

Rapport du PR202

Interface souris (PS/2)

Chok Leong CHAI
Mathieu DAHNE
Ting DU
13/06/2008
Groupe 2
I2

Sommaire

	Page
Introduction	2
a) Le schéma de l'architecture générale	2
b) Le schéma de l'architecture générale avec des signaux entrées et sorties	3
1. Architecture du FPGA	4
1.1. Machine d'états	5
2. Architectures des sous-blocs de FPGA	6
2.1. Registres33	6
2.2. Detect	6
2.3. Compteur	7
2.4. Enable_test	7
2.5. Test_souris	8
2.6. Data_ready	9
2.7. Position	10
2.7.1. X_position	11
2.7.2. Y_position	12
2.8. VGA_ctrl	13
2.8.1. Diviseur	13
2.8.2. Compteur_h	14
2.8.3. Compteur_v	14
2.8.4. RGB	15
2.8.5. Couleur_ctrl	15
2.9. Afficheurs7seg	16
2.10. Clic_ctrl	17
3. Brochage sur la maquette	18
4. Répartition du travail	19
5. Quelques captures d'écran	20
5.1. Behavioral Simulation	20
5.2. Post-Map Simulation	21
5.2.1. Zoom sur le point 1	22
5.2.2. Zoom sur le point 2	23
6. Difficultés rencontrées	24

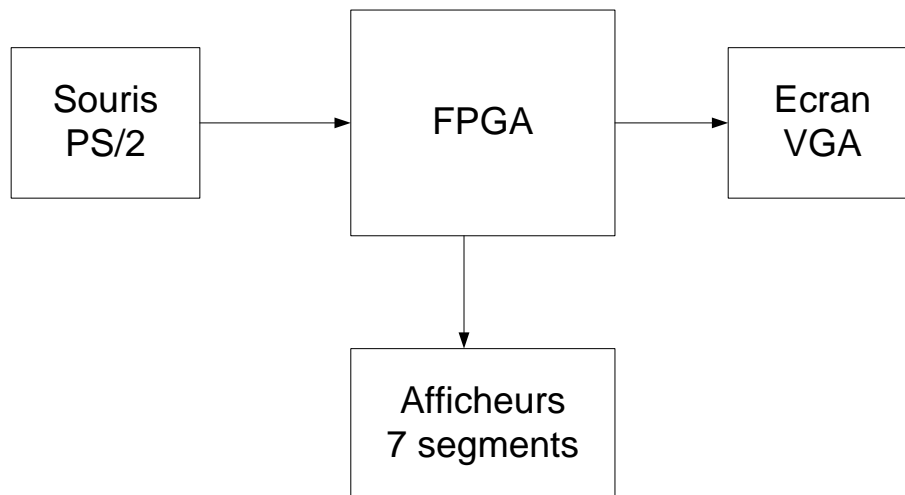
Introduction

Objectif : C'est une version simplifiée de pointeur de souris informatique. On doit pouvoir déplacer un curseur sur un écran VGA à l'aide d'une souris PC de type PS/2. Lorsqu'on bouge la souris, un curseur (sous forme d'un pixel ou d'un simple carré lumineux) se déplace sur l'écran sans dépasser les bords.

En première phase, on pourra d'abord afficher la position du curseur sur les afficheurs 7 segments puis, en deuxième étape, le curseur sera affichée sur un écran PC de type VGA.

(Source : Sujet du projet récupéré de l'adresse <http://intra.esiee.fr/~jeangeoa/>)

a) Le schéma de l'architecture générale :



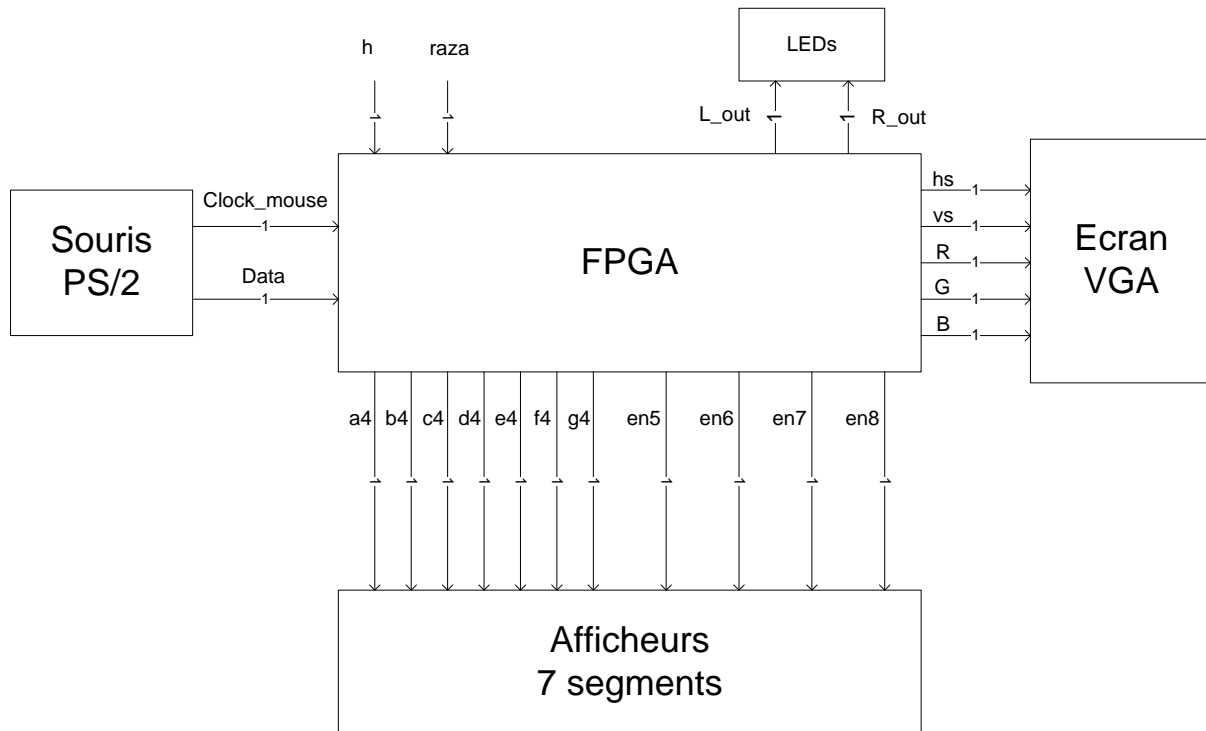
On commence le projet en réunissant et analysant les documentations récupérées du site <http://intra.esiee.fr/~jeangeoa/>

Les docs à notre disposition sont :

- Spartan-3 Starter Kit (pdf)
- « The PS/2 Mouse/Keyboard Protocol » sur le site <http://www.computer-engineering.org/ps2protocol/>
- « The PS/2 Mouse Interface » sur le site <http://www.computer-engineering.org/ps2mouse/>

On commence à construire des schémas fonctionnels de plus en plus fins (méthodologie TOP/DOWN).

b) Le schéma de l'architecture générale avec des signaux entrées et sorties :



On décide d'utiliser le schéma ci-dessus avec 4 entrées et 18 sorties.

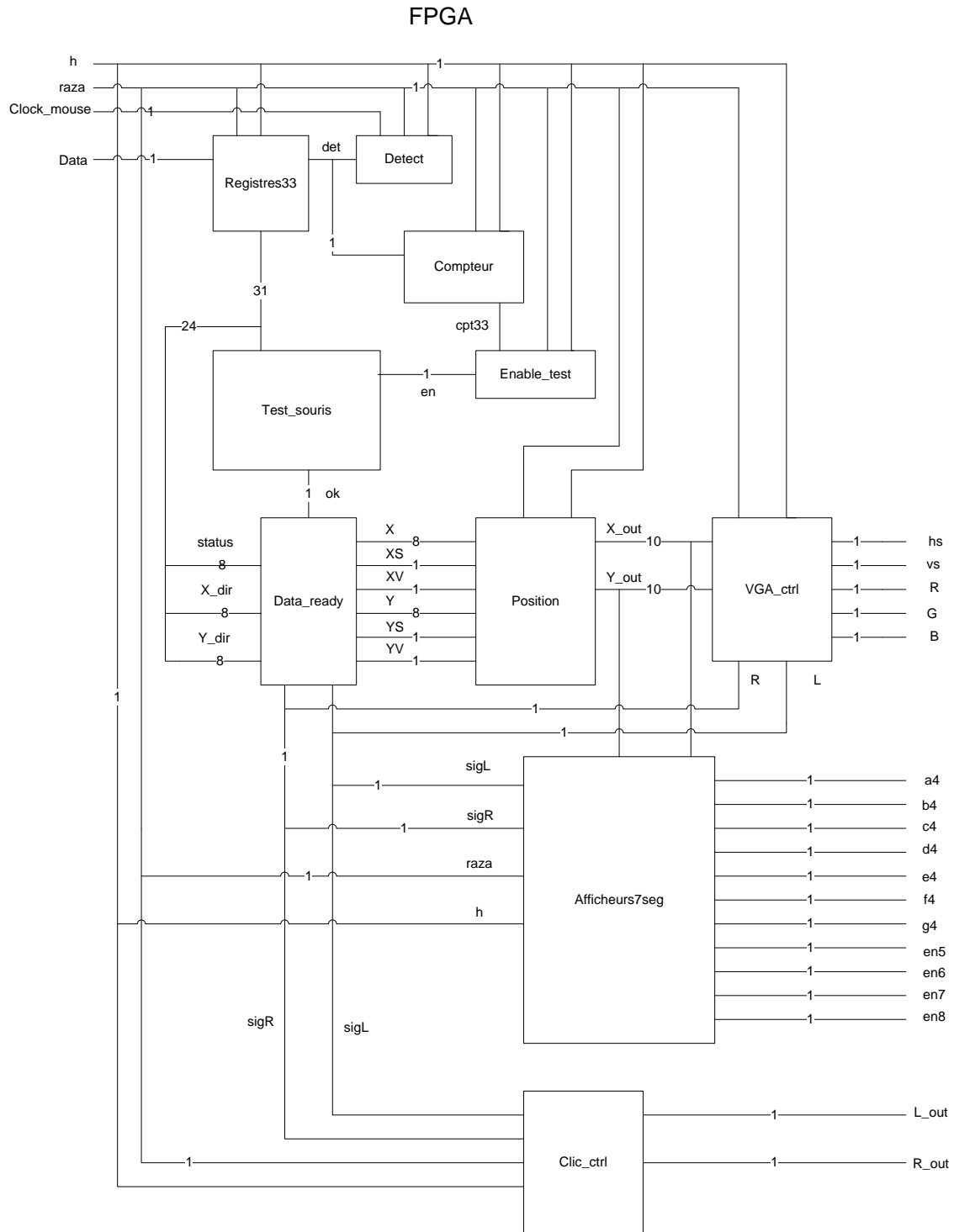
Entrées :

- h
- raza
- Clock_mouse
- Data

Sorties :

- L_out
- R_out
- hs
- vs
- R
- G
- B
- a4
- b4
- c4
- d4
- e4
- f4
- g4
- en5
- en6
- en7
- en8

1. Architecture du FPGA



Nous avons construit des blocs de contrôles pour gérer chaque type de signaux envoyés par la souris :

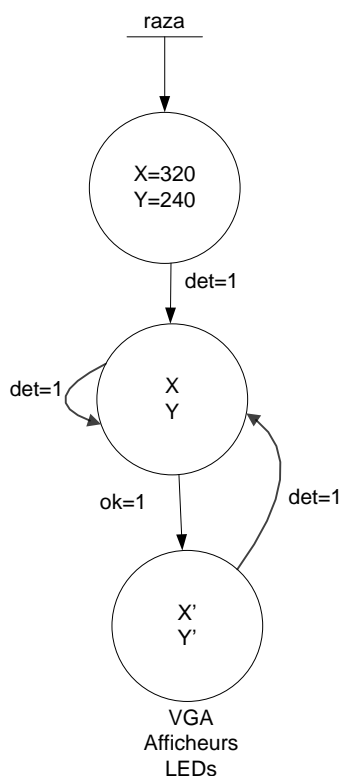
- L'horloge « h » et le signal de remise à zéro « *raza* »
- L'horloge de la souris « *clock_mouse* »
- Les informations envoyées par la souris pour ses mouvements et clics que nous avons appelé « *Data* »

Pour ce dernier, le **Registres33** enregistre les 33 bits de données en séries envoyées par *Data*. Le module **Detect** sert à détecter un front descendant de la *Clock_souris* puis envoie un signal d'activation au **Registres33** pour décaler les données d'un bit dès que le **Detect** passe à 1. Le **Compteur** compte de 0 à 32 (correspond à 33 entiers, compteur modulo 33) et il est incrémenté de 1 dès que le **Detect** passe à 1 ce qui signifie qu'il arrive un nouveau bit de *Data* de la souris. Dès que le **Compteur** compte à 32, le signal *cpt33* est activé à 1 ce qui veut dire que le **Registres33** vient de récupérer tous les 3 paquets de données envoyées par la souris.

Un protocole de test est ensuite réalisé, pour tester l'exactitude des informations envoyées par la souris, grâce à **Enable_test**, qui signale que les informations sont prêtes à être testées, **Test_Souris** qui effectue le test, et enfin **Data_ready** qui indique que le signal est bon. Ensuite, d'après les données envoyées par **Data_ready**, nous mettons à jour la position du curseur dans **Position**, qui sera enfin envoyée au **VGA_CTRL** pour un affichage sur écran VGA, et à **Afficheurs7seg** pour afficher la position numériquement sur les 4 afficheurs de 7 segments de la maquette.

Et enfin, le bloc **Clic_ctrl** contient 2 bascules de type T pour maintenir le clic de la souris. Une bascule pour le clic gauche et une pour le clic droite. Les sorties de **Clic_ctrl** passent à l'état haut pour le première clic pour allumer la LED correspondante et puis passent à l'état bas pour le deuxième clic.

1.1. Machine d'états



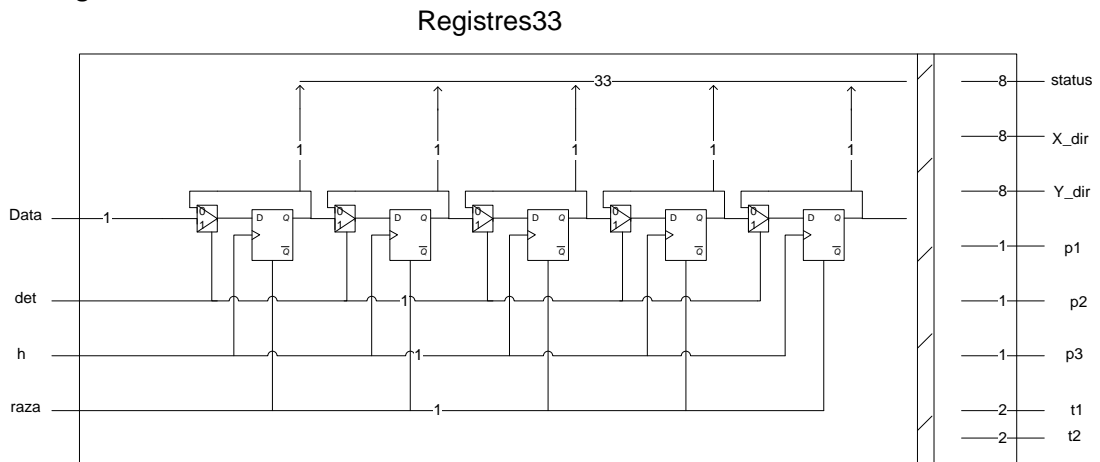
Quand le *raza* passe à l'état haut, la position du curseur est initialisée au milieu de l'écran (X=320, Y=240).

Le signal *det* est activé lorsqu'on bouge ou clique la souris. On traite les données envoyées par la souris et on les vérifie s'il y a des erreurs. Sinon, la position actuelle de la souris sur l'écran est mise à jour avec les données reçues.

Puis, on affiche des infos sur l'écran VGA, les afficheurs de 7 segments et/ou les LEDs.

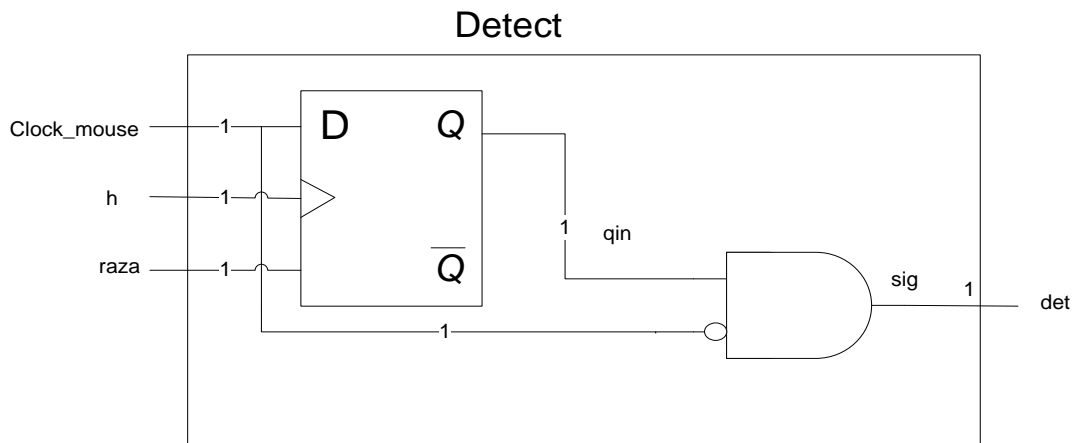
2. Architectures des sous-blocs de FPGA

2.1. Registres33 :



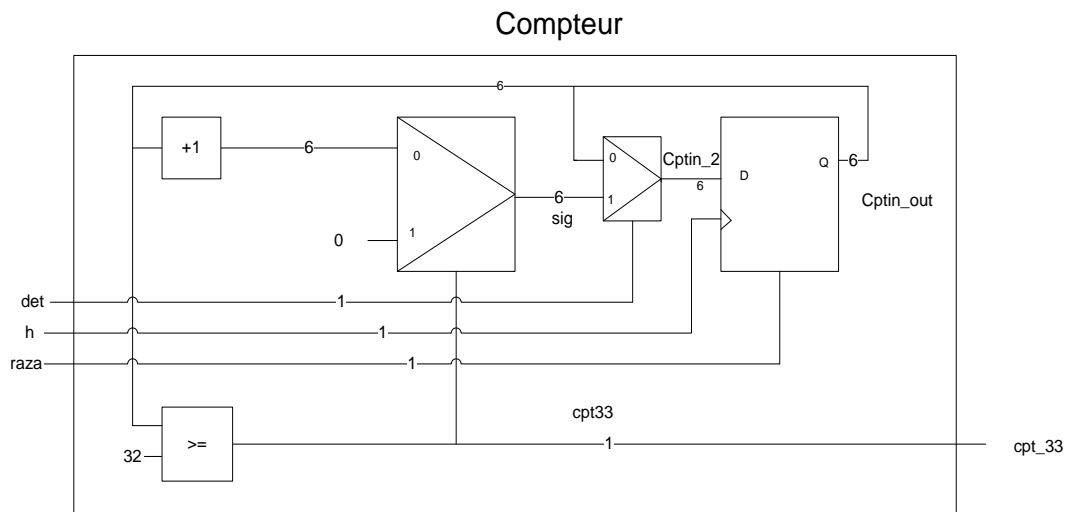
Ce bloc **Registres33** est construit par 33 bascules en séries précédées d'un multiplexeur pour chacune des bascules. Les multiplexeurs servent à maintenir les données dans les bascules quand le signal *det* est à l'état bas ou ils décalent les données à droite pour prendre en compte l'arrivée de nouvelle donnée quand le signal *det* passe à l'état haut.

2.2. Detect :



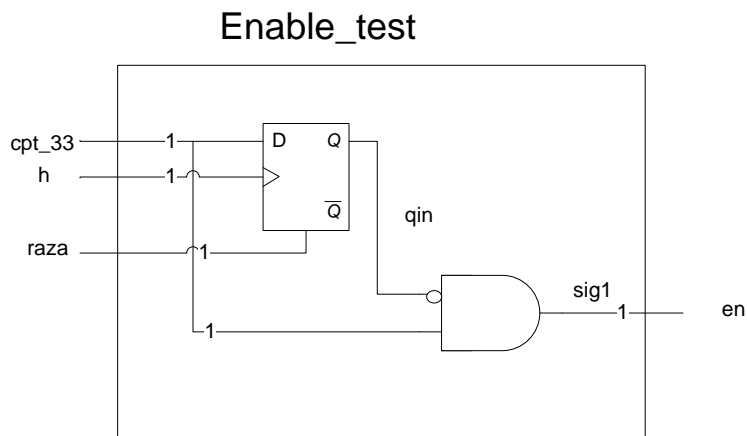
Le bloc **Detect** a pour objectif de détecter le front descendant de l'horloge de la souris. La sortie *det* passe à '1' quand il y a front descendant de l'horloge de la souris pour signaler l'arrivée de nouvelle donnée.

2.3. Compteur :



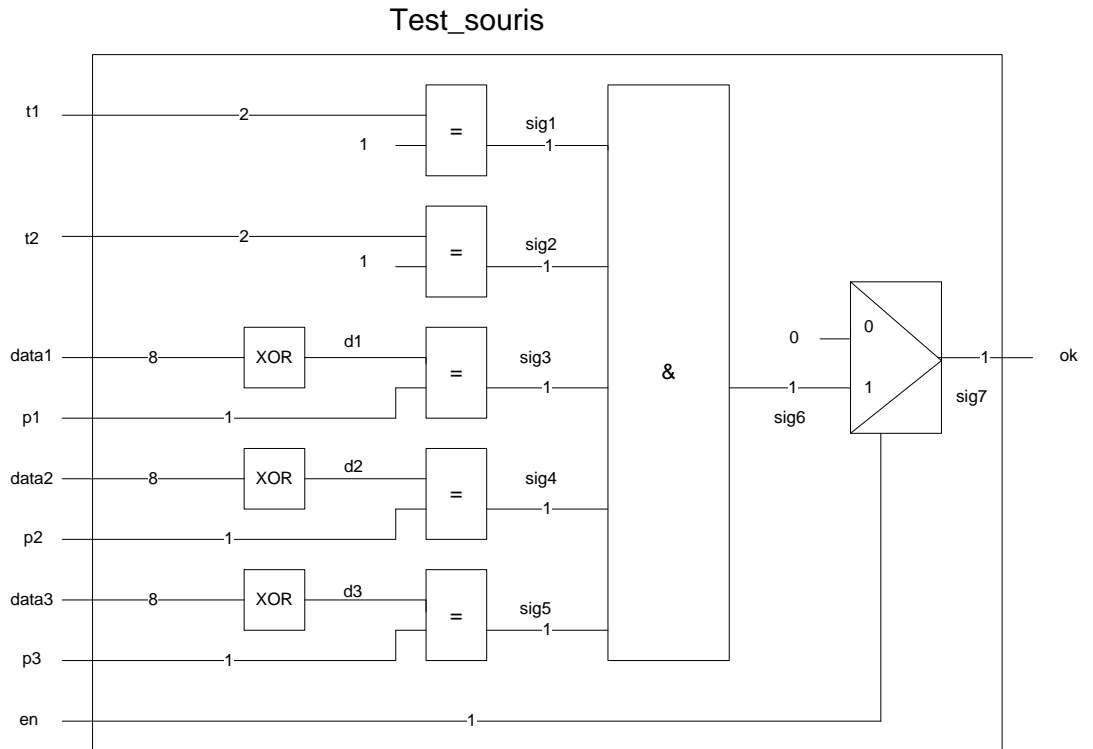
Ce **Compteur** est un compteur modulo 33 pour compter le nombre de bits reçus de la souris. Il envoie un signal '1' quand il finit de compter de 0 à 32. Cela signifie que le **Registres33** vient de recevoir tous les 3 paquets de données (33 bits) de la souris.

2.4. Enable_test :



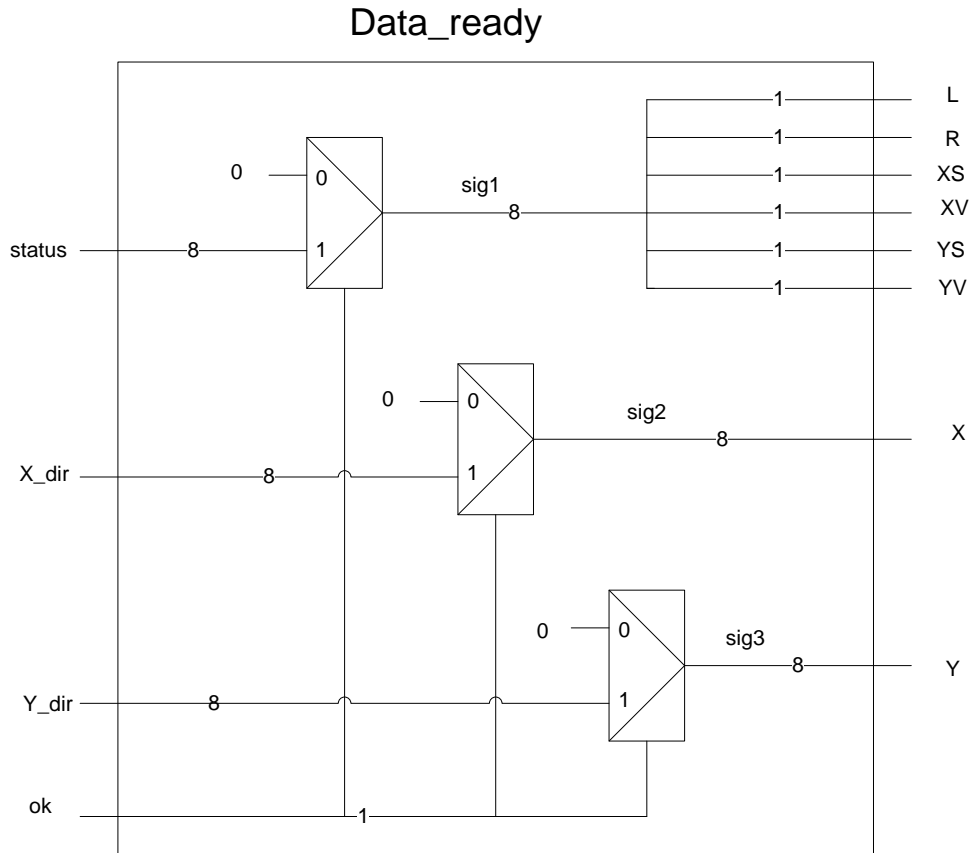
Le bloc **Enable_test** prend en compte le signal *ctp_33* du **Compteur** (cf. 2.3.). Il fonctionne comme un détecteur qui envoie à la sortie *en* un signal '1' sur un cycle d'horloge du système quand le signal *ctp_33* passe à '1'.

2.5. Test_souris :



Ce bloc **Test_souris** sert à tester et vérifier les données récupérées dans le **Registres33**. On compare les paquets de données avec les bits de parité de chacun. On vérifie aussi les bits qui sont toujours à '1' ou à '0'. Il envoie un signal *ok* à '1' quand le test vérifie que les données sont bonnes.

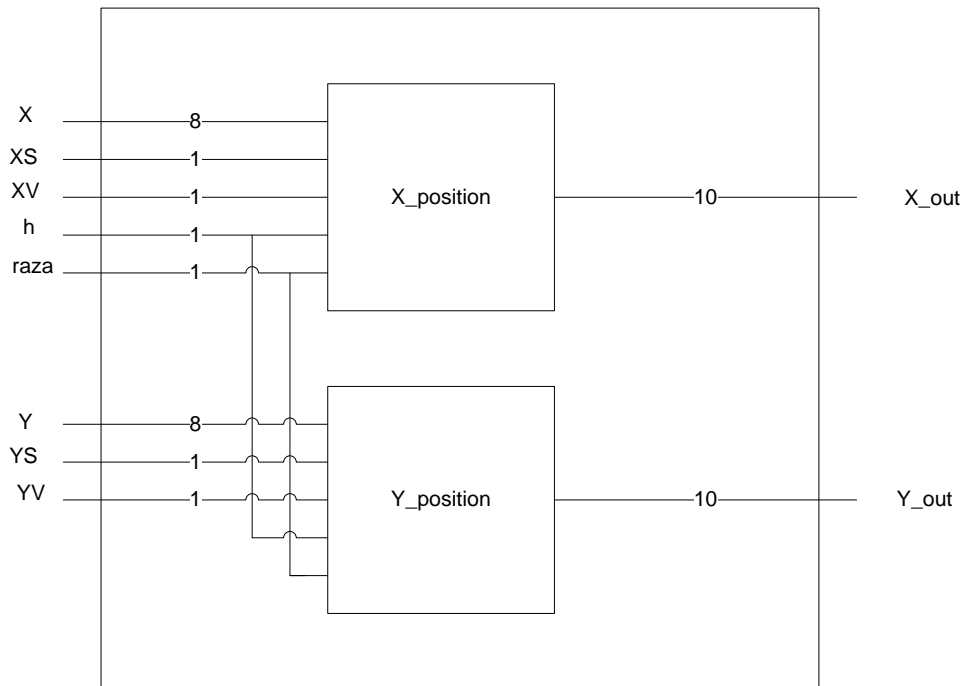
2.6. Data_ready :



Dès que le **Data_ready** reçoit le signal d'autorisation *ok*, il laisse passer les paquets de données récupérés du **Registres33** sinon les sorties de **Data_ready** sont forcées à l'état bas.

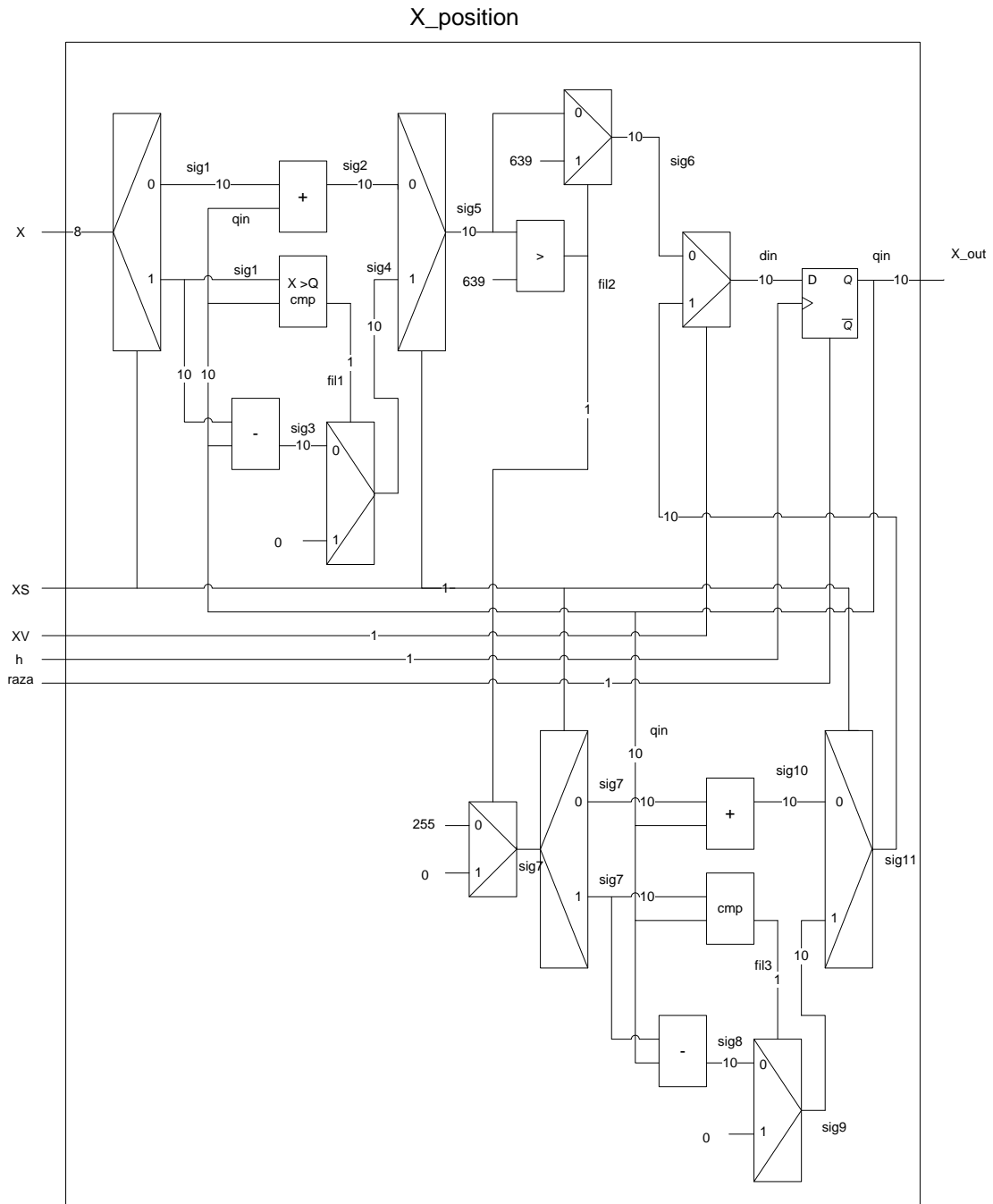
2.7. Position :

Position



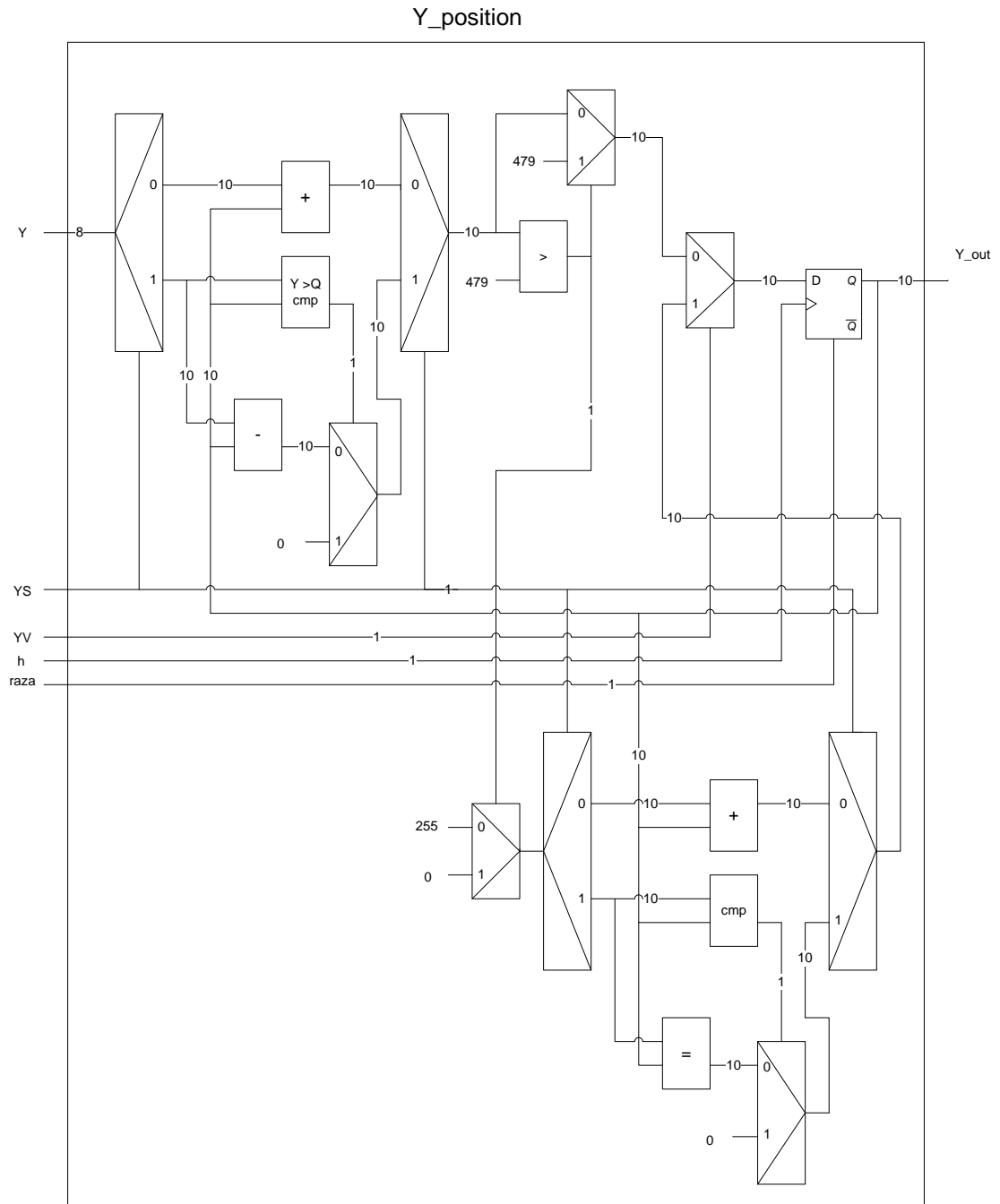
Comme son nom indique, le bloc **Position** sert à gérer la position de la souris. Il est composé de deux sous-blocs : **X_position** pour la position horizontale, et **Y_position** pour la position verticale.

2.7.1. X_position :



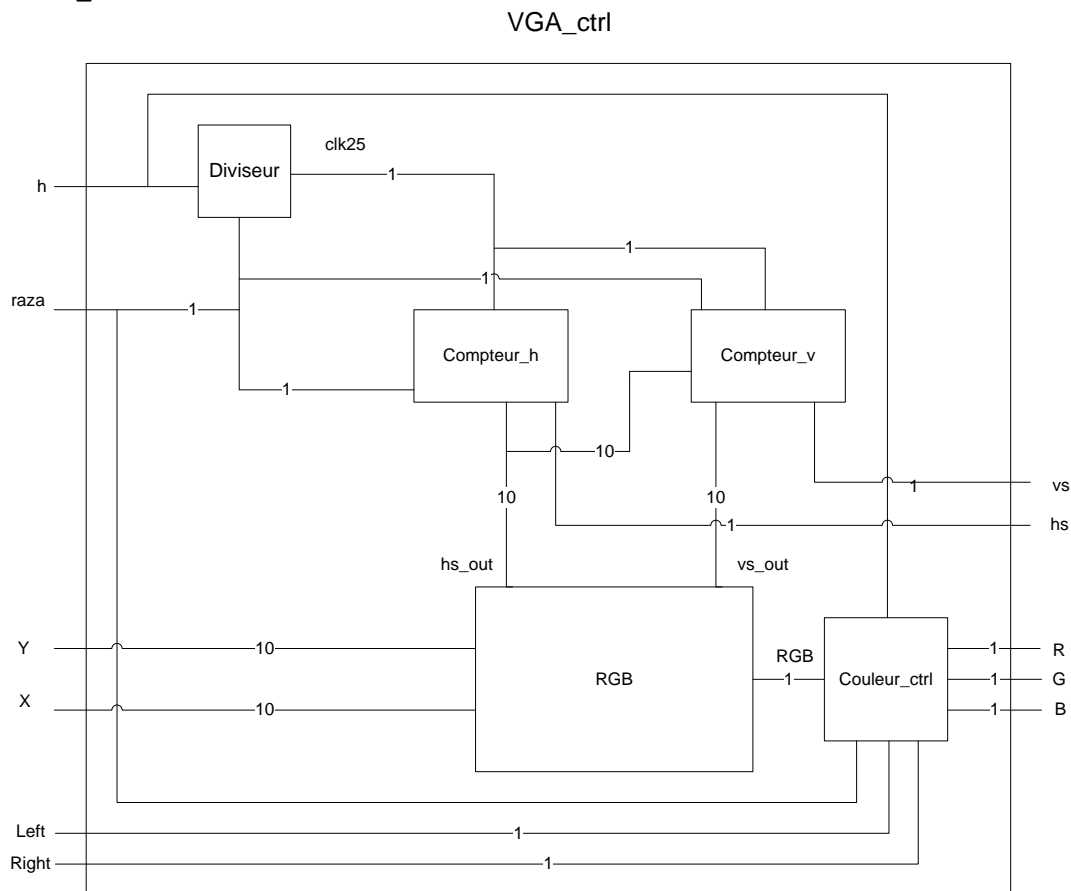
Ce bloc est peut-être le bloc le plus compliqué. On le voit sur le schéma mais le principe de fonctionnement est simple. Premièrement, quand le *raza* est à '1', on initialise la sortie *X_out* à la valeur 320 (c'est le milieu de la droite horizontale allant de 0 à 639). Quand le *raza* est autre que '1', on compare cette valeur à la valeur *X* en entrée. Selon le signal *XS*, on ajoute ou soustrait ces 2 valeurs. Dans le cas où on doit faire une soustraction, c'est-à-dire quand *XS* est à '1' et pour éviter le problème avec le signe, on fait d'abord une comparaison. Si la valeur de la position actuelle est supérieure à la valeur *X* en entrée, on fait une soustraction. Sinon, on force la valeur de la position à 0. Dans le cas où il y a overflow (*XV*=1), on déplace le curseur à vitesse maximale. On vérifie aussi que le curseur ne dépasse pas le bord de l'écran en faisant une comparaison avec la valeur 639.

2.7.2. Y_position :



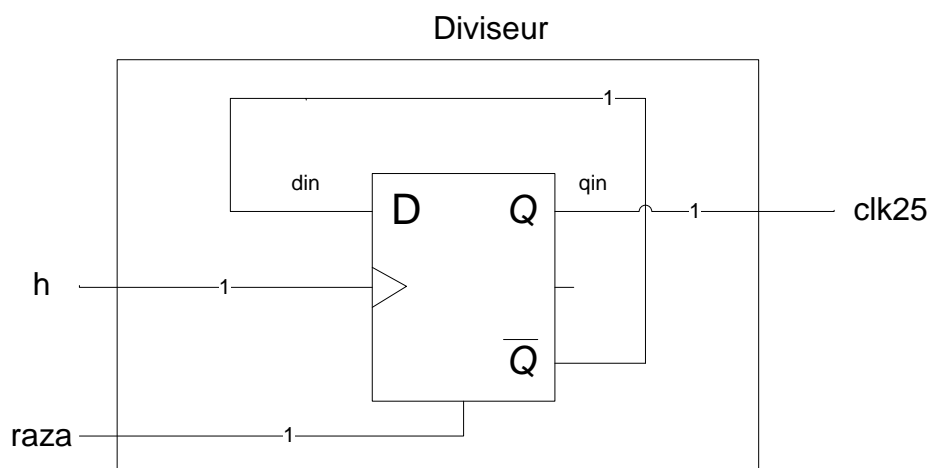
Idem pour le bloc **X_position** sauf qu'on compare la valeur de la position avec la valeur 479. Une chose qu'il faut préciser est que la sortie est sur 10 bits (0 à 1023) au lieu de 8 bits (0 à 255) car on doit afficher le curseur sur un écran de 640x480. Il nous faut donc au moins 10 bits pour retenir la position de la souris.

2.8. VGA_ctrl :



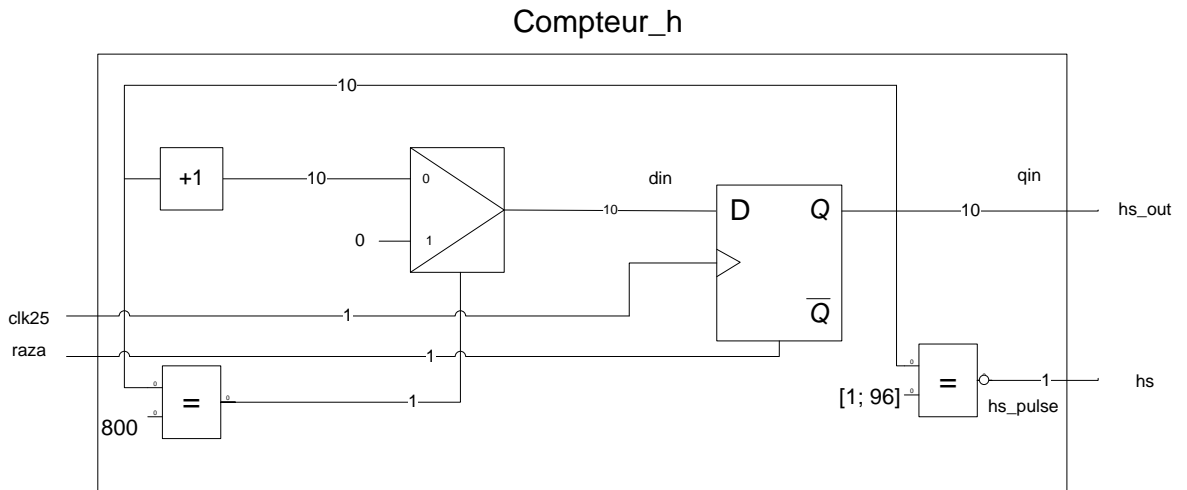
Ce bloc **VGA_ctrl** est composé de 5 sous_blocs. En général, il sert à générer des impulsions aux sorties *vs* et *hs* pour contrôler l’affichage sur un écran. Les sorties *R*, *G* et *B* sont des signaux de couleur qui sont activés au bon moment et au bon endroit pour qu’on puisse voir la position exacte de la souris sur un écran.

2.8.1. Diviseur :



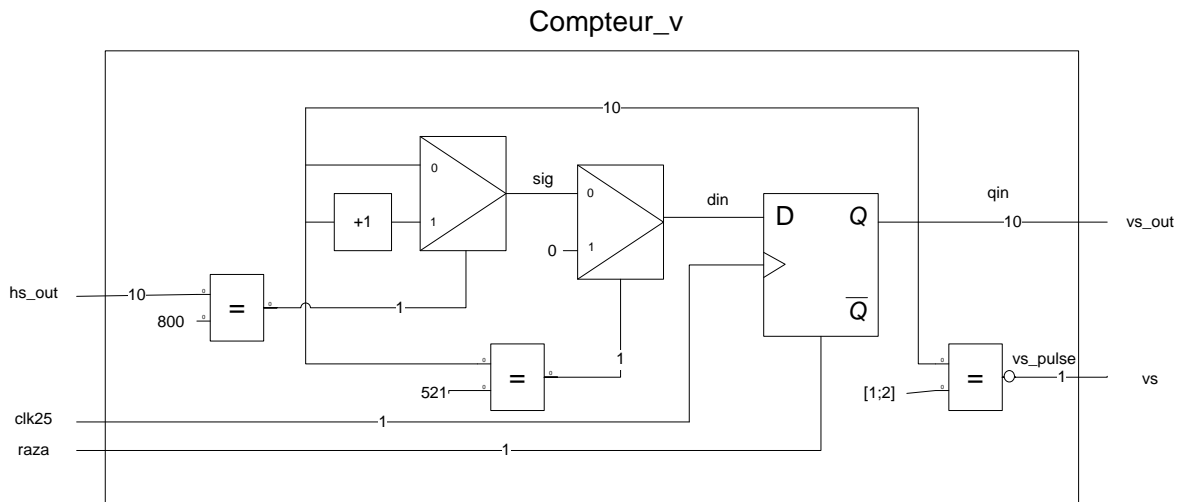
On a une horloge du système de 50 MHz. Or, d’après la doc, pour faire fonctionner un écran ou pour pouvoir afficher des images sur un écran, il nous faut une horloge de 25 Mhz pour arriver à piloter l’écran. D’où l’intérêt de ce **Diviseur** qui divise l’horloge du système par 2.

2.8.2. Compteur_h :



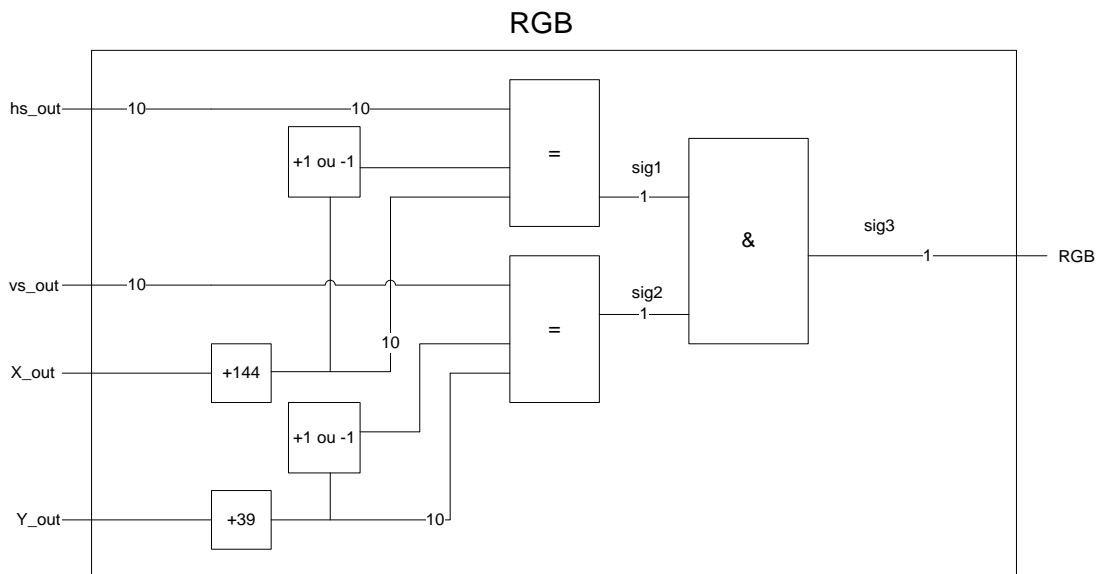
C'est un compteur qui calcule la durée d'une impulsion à l'état bas envoyée à la sortie *hs*. Ce **Compteur_h** compte de 0 à 800 (modulo 801). Il envoie aussi à la sortie *hs_out* d'un signal sur 10 bits la valeur du comptage.

2.8.3. Compteur_v :



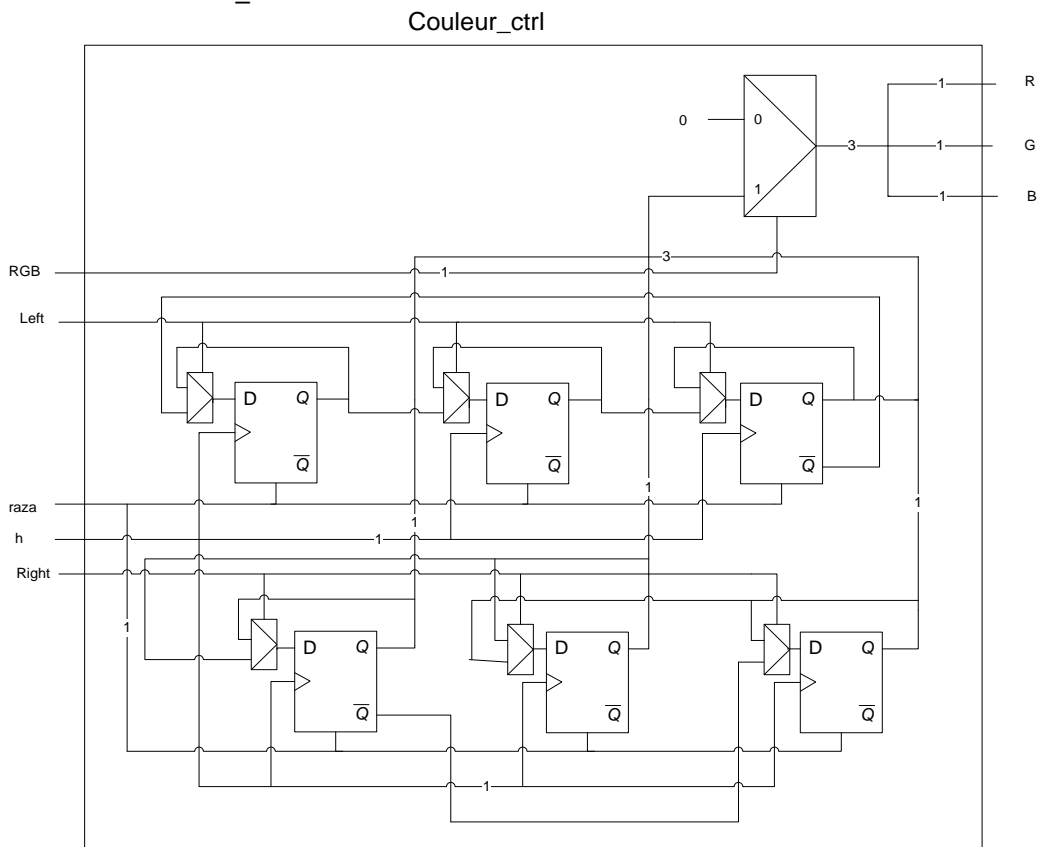
Ce **Compteur_v** prend en compte le signal *hs_out* du **Compteur_h**. Il est incrémenté de 1 quand la valeur du signal *hs_out* passe à 800 ce qui signifie le déplacement horizontale sur écran arrive à la fin de la ligne et il faut faire un retour à la ligne. Ce **Compteur_v** est un compte modulo 522 qui a les sorties *vs_out* (valeur du comptage) et *vs* (impulsion)

2.8.4. RGB :



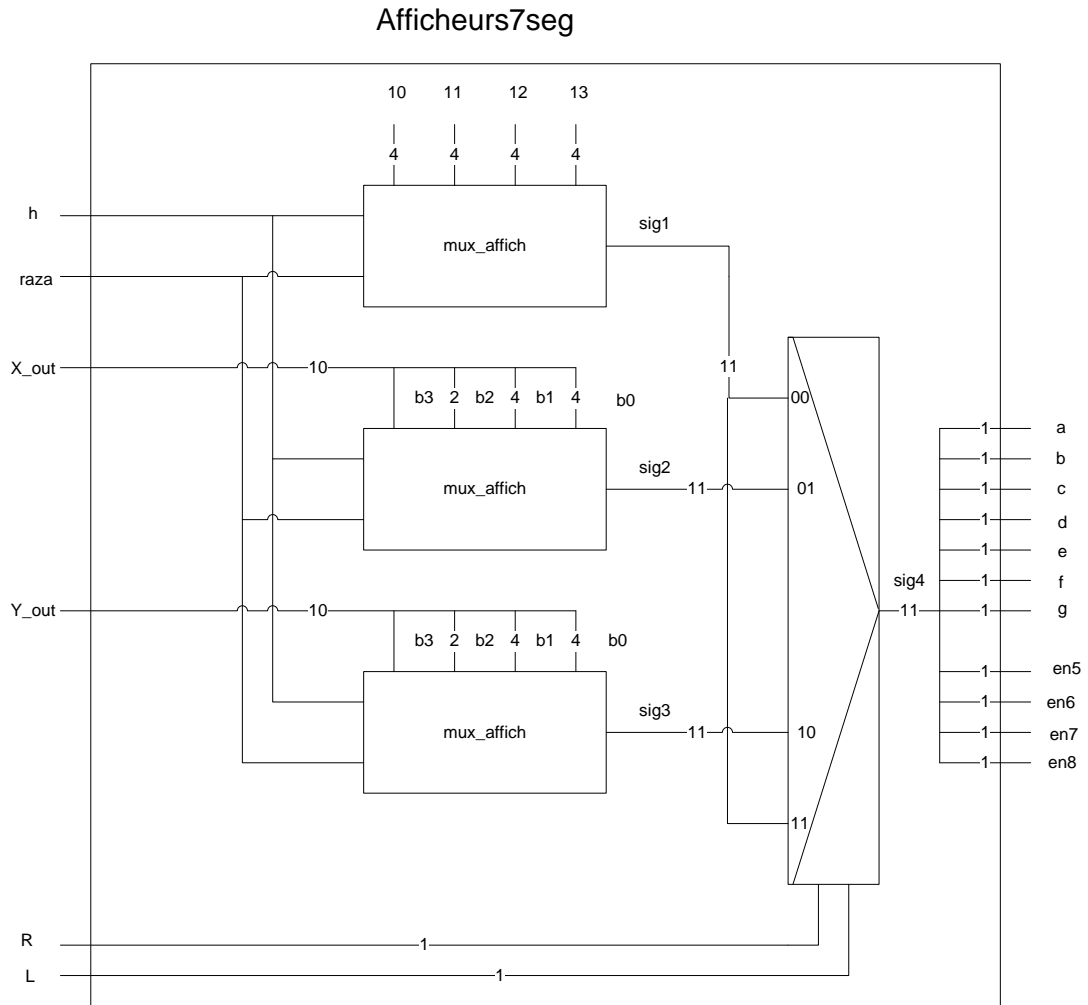
Ce bloc **RGB** envoie un signal **RGB** à l'état '1' quand la valeur de **hs_out** correspond à la valeur **X_out** plus 143 ou 144 ou 145 **ET** la valeur de **vs_out** correspond à la valeur **Y_out** plus 38 ou 39 ou 40. Au lieu d'afficheur un seul pixel sur écran, on affiche un carré de 9 pixels. Ceci pour qu'on voit mieux la position du curseur.

2.8.5. Couleur_ctrl :



Ce bloc **Couleur_ctrl** gère la couleur de curseur en utilisant 2 compteurs de Johnson précédés d'un mux. L'un pour un décalage à droite et l'autre à gauche si on clique la souris.

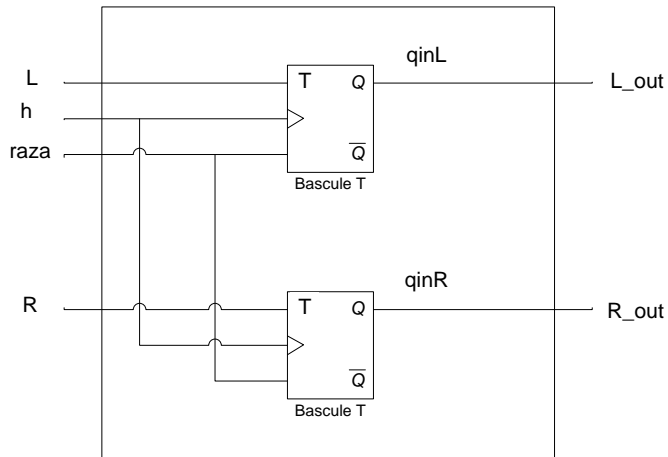
2.9. Afficheurs7seg :



En faisant et maintenant un clic (gauche ou droite), ce bloc **Afficheurs7seg** affiche la position du curseur sur les 4 afficheurs de 7 segments. Les sous-blocs **mux_affich** sont récupérés du site <http://intra.esiee.fr/~jeangeoa/>

2.10. Clic_ctrl :

Clic_ctrl



Ce bloc **Clic_ctrl** utilise 2 bascules T, l'une pour le clic gauche et l'autre pour le clic droite. Un simple clic fait passer la sortie de la bascule à l'état haut et reste à cet état jusqu'au prochain clic. Cela sert à allumer la LED sur la maquette.

3. Brochage sur la maquette :

Entrées		Sorties	
Input	Loc	Output	Loc
h	T9	hs	R9
raza	F12 (SW0)	vs	T10
Clock_mouse	M16	R	R12
Data	M15	G	T12
		B	R11
		a4	E14
		b4	G13
		c4	N15
		d4	P15
		e4	R16
		f4	F13
		g4	N16
		en5	D14
		en6	G14
		en7	F14
		en8	E13
		L_out	P14 (LD1)
		R_out	K12 (LD0)

4. Répartition du travail :

Travail	Elèves
Gestion de l'interface souris PS/2 - Registres33 - Detect - Compteur - Test_souris	Mathieu DAHNE
Stockage et mise à jour de la position du curseur - Enable_test - Data_ready - Position - Afficheurs7seg	Mathieu DAHNE, Chok Leong CHAI
Gestions des clics souris - Clic_ctrl	Ting DU
Gestion de l'affichage sur écran VGA - VGA_ctrl	Chok Leong CHAI
Gestion de la couleur - Couleur_ctrl	Chok Leong CHAI
Rapport - Schémas	Ting DU

5. Quelques captures d'écran sur les simulations sous ModelSim concernant les points les plus importants.

*On travail sous Xilinx version 10.1 et on fait des simulations sous ModelSim version 6.3c.

5.1. Behavioral Simulation

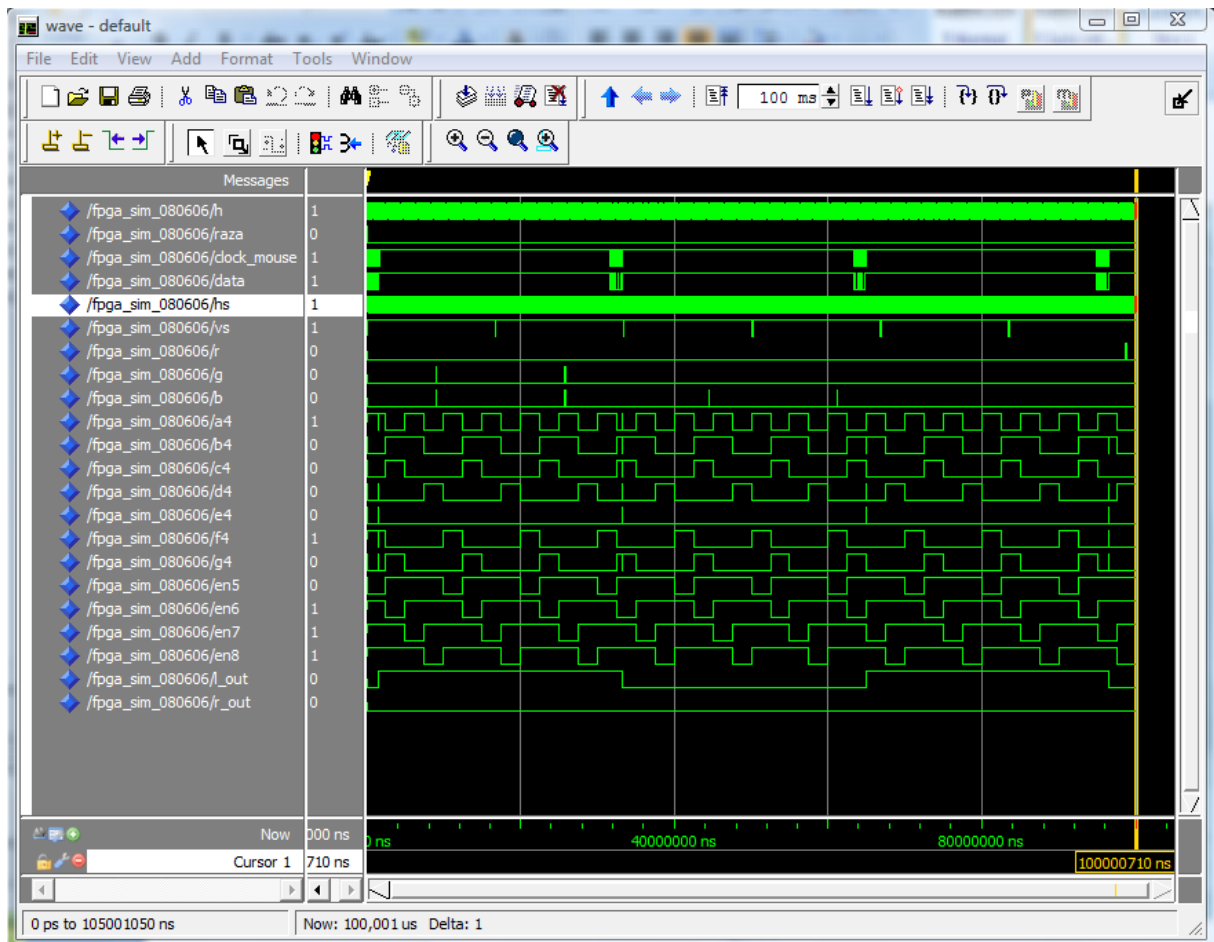


Figure 1

On fait une Behavioral simulation sur une durée de 100 ms. La simulation nous montre le résultat qu'on s'attend à avoir.

5.2. Post-Map Simulation

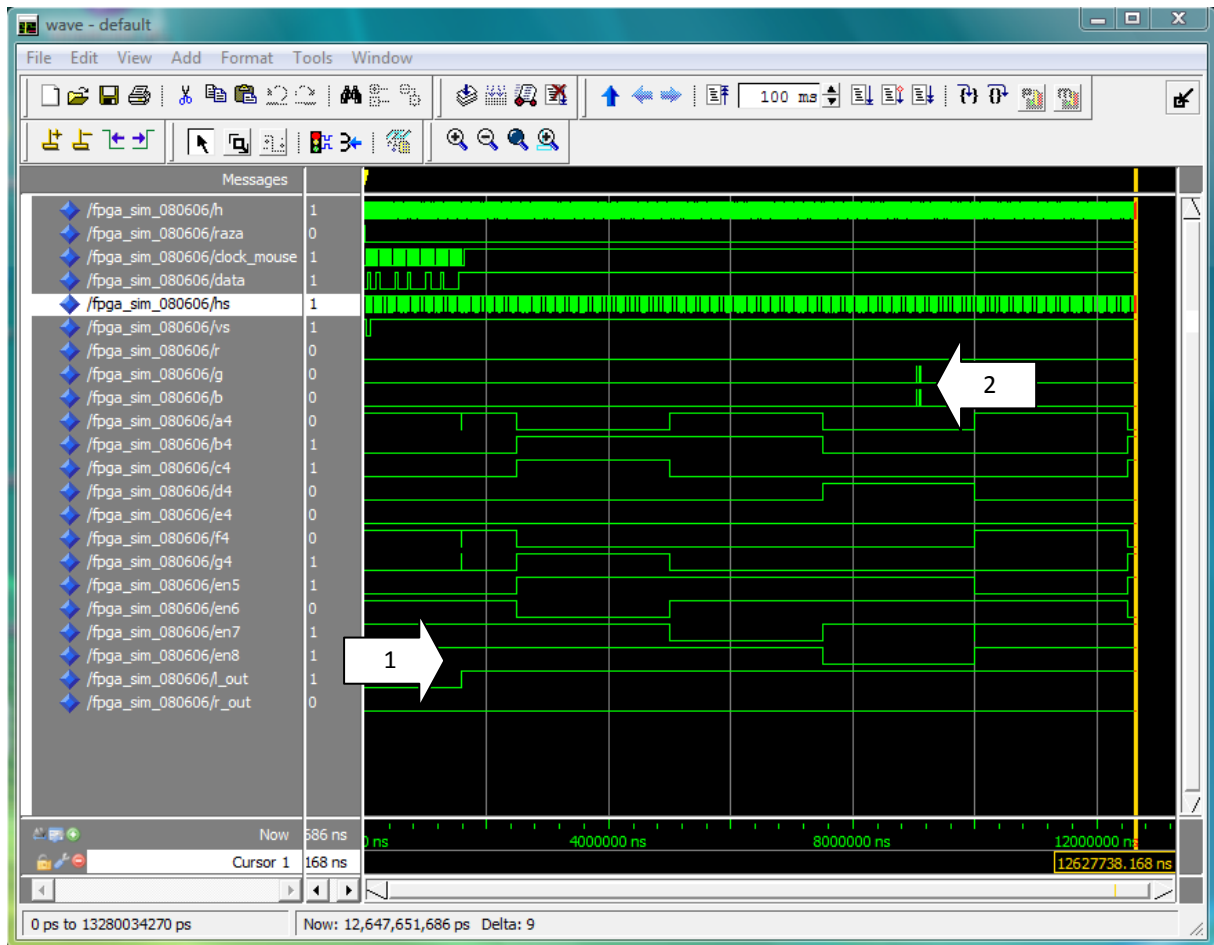


Figure 2

Cette simulation prend un peu plus long temps que prévu. En fait, après plus de 20 heures, on n'arrive qu'à faire une Post-Map simulation d'une durée de 13 ms. Malgré ces longues heures d'attente, on voit que le résultat est bon. On fait des zooms sur les points indiqués par les flèches ci-dessus.

5.2.1. Zoom sur le point 1

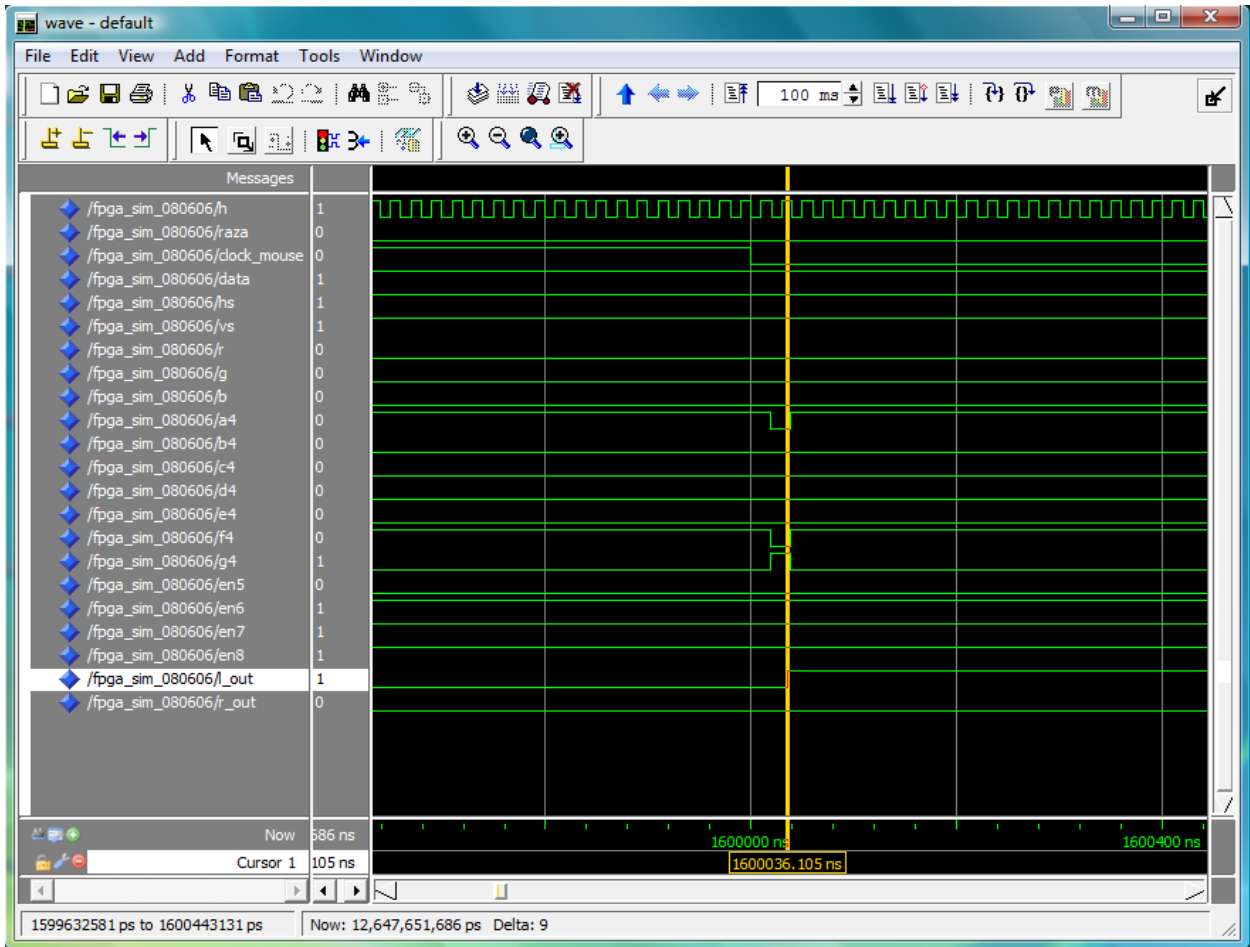


Figure 3

Le signal L_out passe à l'état haut après avoir traité les données envoyées par $Data$.

5.2.2. Zoom sur le point 2

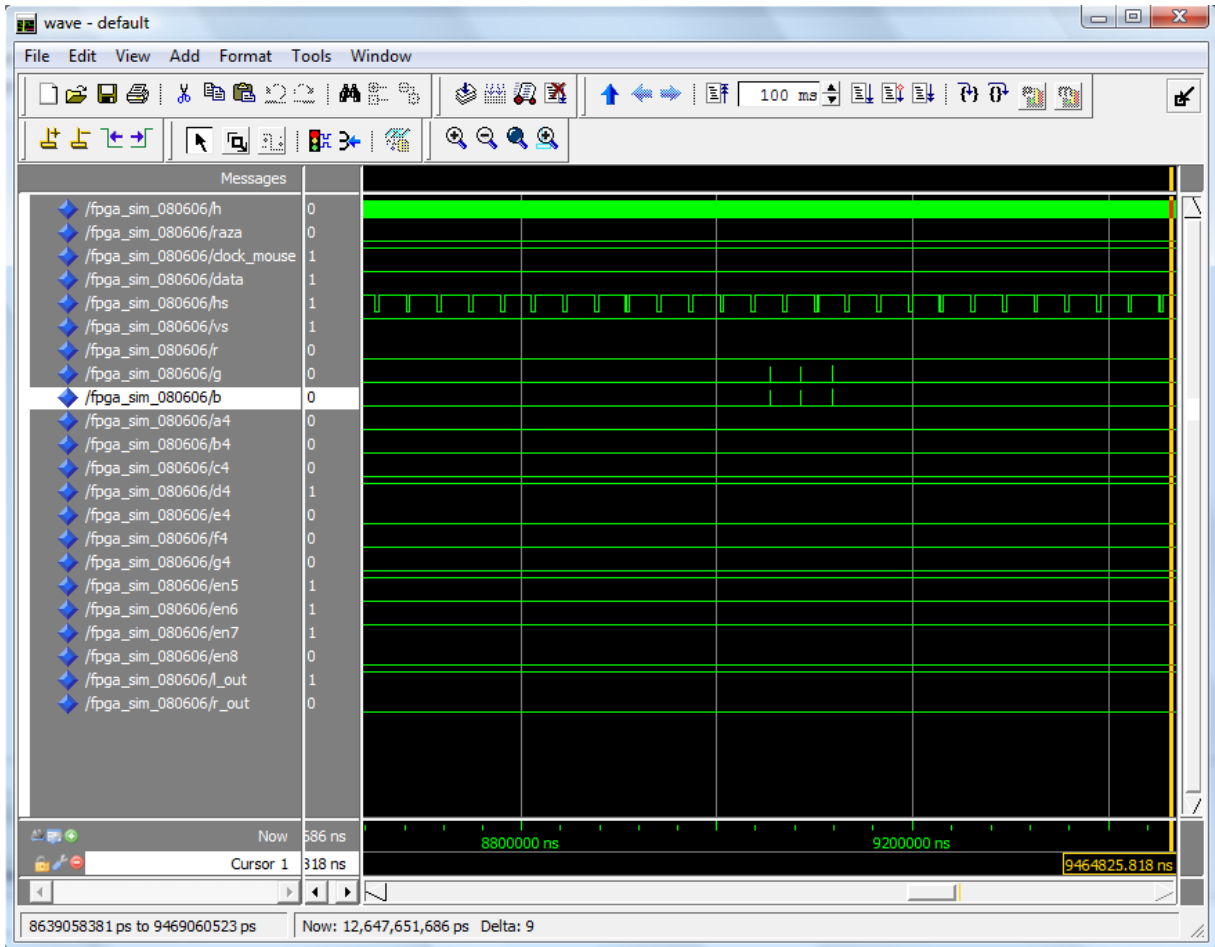


Figure 4

Les signaux G et B sont activés 3 fois ce qui correspond bien à ce qu'on attend.

6. Difficultés techniques rencontrées

6.1. Nombre d'entrées et de sorties :

Au départ, on n'avait que 5 signaux de sortie pour piloter l'écran VGA. Puis, on a ajouté 11 signaux pour afficher la position de la souris sur les 4 afficheurs de 7 segments de la maquette et 2 signaux en plus pour allumer les LEDs.

6.2. Le **Registres33**

On ne s'intéressait de stocker que les 3 paquets de données sur 8 bits sans tenir compte de « bit de parité », de « start bit » et de « stop bit ». Puis on apercevait qu'on avait besoin de ces bits pour la suite pour tester les données de la souris. D'où, on a modifié ce bloc avec 33 bascules.

6.3. Le **Detect**

La souris envoie des données en série sur un fil. D'où l'intérêt de ce bloc pour détecter l'arrivée de chaque nouvelle donnée. On avait des soucis pour réaliser un tel bloc mais heureusement on a vu ce genre de détecteur en TD n°2 en VHDL.

6.4. Les **X_position** et **Y_position**

On a un curseur qui se déplace sur l'axe horizontal de 0 à 639 pixels et l'axe vertical de 0 à 479 pixels. On ne peut pas dépasser ces valeurs. Or, quand on fait une soustraction, il y a des possibilités d'avoir une valeur négative. Pour ne pas s'embêter avec le signe, car on ne sait pas ce qu'on va avoir sur les bits de sortie avec une valeur négative, on a modifié ce bloc de façon à ne pas traiter le cas d'une valeur négative en forçant la valeur à 0 si la valeur à soustraire est plus grande que la valeur initiale.

6.5. Le **VGA_ctrl**

En début, dans le bloc de **VGA_ctrl**, il n'y avait pas de bloc **Couleur_ctrl**. On l'a ajouté après qu'on s'est rendu compte qu'il est possible d'utiliser le compteur de Johnson pour faire varier la couleur du curseur.

6.6. De petits erreurs et problèmes

On a rencontré pas mal de petites erreurs qui nécessitaient des heures pour les trouver pendant la durée du projet. Par exemple, on a oublié de relier un fil qui gère le clic de la souris. Quand on faisait la simulation et on apercevait qu'il y avait de signal de clic à la sortie, on croyait qu'on a mal géré les timings. Mais, en fait, il suffisait de brancher ou relier ce fil au bon endroit. Puis, il y avait des erreurs qui provenaient du code avec de fausse syntaxe, les problèmes du logiciel Xilinx..., etc.