TP3 : Conception d'une UART et Crypteur/Décrypteur série D.E.S. 56 bits

Objectifs

- ➢ Comprendre les liaisons série RS232.
- ➢ Concevoir une UART
- ➢ Utiliser une IP externe d'encryptage/décryptage D.E.S. (code fourni, issu de OpenCores)
- ➢ Construire le séquencement du système complet RS232-Rx / Cryptage / RS232-Tx
- ➢ Construire un banc de test avec Entrées/Sorties fichiers.
- ➢ Comprendre la nécessité d'utiliser un buffer élastique (simulation).
- ➢ Implémenter la solution, essayer et vérifier le problème créé par l'absence de buffer.
- ➢ Ajout d'un buffer élastique : utiliser l'instanciation de macro-fonctions FPGA.

Les étapes de ce projet seront :

- Compréhension des transmissions séries asynchrones.
- Création de l'UART **Réception** et Test par simulation (le modèle comportemental de l'émission, fourni, facilite cette partie).
- Création de l'UART **Émission** et Test par simulation (l'émission est plus simple que la réception, et peut être testée avec la partie réception conçue auparavant et/ou comparée au modèle comportemental fourni).
- Création d'un **Banc de Test**.
- Compréhension du fonctionnement « boîte noire » du module d'encryptage/décryptage D.E.S. 56 bits fourni. Le banc de test qui est fourni permet de voir « vivre ce module » qui est facile d'emploi. Noter qu'il n'est pas demandé une compréhension du fonctionnement interne du bloc D.E.S. (bien que le code soit fourni).
- Création du module de pilotage de l'encrypteur. Ce module gère les flux RS232, accumule les caractères par groupe de 8 (64 bits), demande l'encryptage, récupère le bloc de 8 caractères encryptés, et le retourne par la liaison RS232.
- Test du système par simulation avec le banc de test créé auparavant, et mise en évidence de problème liés à l'absence de buffer élastique (perte de caractères).
- Test sur maquette
- Ajout d'un buffer élastique (Fifo).
- Vérification par simulation, puis sur maquette avec le fichier binaire encrypté fourni.

Ce qu'il faut faire :

UART

NB : on adoptera le format « N81 » : pas de parité, huit bits de données, 1 bit de stop.

- ➢ Chercher (Internet, littérature) le fonctionnement d'une liaison RS232 : transmission asynchrone.
- ➢ Ouvrir le fichier UARTS.vhd
- ➢ Penser à re-synchroniser l'entrée !
- ➢ Écrire le diviseur de fréquence qui construit une impulsion au rythme d'un demi-bit transmis. Pour 115.200 bauds, sa fréquence doit être double. Ce diviseur doit pouvoir être remis à zéro par la machine d'états de réception (pour synchroniser sur le début de caractère).
- ➢ Écrire la Machine d'Etats de la réception UART. On peut par exemple coder les phases :
  - attente de début de caractère, et clear du diviseur sur la descente du signal d'entrée.
  - attente du premier demi-bit : Start
  - attente de huit bits de données
  - attente et vérification du bit de stop, impulsion de donnée disponible, et retour au début.
- ➢ Simuler et tester cette partie, en utilisant le modèle comportemental (procedure) fourni.
- ➢ Écrire le code de l'émission de caractère. Ceci ne devrait pas poser de problème. Une machine d'état très simple suffit. Penser à construire un diviseur indépendant de la réception ! Il pourra fournir la période entière et non la demi période.
- ➢ On pourra tester l'ensemble par exemple par bouclage avec un délai de 1 ms

On pourra éventuellement tester cette partie en construisant une petite application qui guette l'arrivée d'un caractère, l'incrémente, et le renvoie à l'UART. Tester alors cette application sur la carte et avec un Terminal (Puty ou TeraTerm sur Windows et Screen ou Minicom sur Linux). Ceci valide les flux dans les deux sens.

Crypter D.E.S. 56 bits

Cette partie a été extraite du site OpenCores.org. Elle comporte trois fichiers :
- DES_lib.vhd est le package
- DES-round.vhd est le bloc unitaire
- DES_small.vhd est le contrôleur qui gère le cycle de cryptage.

Pour faciliter la mise en oeuvre, nous fournissons un banc de test et un script de simulation permettant de mettre en oeuvre et de voir fonctionner ce module.

- ➢ Analyser rapidement le système de cryptage, tout au moins dans son interface avec le monde extérieur. Simuler pour voir le fonctionner (l'encryptage demande un certain nombre de cycles d'horloge).
- ➢ Ouvrir le fichier Crypter.vhd et coder sa fonctionnalité :
  - recevoir les données 8 bits et les assembler en mot de 64 bits,
  - envoyer ce mot à DES_small,
  - attendre son encryptage,
  - retourner la donnée encryptée de 64 bits par mots de huit bits au fur et à mesure de la disponibilité du système externe.

➢ Estimer le flux maximal qu'il est possible d'encoder ainsi.
Comparer avec des solutions logicielles.

Application complète

➢ Coder Top.vhd. En principe, il n'y a pratiquement rien d'autre à faire que de connecter entre eux UARTS et le CRYPTER.

➢ Simuler l'application complète.
Une première passe en mode encryption doit permettre de construire un fichier (binaire ! On pourra utiliser une notation hexadécimale. En effet, l'encryptage transforme des caractères ASCII 7 bits en n'importe quoi, huit bits et caractères de contrôle compris).
Une deuxième passe en mode décryptage doit décoder le fichier construit dans la première passe.

➢ Il est très probable que l'on constate qu'il manque des caractères à la restitution !
Expliquer le phénomène et chercher un correctif (plusieurs méthodes sont envisageables).

➢ Tester sur la carte avec le fichier crypté fourni « test.txt ».
Noter ici le texte en clair :

_____

➢ Pour terminer, ajouter une mémoire tampon élastique : FIFO.

➢ Vérifier qu'il n'y a plus de perte de caractère et que le système fonctionne sur la carte

Option : on pourra chaîner deux cartes, l'une en encodeur et l'autre en décodeur et transmettre des fichiers volumineux.