

Compte rendu TP Information numérique : Unité de calcul arithmétique

Sommaire :

Introduction :	Page 2
Réalisation d'additionneurs	Page 3
Additionneur à propagation de retenue	Page 3
Additionneur à retenue anticipée	Page 9
Comparaison du système à retenue anticipée et à propagation	Page 12
Réalisation de multiplieurs	Page 13
Conclusion	Page 19

Note obtenue : 17/20

Introduction :

Depuis toujours, nous utilisons nos calculatrices, sans jamais nous demander comment cela fonctionne. Lors de ce projet, nous nous sommes demandés « De quelles façons nos ordinateurs réalisent-ils ces opérations ? »

Nous avons donc étudié le fonctionnement de 2 types d'additionneurs : La propagation de retenue et la retenue anticipée

Puis, nous avons étudié un système multiplicatif à propagation de retenue.

Séance n°1 : Réalisations d'additionneurs

Réalisation d'un additionneur à propagation de retenue

Préparation :

Retenues : 1 0111 1000|2

Opérande A : 1011 1101|2

Opérande B : + 1011 0100|2

Somme Totale : = 1 0111 0001|2

La retenue entrante de poids 0 est la retenue de bit de poids faible sur la ligne "retenues". Ici, on additionne deux mots de 8 bits, on obtient donc un nombre de 9 bits. La retenue se propage du bit de poids le plus faible vers le bit de poids le plus fort.

Demi-additionneur (DA)

Voici la table de vérité d'un demi-additionneur de deux mots de 1 bit, avec S la sortie et R la retenue :

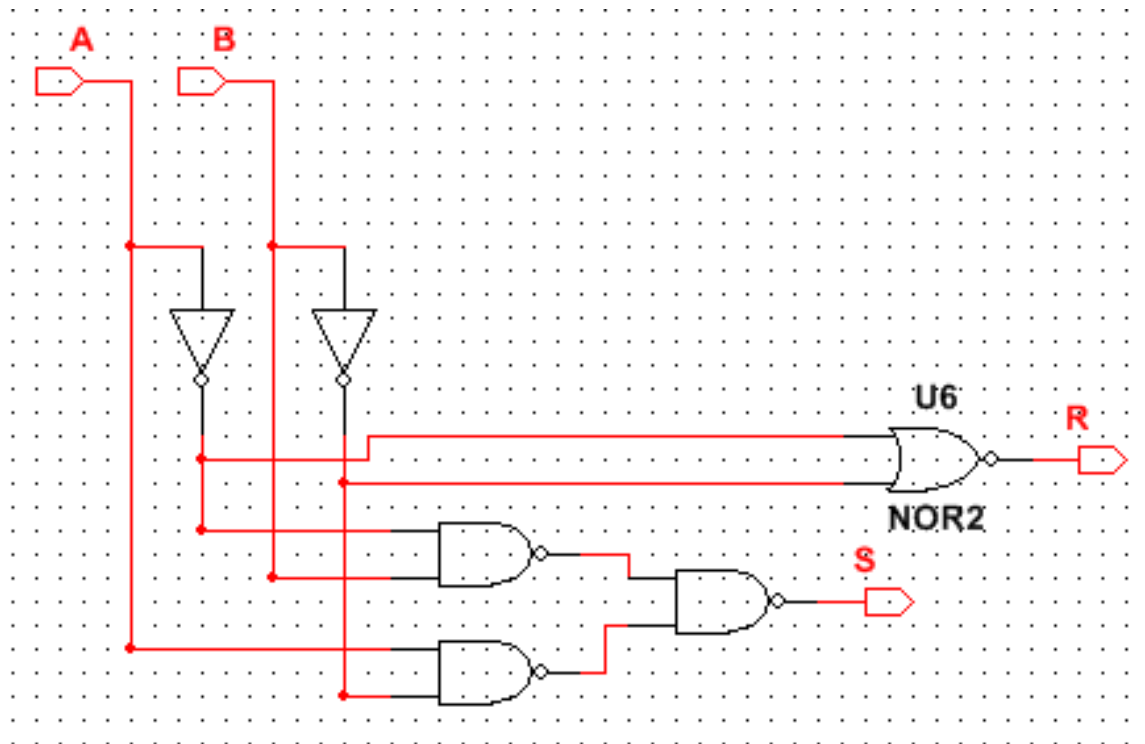
A	B	S	R
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

On obtient les équations logiques suivantes :

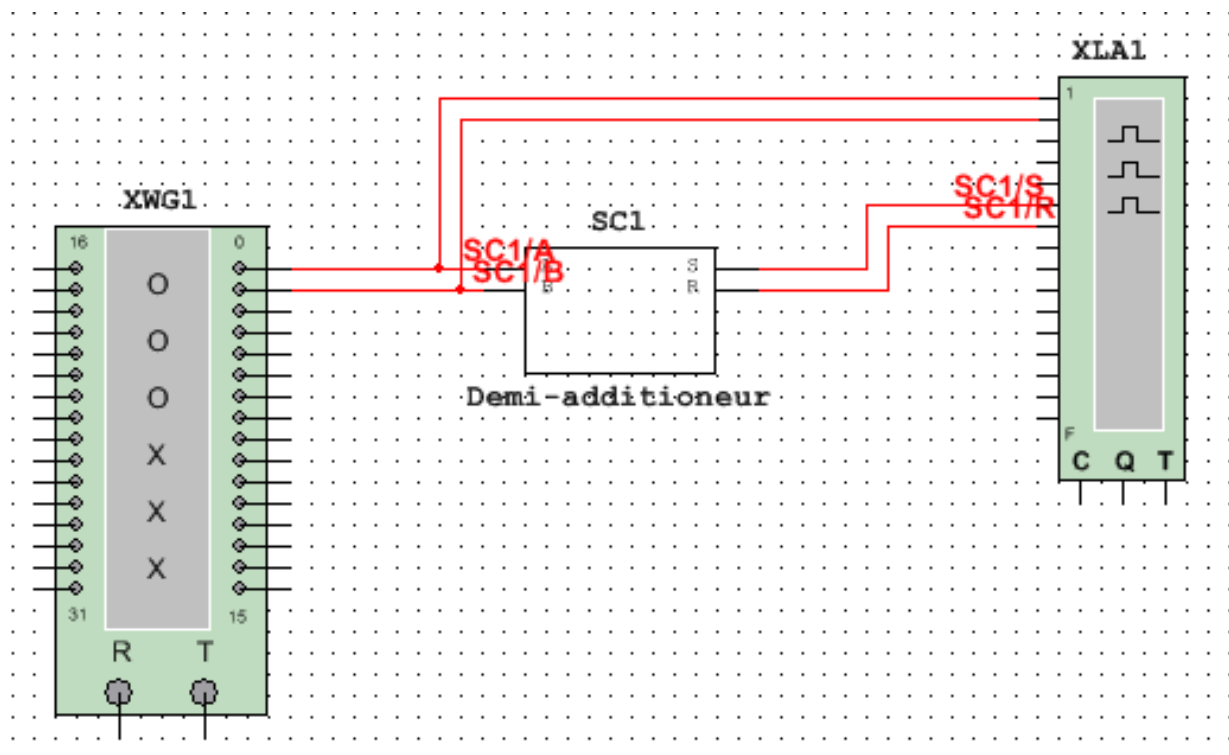
$$S = A'B + AB'$$

$$R = AB$$

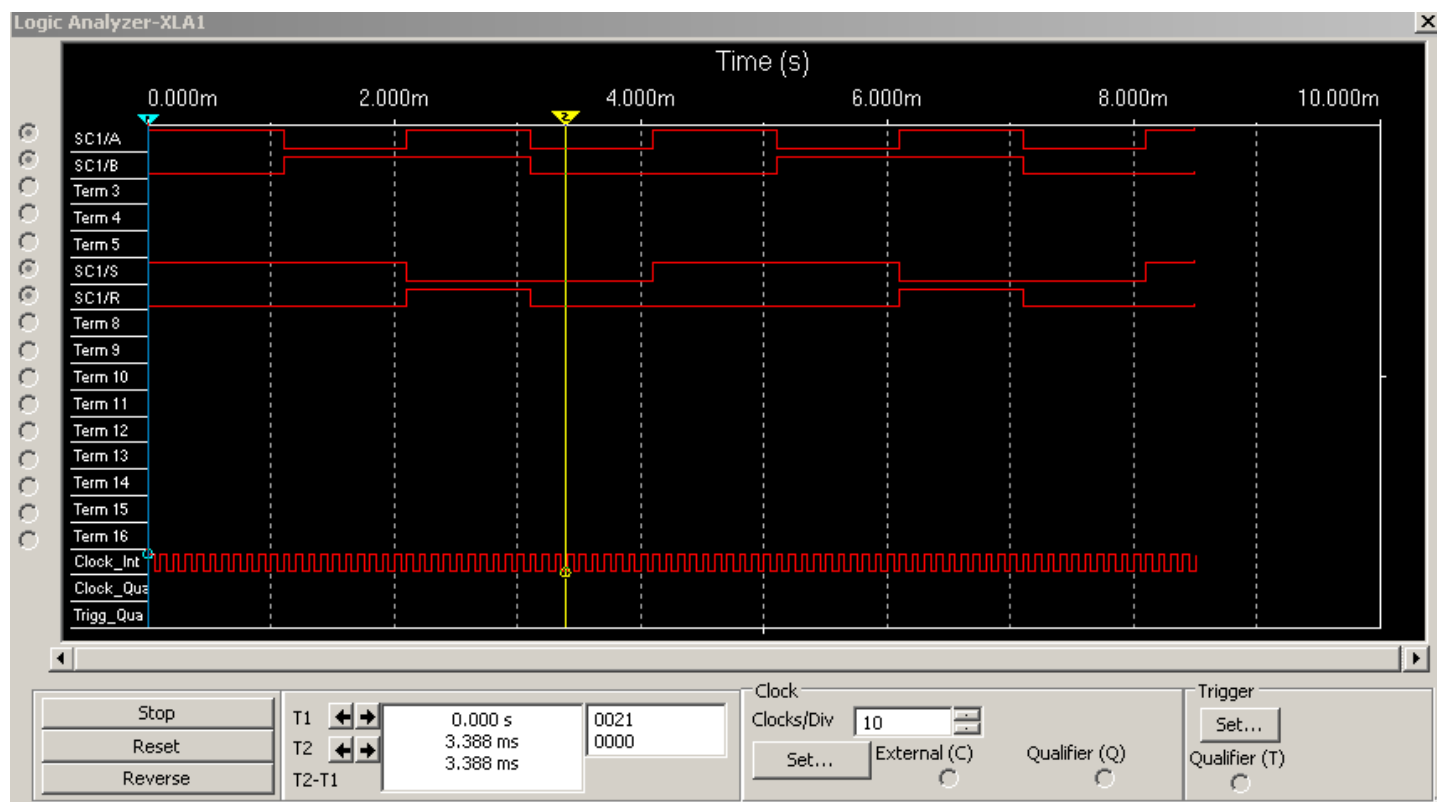
Un schéma logique simplifié de S et de R avec des portes INV, NAND et NOR est le suivant :



On réalise le montage suivant avec un *Word Generator* et un *Logic Analyser* pour vérifier que le demi-additionneur marche. Nous avons directement placé le montage du demi-additionneur dans un sous-circuit :



Voici ce que donne le *Logic Analyser*, sur lequel on peut vérifier la table de vérité de du demi-additionneur :



Additionneur complet (AC)

En préparation, nous avons du réaliser la table de vérité d'un additionneur complet :

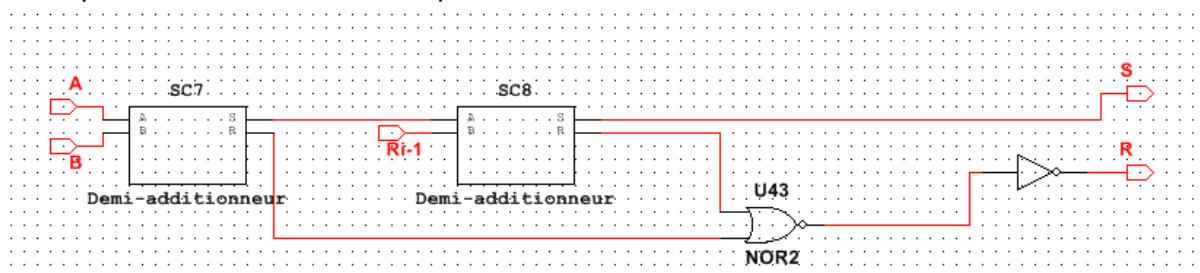
A	B	Ri-1	S	R
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

On obtient donc ces équations logiques simplifiées pour S et R

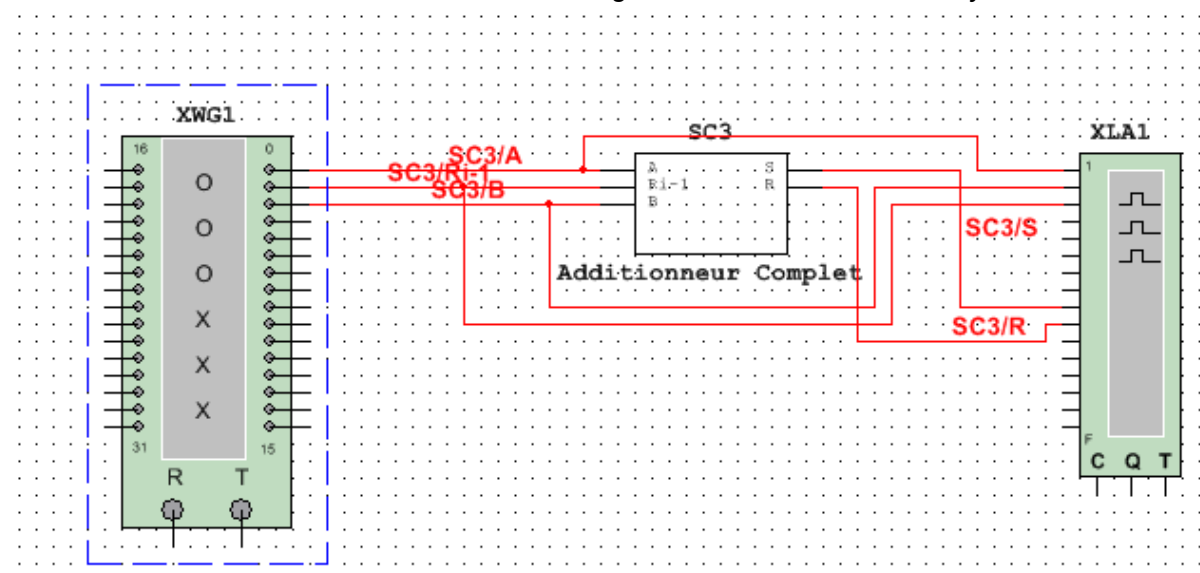
$$S = A'B'Ri-1 + A'BRi-1' + AB'Ri-1' + AB Ri-1$$

$$R = Ri-1B + ARi-1 + AB$$

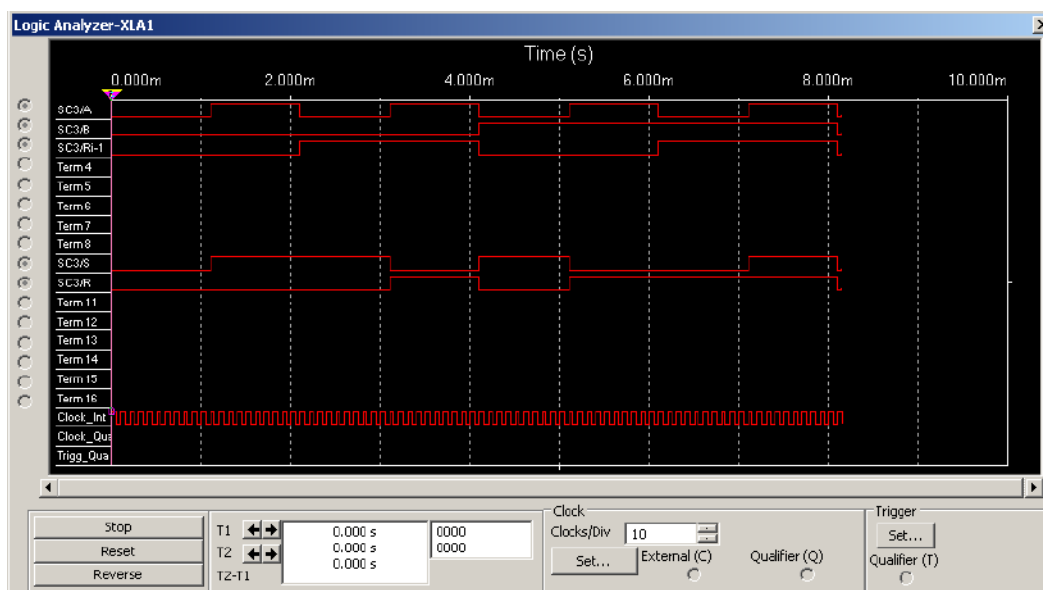
Nous avons ensuite réalisé le montage de l'additionneur complet avec deux demi-additionneurs et des portes INV et NOR à partir de la table de vérité de l'Ac et de celle du DA.



En plaçant ce montage dans un sous-circuit, on réalise ces branchements avec le *Word Generator* et le *Logic Analyser* :

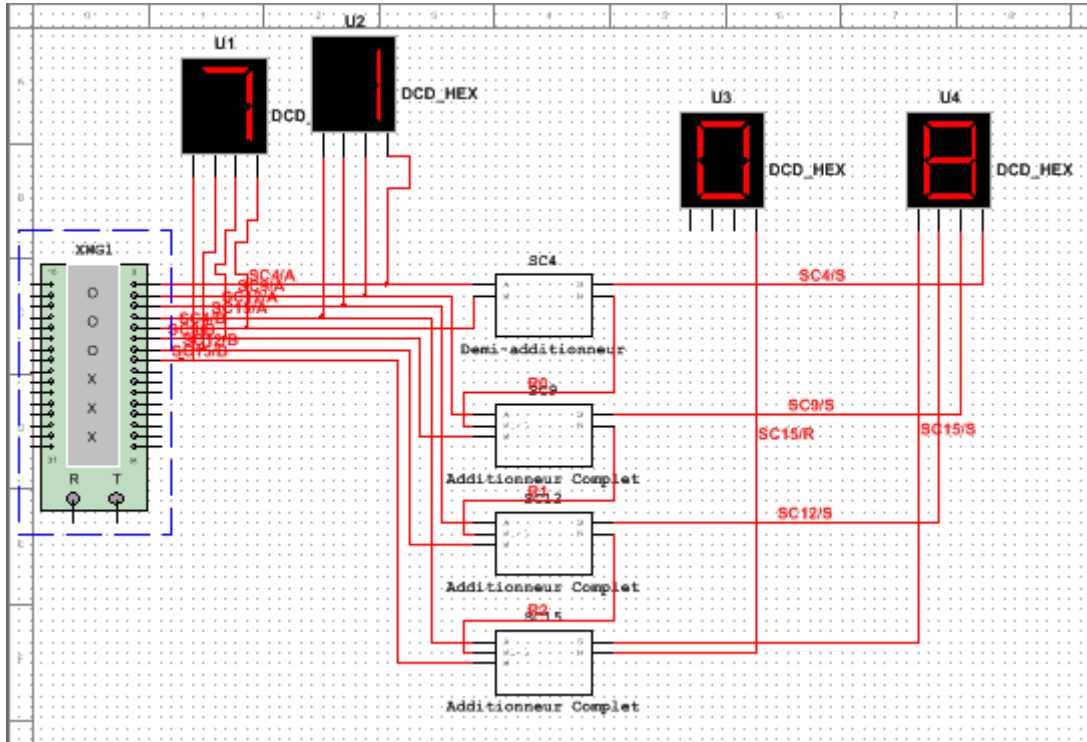


Dans le *Logic Analyzer*, on a ces courbes qui montrent que l'additionneur complet fonctionne correctement :



Additionneur de 2 nombres de 4 bits à propagation de retenue

Le circuit A4b correspond à ce montage avec un DA et trois AC. On connecte des afficheurs 7 segments pour pouvoir vérifier que l'additionneur fonctionne correctement :



Grâce à cette image, on peut vérifier que l'additionneur est bon puisque $7 + 1 = 8$ en sortie et la retenue est nulle.

Réalisation d'un additionneur à retenue anticipée

D'après la table de vérité de l'additionneur complet,
 $R = AR_{i-1} + BR_{i-1} + AB$ $S = A'B'R_{i-1} + A'BR_{i-1}' + AB'R_{i-1}' + AB R_{i-1}$
 $R = R_{i-1}(B+A) + AB$ Or $G_i = AB$ et $P_i = A+B$

Donc $R = R_{i-1}P_i + G_i$

$S = A$ OU exclusif B OU exclusif R_{i-1}

Réalisation d'un additionneur 4 bits à retenue anticipée

En utilisant la relation de récurrence $R = R_{i-1}$ on exprime R_0 , R_1 , R_2 et R_3

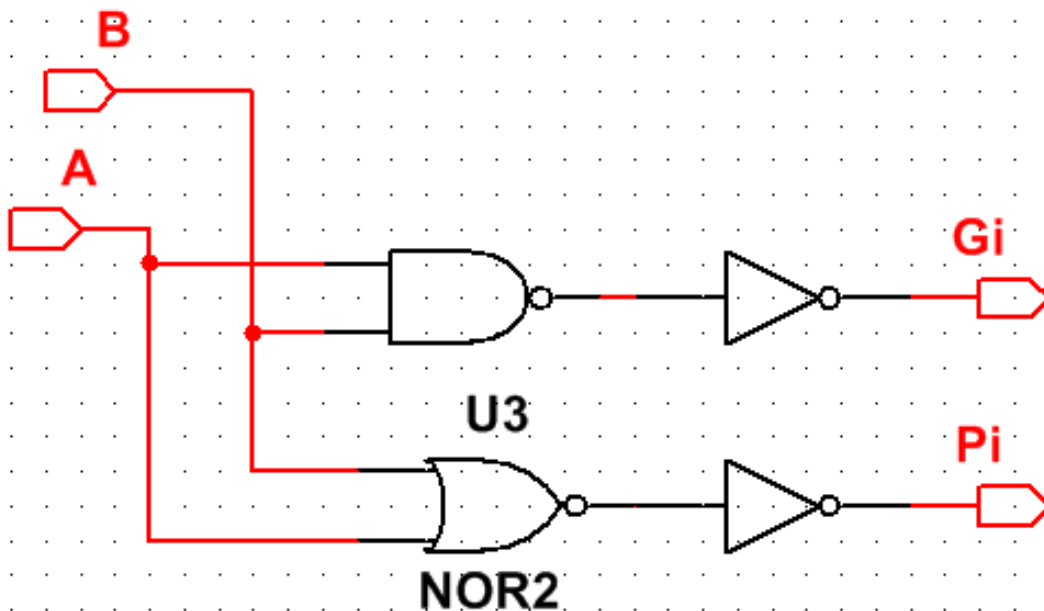
$$R_0 = G_0$$

$$R_1 = R_0P_1 + G_1 = P_1G_0 + G_1$$

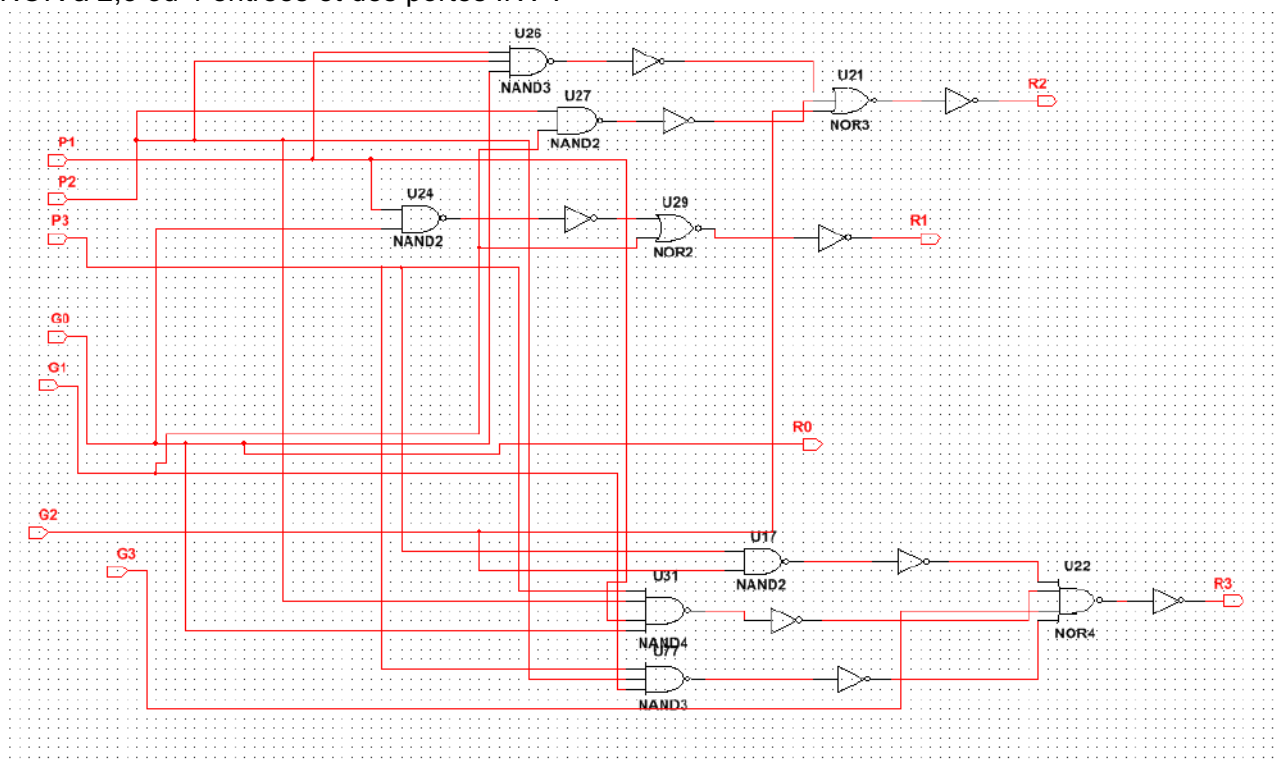
$$R_2 = R_1P_2 + G_2 = P_2P_1G_0 + P_2G_1 + G_2$$

$$R_3 = R_2P_3 + G_3 = P_3P_2P_1G_0 + P_3P_2G_1 + P_3G_2 + G_3$$

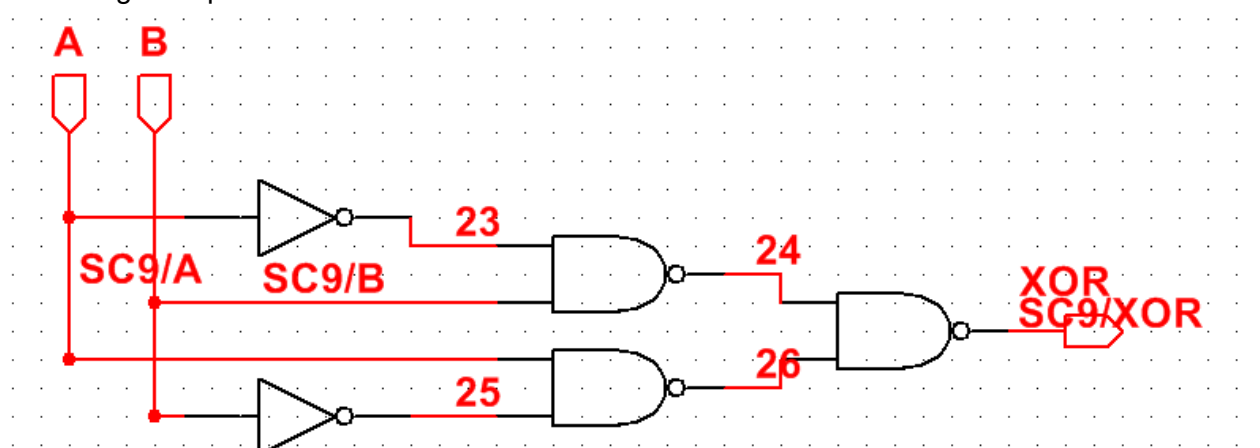
On réalise ce montage pour les G_iP_i avec des portes INV, NAND et NOR :



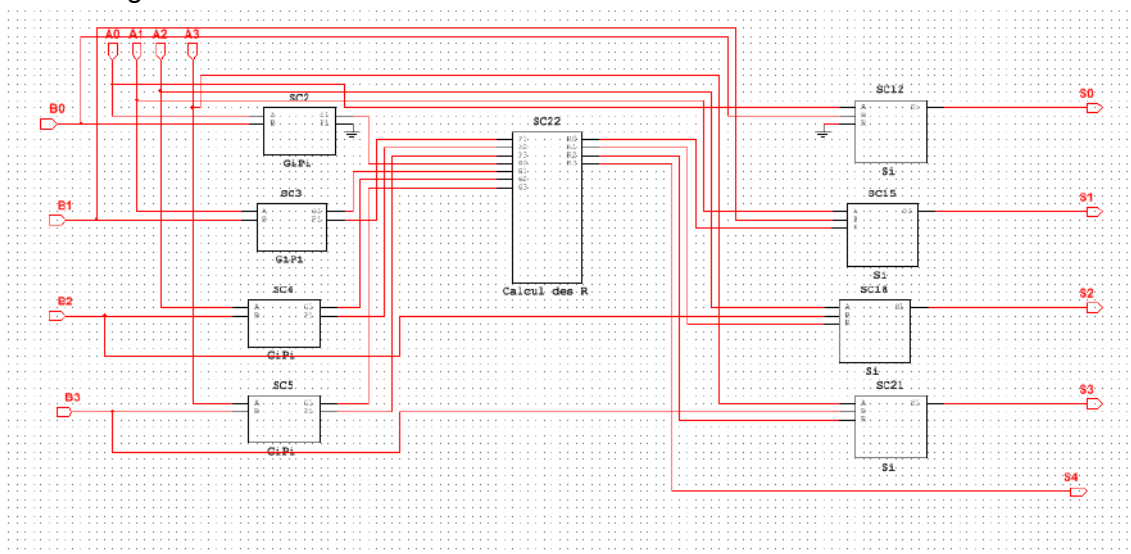
On réalise aussi le montage du calcul des R avec des portes NAND à 2,3 ou 4 entrées, des portes NOR à 2,3 ou 4 entrées et des portes INV :



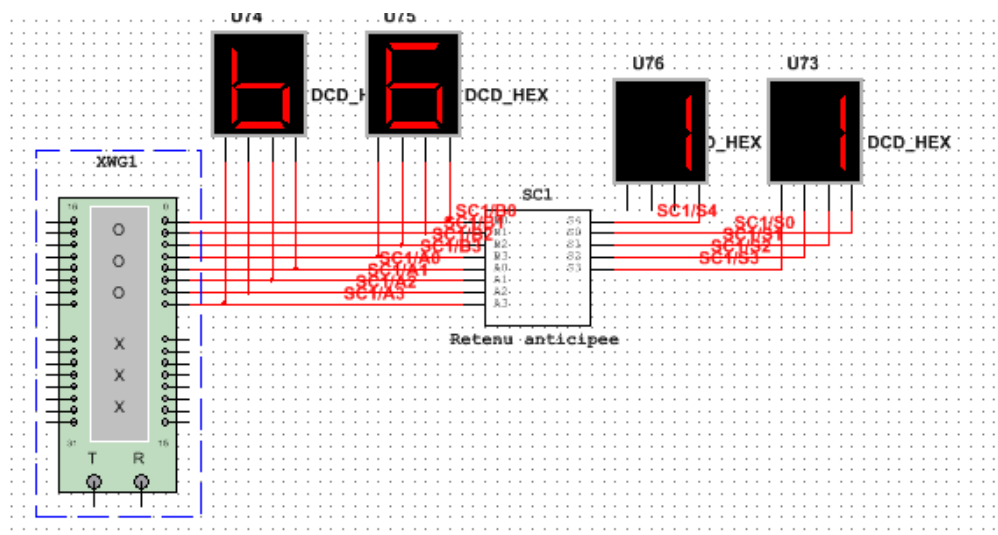
Le circuit générique des XORi est le suivant :



Le montage final avec les Si est celui-ci :



On vérifie le bon fonctionnement de ce montage avec des afficheurs, un *Word Generator* et un *Logic Analyser* :



Comparaison entre le système à propagation de retenu et à retenu anticipé :

On cherche ensuite à comparer les deux additionneurs que nous avons fait :

En supposant que le passage dans une porte logique dure 10ns entre l'entrée et la sortie, le temps de passage maximal est :

- Dans un demi-additionneur : 30ns
- Dans un additionneur complet : 80ns
- Dans un additionneur de 4 bits à propagation : 270ns
- Pour calculer les GiPi : 20ns
- Pour le calcul de R_3 : 30ns
- Pour le calcul de S_3 : 60ns
- Additionneur de 4 bits à anticipation de retenue : 110ns

On voit donc que le système à retenue anticipée est nettement plus rapide que celui à propagation de retenue. Cependant, le système à propagation de retenue n'utilise que 2 circuits différents (demi additionneur et additionneur complet) alors que le système à retenue anticipée utilise 3 circuits (Calcul des GiPi, Calcul des R et calcul des Si (qui regroupe deux circuits XOR)).

L'additionneur à propagation de retenue utilise donc moins de portes, consomme moins d'énergie et est moins cher à produire. Mais il est plus de 2 fois plus lent.

Partie 2 : Réalisation de multiplieurs

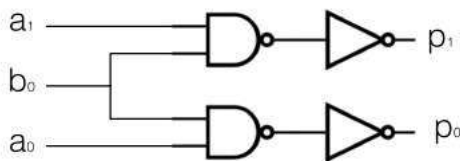
Brique élémentaire M (Multiplieur 2x1) :

On cherche à multiplier un mot d'un bit avec un mot de 2 bits.

Le produit de 2 opérandes A(a_1, a_0) et B (b_0) est P(p_1, p_0) :

A	a_1	a_0
B	x	b_0
P	$p_1 = a_1 b_0$	$p_0 = a_0 b_0$

On obtient alors le circuit suivant :



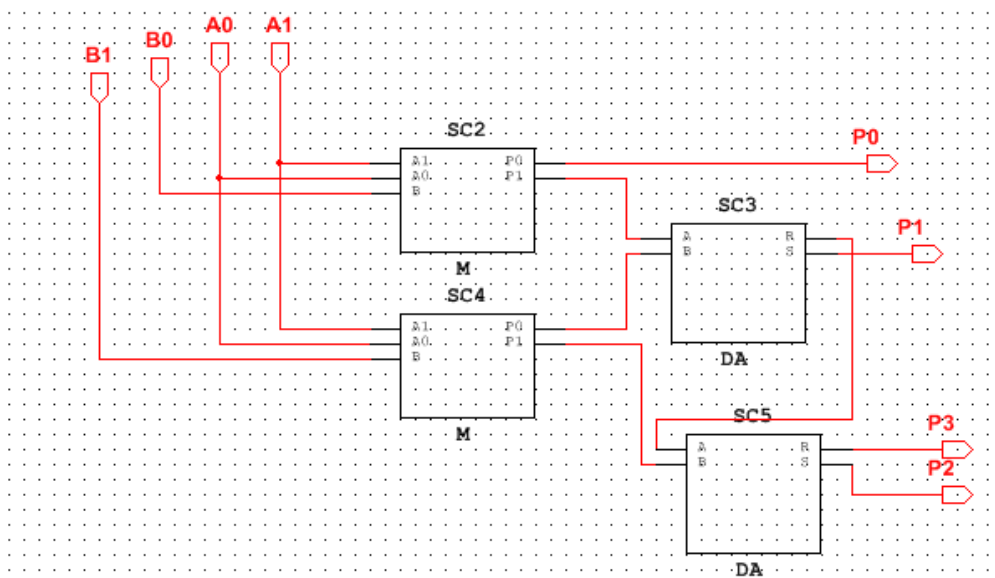
Multiplieur 2x2 :

On cherche maintenant à multiplier 2 mots de 2 bits :

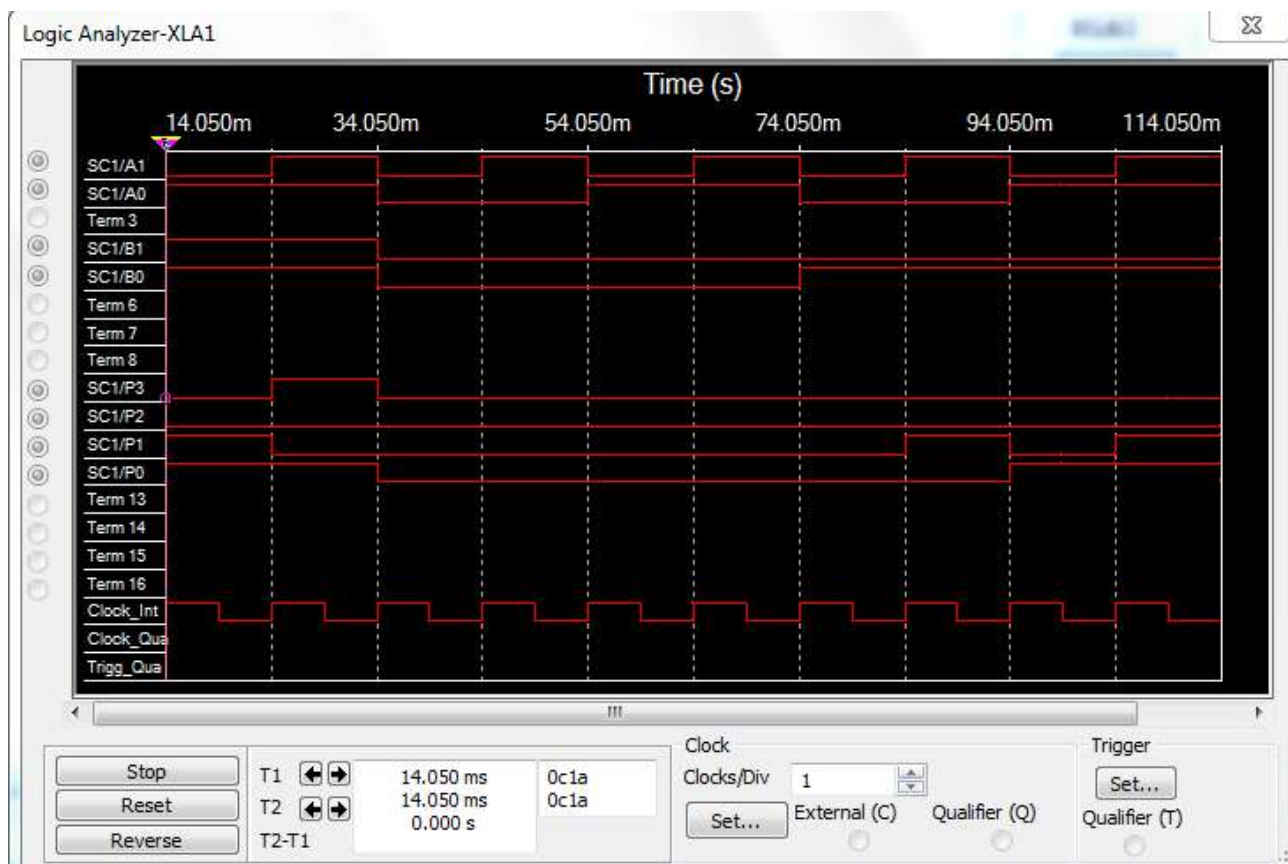
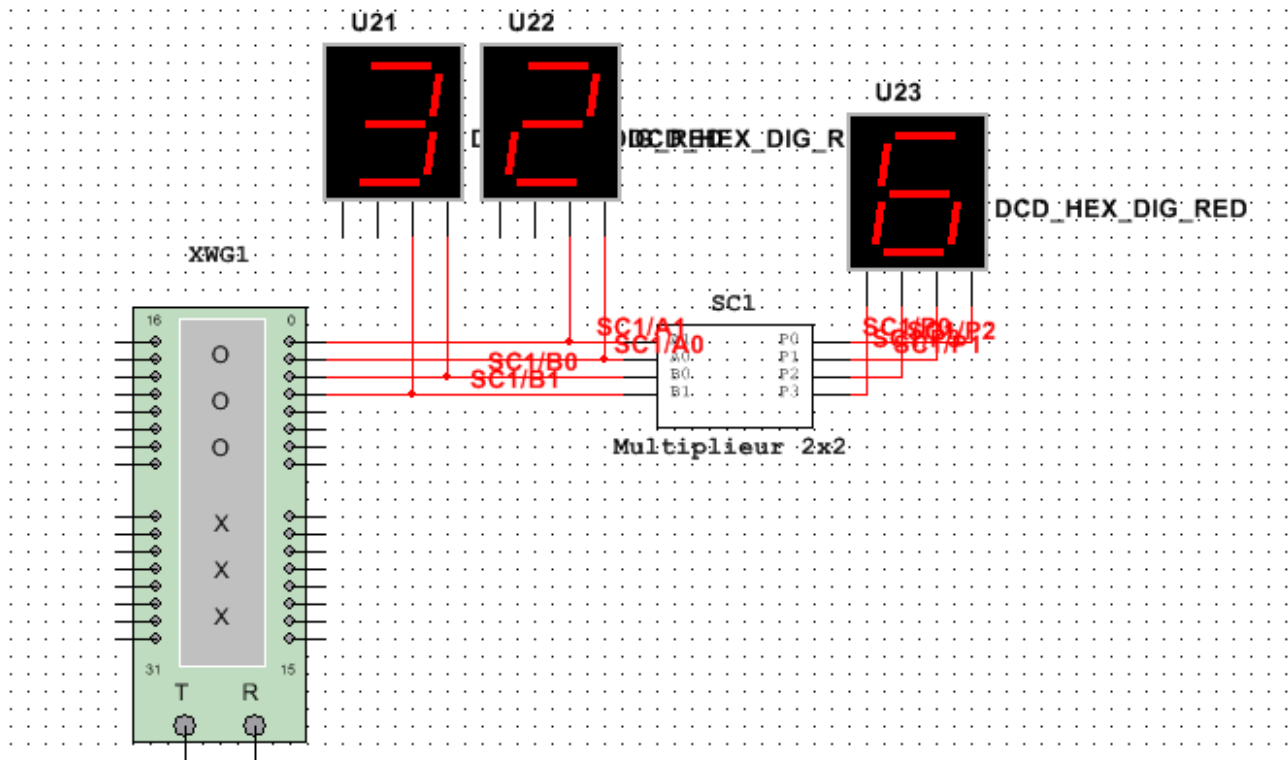
A			a_1	a_0
B		x	b_1	b_0
P	$p_3 = rp_2$	$p_2 = rp_1 + a_1 b_1$	$p_1 = a_1 b_0 + a_0 b_1$	$p_0 = a_0 b_0$

(rp_x : retenu de p_x)

On obtient le circuit suivant grâce à 2 demi-additionneurs et 2 multiplieurs élémentaires) :



On vérifie le bon fonctionnement de ce montage avec des afficheurs, un *Word Generator* et un *Logic Analyzer* :

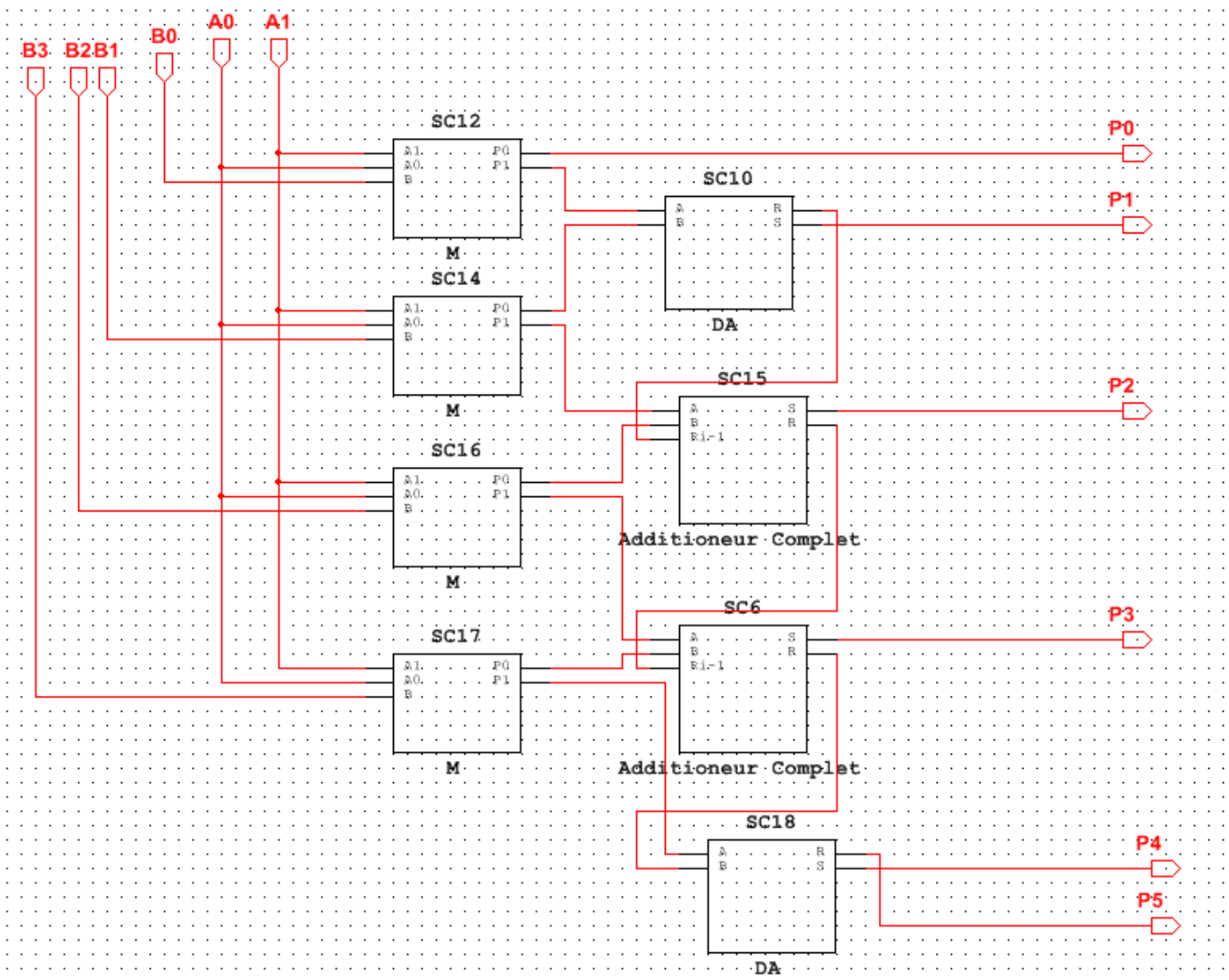


Multiplieur 2x4 :

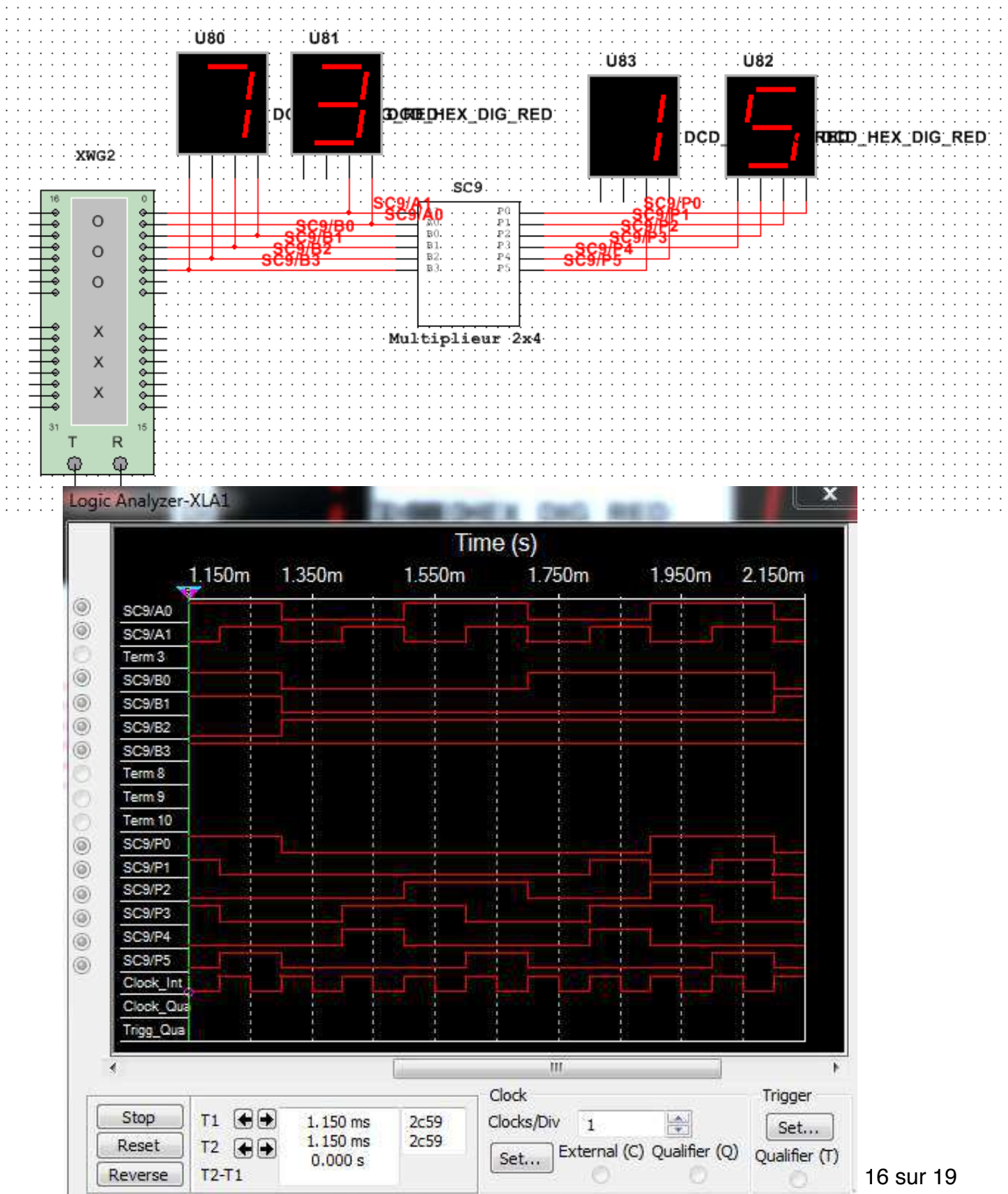
On cherche ensuite à multiplier un mot de 4 bits et un mot de 2 bits :

A					a_1	a_0
B			b_3	b_2	b_1	b_0
P	$p_5 = r p_4$	$p_4 = r p_3 + b_3 a_1$	$p_3 = r p_2 + b_2 a_1 + b_3 a_0$	$p_2 = r p_1 + a_1 b_1 + b_2 a_0$	$p_1 = a_1 b_0 + a_0 b_1$	$p_0 = a_0 b_0$

On obtient le circuit suivant grâce à 2 demi- additionneurs, 2 additionneurs complets et 4 multiplieurs élémentaires :



On vérifie le bon fonctionnement de ce montage avec des afficheurs, un *Word Generator* et un *Logic Analyzer* :



Multiplieur 4x4 :

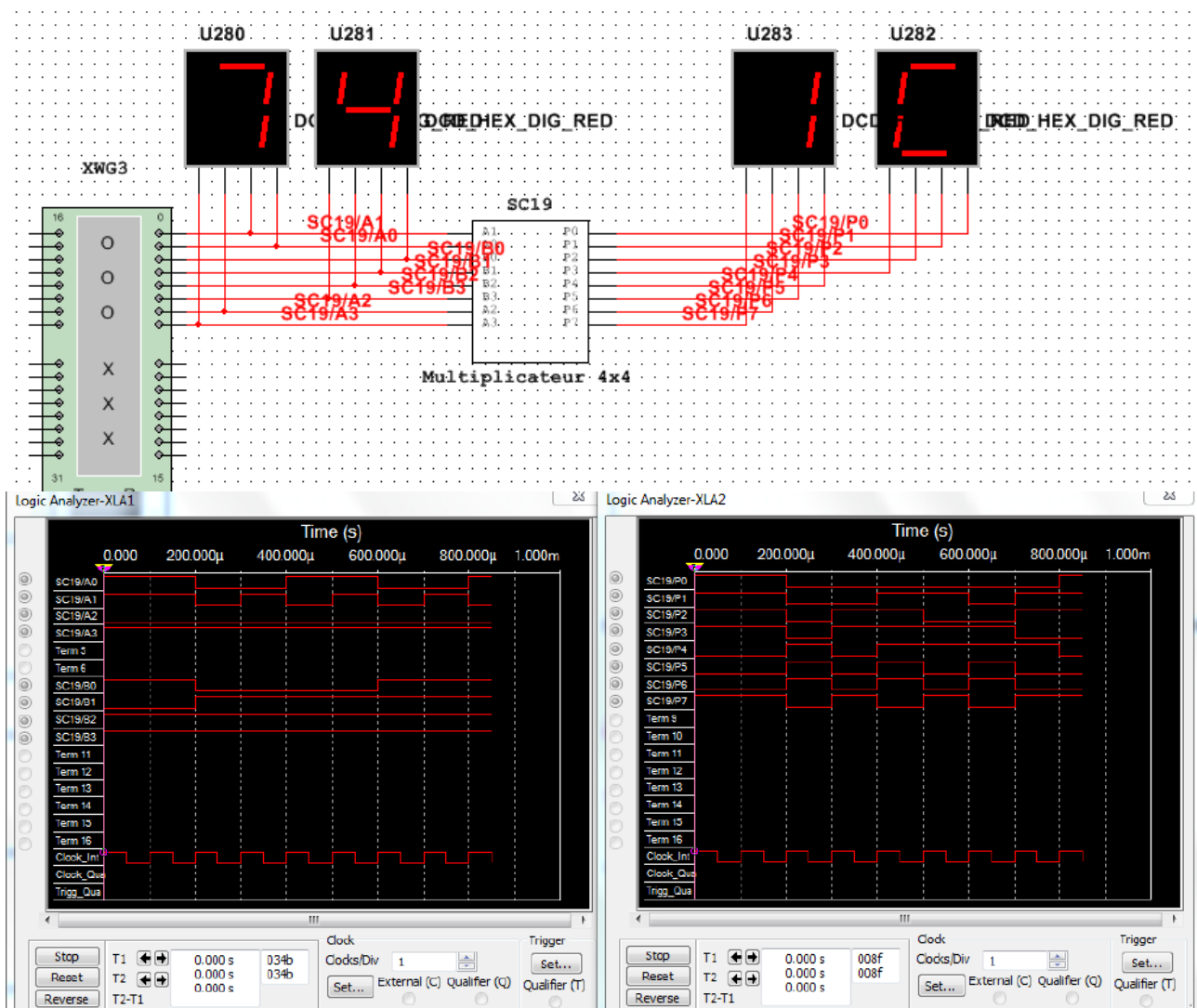
Pour terminer, nous cherchons à faire un multiplieur de 2 mots de 4 bits :

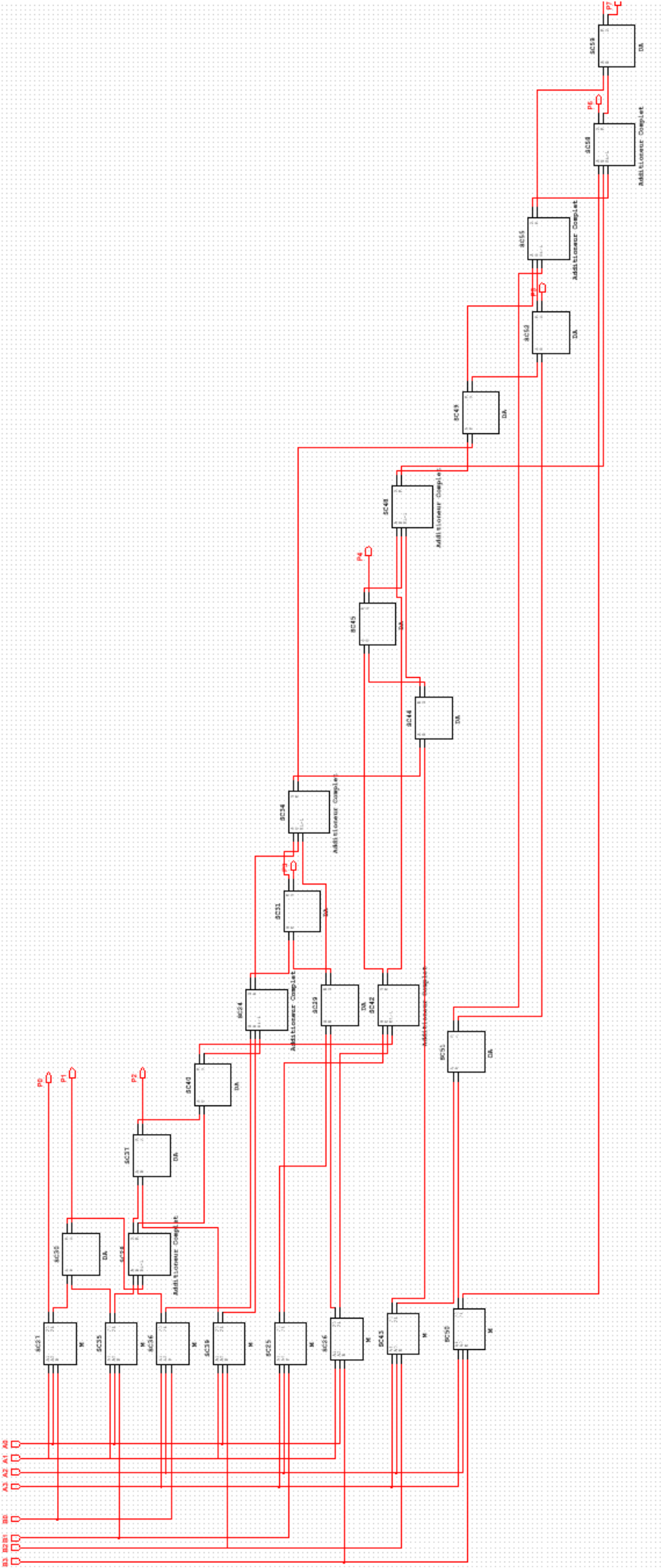
A					a_3	a_2	a_1	a_0
B					b_3	b_2	b_1	b_0
P	$p_7 = rp_6$	$p_6 = rp_5 + a_3b_3$	$p_5 = rp_4 + b_2 a_3 + b_3 a_2$	$p_4 = rp_3 + b_3 a_1 + b_1 a_3 + b_2 a_2$	$p_3 = rp_2 + b_2 a_1 + b_3 a_0 + b_0 a_3 + b_1 a_2$	$p_2 = rp_1 + a_1 b_1 + b_2 a_0 + b_0 a_2$	$p_1 = a_1 b_0 + a_0 b_1$	$p_0 = a_0 b_0$

On obtient le circuit suivant grâce à 8 multiplieurs (les sous-circuits de gauche), 7 additionneurs complets (les sous-circuits avec un nom long) et 11 demi additionneurs (les sous circuits avec un nom court).

Par soucis de place, le circuit est sur la page suivante.

On vérifie le bon fonctionnement de ce montage avec des afficheurs, un *Word Generator* et un *Logic Analyser* :





Conclusion :

Le but de ce projet était d'apprendre les bases des portes logiques à travers le logiciel Multisim. Nous avons dû réaliser différents types d'additionneurs et de multiplieurs. Les différents montages réalisés nous ont permis de réduire le nombre de portes logiques, augmentant la rapidité d'exécution et réduisant les ressources nécessaires à ces circuits. Cependant, nous pouvons nous demander si à la manière de l'additionneur à retenue anticipée, il n'existe pas un multiplicateur à retenue anticipée permettant d'augmenter la vitesse d'exécution.