TP2 : Conception d'un UART

Les étapes de ce TP seront :

- Compréhension des transmissions séries asynchrones.
- Création de l'UART **Émission** et Test par simulation (l'émission est plus simple que la réception)
- Création de l'UART **Réception** et Test par simulation
- Tests sur carte FPGA
- **Crypter et décrypter** le message série avec une IP DES 56 bits de opencores.

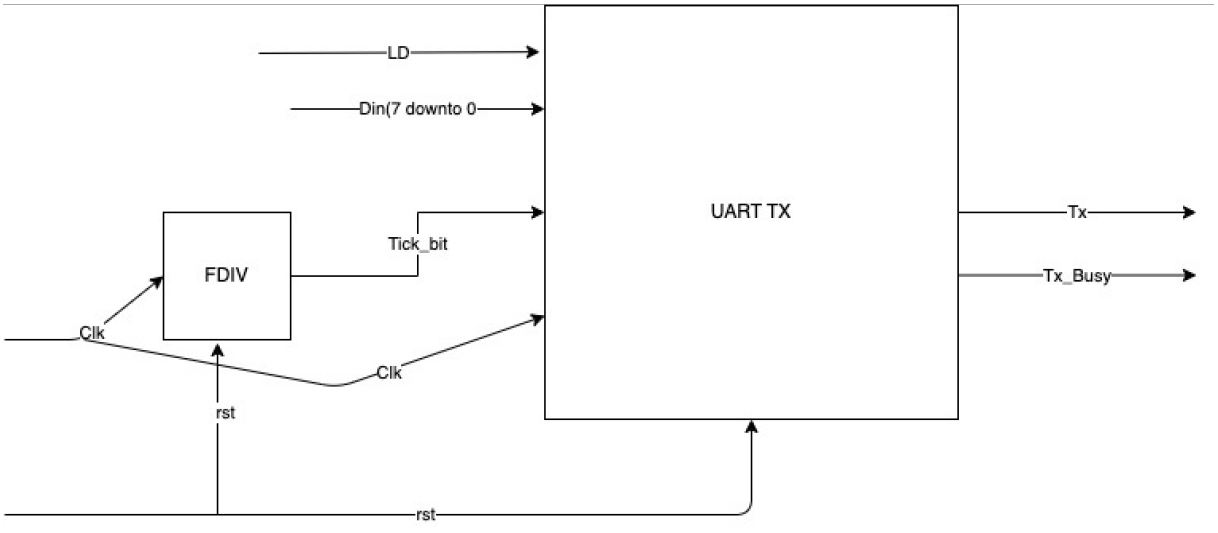
Partie 1 : Conception de l'UART (16 points)

NB : on adoptera le format « N81 » : pas de parité, huit bits de données, 1 bit de stop avec un baud rate de 115200 bps.

Chercher sur internet le fonctionnement d'une liaison RS232 : transmission asynchrone.

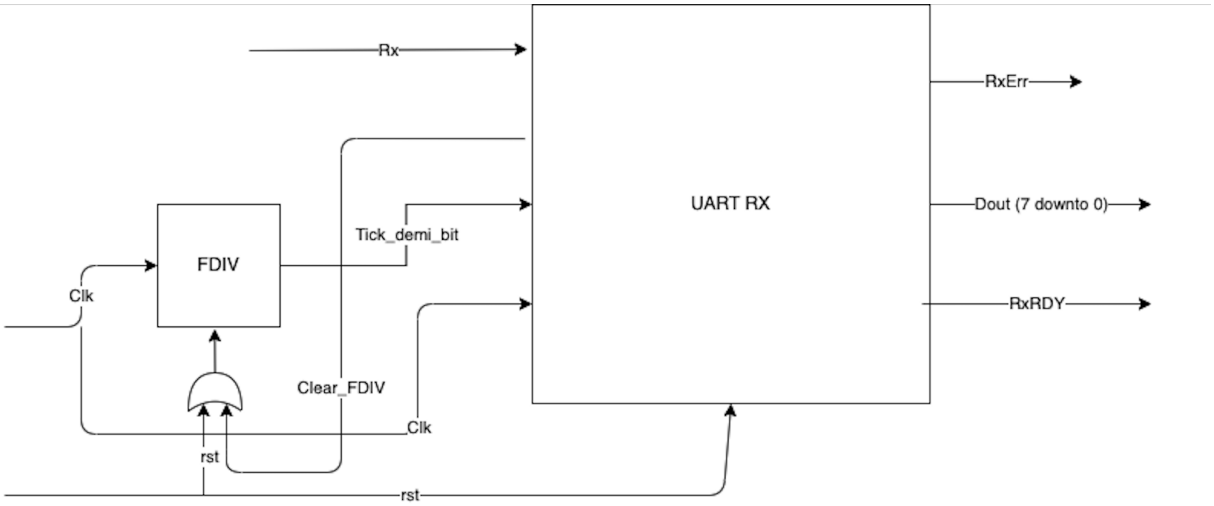
Quelques conseils pour la partie émission :

- ➢ Écrire le code de l'émission de caractère. Une machine d'état très simple suffit.
- ➢ Le diviseur de fréquence qui génère une impulsion au rythme d'un bit transmis est déjà fourni. Il permet de générer la période (baud rate avec Tick_bit) pour l'émission des bits.
- ➢ Simuler la partie émission avec le banc de test auto vérifiant et le script de simulation qui sont déjà fourni dans le dossier simu.



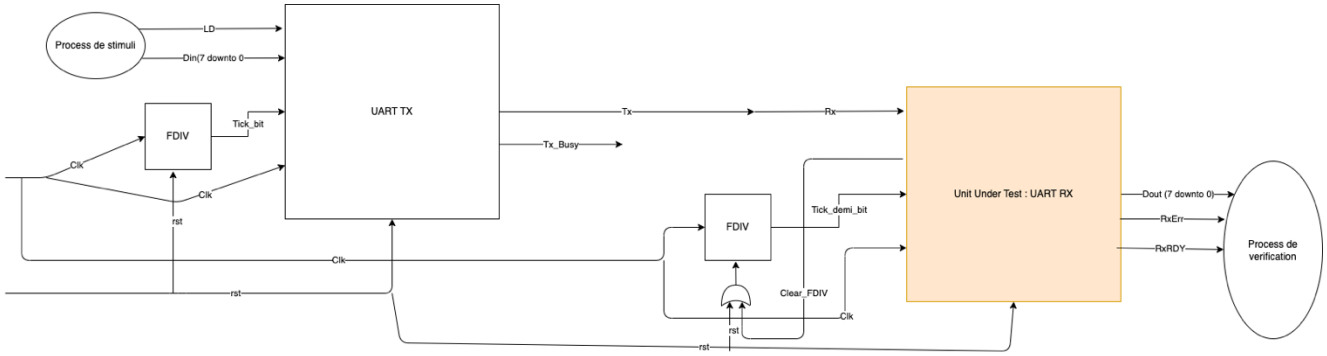
Quelques conseils pour la partie réception :

- ➢ Penser à resynchroniser l'entrée !
- ➢ Le diviseur de fréquence (FDIV) construit une impulsion au rythme d'un demi-bit transmis. Pour 115.200 bauds, sa fréquence doit être double. Ce diviseur doit 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.

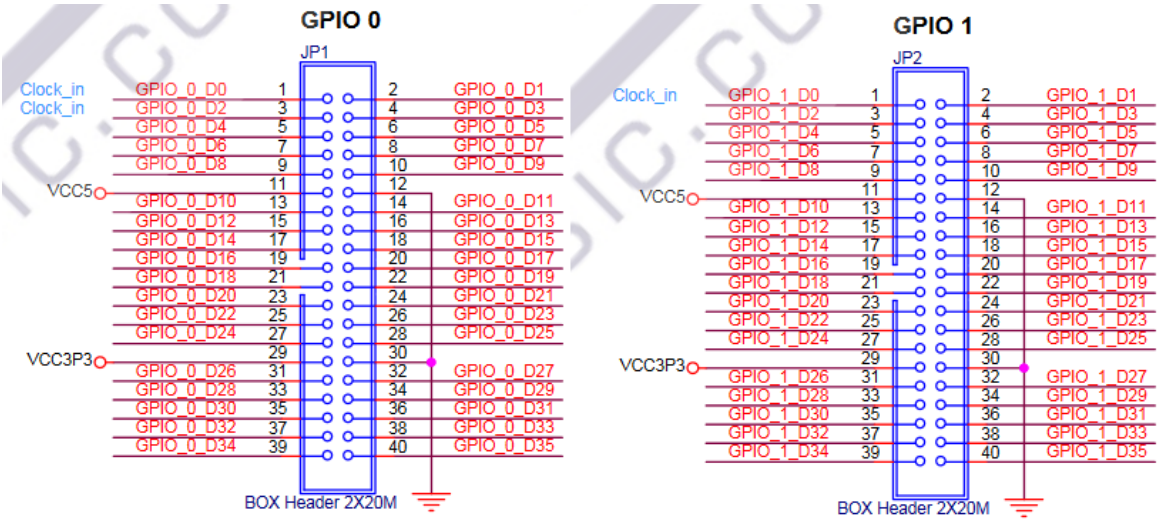


- ➢ Simuler la partie réception avec un banc de test et un script de simulation. Vous pouvez utiliser la partie émission pour tester par simulation la partie réception.

Voici un schéma exemple d'un banc de test pour vérifier votre partie réception :



Ports GPIO sur la carte DE1_SoC :



Liste des étapes à valider :

1 / Validation de la partie émission FPGA → PC (10 points)

Tester la partie émission de l'UART sur la carte FPGA en connectant la sortie Tx sur une pin du port GPIO (GPIO_(0) par exemple), l'entrée data sur SW(7 ..0) et l'entrée Go sur KEY(1). Ensuite utiliser un convertisseur série/USB pour recevoir les caractères sur le terminal de votre PC (Puty sur Windows ou Screen sur Linux et Mac). Il faudra connecter la sortie Tx (GPIO(0)) sur le pin Rx du convertisseur série/USB.

Sur la carte DE1_SoC : GPIO_0_D0 → GPIO_0(0) → PIN_AC18

2 / Validation de la partie réception PC → FPGA (4 points)

Tester la partie réception de l'UART sur la carte FPGA en connectant l'entrée Rx sur une pin du port GPIO (GPIO(1) par exemple), la sortie Data sur les LEDs(7 ..0), la sortie DAV sur la LED(8) ou sur des afficheurs 7 sehments (HEX0 et HEX1) et la sortie erreur sur le LED(9). Ensuite utiliser un convertisseur série/USB pour envoyer un caractère depuis le terminal de votre PC (Puty sur Windows ou Screen sur Linux et Mac). Il faudra connecter l'entrée Rx (GPIO(1)) sur le pin Tx du convertisseur série/USB.

Sur la carte DE1_SoC : GPIO_0_D1 → GPIO_0(1) → PIN_Y17

3/ Validation de l'émission / réception (1 point)

Construisez une petite application qui guette l'arrivée d'un caractère sur Rx, l'incrémente, et le renvoie sur Tx. Tester alors cette application sur la carte FPGA avec un Terminal (Puty sur Windows et Screen sur Linux). Ceci valide les flux dans les deux sens.

4/ Amélioration envisageable (1 point)

Une entrée Baud sur un bit doit permettre de choisir entre 2 vitesses de transmission qui sont configuré grâce à des paramètres génériques.
Par exemple :

Baud = ‘0’ → baud rate = 115200, (paramètre générique Baud1)
Baud = ‘1’ → baud rate = 19200, (paramètre générique Baud2)

Deux paramètres génériques permettant de rajouter un bit de parité : Parity et Even
Si Parity = False → pas de bit de parité
Si Parity = True, Even = True → Bit de parité paire
Si Parity = True, Even = False → Bit de parité impaire

Partie 2 : Crypteur/Décrypteur série D.E.S. 56 bits (4 points)

- ➢ 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 cette partie seront :

- 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.

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 œuvre, nous fournissons un banc de test (**DES_small_tb.vhd**) et un script de simulation (**simu_DES.do**) permettant de simuler et de voir fonctionner ce module.
Analyser rapidement le système de cryptage, tout au moins dans son interface avec le monde extérieur.
Simuler pour le voir fonctionner (l'encryptage demande un certain nombre de cycles d'horloge).

Le banc de test permet d'encoder un vecteur de 64 bits puis le décoder avec plusieurs valeurs d'entrées et plusieurs clefs générés pseudo aléatoirement.

Exercice 1 : Simulation du cryptage et décryptage d'un message

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.

Pour tester cette partie vous avez un banc de test (**Crypter_tb.vhd**) et un script de simulation (**simu_Crypter.do**).

Le banc de test comporte 2 architectures :

- Une première (test1) qui permet d'encoder un message et ranger le résultat en code hexadécimal dans un fichier texte **fout.txt**
- Une deuxième (test2) qui permet de décoder le message et ranger le résultat dans un fichier texte **decoded.txt**

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

Exercice 2 : Application complète

➢ Coder **DE_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.