

Réalisation d'un désobfuscateur vidéo

1 Introduction

Le démarrage d'une chaîne de télévision cryptée dans le début des années 1980 a nécessité la mise en place de techniques de brouillage du son et de la vidéo. Les techniques utilisées étaient relativement « frustrées » compte tenu du contexte technologique de l'époque (numérisation de la vidéo balbutiante, puissance des microcontrôleurs « grand public » très limitée. En ce qui concerne la vidéo, le principe choisi était très simple et exploitait le principe de transmission de l'image que nous rappelons ici :

- Une image est constituée de pixels affichés ligne par ligne, puis colonne par colonne dans chaque ligne.
- Pour « laisser le temps » aux équipements de se synchroniser en début de ligne et en début d'image, des temps morts sont organisés entre chaque ligne et chaque image et des signaux de synchronisation sont générés en début de ligne et d'image

Compte tenu de ces caractéristiques les ingénieurs ont retenu le principe de brouillage suivant:

- Chaque ligne de l'image transmise est retardée d'un temps $0 \Delta t$ ou $2\Delta t$ (avec Δt petit devant la durée d'une ligne de l'image mais suffisamment grand pour gêner la visibilité).
- Le choix du retard pour une ligne est fourni par un générateur pseudo-aléatoire dont la séquence est déterminée par la clef fournie au décodeur.

Ainsi, un petit microcontrôleur était suffisamment puissant pour calculer la séquence pseudo-aléatoire ligne à ligne, et les retards étaient obtenus de manière analogique par des assemblages de circuits LC....

Notre objectif est de réaliser un décodeur vidéo de ce type (essentiellement la gestion des retards) et d'explorer les premières techniques de piratage ainsi que les premières contremesures mises en place. Vous disposez d'un environnement déjà organisé (outil Quartus, fichiers source, ce sujet). La première compilation du projet étant un peu longue, **lancez dès maintenant une synthèse dans l'outil Quartus, et lisez la suite pendant ce temps.**

2 Prise en main de la maquette

Le projet Quartus pour la carte DE2 contient:

- Un générateur d'images (modules « mire » et « uncompress »)
- Un brouilleur d'images simulant la transmission brouillée, et la génération des séquences pseudoaléatoires du décodeur (module « scrambler »)
- Un décodeur squelette que vous devrez compléter.(module « descrambler »)
- Un contrôleur vidéo connecté à l'écran du PC (module « vga »)

Dès que le projet est compilé, **programmez le FPGA et observez le fonctionnement à l'écran.** Différents interrupteurs permettent de configurer la maquette. Testez ces différentes configurations pour vous familiariser avec le dispositif.

Vérifiez, tout d'abord que tous les interrupteurs sont en position « 0 ».

Le tableau suivant détaille le rôle des interrupteurs :

Interrupteur	Utilisation
sw[17]	→ 0: affichage d'une mire de carrés de 16x16 → 1: affichage d'un lapin...
sw[16:15]	→ 00: mode test, aucune ligne n'est retardée → 01: mode test, toutes les lignes sont retardées de Δt → 10: mode test, toutes les lignes sont retardées de $2\Delta t$ → 11: mode brouillage : le retard des lignes est aléatoire
sw[14:13]	→ 0x: Etape1 du contrôle : retards générés par un décodeur « officiel » → 10: Etape2 du contrôle : retards générés par un décodeur « pirate » → 11: Etape3 du contrôle : retards générés par un décodeur « pirate++ »
sw[1:0]	→ 00: Affichage de l'image originale en clair. → 01: Affichage de l'image brouillée → 1x: Affichage de l'image décodée (par vos soins)

3 Etape1

Il s'agit de compenser les retards des lignes en s'appuyant sur les informations envoyées par le décodeur. Pour cela vous éditez le fichier « descrambler.sv ». Les interfaces de ce module sont précisées dans le tableau suivant. **Attention : tous les signaux fournis ne sont pas forcément nécessaires à votre travail.**

Signal	Type	Rôle/Utilisation
clk	entrée	Horloge donnant le rythme d'arrivée des pixels (un pixel par cycle)
nreset	entrée	Signal pouvant être utilisé pour initialiser les blocs séquentiels. Actif à l'état 0
x (10bits)	entrée	Position de la colonne courante dans l'image. 0 signifie la première colonne à gauche de l'image. Il n'y a que 640 pixels visibles. La valeur 1023 correspond à un cycle d'horloge précédent le premier pixel.
y (10bits)	entrée	Position de la ligne courante dans l'image. 0 signifie la première ligne en haut de l'image. Il n'y a que 480 lignes visibles.
pixel_valid	entrée	Indique si le pixel courant est valide (dans la zone affichable), c'est à dire si x est compris entre 0 et 639 et si y est compris entre 0 et 479.
pixel_in (8 bits)	entrée	Le pixel entrant courant
pixel_out (8 bits)	sortie	Le pixel sortant courant
retard (2bits)	entrée	Indicateur provenant du décodeur indiquant le retard de la ligne en cours: <ul style="list-style-type: none"> • 00 signifie que la ligne n'est pas retardée • 01 signifie que la ligne est retardée de 20 pixels • 10 signifie que la ligne est retardée de 40 pixels
sel (2bits)	entrée	Indicateur provenant des switches sw[16:15] permettant de passer en mode test ou non. Cet indicateur est prioritaire sur le signal retard <ul style="list-style-type: none"> • 00 signifie qu'un retard fixe de 0 pixel est sélectionné • 01 signifie qu'un retard fixe de 20 pixels est sélectionné • 10 signifie qu'un retard fixe de 40 pixels est sélectionné • 11 signifie que le retard est déterminé par le signal « retard »

Votre travail consiste à générer des pixels sortants (pixel_out) dont le retard temporel est constant quel que soit le retard initial de la ligne indiqué par les signaux « retard » ou « sw »:

- Si le retard initial est nul , il faut retarder la ligne en cours de 40 pixels

- Si le retard initial est de 20 pixels , il faut retarder la ligne en cours de 20 pixels
- Si le retard initial est de 40 pixels , il ne faut pas retarder la ligne en cours.

Pour cela, nous vous proposons d'utiliser le principe des « mémoires tampon » circulaires. Vous disposez (voir dans le code source) de 2 mémoires de 32 mots de 8 bits. Ces mémoires sont des mémoires à écriture et lecture simultanées et synchrone, au front d'horloge :

- La donnée « mem_in » est écrite à l'adresse « write_add »
- La donnée « mem_out » est lue à l'adresse « read_add »
- ATTENTION : la donnée lue n'est présente qu'au cycle suivant la requête.

L'utilisation de ce type de mémoire en « ligne à retard » se fait suivant le principe suivant:

- Un premier compteur génère une adresse d'écriture en bouclant en permanence dans la mémoire
- Un deuxième compteur génère une adresse de lecture, en retard de N-1 cycles (N étant le retard voulu pour la ligne à retard) par rapport au compteur d'écriture.

Travail à effectuer, dans le fichier « descrambler.sv »

- *Coder un système de retards de 20 + 20 cycles sur les pixels entrants à l'aide des deux mémoires proposées et des compteurs biens conditionnés.*
- *Mettre en place un système de sélection des retards répondants aux contraintes provenant des signaux « retard » et « sw »*
- *Synthétiser le projet, corriger les erreurs et tester sur la maquette. Vous pouvez utiliser le mode « test » pour vérifier en détail le comportement de votre système.*

4 Etape 2

Placez les switches sw[14:13] de façon à passer à l'étape 2. Vous ne disposez plus des séquences pseudoaléatoires du décodeur.

Avec le système de brouillage original, les signaux respectant la norme de transmission de télévision de l'époque, il était relativement facile de détecter la position du début des lignes en examinant l'amplitude des signaux qui était très faible en début de ligne. Des « petits malins » ont exploité cette propriété pour réaliser des décodeurs pirates simplistes.

Nous avons reproduit ce schéma, en forçant la valeur des pixels en début de ligne à la valeur 0.

Le module « cherche_retard », disposant des mêmes entrées que le module « descrambler » sera en charge de déterminer la valeur du retard dans les lignes. Son seul signal de sortie est ce retard.

Travail à effectuer, dans le fichier « cherche_retard.sv »

- *imaginer un dispositif examinant les premiers pixels de la ligne en cours et déterminant son retard en conséquence*
- *coder ce dispositif en lieu et place de la valeur constante 0 indiquée dans le code original*
- *synthétiser le projet, corriger les erreurs et tester sur la maquette pour vérifier le bon fonctionnement.*
- *Remarque: suivant la finesse du dispositif, il peut y avoir quelques erreurs résiduelles qui ne seront pas considérées.*

5 Etape 3

Placez les switches sw[14:13] de façon à passer à l'étape 3.

Les ingénieurs de la chaîne cryptée se sont remis à l'ouvrage et ont constaté que l'on pouvait s'affranchir du respect des normes de transmission sans que les téléviseurs de l'époque en soit affectés. En ajoutant du bruit dans la zone de début de ligne, il n'est plus possible d'utiliser d'algorithme simpliste.

Nous avons reproduit ce schéma, en générant du bruit en début de ligne. Vous pouvez constater que votre décodeur pirate n'est plus opérationnel.

Pour surmonter le problème, il est possible de chercher la corrélation entre deux lignes consécutives et en déduire l'écart de retard.

Pour vous aider, nous avons systématiquement laissé la ligne 0 de l'image avec un retard nul.

Voici la démarche à suivre:

- Le code modifié sera dans le module « `cherche_correlation.sv` »
- Choisir un retard de 40 pixels pour la ligne 0.
- Stocker dans une mémoire (en s'inspirant du code du « `descrambler` ») une tranche de 64 pixels situés à partir de l'adresse 128 dans la ligne 0
- Comparer simultanément
 - 64 pixels de la ligne 1 commençant à l'adresse 128
 - 64 pixels de la ligne 1 commençant à l'adresse 108
 - 64 pixels de la ligne 1 commençant à l'adresse 88
- La comparaison se fera en faisant la somme des valeurs absolues des différences des pixels comparés (attention à la dynamique des calculs)
- En déduire le retard de la ligne 1 en déterminant la valeur minimale des 3 tests effectués.
- Stocker « les bons » pixels de la ligne 1 dans la mémoire précédente pour réitérer le travail en ligne 2, et ainsi de suite de ligne en ligne.

Travail à effectuer dans le fichier « `cherche_correlation` »:

- Coder un dispositif suivant l'algorithme décrit précédemment
- Tester sur la maquette. Normalement, la partie droite de l'image (après la colonne 128) doit être décodée

6 Etape 4 (optionnelle)

La méthode décrite en « étape 3 » permet de trouver le retard, mais « trop tard » car une bonne partie de la ligne est déjà passée. Pour pouvoir une image correcte, il faut faire les modifications suivantes :

- Dans « `cherche_correlation` », resynchroniser l'annonce du nouveau retard juste avant le début de la ligne suivante.
- Dans « `descrambler` », retarder les pixels entrants d'une ligne entière (640 pixels actifs) de manière à rendre cette ligne cohérente avec l'info de retard. L'image se déplacera verticalement d'une ligne..
- **ATTENTION : Ne vous engagez pas dans cette modification sans avoir fait constater par votre encadrant que l'étape 3 est valide.**