

Rapport Communication Bluetooth

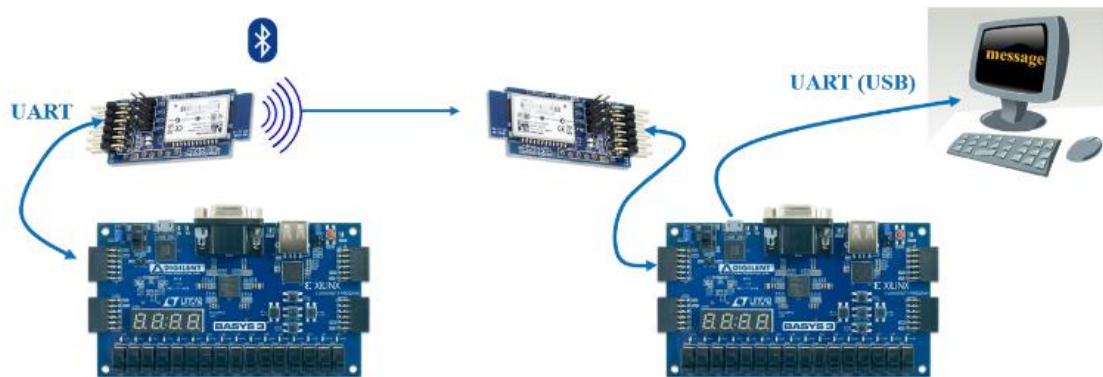
Elio KHAZAAL - Abel DIDOUH

Sommaire :

- Description du TP
- Architecture de l'émetteur
- Simulation de l'émetteur
- Implémentation de l'émetteur
- Récepteur
- Bonus

Description du TP :

L'objectif de ce travail pratique (TP) consiste à envoyer un message sous forme de caractères ASCII stockés sur un FPGA à un autre FPGA en utilisant des modules Pmod BT2 pour la communication Bluetooth. Le FPGA récepteur affichera le message sur un terminal PuTTY d'un PC via USB/UART, vérifiera l'intégrité du message en calculant le code de redondance cyclique (CRC) et renverra au FPGA émetteur, via Bluetooth, le caractère 0x06 (ASCII ACK, pour acknowledge) si le CRC est correct ou le caractère 0x15 (ASCII NAK, pour not acknowledge) si le CRC est incorrect.



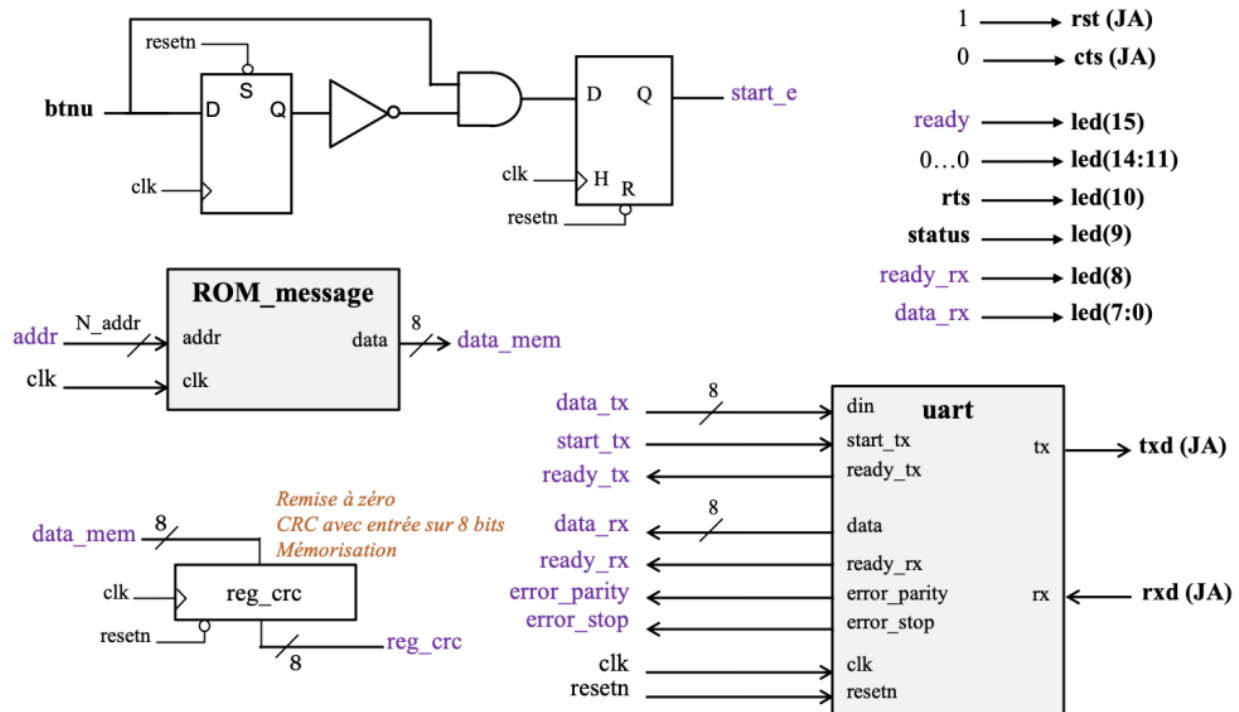
2. Architecture et VHDL

A. RTL Structure

Le but de notre structure RTL est de nous permettre de communiquer avec le PmodBT2 via le protocole suivant:



On nous a donné une structure RTL incomplète que nous pouvons voir sur l'image suivante :



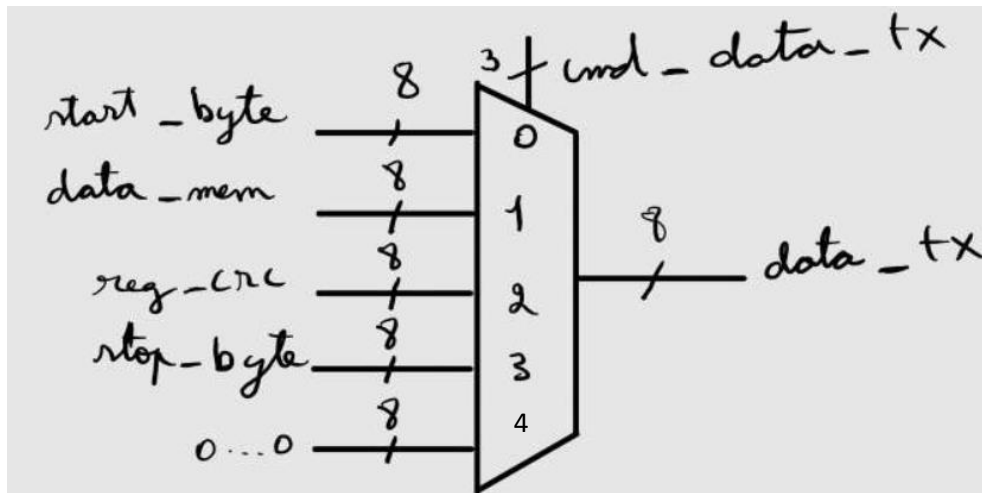
La structure RTL fournie comprend :

- Une entité "uart" responsable de la transmission et de la réception de données à l'aide du protocole de communication UART.
- Une mémoire dans laquelle notre message à transmettre est stocké.
- Un registre de 8 bits qui calcule la valeur du CRC à chaque cycle d'horloge UART.
- Un composant qui détecte le front montant de btneu lorsqu'on appuie sur le bouton btneu pour signaler le début de la transmission.

Nous devons compléter cette structure RTL afin de transmettre les données selon le protocole donné.

Multiplexeur data tx :

Comme nous l'avons vu dans le protocole de communication donné, nous devons envoyer les données dans l'ordre suivant sur data_tx : start_byte, data_bytes, crc_byte et aussi stop_byte. Pour ce faire, nous aurons besoin d'un multiplexeur pour passer à la valeur que nous devons insérer sur data_tx en fonction de la valeur de cmd_data_tx qui sera contrôlée par la machine à états. Si la valeur de cmd_data_tx est différente de 1, 2, 3 ou 4, data_tx sera mise à 0.

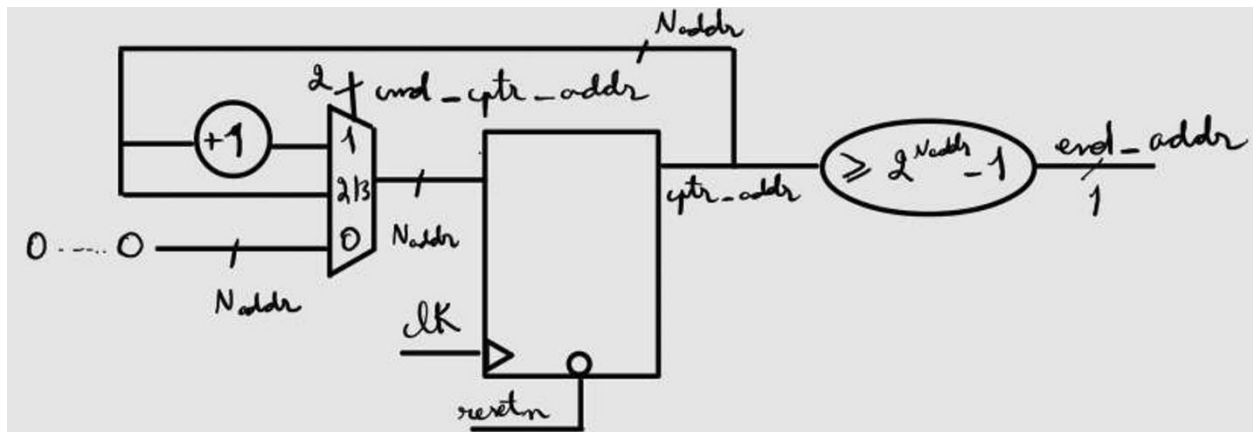


Compteur d'Adresse :

Nous avons besoin d'un compteur pour augmenter la valeur de l'adresse de 0 à $2^{N_{addr}} - 1$ afin de pouvoir lire toutes les valeurs stockées dans la mémoire ROM.

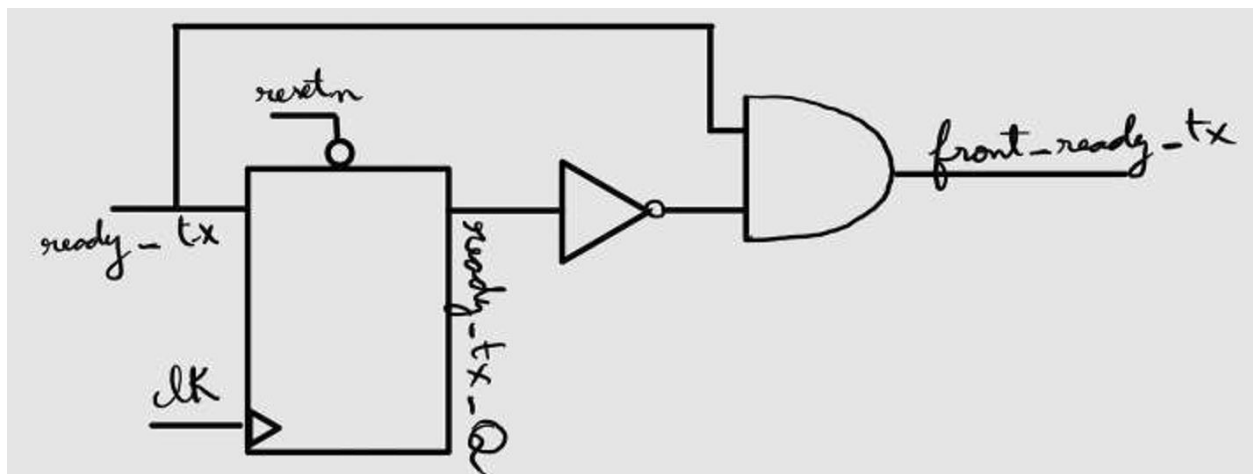
Ce compteur aura trois fonctions principales en fonction de la valeur de "cmd_cpnr_addr" :

- Incrémenter l'adresse de 0 à $2^{N_{addr}} - 1$
- Mémoriser la valeur de cptr_addr puisque l'horloge système "clk" est beaucoup plus rapide que l'horloge UART, et nous voulons que notre compteur s'incrémente au rythme de l'horloge UART.
- Remise de la valeur "cptr_addr" à 0 lorsqu'elle dépasse la valeur $2^{N_{addr}} - 1$.



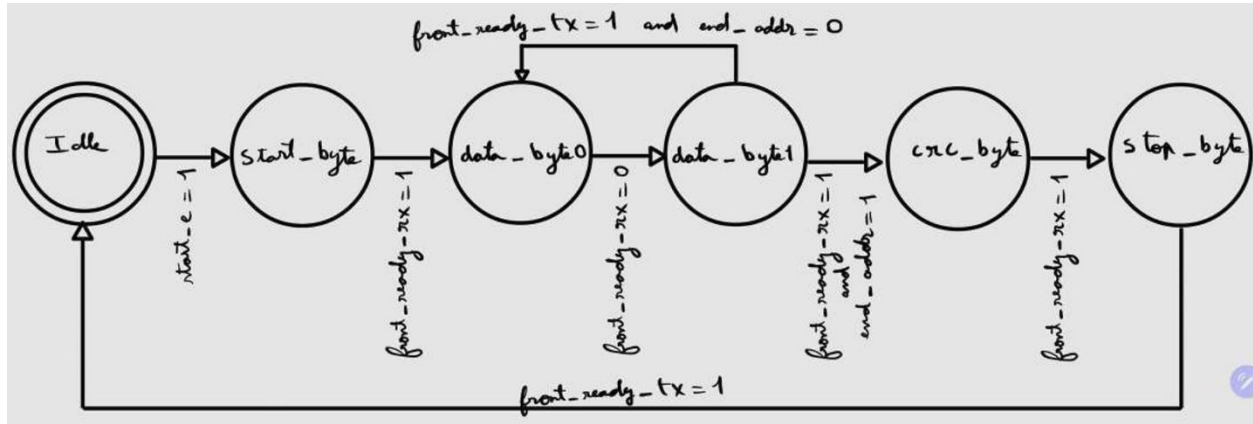
Détecteur de ready_tx rising edge :

Le signal "ready_tx" est activé lorsque qu'un octet est transmis et que le système est prêt pour une nouvelle transmission. Ainsi, nous aurons besoin d'un composant qui détectera le front montant de "ready_tx" afin de savoir quand envoyer le prochain octet sur "data_tx" dans notre protocole de communication.



B. Machine à état (FSM)

La figure suivante montre la machine à états qui pilotera notre architecture RTL :



Dans les parties suivantes, nous expliquerons les commandes que nous contrôlerons avec notre machine à états ainsi que leurs fonctions. Nous expliquerons également le fonctionnement de la machine à états à chaque étape.

❖ Table de Commandes

La table suivante mettra en évidence les commandes qui contrôlent notre structure RTL ainsi que les composants RTL qu'elles contrôlent, les fonctions qu'elles réalisent et les valeurs des commandes.

| Command Name | RTL Component | Fonctions | Valeur |
|---------------|---------------------|-------------------|--------|
| cmd_crc | CRC register | Set to "0" | "00" |
| | | Calculate CRC | "01" |
| | | Memorize | others |
| cmd_data_tx | Data Tx Multiplexer | Select start_byte | "000" |
| | | Select data_mem | "001" |
| | | Select reg_crc | "010" |
| | | Select stop_byte | "011" |
| | | Set to "0" | others |
| cmd_cptr_addr | Address Counter | Set to "0" | "00" |
| | | Increment | "01" |
| | | Memorize | others |

❖ Séquence de la machine à états

À l'état initial, le système est en mode "idle", "ready" est mis à 1, indiquant que le système est prêt pour une nouvelle transmission, "data_tx" est mise à 0 et le système attend que l'utilisateur appuie sur le bouton "btneu" pour passer à l'état suivant, "start_byte".

À l'état "start_byte", "data_tx" est réglé sur l'octet de départ 0x02. Lorsque nous détectons le front montant de "ready_tx", nous remettons "start_tx" à 1 afin d'envoyer le prochain octet de données et nous passons à l'état suivant, "data_byte0".

À l'état "data_byte0", nous réglons le multiplexeur "data_tx" sur "data_mem" afin d'envoyer le contenu de la mémoire ROM. Lorsque "front_ready_tx" est 0, nous incrémentons le compteur d'adresses et calculons la valeur du CRC avant de passer à l'état suivant, "data_byte1".

À l'état "data_byte1", si nous avons un front montant de "ready_tx" et si "end_addr" est différent de 1, nous retournons à l'état précédent, "data_byte0", afin de calculer la nouvelle valeur du CRC pour les nouvelles données entrantes. Si nous avons un front montant de "ready_tx" et si "end_addr" est égal à 1, nous passons à l'état suivant, "crc_byte".

À l'état "crc_byte", nous réglons le multiplexeur "data_tx" sur "reg_crc" afin de transmettre la valeur du CRC calculé, puis nous passons à l'état final, "stop_byte".

À l'état "stop_byte", nous réglons le multiplexeur "data_tx" sur l'octet d'arrêt 0x04 afin de transmettre l'octet d'arrêt, puis nous retournons à l'état initial, "idle", en attendant une nouvelle pression de "btneu".

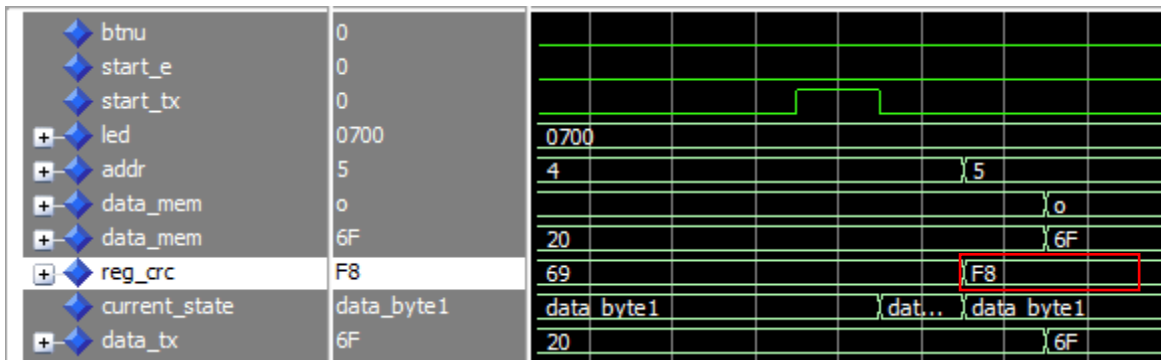
| Etat | Action |
|------------|---|
| idle | ready set to 1 cmd_data_tx set to 111 |
| | If start_e = 1 next_state <= start_byte; start_tx set to 1 cmd_data_tx set to 000 |
| start_byte | cmd_data_tx set to 000 |
| | if front_ready_tx = 1 next_state <= data_byte0 start_tx set to 1 cmd_data_tx set to 001 |

| State | Action |
|-------------------|--|
| data_byte0 | cmd_data_tx set to 001 |
| | if front_ready_tx = 0 next_state <= data_byte1 cmd_cpnr_addr set to 01 cmd_crc set to 01 cmd_data_tx set to 001 |
| data_byte1 | cmd_data_tx set to 001 |
| | if front_ready_tx = 1 and end_addr = 0 next_state <= data_byte0 cmd_cpnr_addr set to 11 cmd_crc set to 11 start_tx set to 1 cmd_data_tx set to 001 |
| | if front_ready_tx = 1 and end_addr = 1 next_state <= crc_byte cmd_crc set to 01 start_tx set to 1 cmd_data_tx set to 001 |
| | else cmd_cpnr_addr set to 11 cmd_crc set to 11 cmd_data_tx set to 001 |
| crc_byte | if front_ready_tx = 1 next_state <= stop_byte start_tx set to 1 cmd_data_tx set to 010 |
| | else cmd_crc set to 11 cmd_data_tx set to 010 |
| stop_byte | if front_ready_tx = 1 next_state <= idle start_tx set to 1 cmd_data_tx set to 011 |
| | else cmd_data_tx set to 011 |

3. Simulation de l'émission :

Nous allons prouver le bon fonctionnement de notre code par une simulation. Nous utilisons le testbench de la simulation fourni (scripts TCL).

La valeur du CRC après avoir traité "Simu " est la même que celle relevée lors du TP CRC qui est : 0xF8.

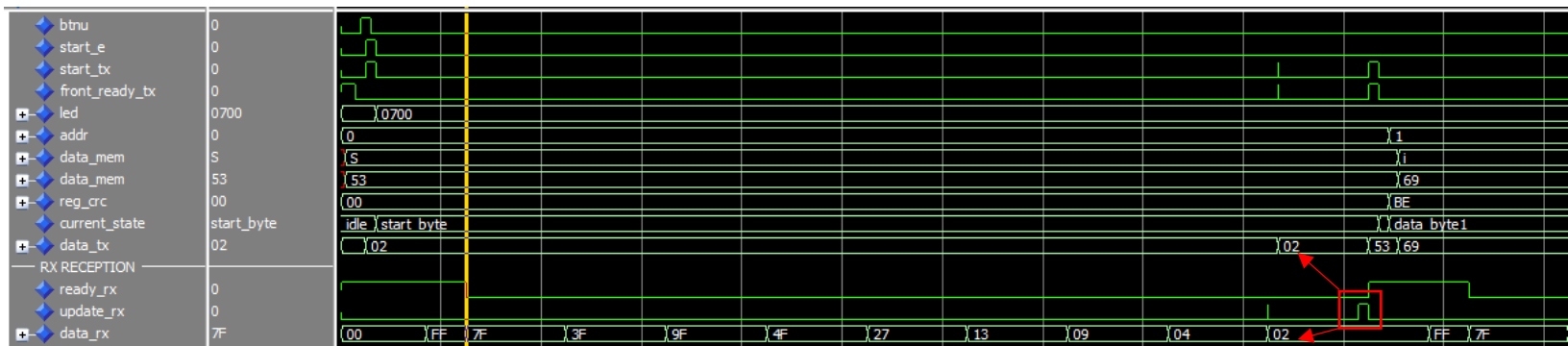


Nous regardons le transcript de Questasim :

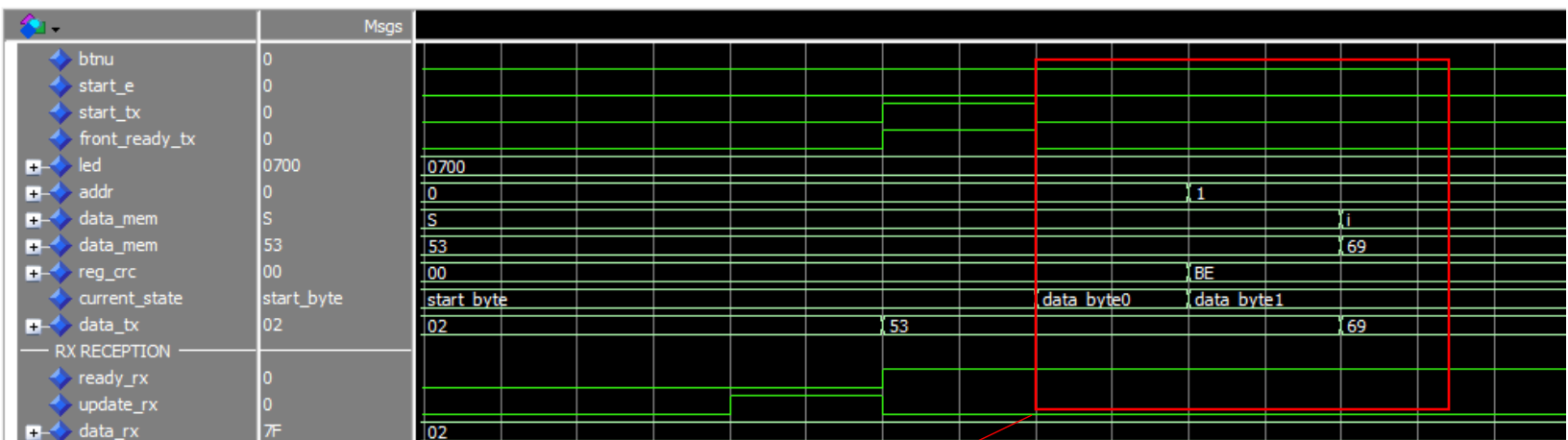
```
# ** Note: data_rx : 0x02 soit en ASCII
#   Time: 1025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x53 soit en ASCII S
#   Time: 2025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x69 soit en ASCII i
#   Time: 3025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x6D soit en ASCII m
#   Time: 4025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x75 soit en ASCII u
#   Time: 5025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x20 soit en ASCII
#   Time: 6025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x6F soit en ASCII o
#   Time: 7025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x6B soit en ASCII k
#   Time: 8025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x21 soit en ASCII !
#   Time: 9025 ns   Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x3A soit en ASCII :
#   Time: 10025 ns  Iteration: 2   Instance: /tb_emission/retour
# ** Note: data_rx : 0x04 soit en ASCII
#   Time: 11025 ns  Iteration: 2   Instance: /tb_emission/retour
# ** Note: ACK
```

Après analyse du transcript, nous envoyons le message "Simu ok!" avec au début 0x02 et à la fin 0x04. Le CRC est envoyé après la donnée utile. Nous envoyons 0x3A qui est la valeur finale du CRC.

Nous allons analyser le chronogramme afin de corroborer nos résultats.



Au début, le composant est dans l'état *idle*. Le bouton `btnu` est actionné ainsi, le signal `start_e` est à l'état haut et le signal `start_tx` s'active. Le composant passe dans l'état *start_byte*. Le signal `data_tx` reçoit 0x02. Nous regardons du côté de la réception rx. Lorsque le signal `update_rx` est actif, le signal `data_rx` est égale à 0x02.

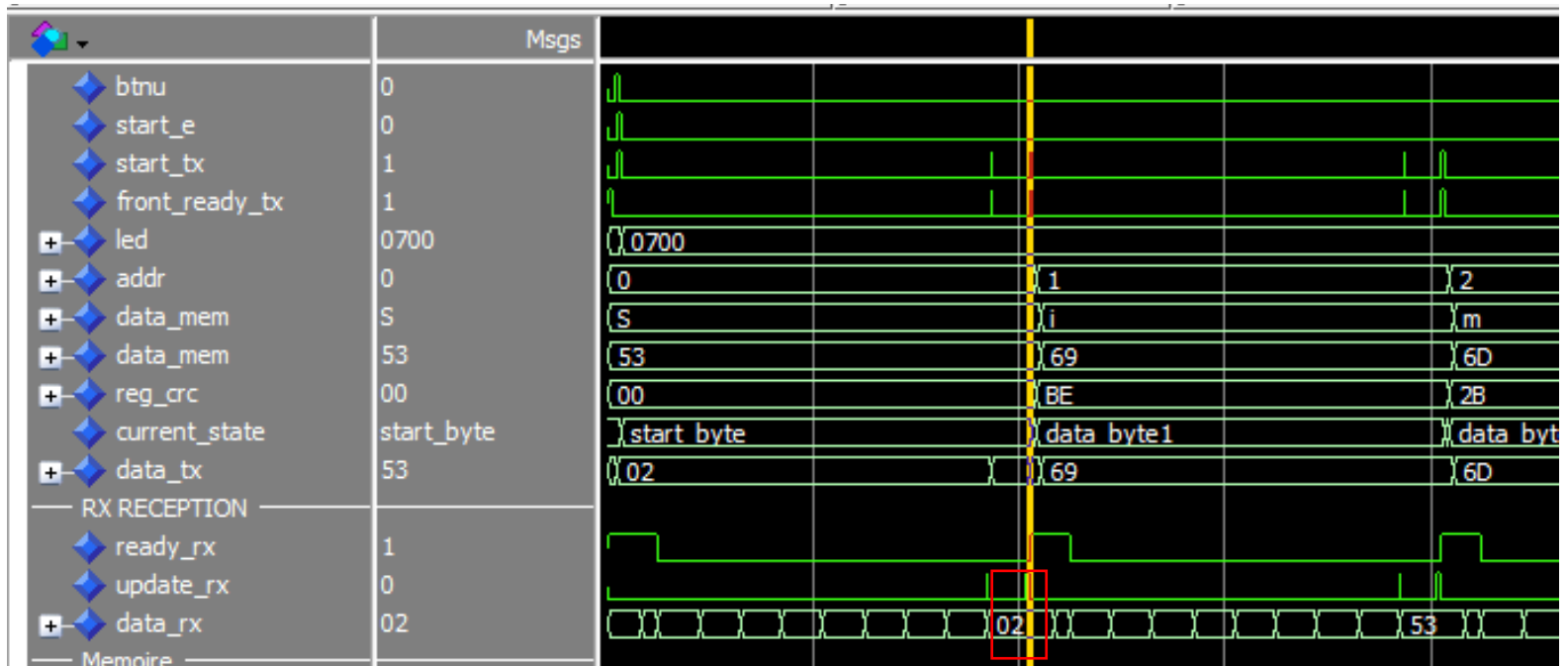


Le composant sort de l'état *start_byte* lorsque le signal `front_ready_tx` s'active. Le nouvel état est *data_byte0*. Dans cet état, l'adresse est incrémentée et le CRC est calculé. Puis le composant change d'état pour être dans l'état *data_byte1*. Dans cet état, l'adresse est mise à jour et le résultat du CRC est mémorisé.

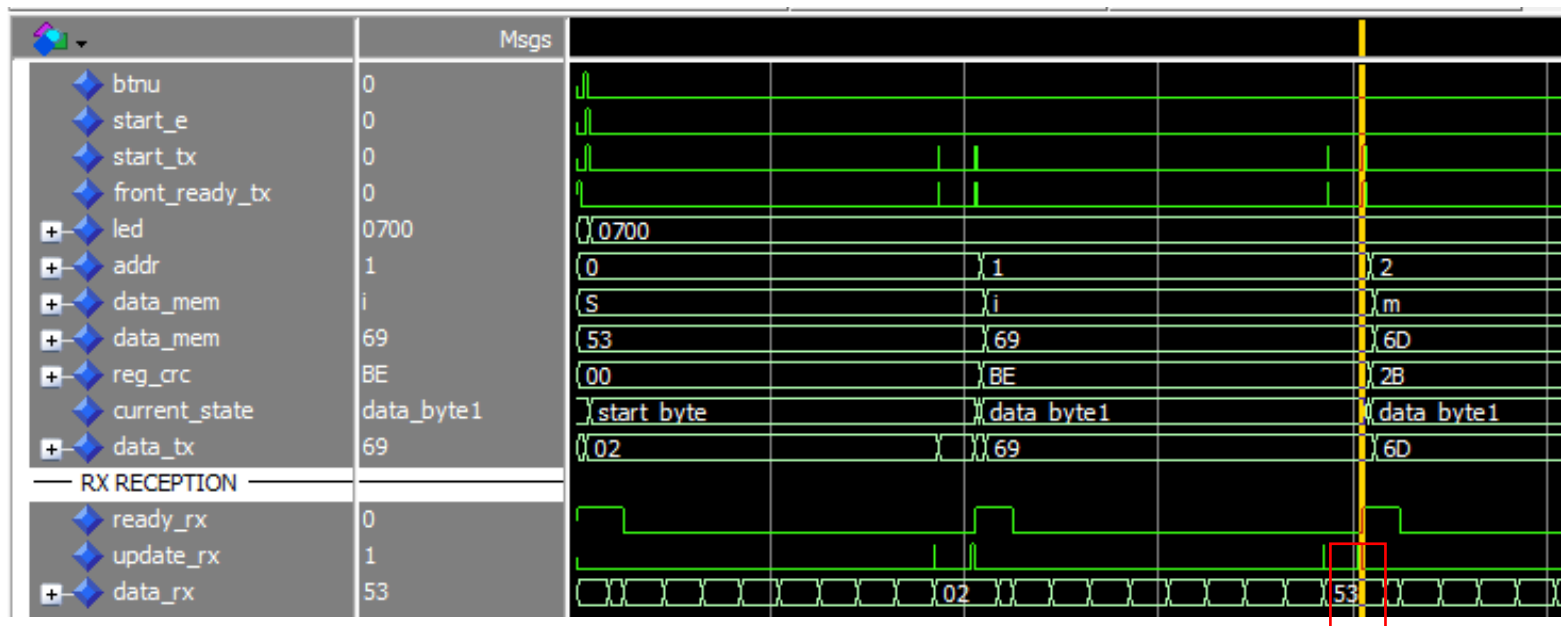
Le composant réitère pour chaque adresse.

Regardons sur le chronogramme lorsque le signal `update_rx` est actif quelle donnée obtenons nous sur `data_rx` :

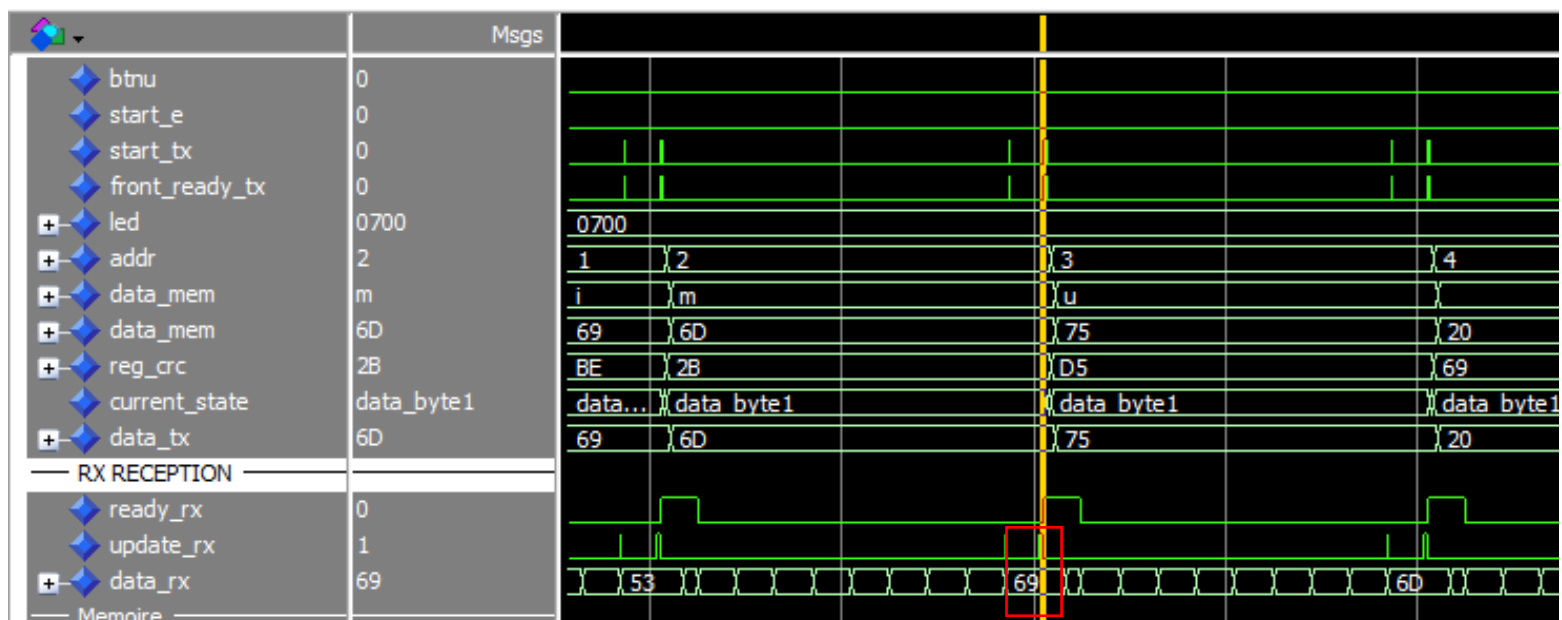
Pour le premier update_rx nous obtenons sur data_rx : 0x02.



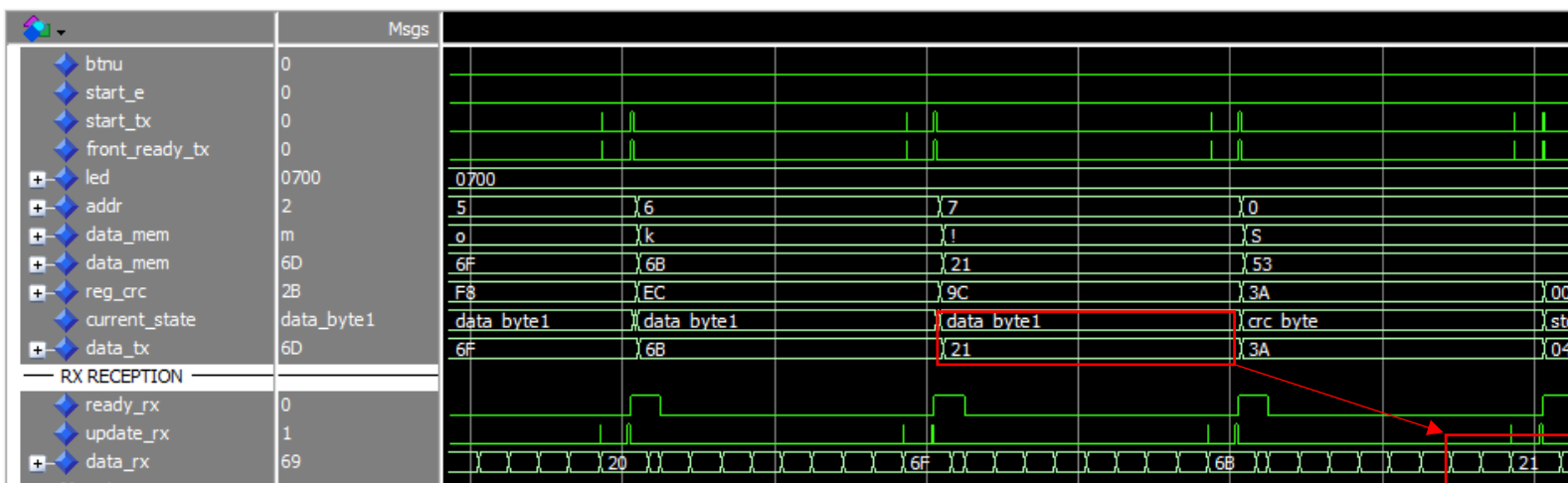
Pour le second update_rx, nous obtenons data_rx : 0x53.



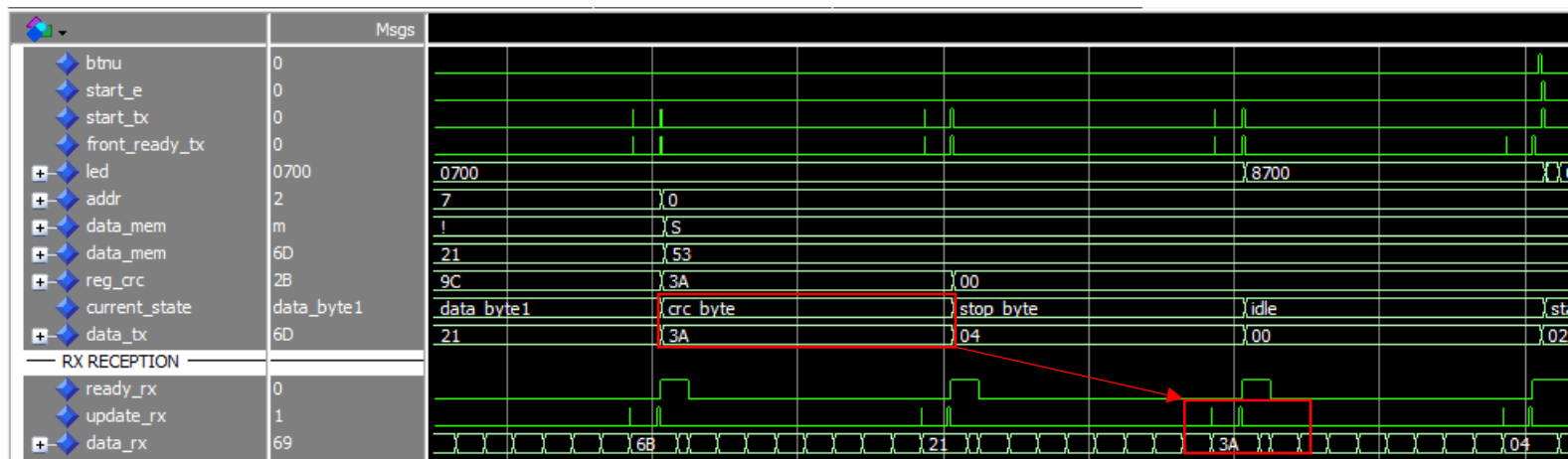
Pour le troisième update_rx, nous obtenons data_rx : 0x69.



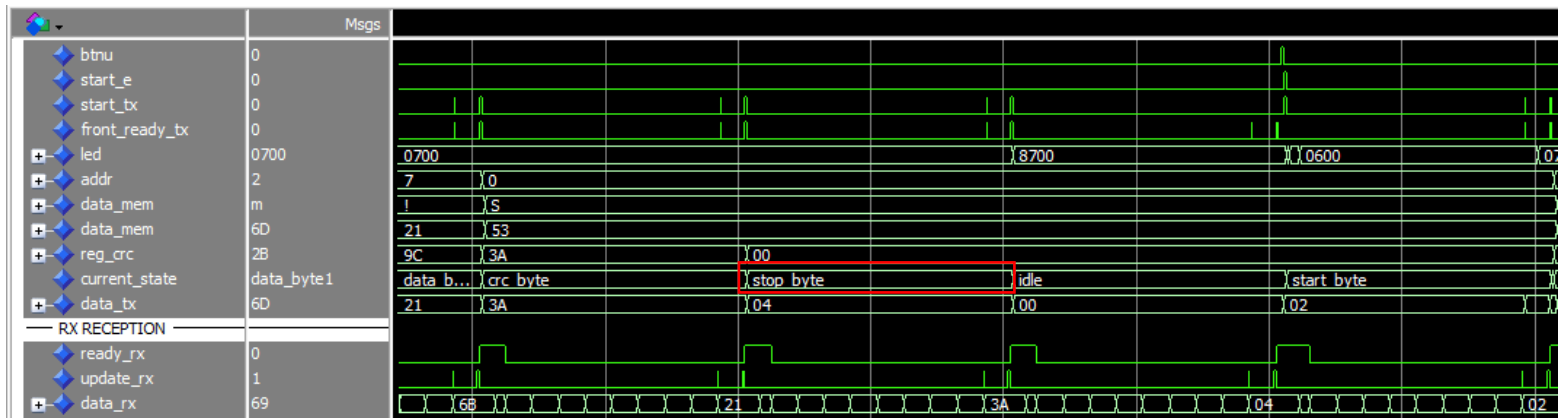
Puis, nous avons analysé le last_data_byte.



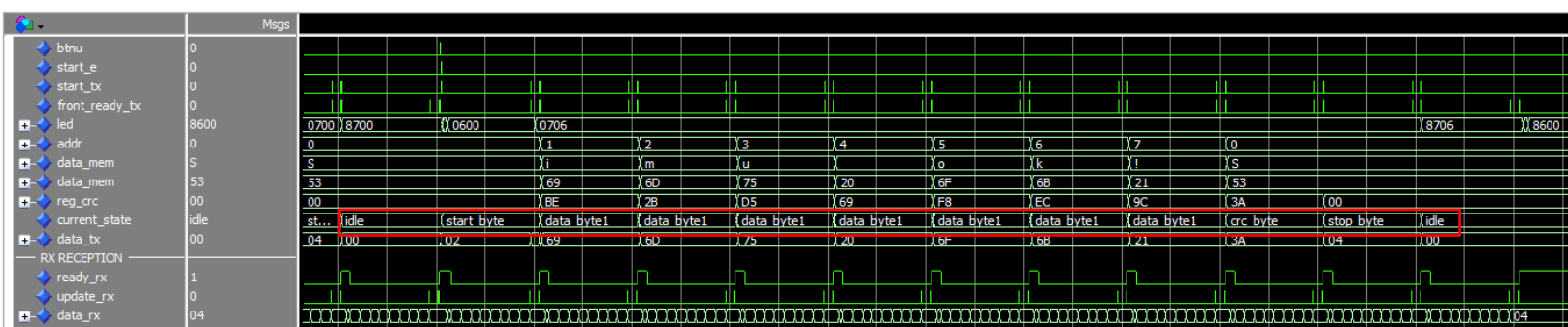
Finalement, nous arrivons au crc_byte :



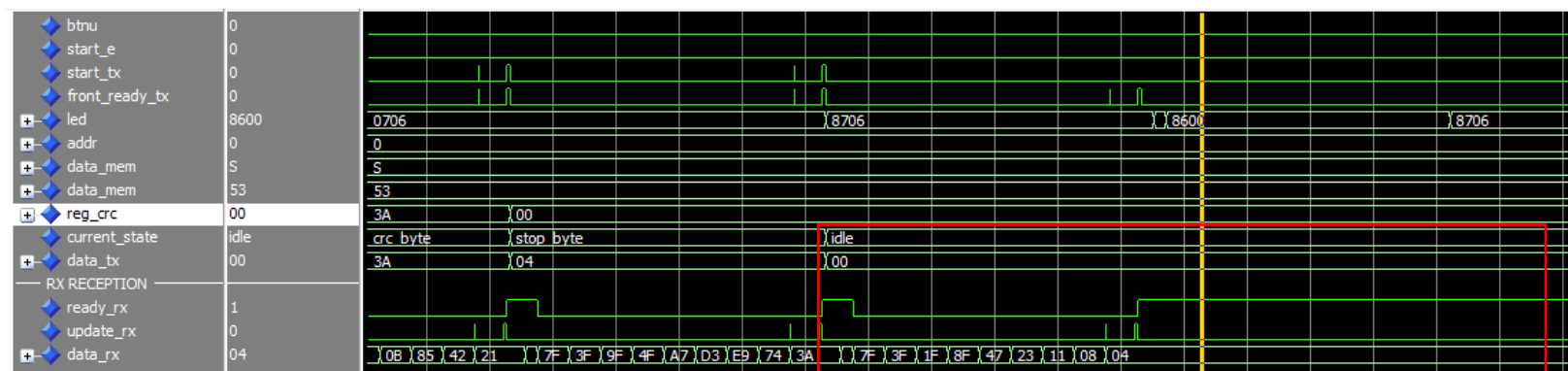
Notre trame se termine par stop_byte :



Le testbench effectue un nouvel envoi :



Nous recevons le stop_byte : 0x04. Le composant revient dans l'état idle data_tx reçoit 0x00 mais il n'y a pas d'envoi.



La trame composée des éléments suivant sont envoyées :

- start_byte : 0x02
- data_byte : donnée utile : adresse numéro 0
- ...
- data_byte : donnée utile : adresse numéro $2^{**}N_addr - 1$
- crc_byte : CRC du payload
- stop_byte : 0x04

Nous pouvons affirmer que notre émetteur effectue un envoi sans erreur.

4. Implémentation

2) Le rapport de synthèse pour l'émetteur :

Report Check Netlist:

| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | | |
|---------------------------------------|-------------------|--|--------|--|----------|--|--------|--|-------------------|--|
| | Item | | Errors | | Warnings | | Status | | Description | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | | |
| 1 | multi_driven_nets | | 0 | | 0 | | Passed | | Multi driven nets | |
| +-----+-----+-----+-----+-----+-----+ | | | | | | | | | | |

Vivado informe qu'il n'y a aucun court-circuit

| ----- | | | | |
|------------|--|--------------|--|-------------------|
| State | | New Encoding | | Previous Encoding |
| ----- | | | | |
| idle | | 000 | | 000 |
| start_byte | | 001 | | 001 |
| data_byte0 | | 010 | | 010 |
| data_byte1 | | 011 | | 011 |
| crc_byte | | 100 | | 100 |
| stop_byte | | 101 | | 101 |
| ----- | | | | |

Vivado a bien reconnu nos états de notre FSM.

```
-----
Start RTL Hierarchical Component Statistics
-----
Hierarchical RTL Component report
Module emission
Detailed RTL Component Info :
+---Adders :
      2 Input      8 Bit      Adders := 1
+---XORs :
      2 Input      1 Bit      XORs := 24
+---Registers :
           8 Bit      Registers := 2
           1 Bit      Registers := 3
+---Muxes :
      2 Input      8 Bit      Muxes := 6
      4 Input      8 Bit      Muxes := 1
      2 Input      3 Bit      Muxes := 4
      6 Input      3 Bit      Muxes := 2
      2 Input      2 Bit      Muxes := 3
      3 Input      2 Bit      Muxes := 1
      6 Input      2 Bit      Muxes := 2
```

```

        2 Input    1 Bit      Muxes := 7
        6 Input    1 Bit      Muxes := 3
Module rom_message
Detailed RTL Component Info :
+---Registers :
                8 Bit      Registers := 1
+---ROMs :
                ROMs := 1
Module uart_tx
Detailed RTL Component Info :
+---Adders :
        2 Input    10 Bit      Adders := 1
+---XORs :
        2 Input     1 Bit      XORs := 1
+---Registers :
                10 Bit      Registers := 1
                9 Bit      Registers := 1
                1 Bit      Registers := 2
+---Muxes :
        2 Input    10 Bit      Muxes := 1
        2 Input     9 Bit      Muxes := 2
        2 Input     2 Bit      Muxes := 4
        4 Input     2 Bit      Muxes := 3
        2 Input     1 Bit      Muxes := 5
        4 Input     1 Bit      Muxes := 4
Module uart_rx
Detailed RTL Component Info :
+---Adders :
        2 Input    10 Bit      Adders := 1
+---XORs :
        2 Input     1 Bit      XORs := 1
+---Registers :
                10 Bit      Registers := 1
                8 Bit      Registers := 1
                2 Bit      Registers := 1
                1 Bit      Registers := 3
+---Muxes :
        2 Input    10 Bit      Muxes := 2
        2 Input     8 Bit      Muxes := 2
        2 Input     2 Bit      Muxes := 2
        3 Input     2 Bit      Muxes := 1
        4 Input     2 Bit      Muxes := 4
        4 Input     1 Bit      Muxes := 5
        2 Input     1 Bit      Muxes := 8
        3 Input     1 Bit      Muxes := 2

```

```

-----
Finished RTL Hierarchical Component Statistics
-----
-----

```

Vivado a bien reconnu l'architecture emission, la ROM et également l'IP UART qui se compose d'un module UART_TX et UART_RX.

Nous nous concentrons sur la partie emission :

```
-----
Start RTL Hierarchical Component Statistics
-----
Hierarchical RTL Component report
Module emission
Detailed RTL Component Info :
+---Adders :
      2 Input      8 Bit      Adders := 1
+---XORs :
      2 Input      1 Bit      XORs := 24
+---Registers :
           8 Bit      Registers := 2
           1 Bit      Registers := 3
+---Muxes :
      2 Input      8 Bit      Muxes := 6
      4 Input      8 Bit      Muxes := 1
      2 Input      3 Bit      Muxes := 4
      6 Input      3 Bit      Muxes := 2
      2 Input      2 Bit      Muxes := 3
      3 Input      2 Bit      Muxes := 1
      6 Input      2 Bit      Muxes := 2
      2 Input      1 Bit      Muxes := 7
      6 Input      1 Bit      Muxes := 3
```

Vivado a bien reconnu deux registres de 8 bits.

Vivado a bien reconnu trois registres de 1 bits.

```
ROM: Preliminary Mapping Report
+-----+-----+-----+-----+
|Module Name | RTL Object   | Depth x Width | Implemented As |
+-----+-----+-----+-----+
|emission    | mem/data_reg | 128x8          | Block RAM      |
+-----+-----+-----+-----+
```

Vivado a bien reconnu la ROM dans le module emission.

```
Report Cell Usage:
+-----+-----+-----+
|      | Cell      | Count |
+-----+-----+-----+
| 1     | BUFG      | 1     |
| 2     | CARRY4    | 2     |
| 3     | LUT1      | 2     |
| 4     | LUT2      | 16    |
| 5     | LUT3      | 12    |
| 6     | LUT4      | 35    |
| 7     | LUT5      | 20    |
| 8     | LUT6      | 30    |
| 9     | RAMB18E1  | 1     |
| 10    | FDCE      | 62    |
| 11    | FDPE      | 4     |
```

| | | |
|---------------------|------|----|
| 12 | IBUF | 6 |
| 13 | OBUF | 19 |
| +-----+-----+-----+ | | |

Il n'y a aucun latch.

Warnings :

WARNING: [Synth 8-3917] design emission has port led[14] driven by constant 0

WARNING: [Synth 8-3917] design emission has port led[13] driven by constant 0

WARNING: [Synth 8-3917] design emission has port led[12] driven by constant 0

WARNING: [Synth 8-3917] design emission has port led[11] driven by constant 0

Ces warnings ne sont pas graves. En effet, nous imposons un zéro à chaque entrée pour la led[14], la led[13], la led[12] et la led[11].

0...0 → led(14:11)

WARNING: [Synth 8-3917] design emission has port cts driven by constant 0

WARNING: [Synth 8-3917] design emission has port rst driven by constant 1

Ces warnings ne sont pas graves. En effet, nous imposons un zéro pour le port cts et un pour le port rst.

1 → rst (JA)
0 → cts (JA)

Le rapport d'implémentation pour l'émetteur :

1. Slice Logic

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|-----------------------|------|-------|------------|-----------|-------|
| Slice LUTs | 82 | 0 | 0 | 20800 | 0.39 |
| LUT as Logic | 82 | 0 | 0 | 20800 | 0.39 |
| LUT as Memory | 0 | 0 | 0 | 9600 | 0.00 |
| Slice Registers | 67 | 0 | 0 | 41600 | 0.16 |
| Register as Flip Flop | 67 | 0 | 0 | 41600 | 0.16 |
| Register as Latch | 0 | 0 | 0 | 41600 | 0.00 |
| F7 Muxes | 0 | 0 | 0 | 16300 | 0.00 |
| F8 Muxes | 0 | 0 | 0 | 8150 | 0.00 |

1.1 Summary of Registers by Type

| Total | Clock Enable | Synchronous | Asynchronous |
|-------|--------------|-------------|--------------|
| 0 | - | - | - |
| 0 | - | - | Set |
| 0 | - | - | Reset |
| 0 | - | Set | - |
| 0 | - | Reset | - |
| 0 | Yes | - | - |
| 4 | Yes | - | Set |
| 63 | Yes | - | Reset |
| 0 | Yes | Set | - |
| 0 | Yes | Reset | - |

2. Slice Logic Distribution

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|--|-----------|----------|------------|-------------|-------------|
| Slice | 27 | 0 | 0 | 8150 | 0.33 |
| SLICEL | 16 | 0 | | | |
| SLICEM | 11 | 0 | | | |
| LUT as Logic | 82 | 0 | 0 | 20800 | 0.39 |
| using 05 output only | 0 | | | | |
| using 06 output only | 49 | | | | |
| using 05 and 06 | 33 | | | | |
| LUT as Memory | 0 | 0 | 0 | 9600 | 0.00 |
| LUT as Distributed RAM | 0 | 0 | | | |
| LUT as Shift Register | 0 | 0 | | | |
| Slice Registers | 67 | 0 | 0 | 41600 | 0.16 |
| Register driven from within the Slice | 63 | | | | |
| Register driven from outside the Slice | 4 | | | | |
| LUT in front of the register is unused | 1 | | | | |
| LUT in front of the register is used | 3 | | | | |
| Unique Control Sets | 6 | | 0 | 8150 | 0.07 |

* * Note: Available Control Sets calculated as Slice * 1, Review the Control Sets Report for more information regarding control sets.

Relevé des ressources :

- 27 slices
- 82 LUTS
- 67 DFF

3. Memory

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|----------------|----------|----------|------------|------------|-------------|
| Block RAM Tile | 0.5 | 0 | 0 | 50 | 1.00 |
| RAMB36/FIFO* | 0 | 0 | 0 | 50 | 0.00 |
| RAMB18 | 1 | 0 | 0 | 100 | 1.00 |
| RAMB18E1 only | 1 | | | | |

* Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO18E1. However, if a FIFO18E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB18E1

4. DSP

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|-------------|----------|----------|------------|-----------|-------------|
| DSPs | 0 | 0 | 0 | 90 | 0.00 |

5. IO and GT Specific

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|-----------------------------|------|-------|------------|-----------|-------|
| Bonded IOB | 25 | 25 | 0 | 106 | 23.58 |
| IOB Master Pads | 10 | | | | |
| IOB Slave Pads | 14 | | | | |
| Bonded IPADs | 0 | 0 | 0 | 10 | 0.00 |
| Bonded OPADs | 0 | 0 | 0 | 4 | 0.00 |
| PHY_CONTROL | 0 | 0 | 0 | 5 | 0.00 |
| PHASER_REF | 0 | 0 | 0 | 5 | 0.00 |
| OUT_FIFO | 0 | 0 | 0 | 20 | 0.00 |
| IN_FIFO | 0 | 0 | 0 | 20 | 0.00 |
| IDELAYCTRL | 0 | 0 | 0 | 5 | 0.00 |
| IBUFDS | 0 | 0 | 0 | 104 | 0.00 |
| GTPE2_CHANNEL | 0 | 0 | 0 | 2 | 0.00 |
| PHASER_OUT/PHASER_OUT_PHY | 0 | 0 | 0 | 20 | 0.00 |
| PHASER_IN/PHASER_IN_PHY | 0 | 0 | 0 | 20 | 0.00 |
| IDELAYE2/IDELAYE2_FINEDELAY | 0 | 0 | 0 | 250 | 0.00 |
| IBUFDS_GTE2 | 0 | 0 | 0 | 2 | 0.00 |
| ILOGIC | 0 | 0 | 0 | 106 | 0.00 |
| OLOGIC | 0 | 0 | 0 | 106 | 0.00 |

6. Clocking

| Site Type | Used | Fixed | Prohibited | Available | Util% |
|------------|------|-------|------------|-----------|-------|
| BUFGCTRL | 1 | 0 | 0 | 32 | 3.13 |
| BUFIO | 0 | 0 | 0 | 20 | 0.00 |
| MMCME2_ADV | 0 | 0 | 0 | 5 | 0.00 |
| PLLE2_ADV | 0 | 0 | 0 | 5 | 0.00 |
| BUFMRCE | 0 | 0 | 0 | 10 | 0.00 |
| BUFHCE | 0 | 0 | 0 | 72 | 0.00 |
| BUFR | 0 | 0 | 0 | 20 | 0.00 |

Relevé des ressources :

- 1 block RAM
- 0 DSP
- 1 arbre d'horloge (BUFGCTRL)
- 0 PLL

Analyse du rapport de timings :

```
-----  
| Design Timing Summary  
| -----  
-----
```

| WNS(ns) | TNS(ns) | TNS Failing Endpoints | TNS Total Endpoints |
|---------|---------|-----------------------|---------------------|
| ----- | ----- | ----- | ----- |
| 4.810 | 0.000 | 0 | 106 |

Set-up

| WHS(ns) | THS(ns) | THS Failing Endpoints | THS Total Endpoints |
|---------|---------|-----------------------|---------------------|
| ----- | ----- | ----- | ----- |
| 0.104 | 0.000 | 0 | 106 |

Hold

| WPWS(ns) | TPWS(ns) | TPWS Failing Endpoints | TPWS Total Endpoints |
|----------|----------|------------------------|----------------------|
| ----- | ----- | ----- | ----- |
| 4.500 | 0.000 | 0 | 69 |

Pulse Width

All user specified timing constraints are met.

```
-----  
--  
| Clock Summary  
| -----  
-----  
--
```

| Clock | Waveform(ns) | Period(ns) | Frequency(MHz) |
|---------|---------------|------------|----------------|
| ----- | ----- | ----- | ----- |
| clk_100 | {0.000 5.000} | 10.000 | 100.000 |

Contraintes sur les horloges (issues du xdc) :

On retrouve ce qu'on a spécifié dans le .xdc.

Les contraintes sont respectées.

Relevé :

- Slack = WNS = 4.810 ns
- WNS ≥ 0 (4.810 ns) et WHS ≥ 0 (0.104 ns)

Calcul fréquence maximale :

$f_{\text{max}} = 1/(\text{period} - \text{WNS}) = 1/((10-4.810)*10^{-9}) = 192\,678\,227.4\text{ Hz}$

STA : chemin critique

Point de départ du chemin critique : **l/trans/tempo_reg**

Arrivée du chemin critique : **FSM_sequential_current_state_reg**

Max Delay Paths

```
Slack (MET) : 4.810ns (required time - arrival time)
Source: l/trans/tempo_reg[3]/C
(rising edge-triggered cell FDCE clocked by clk_100 {rise@0.000ns fall@5.000ns period=10.000ns})
Destination: FSM_sequential_current_state_reg[0]/D
(rising edge-triggered cell FDCE clocked by clk_100 {rise@0.000ns fall@5.000ns period=10.000ns})
Path Group: clk_100
Path Type: Setup (Max at Slow Process Corner)
Requirement: 10.000ns (clk_100 rise@10.000ns - clk_100 rise@0.000ns)
Data Path Delay: 4.864ns (logic 1.627ns (33.452%) route 3.237ns (66.548%))
Logic Levels: 4 (LUT4=1 LUT5=2 LUT6=1)
Clock Path Skew: -0.039ns (DCD - SCD + CPR)
Destination Clock Delay (DCD): 4.791ns = ( 14.791 - 10.000 )
Source Clock Delay (SCD): 5.090ns
Clock Pessimism Removal (CPR): 0.260ns
Clock Uncertainty: 0.035ns ((TSJ^2 + TIJ^2)^1/2 + DJ) / 2 + PE
Total System Jitter (TSJ): 0.071ns
Total Input Jitter (TIJ): 0.000ns
Discrete Jitter (DJ): 0.000ns
Phase Error (PE): 0.000ns
```

| Location | Delay type | Incr(ns) | Path(ns) | Netlist Resource(s) |
|---------------|---------------------------|----------|----------|---|
| W5 | (clock clk_100 rise edge) | 0.000 | 0.000 | r |
| | net (fo=0) | 0.000 | 0.000 | r clk (IN) |
| W5 | IBUF (Prop_ibuf_I_0) | 1.458 | 1.458 | r clk_IBUF_inst/0 |
| | net (fo=1, routed) | 1.967 | 3.425 | r clk_IBUF |
| BUFGCTRL_X0Y0 | BUFG (Prop_bufg_I_0) | 0.096 | 3.521 | r clk_IBUF_BUFG_inst/0 |
| SLICE_X10Y44 | net (fo=68, routed) | 1.569 | 5.090 | r l/trans/CLK |
| | FDCE | | | r l/trans/tempo_reg[3]/C |
| SLICE_X10Y44 | FDCE (Prop_fdce_C_Q) | 0.478 | 5.568 | r l/trans/tempo_reg[3]/Q |
| | net (fo=4, routed) | 0.748 | 6.317 | r l/trans/tempo_reg[3] |
| SLICE_X10Y44 | LUT5 (Prop_lut5_I3_0) | 0.321 | 6.638 | r l/trans/FSM_sequential_current_state[1]_i_5/0 |
| | net (fo=2, routed) | 0.602 | 7.239 | r l/trans/FSM_sequential_current_state[1]_i_5_n_0 |
| SLICE_X11Y44 | LUT6 (Prop_lut6_I3_0) | 0.355 | 7.594 | f l/trans/FSM_sequential_current_state[1]_i_3/0 |
| | net (fo=7, routed) | 0.564 | 8.159 | r l/trans/FSM_sequential_current_state[1]_i_3_n_0 |
| SLICE_X9Y44 | LUT4 (Prop_lut4_I1_0) | 0.118 | 8.277 | r l/trans/FSM_sequential_current_state[2]_i_3/0 |
| | net (fo=8, routed) | 0.706 | 8.982 | r l/trans/ready_tx_Q_reg |
| SLICE_X8Y44 | LUT5 (Prop_lut5_I2_0) | 0.355 | 9.337 | r l/trans/FSM_sequential_current_state[0]_i_1_1/0 |
| | net (fo=1, routed) | 0.616 | 9.954 | r l_n_1 |
| SLICE_X8Y44 | FDCE | | | r FSM_sequential_current_state_reg[0]/D |
| | | | | |
| W5 | (clock clk_100 rise edge) | 10.000 | 10.000 | r |
| | net (fo=0) | 0.000 | 10.000 | r clk (IN) |
| W5 | IBUF (Prop_ibuf_I_0) | 1.388 | 11.388 | r clk_IBUF_inst/0 |
| | net (fo=1, routed) | 1.862 | 13.250 | r clk_IBUF |
| BUFGCTRL_X0Y0 | BUFG (Prop_bufg_I_0) | 0.091 | 13.341 | r clk_IBUF_BUFG_inst/0 |
| | net (fo=68, routed) | 1.450 | 14.791 | r clk_IBUF_BUFG |
| SLICE_X8Y44 | FDCE | | | r FSM_sequential_current_state_reg[0]/C |
| | clock pessimism | 0.260 | 15.051 | |
| | clock uncertainty | -0.035 | 15.016 | |
| SLICE_X8Y44 | FDCE (Setup_fdce_C_D) | -0.252 | 14.764 | FSM_sequential_current_state_reg[0] |
| | | | | |
| | required time | | 14.764 | |
| | arrival time | | -9.954 | |
| | | | | |
| | slack | | 4.810 | |

δA

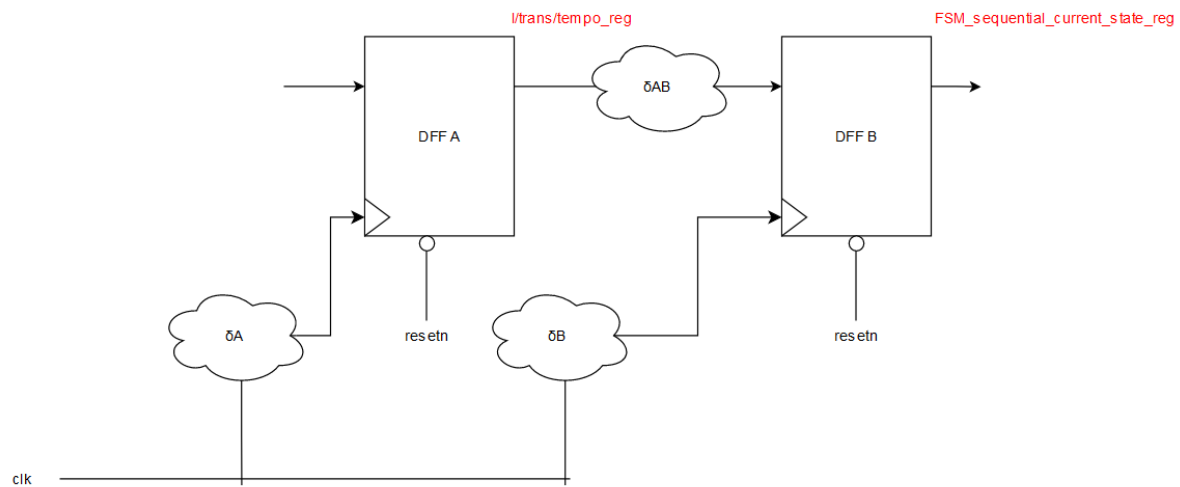
δAB

δB

Le chemin critique se trouve dans le niveau hiérarchique l entre le registre tempo FSM et le registre d'états de la FSM.

Ici il traverse le LUT 5, le LUT 6, le LUT 4 puis le LUT 5.

Nous représentons le chemin critique :



l représente le composant uart.

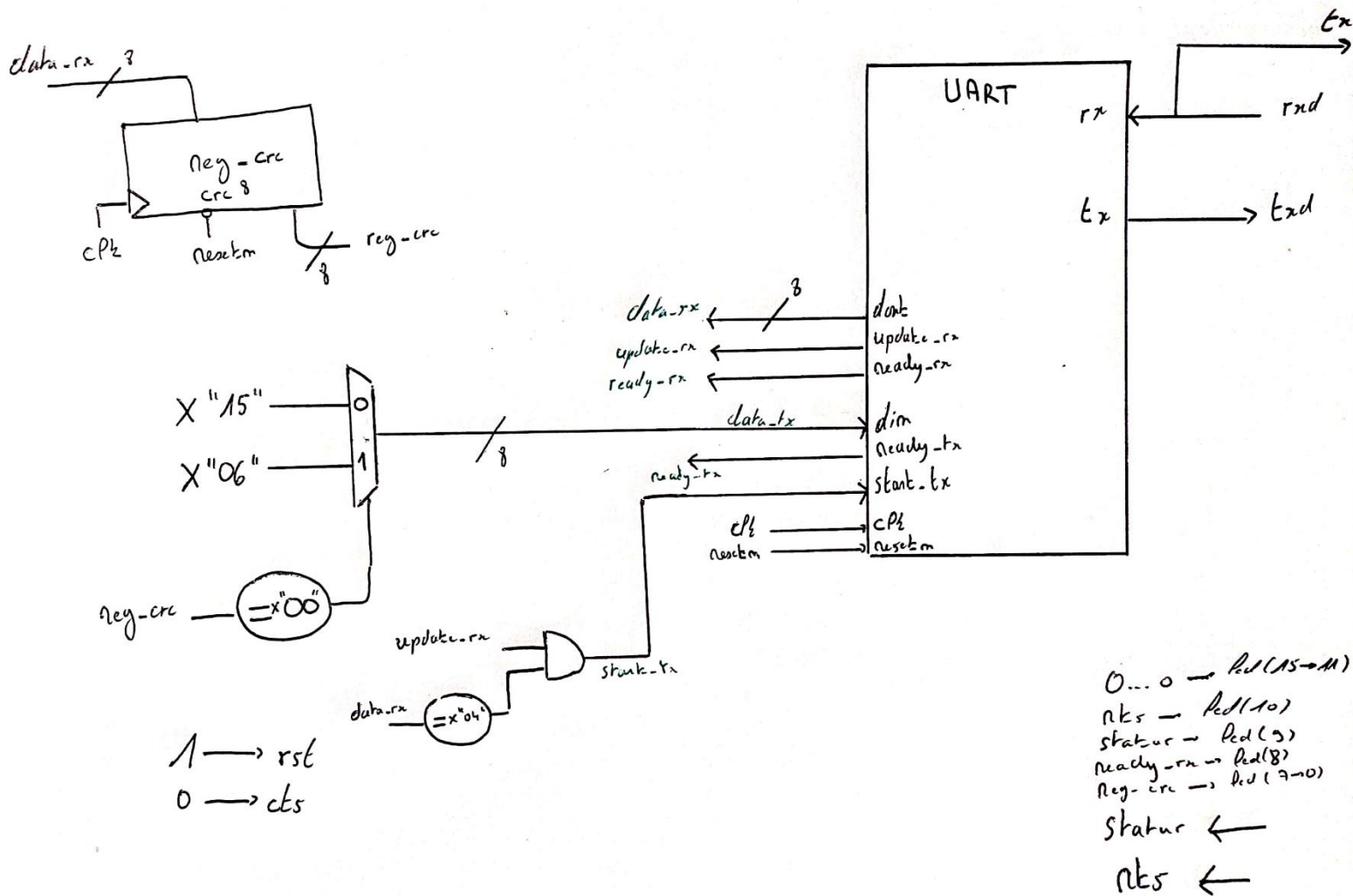
```
Bravo! La transmission est operationnelle. Verifiez CRC OK : 0x06 sur led, si 0x15, erreur de CRC.
```

Dans la suite de cette partie, nous allons tenter de ne pas afficher le CRC.

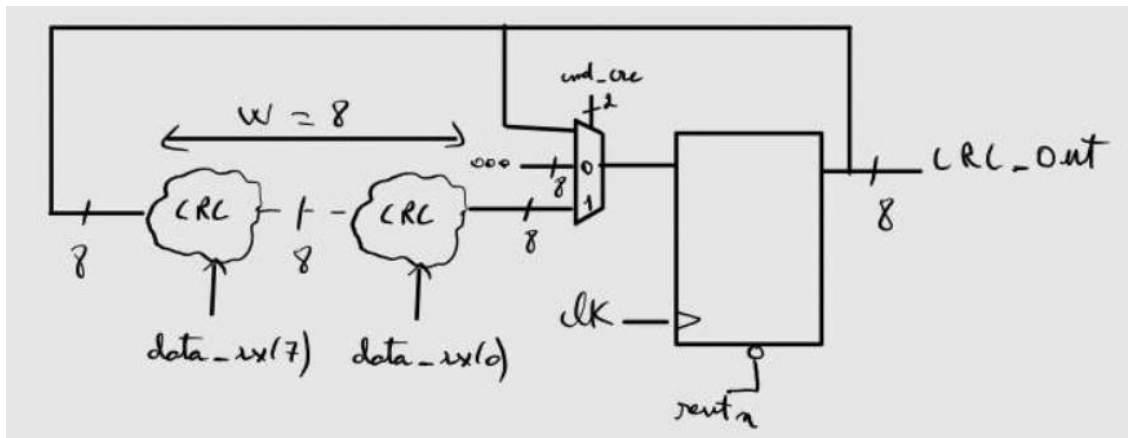
CRC :-

5. Récepteur

- 1) À partir du code VHDL de la partie réception, nous pouvons dessiner les composants RTL suivants :



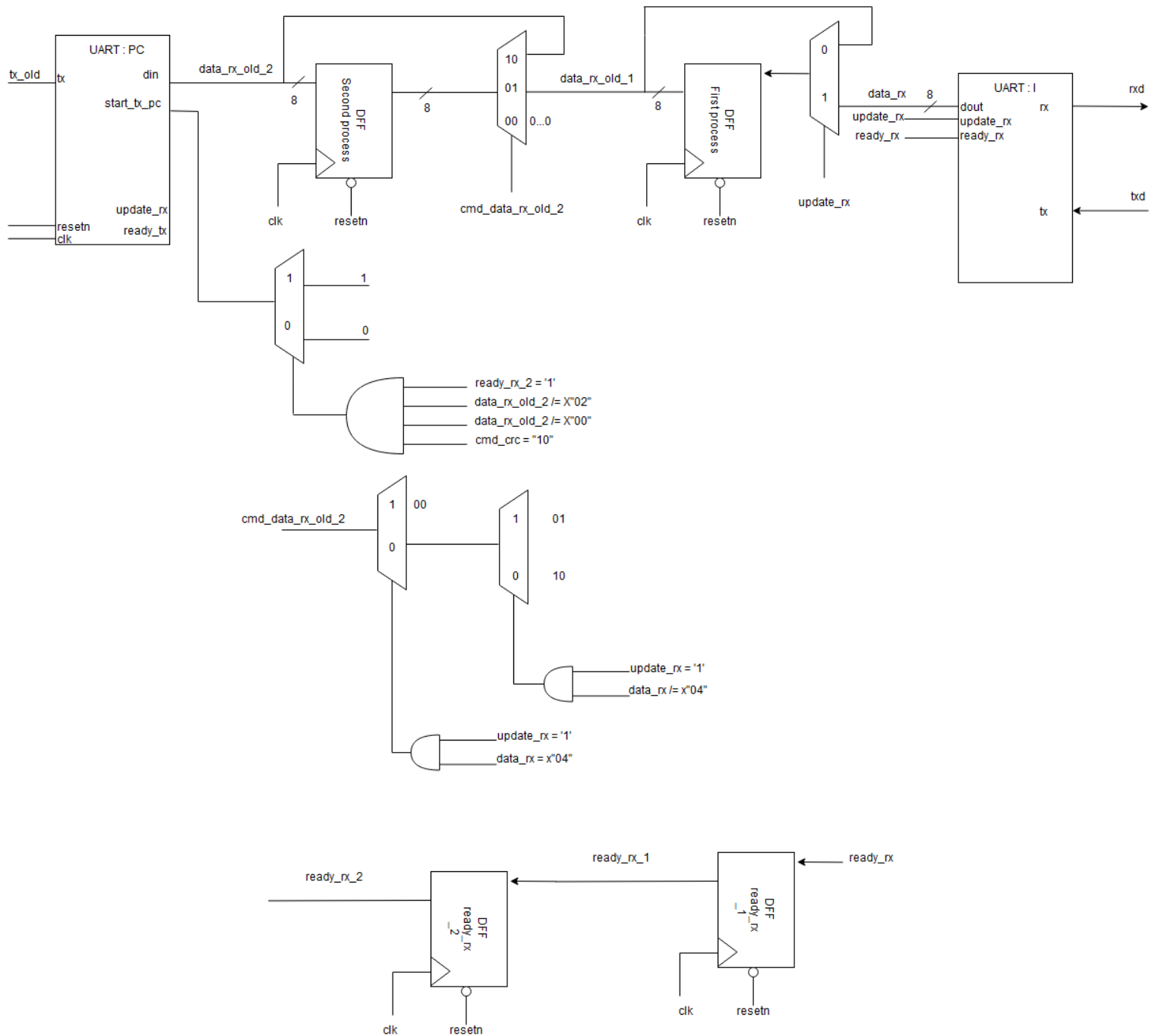
La figure ci-dessus montre un dessin de l'entité UART. Nous avons besoin de cette entité pour recevoir les données UART du PmodBT2 en utilisant rxd, puis pour les désérialiser en données parallèles de 8 bits data_rx afin de les envoyer au FPGA.



La figure ci-dessus montre un dessin du composant RTL CRC. Les données reçues "data_rx" sont transmises par des paquets de 8 bits qui doivent être traités simultanément avec un front de l'horloge UART.

Les nuages intitulés "CRC" représentent les fonctions combinatoires avec un "ou exclusif" basé sur la valeur du polynôme $g(x)$. Pour traiter 8 bits en une période d'horloge, il est nécessaire de mettre en cascade cette fonction 8 fois.

2) Voici l'architecture de réception que nous proposons pour filtrer l'affichage du CRC sans introduire de compteur :



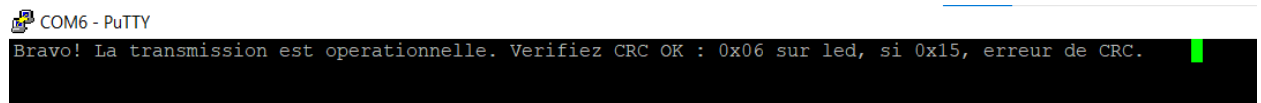
À droite, nous avons l'entité UART qui reçoit les données du PmodBT2. Il reçoit les données sur rxd dans l'ordre suivant : start byte, data bytes, CRC byte et ensuite stop byte.

Afin de séparer le CRC du reste du message, nous avons décidé d'ajouter deux registres de 8 bits en série après la réception de data_rx. Ces deux registres nous permettent d'introduire un délai dans le message qui nous permet de détecter le CRC et de l'éliminer avant qu'il ne soit lu par l'ordinateur.

Lorsque le message est reçu, le dernier octet sur data_rx sera l'octet d'arrêt comme indiqué en rouge dans la figure ci-dessus. Étant donné que nous avons ajouté les deux registres, nous aurons le CRC tel qu'indiqué en rouge sur la sortie du premier registre 'data_rx_old_1' et le dernier octet avant le CRC sera sur la sortie du deuxième registre 'data_rx_old_2'. Dans ce cas, nous insérons une valeur de zéro sur le dernier registre au lieu d'insérer la valeur du CRC en commandant 'cmd_data_rx_old_2'.

Ensuite, nous ajoutons une autre entité UART à gauche, qui sera utilisée pour sérialiser 'data_rx_old_2' afin que nous puissions en lire la valeur sur l'ordinateur via txd. Nous aurons également besoin d'un signal 'start_tx_pc' pour nous permettre d'envoyer chaque octet.

Cette architecture semble fonctionner, nous pouvons voir dans l'image suivante que lorsque nous appuyons sur le bouton sur le PmodBT2 émetteur, nous recevons le signal sur le PmodBT2 récepteur et nous pouvons lire les résultats sur le terminal PuTTY sur le PC.



Cependant, parfois, pour des raisons que nous n'avons pas pu découvrir, nous affichons une lettre plusieurs fois comme on peut le voir sur l'image.

