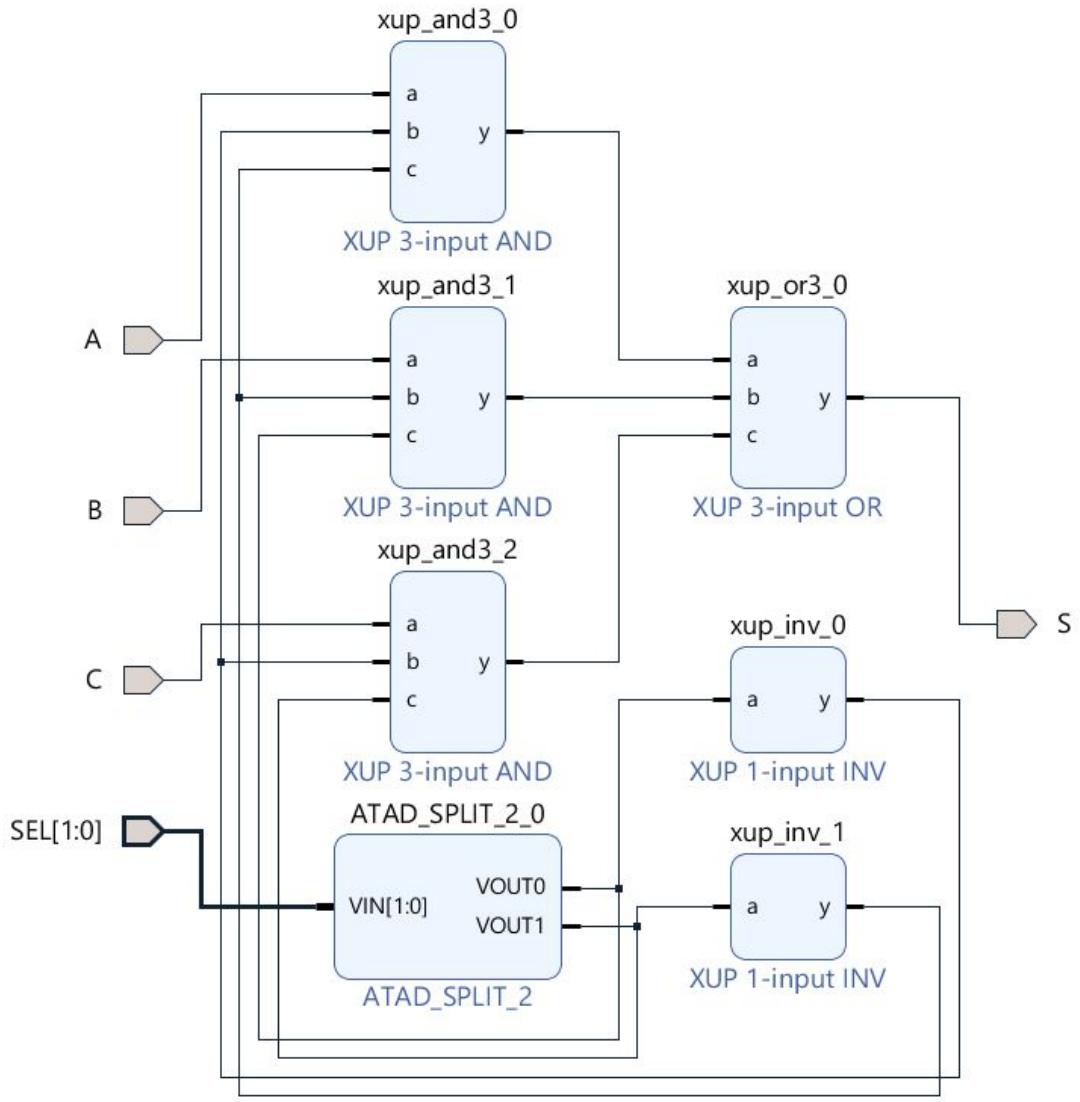


Figure 1.3.0 : MUX_INDEX.

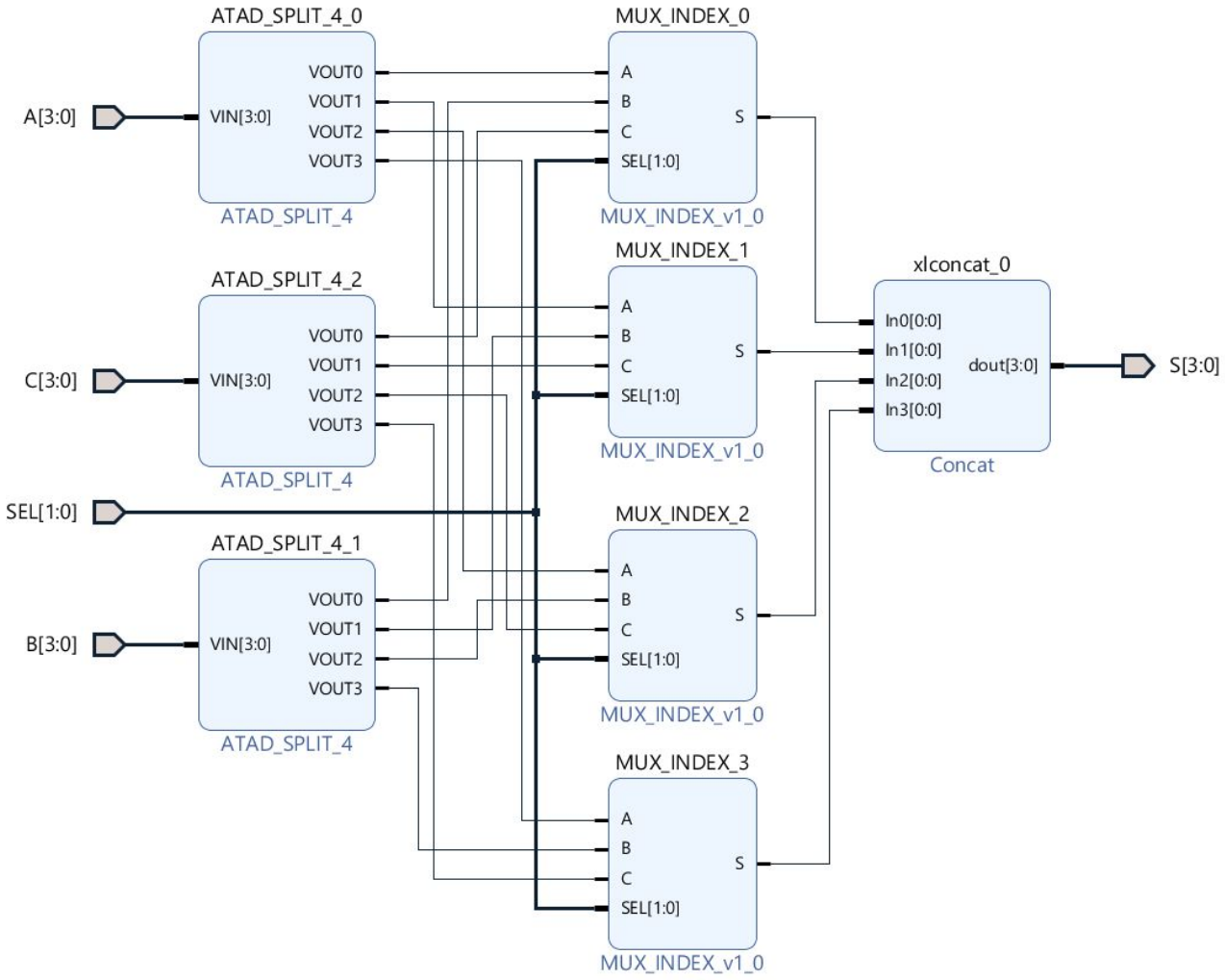


Figure 1.3.1 : Module MUX3_1

Finalement, nous combinons tous les modules et le multiplexeur , pour former un circuit final, le circuit demandé dans ce laboratoire. On a 4 Bus d'entrée : A B C et Sel

A entre dans **BIN_GRAY, B** dans **DIV_4_REST** et **C** dans **Crypto**.

Les entrées **A,B,C** du **MUX3_1** reçoivent respectivement la sorties de **BIN_GRAY** , la sortie du **DIV_4_REST**, la sortie de **CRYPTO** . L'entrée **SEL** du **MUX3_1** est liée au bus d'entrée **SEL**. La figure 1.4 montre le circuit final.

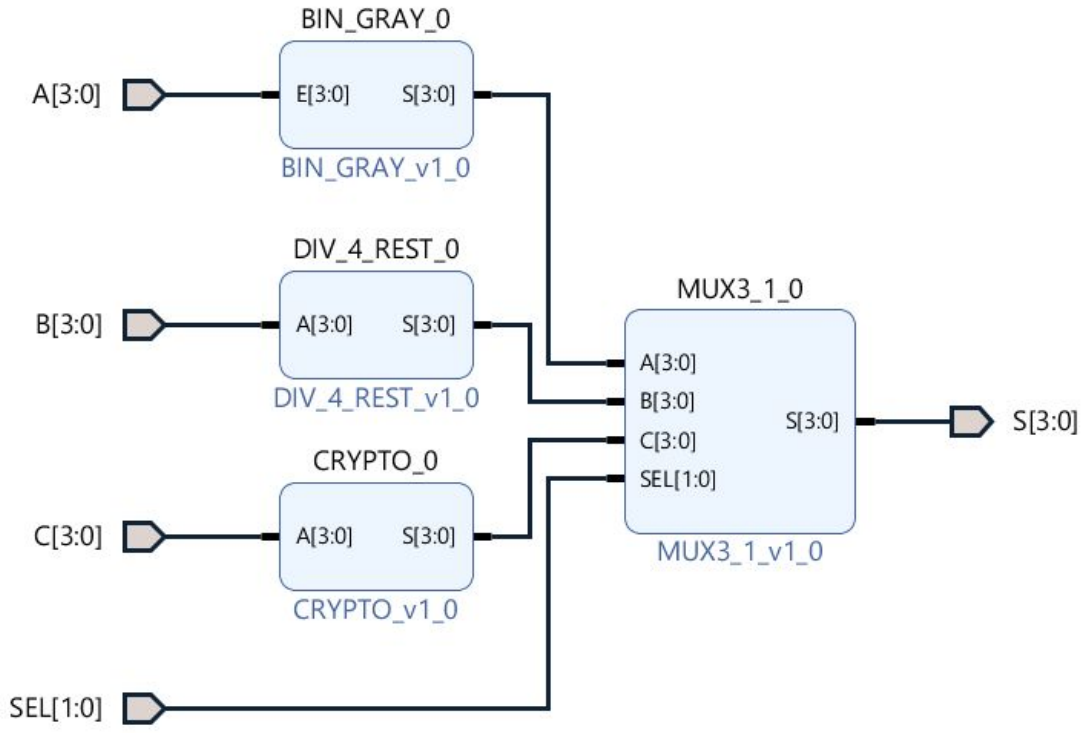


Figure 1.4 : Circuit Final.

2- Vérification du système

Des simulations des différents modules ainsi que du système entier sont nécessaires pour vérifier le bon fonctionnement du système. Un test exhaustif serait utile pour couvrir toutes les possibilités. On utilise le force clock, totalement ou partiellement dans toutes les simulations pour pouvoir valider chaque résultat avec les différentes tables de vérités, et cela sur un écran (une capture).

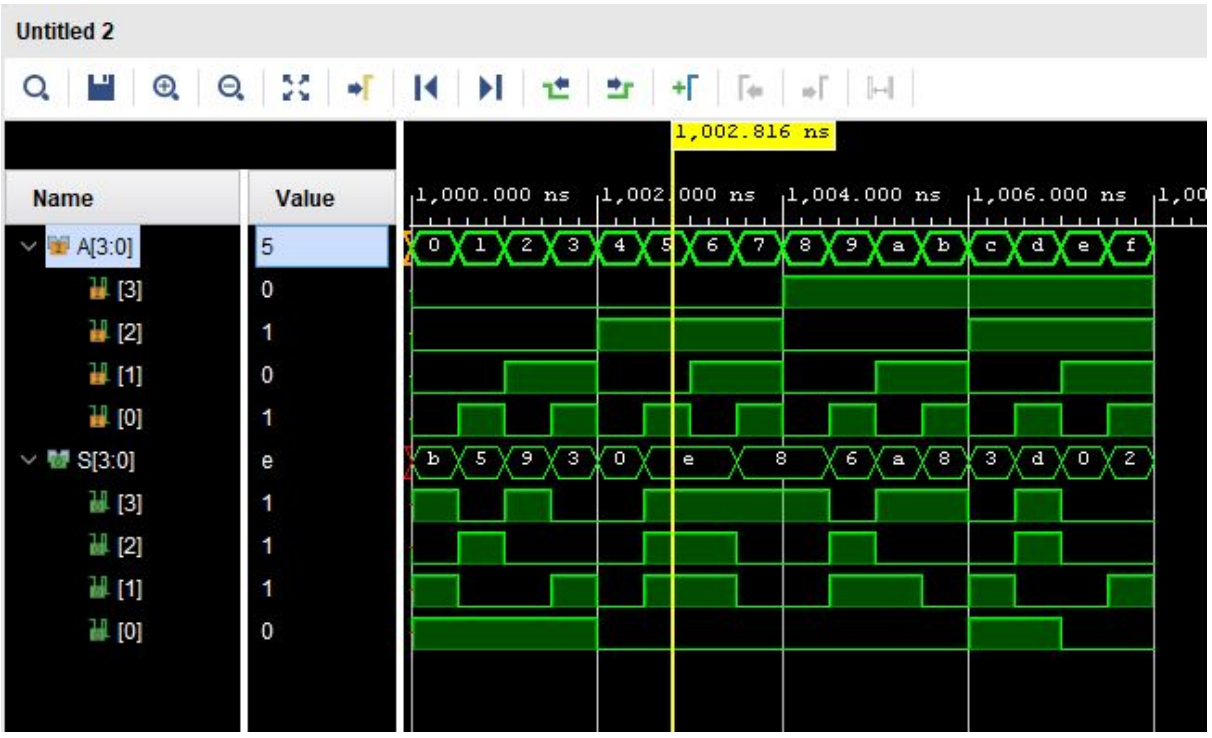


Figure 2.0 : Test exhaustif du CRYPTO.

Pour la vérification du **CRYPTO**, un force clock couvrira 16 possibilités (de **0** à **f** en hexadécimal**.** Pour le bus ***"A"***, nous avons ajusté les périodes d'alternances des bits: ***0*** à 1 ns, ***1*** à 2 ns, ***2*** à 4 ns, ***3*** à 8 ns. La figure 2.0 valide que le sous-système fonctionne correctement.

Par exemple, dans cette simulation, on choisit aléatoirement (ligne jaune), avec l'entrée **0101** (5 en hexadécimal). Tel que prévu théoriquement (tables de vérités et de Karnaugh), la sortie du sous-système **CRYPTO** va retourner **1110** (E en hexadécimal). Ceci affirme que ce sous système fonctionne correctement.

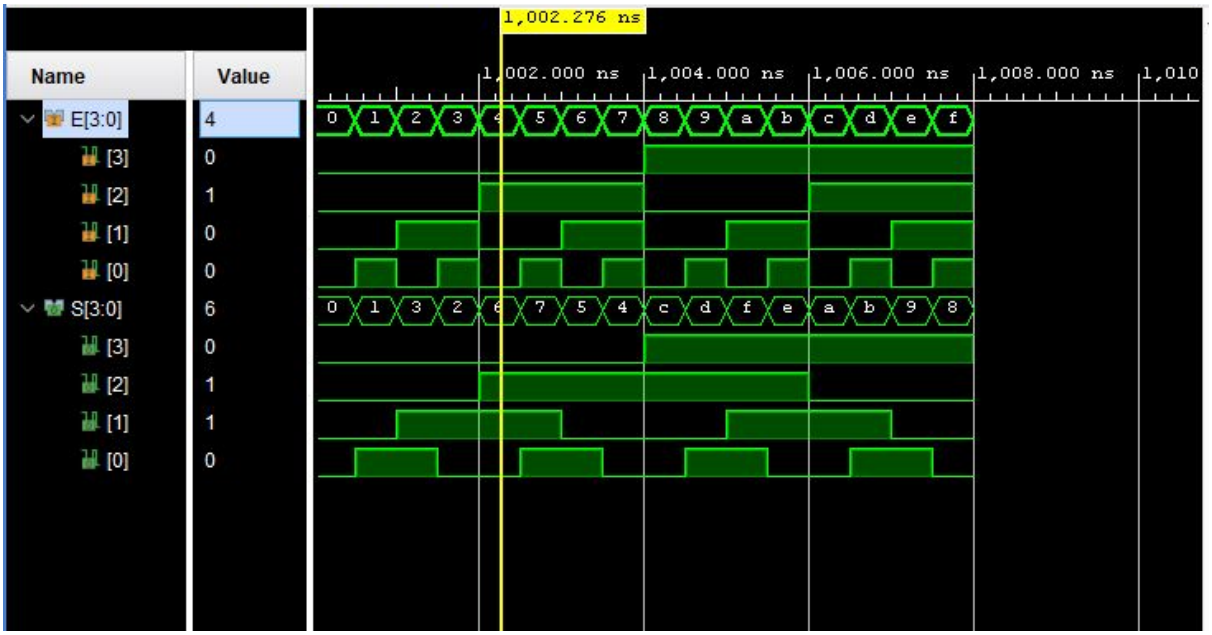


Figure 2.1 : Test exhaustif du BIN_GRAY.

Pour le test du **BIN_GRAY**, on reprend encore la même stratégie de test, c'est-à-dire que l'on va effectuer un force clock qui couvrira 16 possibilités (de **0** à **f** en hexadécimal) **.** Pour le bus ***"A"***, nous avons ajusté les bits: ***0*** à 1 ns, ***1*** à 2 ns, ***2*** à 4 ns, ***3*** à 8 ns. La figure 2.1 valide que le sous-système fonctionne correctement.

Par exemple, dans cette simulation, on choisit aléatoirement (ligne jaune), l'entrée 4 (0100 en hexadécimal). Tel que prévu théoriquement, le sous-système **BIN_GRAY** va effectuer une conversion en code gray. Comme on peut le voir dans la figure 2.1, la sortie est effectivement 6 (0110) . Les résultats concordent parfaitement avec notre table de vérité et de Karnaugh. Il est clair que ce sous-système fonctionne correctement.

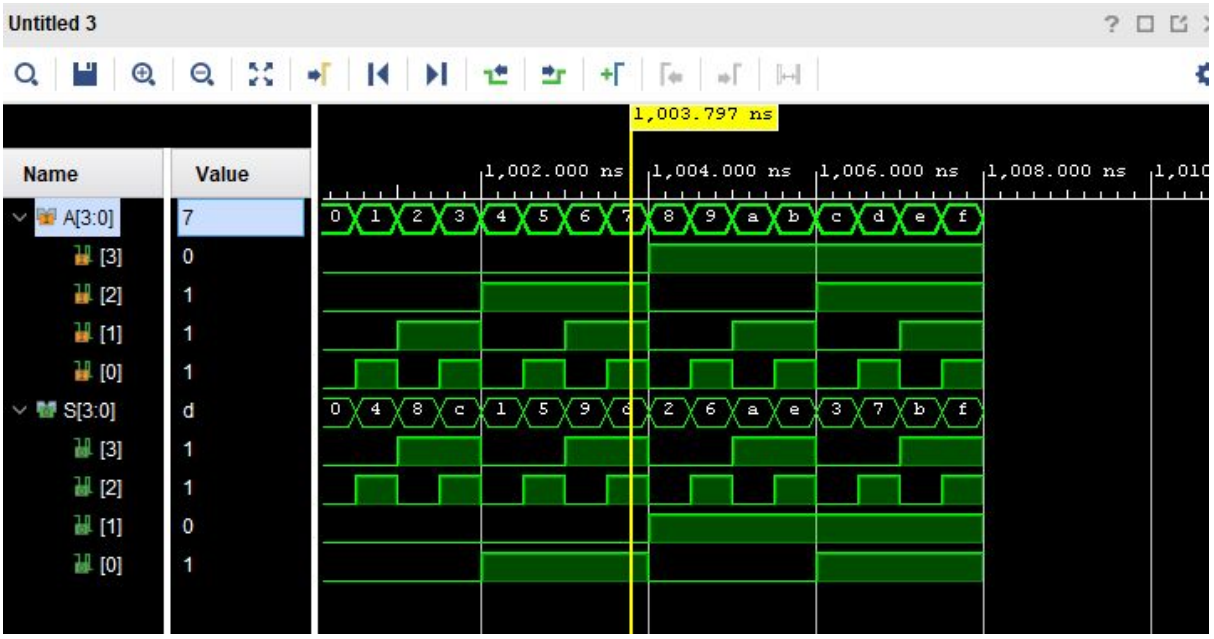


Figure 2.2 : Test exhaustif du DIV_4_REST.

Pour la vérification du diviseur par 4 **DIV_4_REST**, un force clock couvrira 16 possibilité de divisions: les 16 nombres de 4 bits **.** Pour le bus entrée ***"A"***, nous avons forcé les périodes suivantes: ***0*** à 1 ns, ***1*** à 2 ns, ***2*** à 4 ns, ***3*** à 8 ns. La figure 2.2 valide que ce sous-système fonctionne. Par exemple, dans la simulation, une division de **0111** (7 en hexadécimal) par 4, donnera **1101** qui correspond à d en hexadécimal. 7/4 = 1 (01) et reste 3 (11) et c'est pour cela que la sortie est 1101. La figure 2.2 valide le bon fonctionnement de notre système et concorde avec la table de vérité.

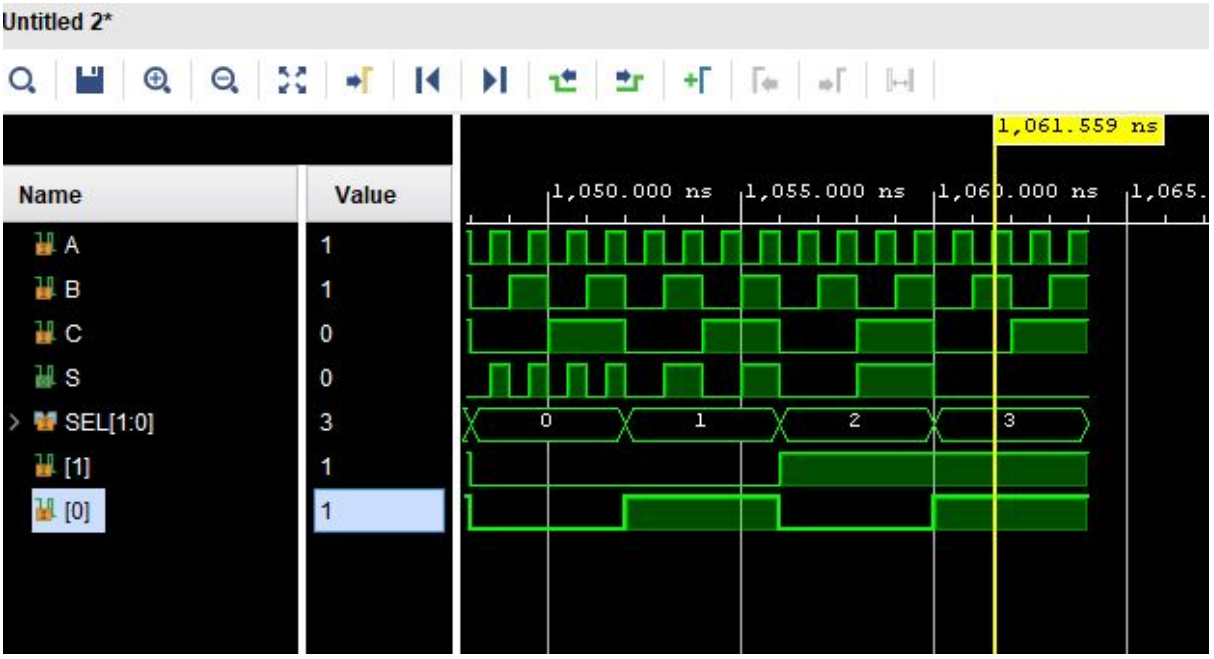


Figure 2.4: TEST du MUX_INDEX.

Pour le Multiplexeur indice, Un force clock est nécessaire pour s'assurer que le fonctionnement est bon, puisque dans ce système la sortie prend juste deux valeurs : 0 ou 1 un force clock est donc nécessaire pour montrer que , pour un certain signal de SEL , la sortie reste égale à l'entrée correspondante dans les différents cas. Exemple: pour SEL = 11, la sortie est toujours 0 quelles que soient les entrées.

Figure 2.5 : Test du MUX3_1

Dans la figure 2.5, une simulation est faite pour le multiplexeur MUX3_1.

- L'entrée A reçoit A(1010).
- L'entrée B reçoit B (1011).
- L'entrée C reçoit C(1100).
- un force clock pour les bits de SEL, pour s'assurer que le sélectionnement est le bon.

La figure 2.5 Montre que pour Sel = 00 la sortie est A, pour SEL =01 la sortie est B, pour SEL=10 la sortie est C, et pour SEL =11 , la sortie est 0.

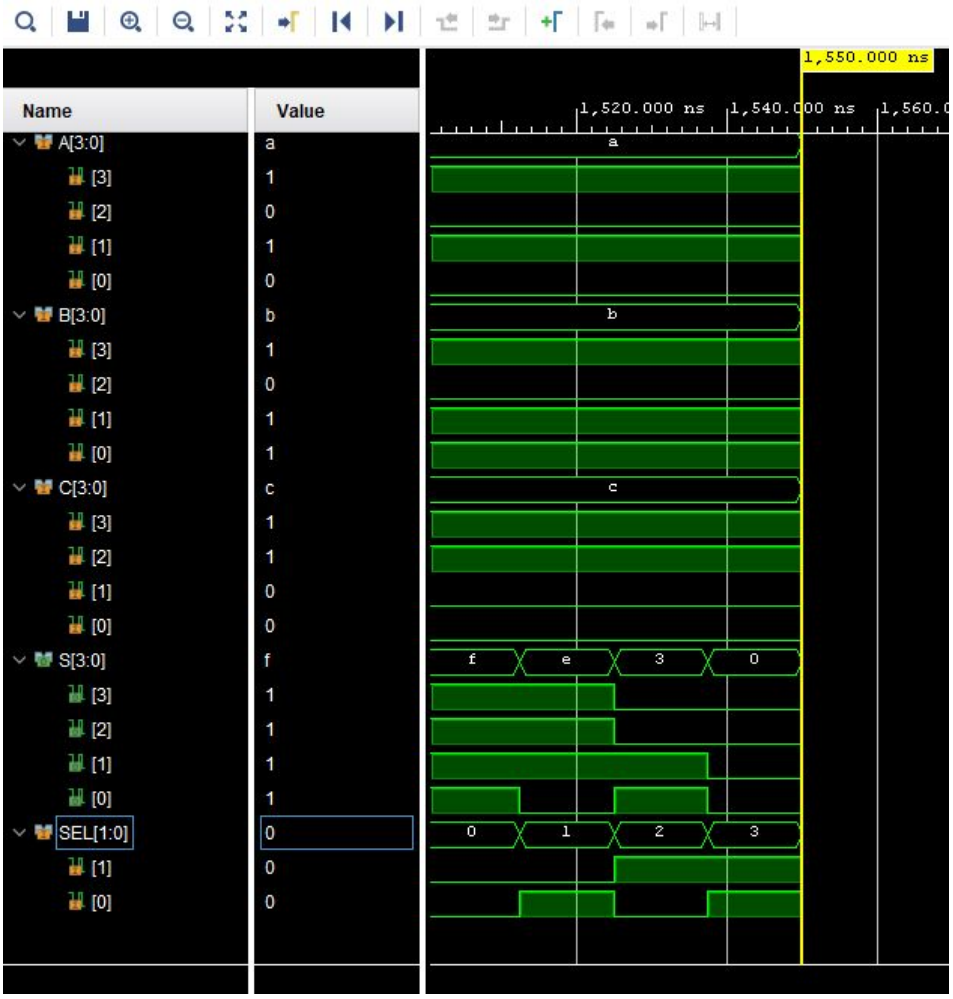


Figure 2.6 : Test exhaustif du système global.

Enfin, pour vérifier que le système en entier fonctionne, nous avons décidé un test partiellement exhaustif. Cette fois, nous avons mis le force clock seulement sur les 2 bits du SEL (le bit **0** à **25ns** et le bit **1** à **50ns**). Les sous-systèmes ont déjà été testé rigoureusement et individuellement, donc pour ce test du système global il n'est pas nécessaire de mettre un force clock pour chaque bit des bus A,B,C. Nous avons simplement ajusté la valeur du bus ***"A"*** à 1010 (A en hexadécimal), pour le bus ***"B"*** à 1011 (B en hexadécimal) et le bus ***"C"*** à 1100 (C en hexadécimal).

- Pour SEL=00, la sortie est F qui est sortie du BIN_GRAY si son entrée est A (1010)
- Pour SEL=01, la sortie est E qui est sortie du DIV_4_REST si son entrée est B(1011)
- Pour SEL =10, la sortie est 3, qui est sortie du CRYPTO si son entrée est C(1100)
- Pour SEL=11, la sortie est 0.

La figure 2.6 confirme le fonctionnement du système global.

Conclusion

Pour conclure, Le circuit obtenu est composé de 3 modules ainsi que d'un multiplexeur pour choisir la sortie. Tel qu'expliqué dans la description du système, la valeur de la sortie va dépendre de la valeur de SEL, et selon cette dernière valeur, le système va soit procéder à la conversion en code gray (module BIN_GRAY), soit faire une division par 4 avec un reste (module DIV_4_REST) soit une conversion crypto (module CRYPTO), soit il va faire sortir 0. Pour réaliser chacun de nos modules, nous avons utilisé les tables de vérités, les tables de Karnaugh ainsi que les équations simplifiés que nous avons trouvé. Finalement, après avoir construit nos circuits sur Vivado, nous les avons tous testé pour vérifier, à l'aide des simulations, le bon fonctionnement du système et des sous-systèmes. Bref, avec l'aide des tables, de la théorie et des simulations, nous avons pu réaliser et valider ce circuit du laboratoire 3.