Règles de déroulement de l'épreuve

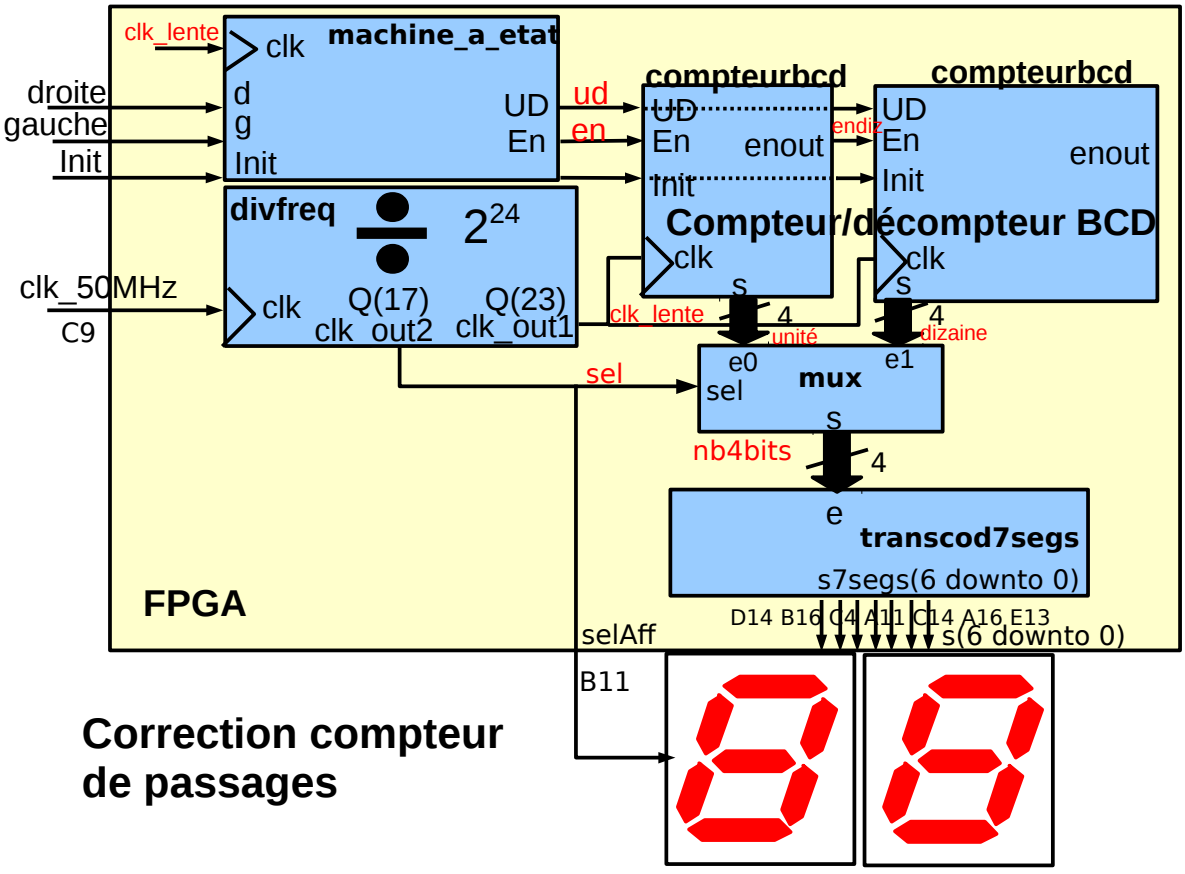
Il n'est pas interdit de communiquer verbalement entre binômes tant que cela ne perturbe pas le déroulement correct de cette séance.
Il vous est absolument interdit, par contre, de vous déplacer, d'échanger des feuilles de brouillons, des notes ainsi que des fichiers.

I) Introduction

Nous allons partir d'une version corrigée du compteur de passages du TP6 (non réalisé) et en faire un certain nombre de modifications. Engagez-vous dans cette épreuve avec comme état d'esprit, de suivre les indications sans trop vous poser de questions sur le pourquoi des modifications mais seulement de les exécuter et de montrer que le travail résultant fonctionne toujours. Le but de l'épreuve est ainsi d'évaluer comment vous appréhendez des modifications à partir d'un schéma complexe décrit en VHDL.
Vous disposez d'une feuille réponse sur laquelle vous répondez et sur laquelle l'enseignant notera ce qu'il a vu fonctionner et sinon, quelques indications sur l'état d'avancement de votre travail.

II) Compteur de passages corrigé.

Pour simplifier la lecture du fichier VHDL, on vous propose un schéma complet de la version corrigée.



Correction compteur de passages

Prenez le temps de remarquer les conventions du dessin, le nom des composants (en gras), le nom des entrées et sortie (à l'intérieur des rectangles bleus et du jaune) et le nom des signaux (en rouge).

Nous allons modifier cet ensemble pour réaliser petit à petit un ensemble capable de simuler le lancer de deux dés. Une des caractéristique d'un dé c'est qu'il affiche un résultat entre 1 et 6.

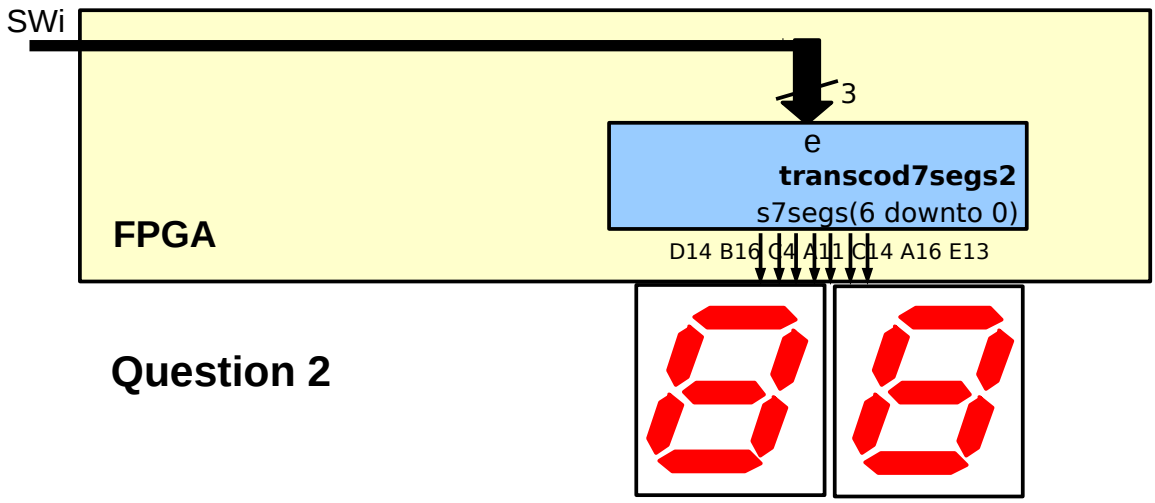
Notre affichage ne se fera pas sur des LEDs correctement disposées mais sur les afficheurs sept segments. Nous allons donc modifier petit à petit ce compteur de passages pour réaliser notre objectif. Votre travail consistera à piocher dans le fichier de correction et y puiser tout ce qui a déjà été réalisé pour une réutilisation et a y apporter toutes les modifications nécessaires et demandées.
Le fichier contenant la version corrigée peut être trouvée dans Ressources2015.zip (sur mon site) dans le répertoire Median2015 (fichier cmptPassageLast.vhd).

1) Travail préliminaire

Vous devez couper le fichier correction en petits morceaux comme indiqués dans les commentaires. Chacun des morceaux sera transformé en symbole et vous allez refaire le compteur de passages sans PORT MAP mais en schématique.

2) Affichage 1 à 6 au lieu de 1 à F

Puisque notre objectif est de réaliser un lancer de dés, on désire maintenant afficher des nombres variant de 1 à 6. Redimensionner et modifier le composant **transcod7segs** destiné à transcoder pour gérer une entrée sur 3 bits seulement. Si l'entrée binaire est 0 ou 7 ce transcodeur devra afficher un tiret sur le segment "g". Pour les tests les trois entrées seront reliées à des boutons poussoir (3 seulement) comme dans le dessin ci-dessous.
On vous demande :
- de prendre le fichier source de **transcod7segs** utilisé en question 1) et de le modifier.
RENOMMEZ SON ENTITE SI VOUS VOULEZ EN FAIRE UN SYMBOLE plus tard !!! C'est pour cela qu'il apparaît sous le nom transcod7segs2 dans la figure ci-dessous
- de construire le fichier ucf
- de réaliser un projet comprenant les deux fichiers (1 VHD + 1 UCF)
- de compiler en corrigeant les éventuelles petites erreurs, d'essayer l'ensemble et de faire valider.



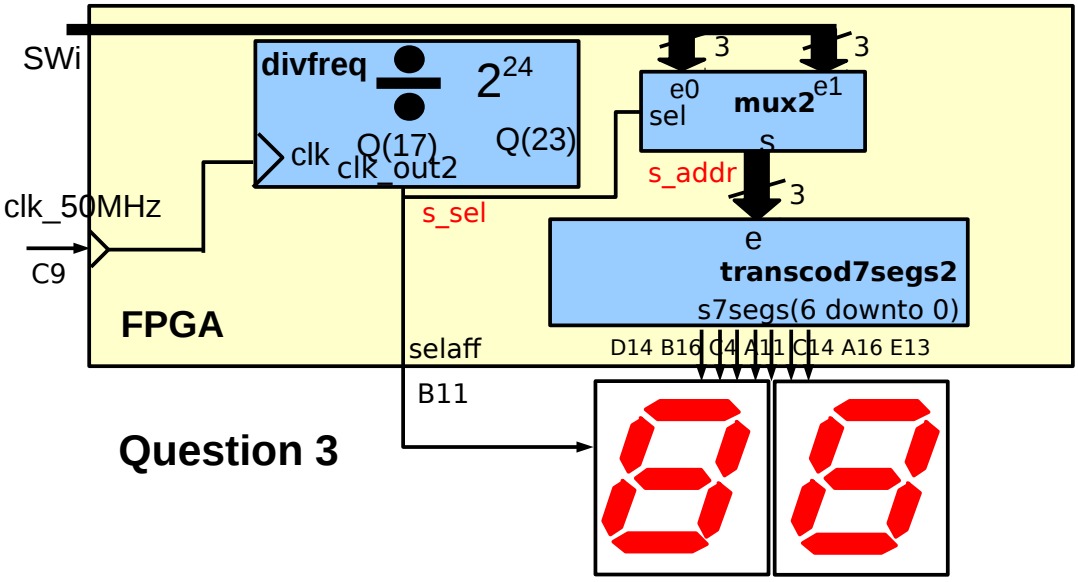
On ne gère pas les deux affichages pour le moment mais un seul... mais cela va changer

3) Travail à réaliser : affichage sur deux Digits

On ajoute maintenant le multiplexeur qui possède toujours deux entrées mais de trois bits seulement.
On vous demande
- de prendre le multiplexeur du fichier source disponible sur internet
- de le dimensionner pour trois bits et de l'ajouter au fichier qui contient le transcodeur
- de prendre le diviseur du fichier source et de l'ajouter aussi dans le fichier global

- de modifier le fichier ucf pour prendre 6 interrupteurs
- de réaliser un projet comprenant les deux fichiers (1 VHD + 1 UCF)
- de compiler en corrigeant les éventuelles petites erreurs, d'essayer l'ensemble et de faire valider.
- **Ce travail peut être fait en schématique si vous le désirez mais RENOMMEZ L'ENTITE DU MULTIPLEXEUR (comme dans la figure ci-dessous) SI VOUS VOULEZ EN FAIRE UN SYMBOLE !!! Il vous faudra réaliser 3 fichiers VHD, un par composant bleu (ci-dessous). Le symbole divfreq est déjà fait (question 1°) !!!!**

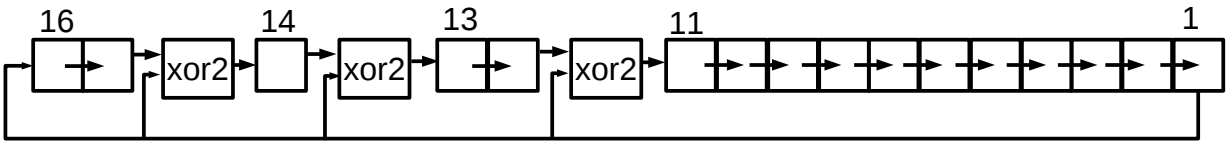
Le schéma de principe est :



III) Réalisation d'un registre LFSR

Le graphe d'états était réalisé par un composant appelé "**machine_a_etat**" dans la correction. Lisez-le pour vous imprégnez du style deux process que l'on va utiliser dans cette question. Autrement référez vous à votre polycopié de cours p 136 et suivantes.

Les LFSR (registres à décalage à rétroaction linéaire) sont une famille de générateurs pseudo-aléatoires. La taille (n en bits) du registre détermine sa périodicité (2^n-1). Même si 6 bits nous suffisent nous allons gérer un générateur de Galois de 16 bits dont voici le schéma tiré de Wikipédia (http://en.wikipedia.org/wiki/Linear_feedback_shift_register):



C'est un grand registre à décalage séparé par des calculs simples avec des ou exclusifs. Si l'on prend 16 bits pour n'en utiliser que 6 c'est uniquement pour avoir une périodicité assez grande. Les flèches matérialisent le passage de l'état présent vers l'état futur.

4) Travail à réaliser : LFSR

1°) Compléter sur la feuille réponse le schéma de calcul de l'état futur en fonction de l'état présent.

2°) En déduire les équations de récurrences bits à bits. Si vous ne voulez pas les écrire bit à bit

vous utiliserez l'opérateur '&' VHDL qui définit une concaténation. Réaliser ce LFSR en utilisant deux process, un qui calcule de manière combinatoire l'état futur en fonction de l'état présent et un qui sur front d'horloge mettra à jour l'état présent.

Indications :

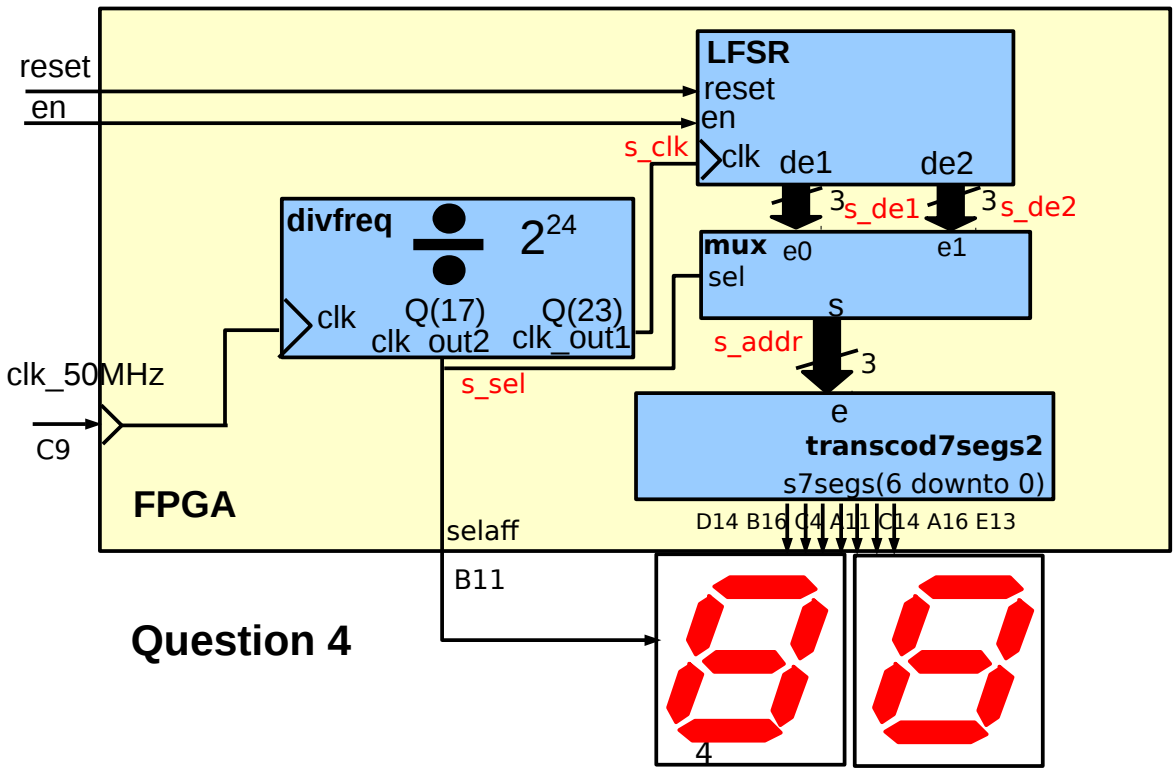
```
architecture aLFSR of LFSR is
signal etatpresent, etatfutur : std_logic_vector(16 downto 1);
signal s_xor1, s_xor2, s_xor3 : std_logic;
begin
  -- Calcul intermediaire des ou exclusifs
  s_xor1 <= etatpresent(15) xor etatpresent(1);
  s_xor2 <= etatpresent(14) xor etatpresent(1);
  s_xor3 <= etatpresent(12) xor etatpresent(1);
  -- Calcul de l'état futur en fonction de l'état présent et des ou exclusifs
  process(etatpresent) begin
    etatfutur(16) <= etatpresent(1);
    -- A compléter

  end process;
  process(clk,reset) begin
    -- A compléter : gestion du "reset" et du "en"
    -- Inspirez-vous du poly de TD/TP ou de cours
  end process;
 -- cablage des deux sorties
  de1 <= etatpresent(16 downto 14);
  de2 <= etatpresent(3 downto 1);
end aLFSR;
```

Il faut gérer un reset qui initialise à une autre valeur que 0 !!! "0000000000000001" est une bonne valeur.
Insérer ce LFSR comme indiqué dans le schéma ci-dessous et faire constater. Il vous faut aussi gérer une entrée "en" pour enable (en français autoriser) "l'incrémentation".

- Ce travail peut être fait en schématique si vous le désirez

Faire constater.



5) Ajout et modification du séquenceur

Pour que le joueur puisse déterminer quand il lance les dés, il nous faut ajouter un graphe d'états.

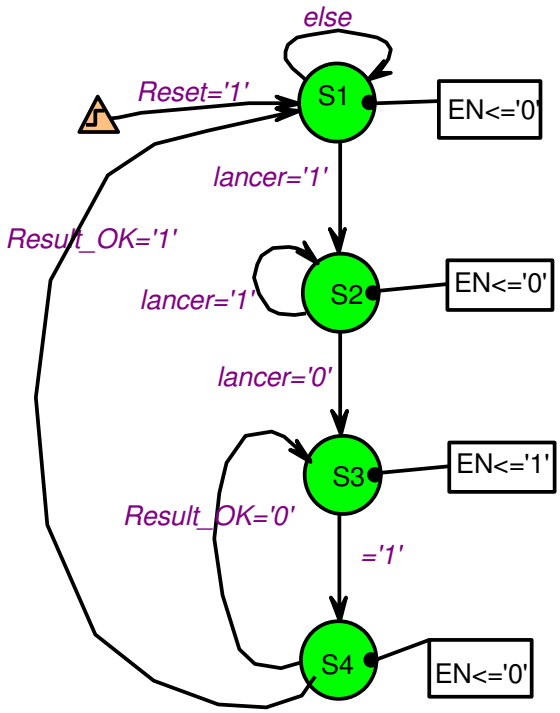
5-1) Détection des mauvais lancers

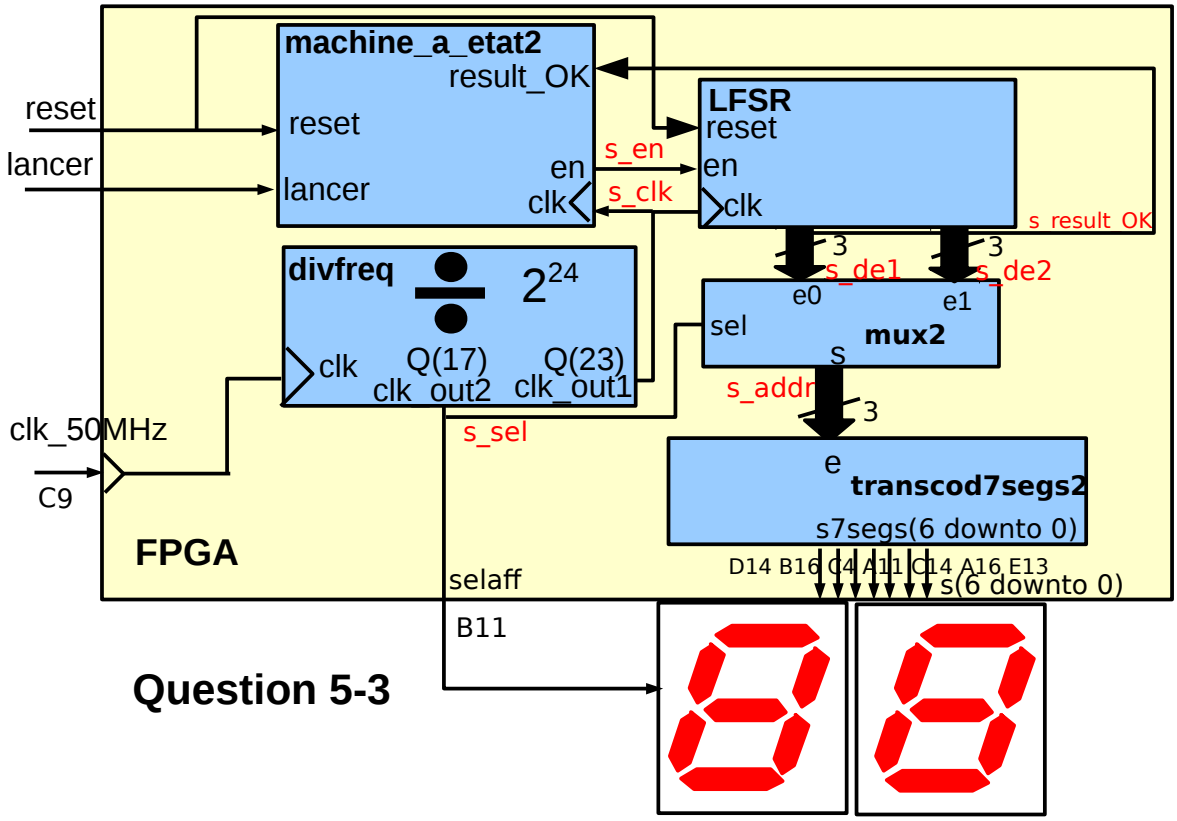
Sur le schéma ci-dessus les résultats de lancer de dés se trouvent dans deux signaux séparés appelés "de1" (pour dé1, les accents étant interdits) et "de2". Construire une fonction combinatoire capable de détecter que ni dé1 ni dé2 sont à 0 ou à 7.

Compléter l'équation correspondante sur votre feuille réponse : Result_NOK et Result_OK. Il est plus facile de raisonner sur Result_NOK qui est à '1' si l'un des dés affiche un résultat incorrect (0 ou 7).

5-2) Présentation du graphe d'évolution

Nous allons ajouter un graphe d'évolution qui a pour objectif de n'afficher les deux lancers que lorsque l'entrée "lancer" est positionnée à un. D'autre part ce graphe sera responsable d'un nouveau lancer au cas où un des résultats est zéro ou sept.
Voici donc le nouveau graphe d'évolution ci-contre.





5-3) Travail à réaliser

Programmer ce nouveau graphe d'évolution en modifiant le composant "**machine_a_etat**". L'insérer ensuite dans le composant global et le câbler. La réalisation du signal "s_Result_OK" sera fait avec une simple équation (en VHDL en tout cas).
Compiler et tester. Faire valider.
Ce travail peut être fait en schématique si vous le désirez mais RENOMMEZ L'ENTITE DE MACHINE_A_ETAT SI VOUS VOULEZ EN FAIRE UN SYMBOLE !!! La réalisation de l'équation pour s_result_OK est alors plus longue....

5-4) Travail final

Si vous avez testé la question 5-3, vous vous êtes aperçu que les résultats intermédiaires non valides sont affichés avec un tiret. Si l'affichage est seulement mis à jour pour un lancé correct, modifier l'ensemble pour que les résultats intermédiaires non valides ne soient plus affichés.

Indications : Cela peut être réalisé de manière séquentielle en ajoutant un état S5 au graphe d'états qui gère l'autorisation d'écriture "enAff" dans un registre d'affichage placé entre le **LFSR** et **mux2**. Ce registre doit être initialisé à une valeur fixe par le "reset" pour affichage au démarrage, et son entrée "en" sera reliée à la sortie "enAff" du séquenceur. Prendre un **FD4CE** et deux **FDCE** (catégorie Flip-Flop) si vous le faites en schématique.

Pouvez-vous dire pourquoi on ne peut pas mettre le registre d'affichage entre **mux2** et **transcod7segs2** ?

Le problème de cette solution est qu'il faut parfois attendre pour avoir l'affichage et que l'utilisateur peut s'impatienter ! Vous pouvez ajouter un chenillard tant que l'affichage n'est pas correct.