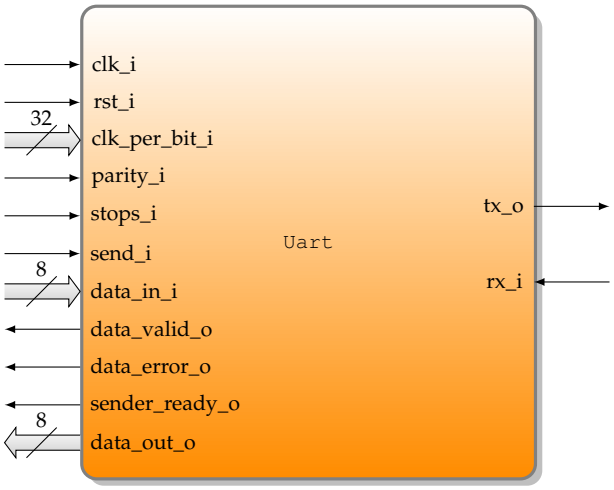
HAUTE ÉCOLE D'INGÉNIERIE ET DE GESTION DU CANTON DE VAUD

Prof. Yann Thoma

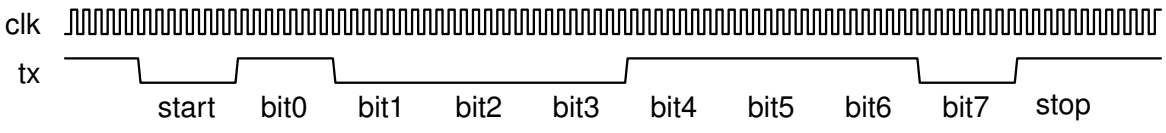
Exercices du cours VSE
Plan de vérification d'un UART
semestre automne 2023 - 2024

Contexte

Soit un UART dont l'entité est la suivante :



Cet UART permet d'envoyer des octets sur une ligne série, et d'en recevoir également.
Pour l'envoi, il suffit de placer `send_i` à '1' pendant un cycle d'horloge, et à ce moment-là la donnée présente sur `data_in_i` au même instant sera ensuite envoyée en série sur la sortie `tx_o`. Attention, une donnée ne peut être envoyée que lorsque `sender_ready_o` est active. Si elle est inactive, la donnée ne sera simplement pas traitée.
Pour la réception, les données sérielles reçues sur `rx_i` sont ensuite transmises sous forme d'octet, via `data_out_o`. La donnée reçue est considérée comme valide lorsque `data_valid_o` est à '1', et ce pour un seul cycle d'horloge.
La sortie `data_error_o` passe à '1' lorsqu'il y a une erreur détectée sur un start bit ou un stop bit.
Autant pour l'envoi que pour la réception, `parity_i` permet d'indiquer la présence d'un bit de parité ('1') ou non ('0'). `stop_bit_i` indique le nombre de stop bit : 1 stop bit si '0', et 2 stop bits si '1'. Enfin, `clk_per_bit_i` donne, sur 32 bits, le nombre de cycles d'horloges nécessaires à l'envoi/réception d'un bit.
Finalement, un paramètre générique, `FIFOSIZE`, permet de choisir la taille de la FIFO présente dans le composant UART en transmission. Cette FIFO interne permet de bufferiser les données et de les envoyer ensuite.
Le chronogramme suivant donne l'exemple d'un envoi de la valeur 0x71 avec `parity_i` à '0', `stops_i` à '0' et `clk_per_bit_i` à 8.



Exemples de bits de parités :

data_i	b_{parity}
00000000	0
00000001	1
00101101	0
01101101	1

Exemples d'envois

data_i	parity_i	stops_i	Observé sur la ligne
01110011	0	0	0 | 11001110 | 1
01110011	1	0	0 | 11001110 | 1 | 1
00110011	1	0	0 | 11001100 | 0 | 1
01110011	0	1	0 | 11001110 | 11
01110011	1	1	0 | 11001110 | 1 | 11

Exercice

A partir de la description ci-dessus, proposez un plan de vérification. Pour ce faire remplissez le fichier `uart_testplan.xml` fourni avec des tests permettant de vérifier des *features* du système. Pour chaque test indiquez la *feature* mise en avant, sa priorité, le scénario à jouer et ce qui doit être vérifié.
Le fichier est formatté pour QuestaSim. Pour l'éditer il faut utiliser Excel (pas LibreOffice, mais bien Excel). En théorie lors de la sauvegarde Excel devrait garder le format XML. Si ce n'est pas le cas il faut le forcer.
⚠ Avant de modifier le fichier, vérifiez les étapes suivantes :
Un fichier `default.rmdb` vous est proposé, et peut être lancé par le Verification Run Manager.
La vérification complète peut être lancée de différentes manières :

1. En pure ligne de commande. Dans le terminal Linux, en étant dans le répertoire `code`, exécuter :

   ```
   vrun directed
   ```

2. En ouvrant la version graphique du Verification Run Manager. Dans le terminal Linux, en étant dans le répertoire `code`, exécuter :

   ```
   vrun -gui directed
   ```

3. Depuis QuestaSim. Dans la version graphique de QuestaSim, depuis le répertoire `code`, exécuter :

   ```
   vrun directed
   ```

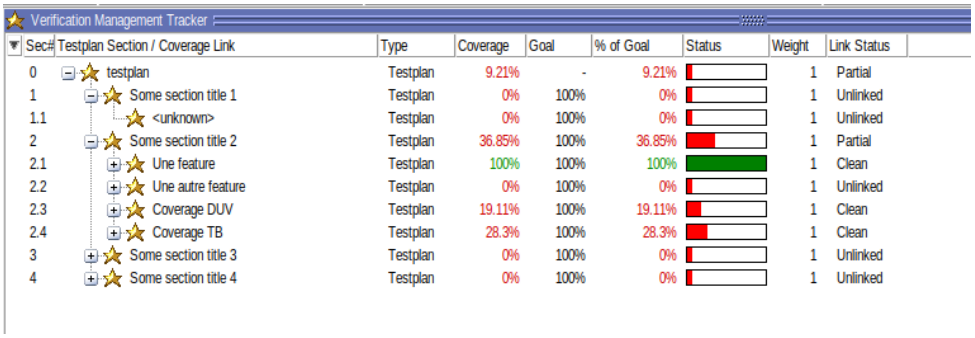
La base de données générée par la vérification est indiquée dans le fichier `default.rmdb`, et en l'occurence sera sauvegardée dans le fichier `VRMDATA/merge.ucdb`.
Pour visualiser ensuite l'état du plan de vérification, dans QuestaSim graphique (ou dans un terminal Linux), exécuter :

```
vsim -viewcov VRMDATA/merge.ucdb
```

Pour cet exercice vous pouvez lancer les deux commandes à la suite dans QuestaSim. Placez-vous dans le répertoire `code` fourni, et exécutez les commandes suivantes :

```
vrun directed
vsim -viewcov VRMDATA/merge.ucdb
```

Affichez ensuite le plan de test et sa couverture grâce au menu *View* → *Verification Management* → *Tracker*. Vous devriez observer ceci :



Observez la correspondance entre le champ *Link* du plan de test, et le nom des *runnable* dans le fichier `default.rmdb`.
Après avoir modifié le plan de test, vous pouvez relancer les deux commandes listées, et vous devriez pouvoir observer les résultats dans le tracker. Evidemment, les tests ne seront pas implémentés, et donc vous aurez une couverture plutôt faible. C'est normal, et pour cet exercice nous en resterons là.

A rendre

Cet exercice peut être réalisé par groupes de deux personnes au maximum. Rendez sur Cyberlearn une archive contenant le fichier `uart_testplan.xml` ainsi qu'un fichier décrivant les modifications que vous voudriez apporter aux spécifications. Nous discuterons en classe des différentes propositions.

Specifications

Les spécifications ont été décomposées en spécifications fonctionnelles et de conception.
Que pensez-vous de ces spécifications ? Sont-elles pertinentes ? Pourraient-elles être améliorées ?

Functional specifications

Sending

REQ001 When a word is sent to the UART from parallel interface, it should be sent in a reasonable future on the TX line, with parity and stop bits corresponding to the ones selected.

Sender FIFO

REQ002 The UART shall have a FIFO of a parameterizable size allowing to store several words, while sending them on the TX line. Every word stored in this FIFO shall eventually be sent, and should not be erased before being sent.

Receiving

REQ003 When a word is received on the RX line and respects the parity and stop bits selected, it should then be available on the parallel interface.

Design specification

Sending

REQ004 When both `sender_ready_o` and `send_i` are active, the word present on `data_in_i` shall eventually be sent on `tx_o`.

Sending Parity

REQ005 The data sent on the serial link shall end up with a parity bit if `parity_i` is active, else no parity bit is sent.

Receiving Parity

REQ006 The receiver shall check the parity bit if `parity_i` is active. In case the parity bit is not valid, it shall emit a 1-cycle '1' on `data_error_o`.

FIFO ready

REQ007 When the internal FIFO is not full, then `sender_ready_o` shall be active.

Sender bit width

REQ008 When transmitting a word on `tx_o`, each bit shall have a number of clock cycles corresponding to `clk_per_bit_i`.

Requirements list