1. Objectif

Dans le TP précédent vous vous êtes familiarisé avec le langage Verilog afin de créer un contrôleur de mémoire RS232. Au cours de ce TP nous allons mettre en place un environnement de validation en System Verilog basé sur une architecture Séquenceur/Driver/Modèle du même type que celle utilisée pour la méthodologie OVM.

Au cours de ce TP vous allez :

- étudier le fonctionnement et le rôle de chaque composant du test
- concevoir un moniteur et un générateur de scénarii de test
- mettre en œuvre les bases du System Verilog (classes, interfaces, …)
- continuer/débugger le contrôleur mémoire RS232

2. Spécification de test

L'objectif d'une spécification de test est de définir les objectifs de validation ainsi que les moyens à employer pour les atteindre. Cela permet de répondre aux questions Quoi?/Comment ?

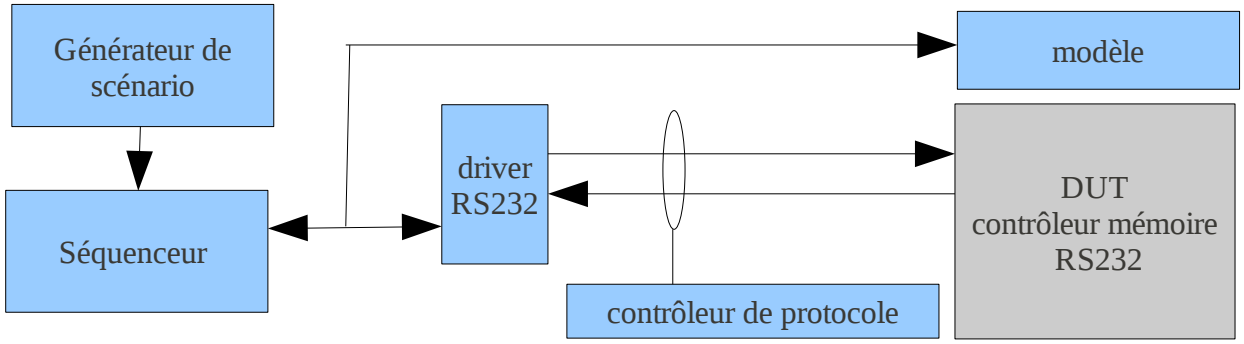
Pourquoi définir des objectifs ? Tout simplement parce que les combinaisons possibles d'entrées/sorties/état interne augmentent exponentiellement avec la complexité du module à tester. Il n'est donc pas envisageable de tester le module dans sa globalité dans un temps raisonnable. Il faut alors définir l'ensemble de critères à vérifier qui donneront le meilleur taux de confiance au regard du temps imposé par les contraintes de time-to-market. Cette ensemble de critères sera consigné dans un document qui servira de feuille de route dans la rédaction du plan de test ; chaque critère devant être vérifié par le test.

Dans le cadre du contrôleur mémoire RS232, les critères peuvent être :

- chaque bit d'adresse de la mémoire doit être exercé / chaque bit de donnée de la mémoire doit être exercé
- une erreur doit être obtenue lorsqu'une écriture est effectué sur une mémoire en lecture seule ou en cours d'effacement
- une erreur doit être obtenue lorsqu'une lecture est effectuée sur une mémoire en cours d'effacement
- une erreur de parité doit conduire à un code d'erreur
- le module doit fonctionner pour les diviseurs 1/2/3/40/100

Une fois les objectifs clairement identifiés, il devient aisé de déterminer une architecture de test.

Dans le cadre du contrôleur mémoire RS232, l'architecture retenue sera la suivante :



Rappel:

Le rôle du **séquenceur** est de « séquencer » les transactions provenant de un ou plusieurs g**énérateurs de scénario** qui joueront le rôle de stimuli de test. Ces transactions sont ensuite transmises au **driver** et au **modèle**. Dans notre cas, nous n'allons utiliser qu'un seul générateur de scénario : le séquenceur aura simplement un rôle de routeur.

Le **modèle** construit une image du module à tester et la compare en permanence aux réponses du DUT.

Le rôle du **driver** est de faire le lien entre les transactions haut niveau (TLM) et le bus (RTL). Le module est bi-directionnel : TLM vers RTL et RTL vers TLM.

Le **contrôleur de protocole** est chargé d'analyser en permanence le trafic sur le bus afin de détecter d'éventuelles erreurs de protocole (aucune fonctionnalité n'est testée ici). Le contrôleur de protocole ne sera pas étudié dans ce TP.

3. Travail à réaliser

1. écrire le squelette de la classe RS232_sequenceur avec pour attribut le driver RS232 ainsi que son constructeur
2. ajouter à la classe RS232_sequenceur les tâches send_data et receive_data. Les tâches send_data et receive_data appelleront dans un premier temps directement les tâches du driver.
3. écrire le squelette de la classe test_scenario en System Verilog ayant pour attribut le séquenceur RS232 ainsi que son constructeur.
4. ajouter à la classe test_scenario les tâches set_memory_protection, erase_memory, write_to_memory et read_from_memory permettant respectivement de configurer les règles de protection, d'effacer la mémoire, d'écrire et de lire dans la mémoire.
5. ajouter à la classe test_scenario la tâche run_test générant aléatoirement des accès en lecture/écriture dans la mémoire.
6. écrire la classe RS232_monitor contenant les fonctions received_data et sent_data. Cette classe a pour but de créer une modèle du contrôleur mémoire RS232. Chaque fois qu'une donnée est reçue ou transmise le moniteur met à jour son état interne et vérifie la validité du comportement du D.U.T
7. modifier la classe RS232_sequenceur afin que les transactions soient en plus transmises au moniteur RS232_monitor.
8. Mettre à jour le test bench afin d'instancier les classes précédemment créées et appeler la fonction run_test().
9. Débugger !!!