Introduction à SystemC

Mini-projet

Tarik Graba

P4 2018-2019

Télécom ParisTech

Table des matières

1 Objectifs :

Le but de ce mini-projet est de vous faire modéliser un système de traitement vidéo, qui prend en entrée un flux vidéo et qui lui applique au vol des transformées basiques.

On cherchera dans ce projet à être le plus efficace possible, en termes de ressources pour la simulation, de types de variables/signaux, de complexité et de lisibilité du code.

Plan du projet

Le travail demandé pour ce mini-projet se décompose en trois phases :

- récupération d'un module de génération de flux vidéo et de son environnement de test
- écriture d'un système de réception de flux vidéo
- écriture du système de transformation vidéo, et test de ce système.

Contexte

On cherche à modéliser et simuler un système de traitement vidéo, qui reçoit un flux vidéo en entrée, et qui restitue en sortie un flux transformé au même format. Les transformations appliquées peuvent êtres multiples :

- filtrage (flou léger)
- zoom in / zoom out
- …

Le format de flux vidéo choisi est une version simplifiée CCIR 601. En voici les grandes lignes (mais le web regorge d'information à ce sujet) :

- Le flux vidéo est transmis sur un bus comprenant 3 parties
  - une horloge à la fréquence pixel
  - les pixels, sous forme d'un quadruplet RGBA (8 bits pour chacune des composantes R, G, et B, et 8 bits pour l'alpha. 0=transparent, 255=opaque).
  - deux signaux de synchronisation : `HREF` et `VREF`.
- Une ligne vidéo se compose **d'environ** 864 pixels, dont **exactement** 720 actifs (c'est-à-dire qui contiennent des pixels valides. Les autres sont en nombre indéterminé, ils correspondent au temps nécessaire pour un retour ligne).

- Une image se compose **d'environ 625 lignes**, dont **exactement** 576 actives (les autres correspondent au temps de retour trame).

Pendant la transmission de pixels actifs d'une ligne, un signal de synchronisation, `HREF`, est mis à 1. Pendant le reste du temps, `HREF` vaut 0.

Au début d'une trame, un signal de synchronisation verticale, `VREF`, est mis à 1 pendant 3 lignes. Le front montant de `VREF` coïncide avec celui de `HREF`.

Les pixels sont transmis à la cadence de `13.5Mhz`, synchrones sur front montant de l'horloge.

Module d'entrée

Pour tester votre système, vous allez avoir besoin de modules d'entrée et de sortie vidéo :

- un module doit permettre de générer un flux vidéo à partir d'une séquence d'images,
- un autre module, doit permettre de générer des images à partir d'un flux pour vérifier visuellement le résultat.

Pour cela, nous vous proposons d'utiliser une série d'images au format PNG. Les fonctions de la bibliothèque `libpng`, une bibliothèque libre, permettront de lire et d'écrire *simplement* des images au format PNG.

On aurait pu utiliser d'autres bibliothèques pour lire directement des flux vidéo, mais cela aurait été un peu plus compliqué.

Le module de génération de flux vidéo vous est fourni. Il est disponible dans le dépôt `Git` suivant :

```
git@gitlab.enst.fr :se207/projet.git
```

Vous pouvez, cloner le dépôt séparément et copier manuellement les fichiers dans votre dépôt personnel ou le fusionner (`merger`) directement avec votre dépôt personnel comme suit :

```
git remote add projet git@gitlab.enst.fr :se207/projet.git
git remote update
git checkout master
git pull projet master --allow-unrelated-histories
```

Vous obtiendrez à la racine du dépôt un dossier `projet_video` contenant :

- un fichier de fonctions pratiques pour accéder simplement aux fonctionnalités de `libpng` (`image.{h,c}`)
- le module de génération de flux vidéo proprement dit (`video_in.{h,cpp}`)
- un environnement de test sommaire (`system.cpp`)
- un `Makefile`
- des images de test, tirées d'un film d'animation bien connu.

Pour des raisons de simplicité on se limite à des images en niveaux de gris. Les pixels venant du générateur de flux ne sont donc codés que sur **8bits**.

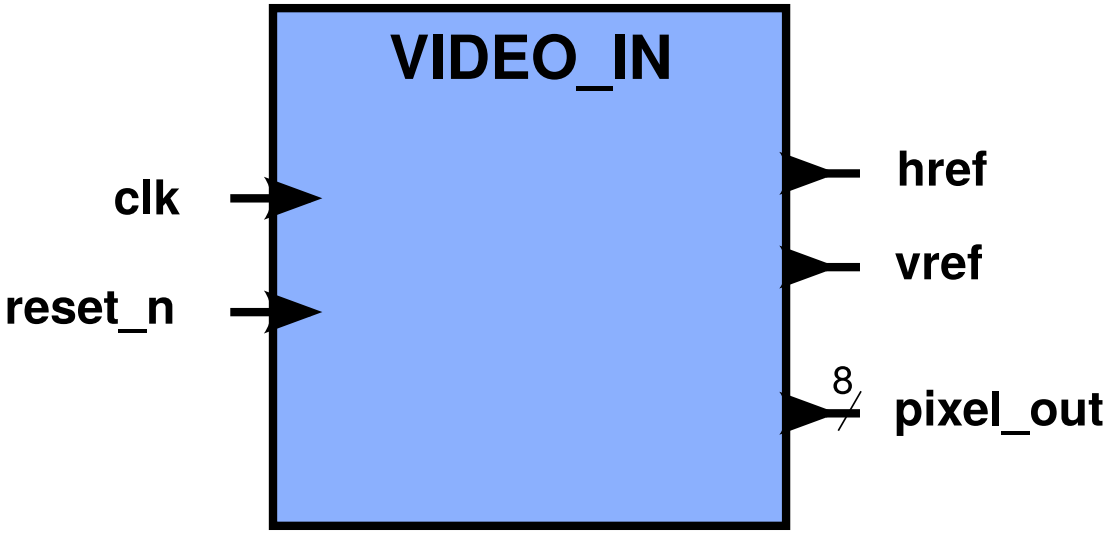

FIG. 1.1: Module générant le flux

Vous pouvez simuler le système tel quel et observer les traces générées (`simulation_trace.vcd`). Prenez le temps d'analyser ces chronogrammes et de vous familiariser avec le format *pseudo-CCIR601*.

2 Au travail :

module de sortie

Maintenant que vous avez un module qui vous fournit un flux vidéo, il serait sympa d'avoir un module qui fait l'inverse, pour vérifier visuellement les transformations vidéo.

C'est le rôle d'un module que vous allez concevoir et vous appellerez `video_out`.

Le travail à faire :

- écrire un module de récupération du flux vidéo, qui crée des images PNG à partir du flux vidéo.
  - Le nom du fichier généré doit dériver du nom du module une fois instancié.
- instancier dans votre environnement de test les deux modules `video_in` et `video_out`
- relier les deux modules directement entre eux et simuler le tout.

Les images de sorties doivent être les mêmes que celles d'entrée. Vous pouvez le vérifier en les affichant successivement. Par exemple :

```
display iinput0.png output0.png
eog iinput0.png output0.png
```
…

Faites attention, vérifiez bien qu'il n'y a pas de décalage, même d'un pixel, en début et fin d'image.

Attention :

Votre module `video_out` ne doit pas compter sur le fait que les trames font 625 lignes ni 864 pixels ! C'est à lui de se synchroniser (à l'aide des signaux HREF et VREF) sur le flux entrant. Il peut par contre compter sur le fait que dans chaque trame il y aura **exactement** `720×576` pixels valides.

Effets spéciaux

Le but du jeu est maintenant de construire un système qui va opérer, à la volée, sur un flux sortant du premier module.

Vous avez d'ores-et-déjà un module qui fournit des flux vidéos et un module qui en génère des PNG. Reste à écrire ce qui va s'interposer entre les deux.

On implémentera ces filtres (dans l'ordre) :

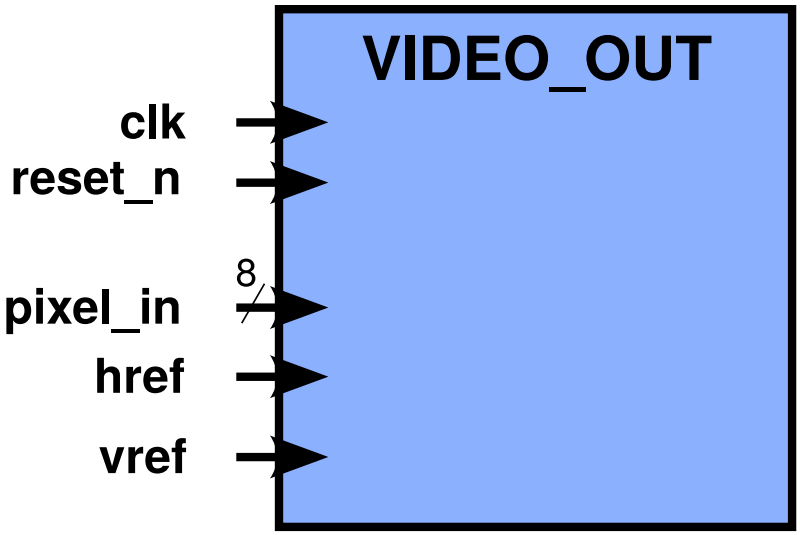

FIG. 2.1: Module récupérant le flux

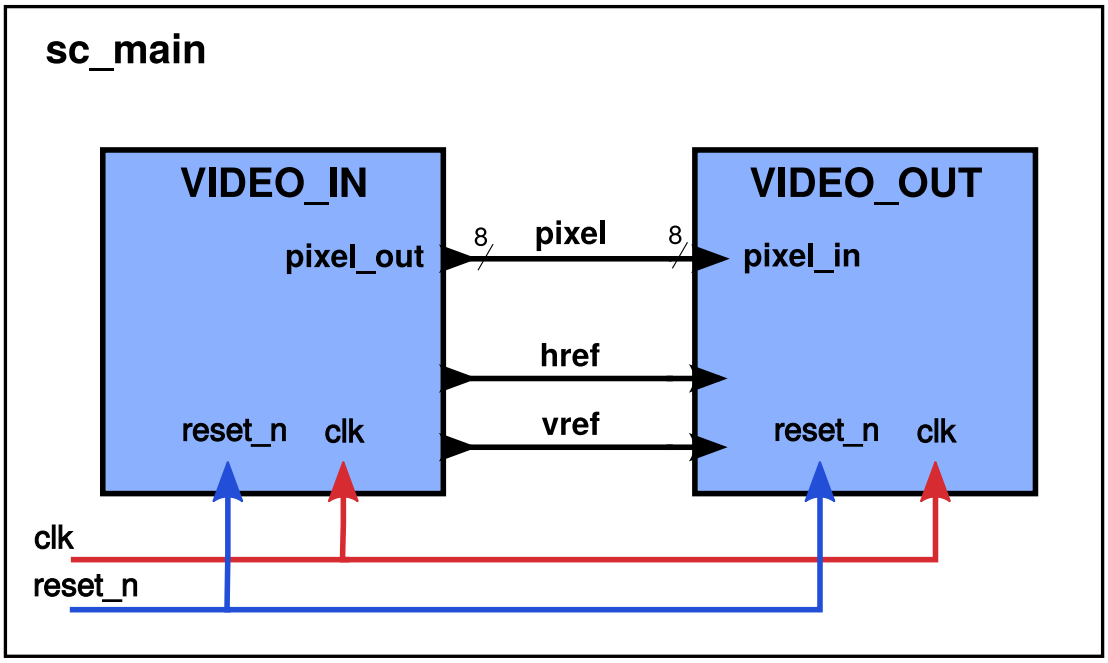

FIG. 2.2: Premier testbench

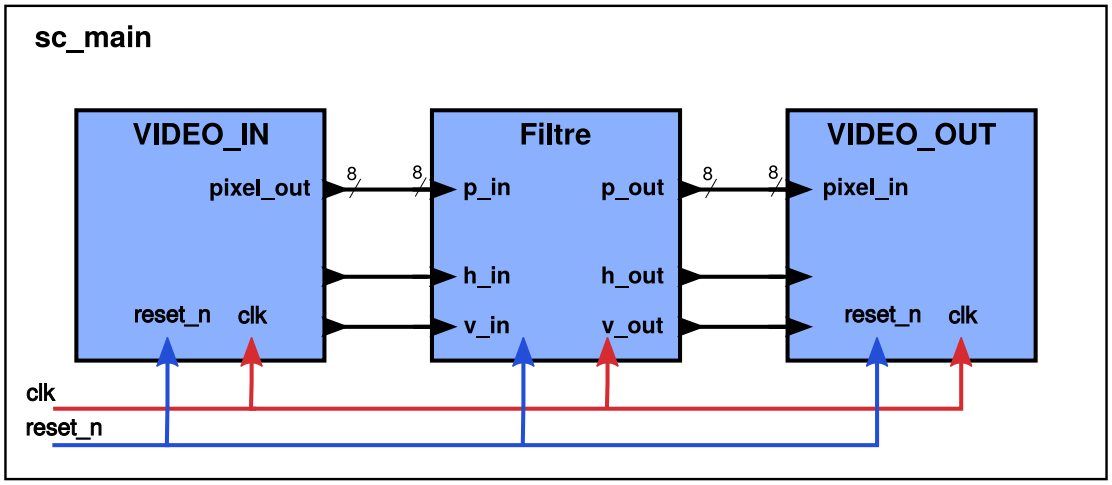

FIG. 2.3: Testbench d'un module de filtrage

- filtre moyenneur sur 8 pixels adjacents (pixel central + ses 8 voisins) : dans la mesure du possible, ce filtre doit effectuer son traitement au vol.
- zoom ×2 : on fera un zoom centré, en dupliquant chaque pixel. Un traitement au vol est-il possible et/ou adapté ?

Vérifiez visuellement que tout marche bien et que les signaux de synchronisation HREF et VREF respectent bien le format initial.

Si tout fonctionne correctement, vous devriez pouvoir chainer les différents modules en les instanciant les uns après les autres.

- **Bonus :** Le modèle du filtre moyenneur pourrait être utilisé pour tous les filtres agissant sur le même type de voisinage 3×3 (Sobel, Gaussien…). Modifiez le filtre moyenneur pour pouvoir préciser les paramètres du filtre à la construction du module.

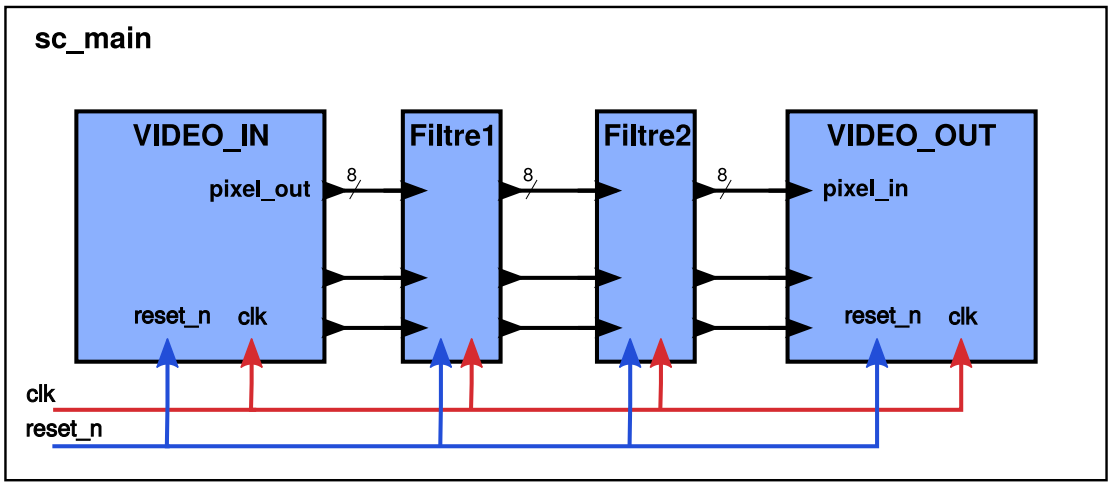

FIG. 2.4: Testbench avec plusieurs modules de filtrage