TP2 : compteurs et mémoires

Travail Préparatoire : Lire l'énoncé avant l'arrivée en séance de TP.
Objectifs du TP :

- Approfondir les connaissances en VHDL (manipulation de vecteurs de bits, arithmétique, affectation concurrente conditionnelle)
- Apprendre à utiliser et fabriquer des composants *génériques* en VHDL
- Comprendre et pratiquer les accès mémoire.

Ex. 1 : Compteurs

Dans l'exercice 4 du TP1, on vous demandait de réfléchir à un circuit composé de 4 composants AC et 4 bascules. Grâce à l'exercice 2 du TD 4, vous devriez comprendre que ce circuit bonus réalise en fait un compteur 4 bits (i.e. un circuit dont la sortie s'incrémente modulo 16 à chaque cycle d'horloge : 0, 1, 2, ..., 14, 15, 0, 1, 2, ...).

Dans cet exercice nous allons introduire les vecteurs et les opérateurs arithmétiques, qui permettent de décrire de tels circuits de manière beaucoup plus abstraite, sans toujours revenir à la connexion des additionneurs 1 bit.

Question 1

— Lisez le fichier `vhd/compteur4.vhd` ainsi que ses commentaires.
— Complétez le fichier en instanciant un registre 4 bits et en écrivant les équations nécessaires pour réaliser le compteur 4 bits avec reset.
Remarque : les variables `dd` et `curval` sont des entiers (type `unsigned`) donc on peut utiliser des opérateurs arithmétiques.

Pour tester le circuit, on utilise le testbench `vhd/tb_compteur4.vhd`, qui provoque un reset avant que le compteur ait atteint la valeur 15.

Question 2

— Pouvez-vous deviner quelle séquence de valeurs vous allez observer en sortie du compteur avec ces stimuli d'entrée ?
— Simulez votre circuit (`make run_simu TOP=compteur4`) et vérifiez si vous avez deviné juste.

Question 3

Réalisez la synthèse (`make synthese TOP=compteur4`) et analysez le résumé des rapports de synthèse affiché dans la console.

Dans les quesions précédentes, la taille choisie (4) intervient souvent mais de manière très systématique dans le code VHDL. Nous allons ici introduire la notion de *generic* en VHDL, qui est faite pour généraliser des descriptions de circuits. L'idée ici est de remplacer 4 par n partout. Regardons ce que ca donne pour réaliser un registre générique.

Question 4
— Observez `vhd/reggene.vhd` pour voir comment décrire un composant générique (et comparez si besoin à `vhd/reg4b.vhd`).
— Observez `vhd/compteur4avecreggene.vhd` pour voir comment instancier un composant générique (et comparez si besoin à `vhd/compteur4.vhd`).

Il s'agit maintenant de définir un compteur lui-même de taille générique, fabriqué avec le composant générique registre n bits.

Question 5
— Complétez le fichier `vhd/compteurgene.vhd`.
— Simulez le comportement de votre compteur générique avec le banc de test `vhd/tb_compteurgene.vhd` (`make run_simu TOP=compteurgene`).
— Réalisez la synthèse (`make synthese TOP=compteurgene`) et analysez le résumé des rapports de synthèse affiché dans la console.

Ex. 2 : Lecture de mémoire

Vous disposez d'une mémoire synchrone de 15 bits d'adresse, et de 32 bits de données [1]. On vous fournit un compteur générique enrichi d'une nouvelle entrée.

Question 1 Regardez le code source (`vhd/Compteur.vhd`) pour comprendre l'intérêt du port `max` et pour découvrir une manière de représenter des multiplexeurs en VHDL.

Avec ce compteur, nous souhaitons construire un composant capable de lire mot par mot le contenu de la mémoire. Le composant construit prend une entrée d'horloge `clk`, une entrée d'initialisation `reset` et une sortie `data` correspondant aux données successives lues.

Question 2 Décrivez ce composant en complétant le fichier `vhd/lecture_memoire.vhd`. Il s'agit de connecter la sortie du compteur sur le bus d'adresse de la mémoire donnée dans `vhd/RAM_Video.vhd`, en veillant à ce que la mémoire soit en lecture pour chaque cycle.

Question 3 Simulez votre composant avec la commande `make run_simu TOP=lecture_memoire`. Vous pouvez consulter les données produites en hexadécimal puis en ASCII (`radix/hexadecimal` ou `radix/ASCII`). Sachant que chaque octet code un caractère en ASCII, décodez le message inclus dans les 9 premières adresses de la mémoire. Quelle est l'adresse mémoire contenant la séquence de caractère "reus" ? Ajoutez à votre simulation l'adresse générée par le compteur pour observer le comportement synchrone de cette mémoire.

Pour aller plus loin...
Question 4 On souhaite à présent lire seulement un mot sur 16 dans la mémoire, en partant de l'adresse 0 et jusqu'à l'adresse 176. Modifiez votre circuit de lecture, simulez et indiquez le message contenu dans la mémoire. Chaque octet code un caractère en ASCII, décodez le message correspondant.

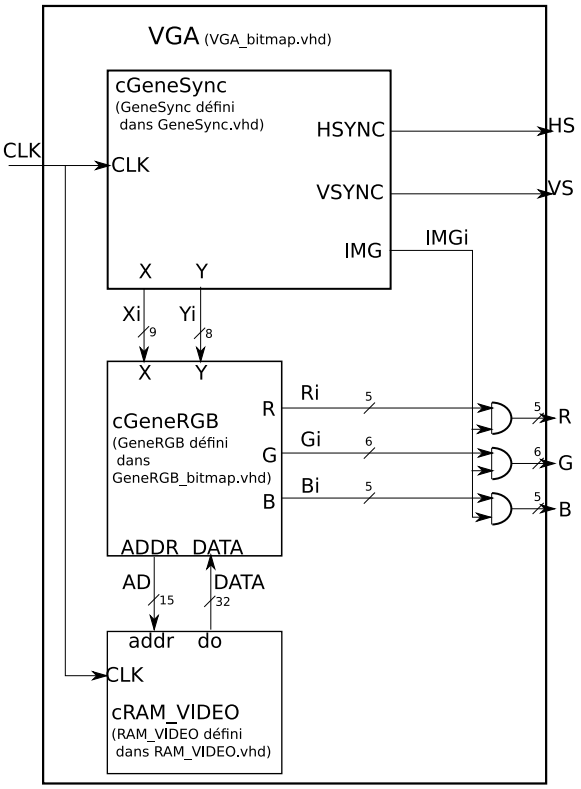
1. Il ne vous aura donc pas échappé que cette mémoire a une capacité de 32k mots soit 128ko !

Ex. 3 : Carte VGA

Dans cet exercice, on va afficher à l'écran une image bitmap sur un écran via le port VGA. Une image bitmap est décrite sous la forme d'une matrice 2D dont chaque case (pixel) correspond à la couleur du point de coordonnées correspondantes. L'image que l'on va utiliser est une image de résolution 320x240 ayant 8 bits par pixel. Cette image est contenu dans le composant mémoire `RAM_Video` étudié dans l'exercice précédent. Chaque mot de cette mémoire de 32 bits regroupe donc 4 pixels consécutifs. Dans le mot, le pixel 0 est sur les 8 bits de poids fort et le pixel 3 est sur les 8 bits de poids faible. Chaque pixel est codé sur 8 bits consécutifs : 2 bits rouge, 3 bits vert et 3 bits bleu. Le bit de poids fort (MSB) correspond au MSB rouge et le bit de poids faible(LSB) correspond au LSB bleu.

La mémoire est remplie sans vide en parcourant l'image de gauche à droite et du haut vers le bas. Ainsi, on obtient l'adresse du mot contenant le pixel désigné par X et Y avec la formule $(X + 320 * Y)/4 = 80 * Y + X/4 = 64 * Y + 16 * Y + X/4$. $X\%4$ donne la position du pixel dans le mot [2].

On vous fournit le système complet qui intègre la mémoire `RAM_Video`, un composant générant la synchronisation du protocole VGA (HS et VS : signaux de synchronisation horizontale et verticale, X et Y : coordonnées du pixel actuellement balayé, IMG : bit de validité) et un composant qui génère les signaux RGB (commande des couleurs). Ce système est décrit dans le schéma ci-dessous. Les vecteurs de sortie RGB (5 ou 6 bits) sont chacun connectés à un convertisseur numérique/analogique présent sur la carte en amont du connecteur VGA (les détails sont dans la documentation à votre disposition sur Chamilo). Pour avoir des couleurs propres, il faut fixer une valeur à tous même si les pixels issus de l'image ne sont codés que sur 8 bits.



Question 1 Complétez si besoin le schéma fourni pour le module `GeneRGB` et implantez votre circuit dans le fichier *vhd/GeneRGB_bitmap.vhd* en vous aidant des conseils fournis dans ce fichier.

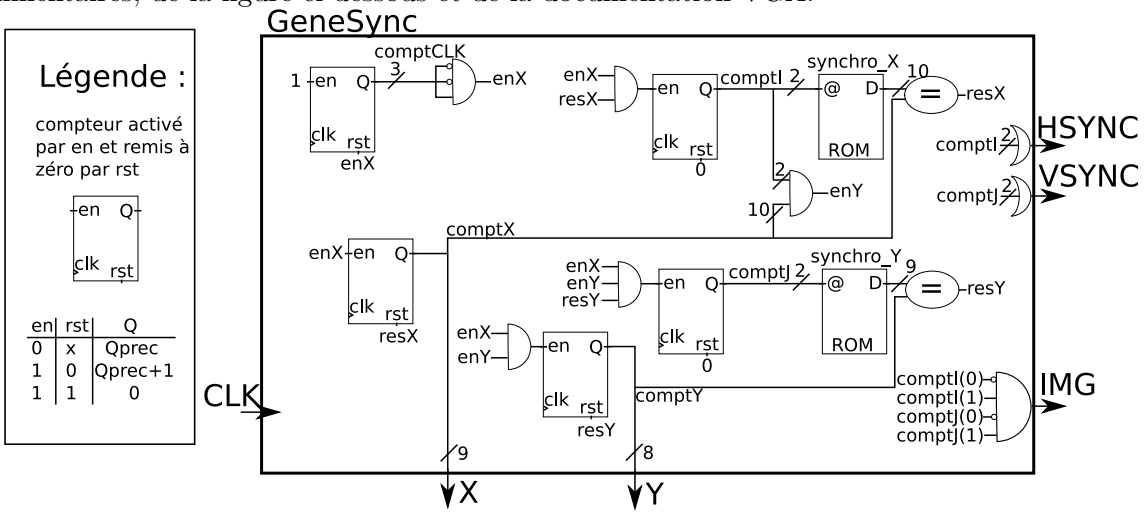
2. On remarquera que $X\%4$ correspond aux 2 bits de poids faibles de X.

Question 2 Simulez le circuit VGA complet (décrit dans le fichier *vhd/VGA_bitmap.vhd*) pendant une durée de simulation de 17ms [3] et vérifiez que le début de l'image correspond aux captures des figures 1 et 2. Pour cela taper la commande `make run_simu TOP=VGA`.

Question 3 Implémentez sur carte. Pour cela tapez la commande `make run_fpga TOP=VGA`.

Pour aller plus loin...
Question 4 Modifiez le composant GeneRGB pour réaliser diverses transformations de l'image. Par exemple échanger le vert et le bleu, inverser le haut et le bas, inverser la droite et la gauche, produire une image miroir, etc. Testez sur carte. Nous donnons figures 3, 4 et 5 quelques exemples d'images à obtenir.

Pour aller plus loin...
Question 5 Lisez la description du composant `GeneSync` et comprenez-la en vous aidant des commentaires, de la figure ci-dessous et de la documentation VGA.



3. Le protocole VGA implanté a une fréquence de rafraichissement de 60Hz.

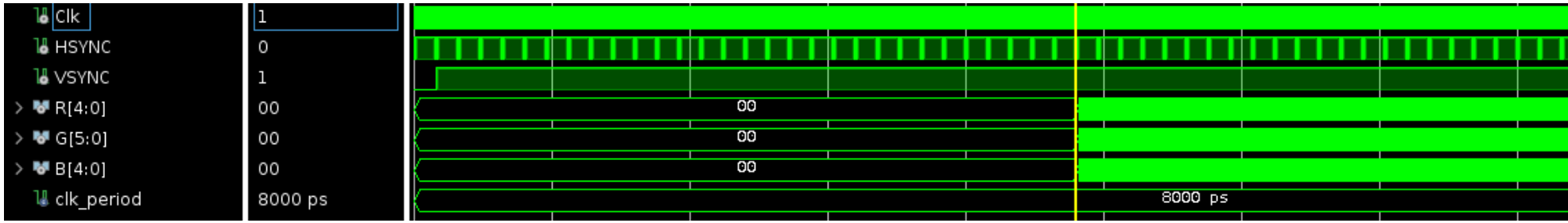


FIGURE 1 – Simulation sur une fenêtre de 1,6ms (affichage en hexa)

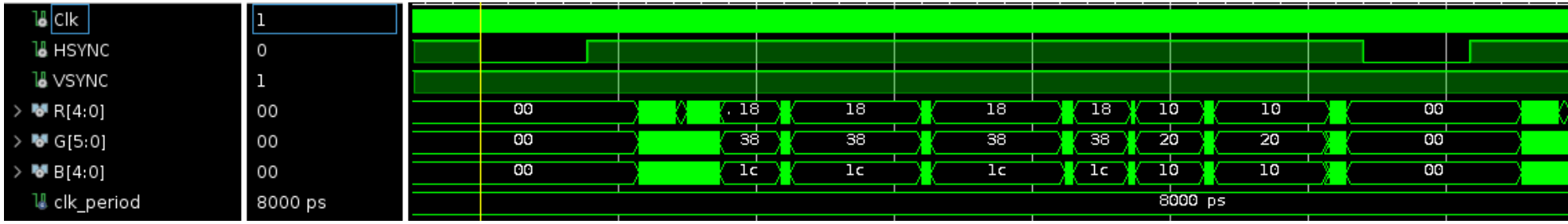


FIGURE 2 – Simulation en zoomant autour de 0,96ms (affichage en hexa)

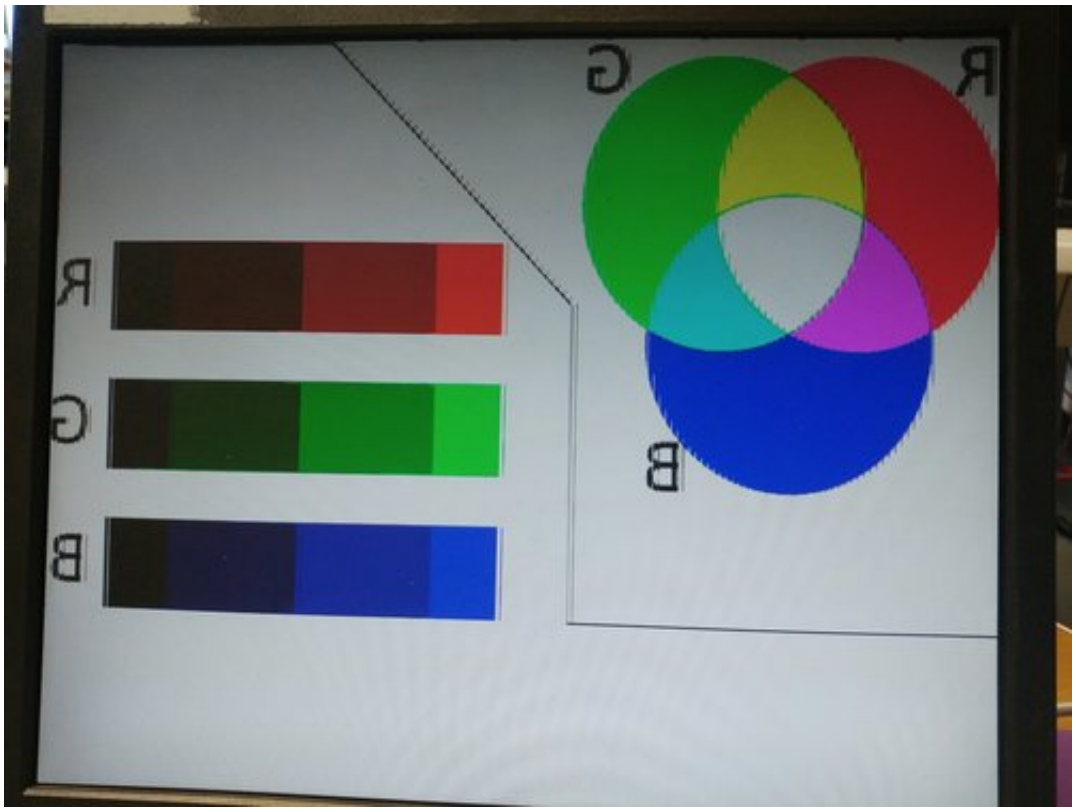


FIGURE 3 – miroir

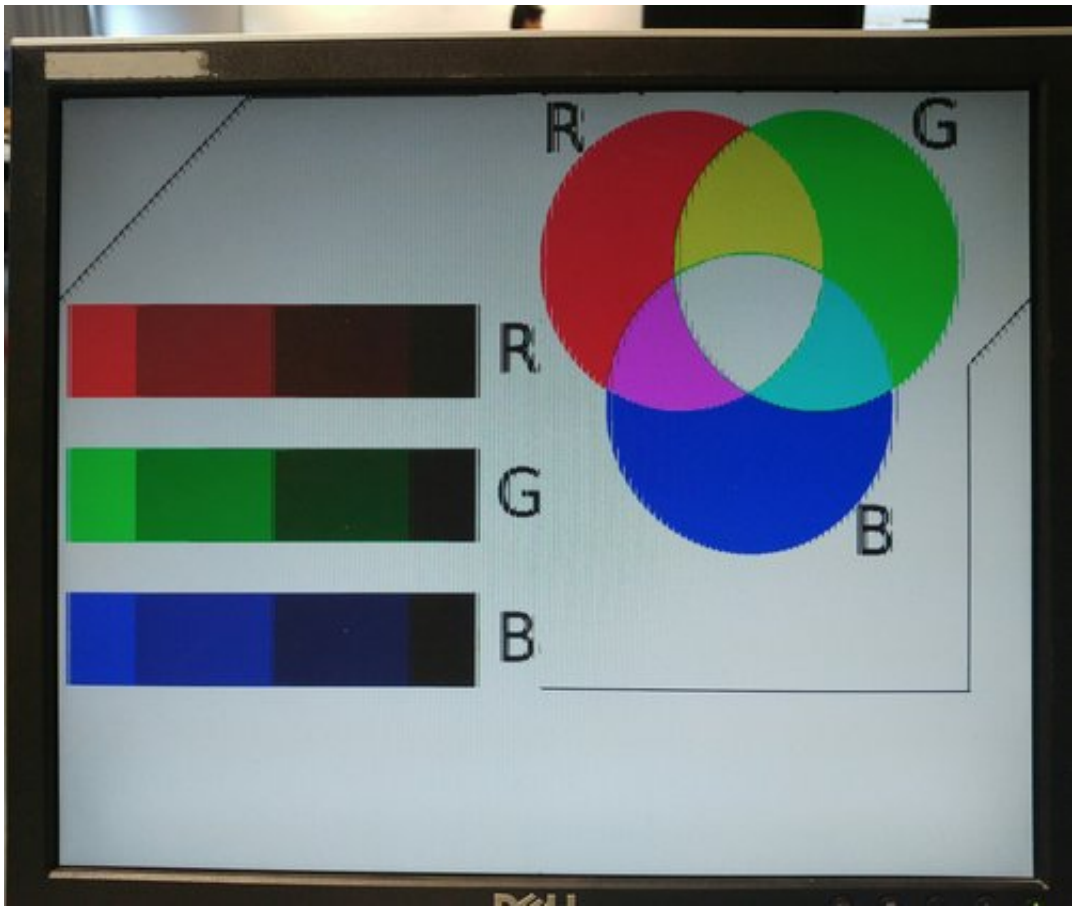


FIGURE 4 – symétrie horizontale

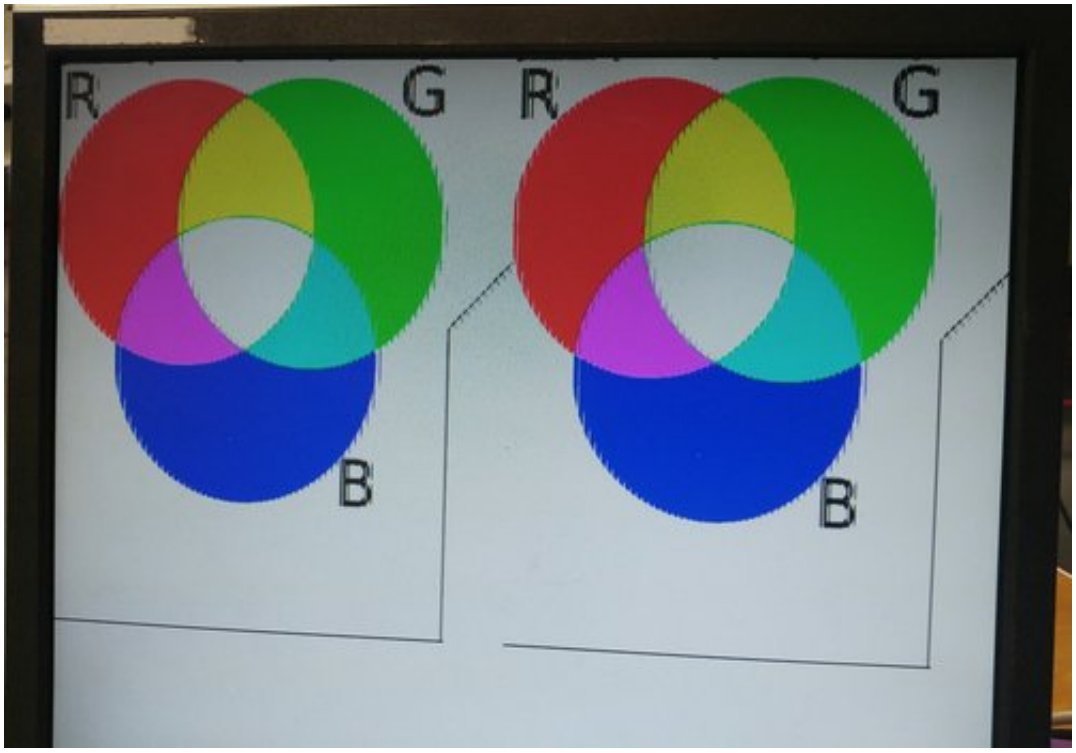


FIGURE 5 – recopie horizontale