|  |
| --- |
| **Description Firmware RAS** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Prepared by** | *Signature* |  | **Accepted by** | *Signature* |
|  | |  |  | |
| **Noémie Rolland** | |  |  | |
|  | |  |  | |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Concerned Models** | **BB** |  | **DM** |  | **EM** |  | **STM** |  | **QM** |  | **FM** |  | **FS** |  | **All** |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Document under Configuration Control** |  |  |  | **Approval request** |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Approved by** | ***Function*** | ***Date*** | ***Signature*** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |
| --- | --- |
| **Summary** |  |
| **Annexes** |  |

|  |  |
| --- | --- |
| **Keywords** |  |

|  |  |
| --- | --- |
| **Distribution** | ***See Distribution list at the end of this document*** |

*Template V1.0*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Document Change Record** | | | | |
| **Edition** | **Revision** | **Date** | **Modified pages** | **Observations** |
| **1** | **0** |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Applicable Documents (AD)** | | | |
| **AD** | **Title** | **Reference** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Reference Documents (RD)** | | | |
| **RD** | **Title** | **Reference** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **List of Abbreviations** | | | |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**- Table of Contents -**

Table des matières

[1. INTRODUCTION 4](#_Toc78555583)

[1.1. Scope of the document 4](#_Toc78555584)

[1.2. Div\_freq2.vhd 4](#_Toc78555585)

[1.3. Div\_freq.vhd 5](#_Toc78555586)

[1.4. Read\_5MHz.vhd 6](#_Toc78555587)

[1.5. Shift\_register\_15b.vhd 7](#_Toc78555588)

[1.6. Mux\_overlap\_neg.vhd 7](#_Toc78555589)

[1.7. Mux\_overlap\_pos.vhd 8](#_Toc78555590)

[1.8. Mux\_overlap.vhd 8](#_Toc78555591)

[1.9. Sequence\_treatment.vhd 9](#_Toc78555592)

[1.10. Row\_addressing.vhd 10](#_Toc78555593)

# INTRODUCTION

## Scope of the document

Ce document a pour objectif de faire une description détaillée des blocs VHDL du Firmware RAS. Il est valable pour la version v1.3.0 du Firmware, v0.3 de Xifu Studio sur la carte Opal Kelly XEM7310.

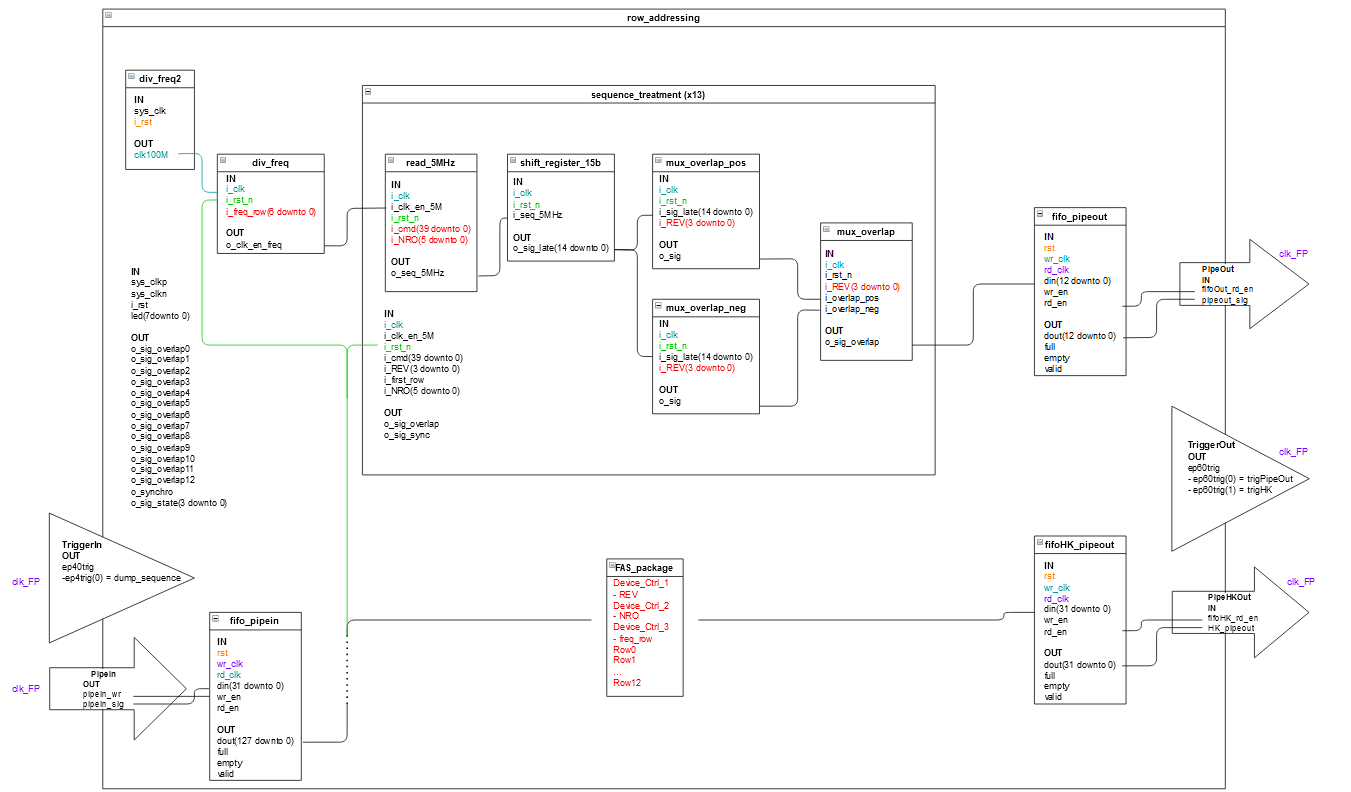


Figure 1: Schéma d'architecture FW RAS

## Div\_freq2.vhd

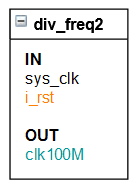


Figure 2: Bloc div\_freq2

|  |  |  |  |
| --- | --- | --- | --- |
| IN | sys\_clk | STD\_LOGIC | Signal d’horloge du FPGA 200MHz |
| IN | i\_rst | STD\_LOGIC | Signal de reset du FPGA actif HAUT |
| OUT | clk100M | STD\_LOGIC | Signal d’horloge 100MHz |

L’objectif de ce bloc est de diviser par 2 l’horloge interne du FPGA de fréquence 200 MHz pour obtenir une horloge de fréquence égale à 100 MHz.

Signal interne :

|  |  |  |
| --- | --- | --- |
| cmp | UNSIGNED(1 downto 0) | Compteur interne |

Lorsque i\_rst=’0’, le compteur cmp est incrémenté à chaque coup d’horloge. Tant que le compteur cmp est plus petit que 1 le signal d’horloge clk100M est égal à 0. Lorsque le compteur cmp est égal à 1 (le cmp est initialisé à 0 au début), le signal clk100M est égal à 1 et le compteur cmp est remis à 0.

## Div\_freq.vhd

Figure 3: Bloc div\_freq

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_freq\_row | STD\_LOGIC\_VECTOR(7 downto 0) | Paramètre fréquence row |
| OUT | o\_clk\_en\_freq | STD\_LOGIC | Clock enable à la fréquence souhaitée |

L’objectif de ce bloc est de générer un signal de clock enable à la fréquence souhaitée par l’utilisateur, cette fréquence est saisie par commande.

Signaux internes :

|  |  |  |
| --- | --- | --- |
| cmd | UNSIGNED(6 downto 0) | Compteur interne |

Lorsque i\_rst\_n est égal à 1 c’est à dire lorsque mode = 1 et que i\_rst = 0, un compteur est incrémenté à chaque coup d’horloge. Tant que le compteur est inférieur à (le compteur est initialisé à 0), le signal o\_clk\_en\_freq est égal à 0. Lorsque le compteur est égal à , le signal o\_clk\_en\_freq est égal à 1 et le compteur cm pest remis à 0.

## Read\_5MHz.vhd

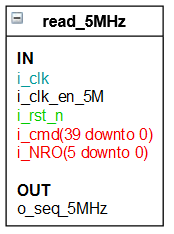


Figure 4: Bloc read\_5MHz

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_clk\_en\_5M | STD\_LOGIC | Clock enable |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_cmd | STD\_LOGIC\_VECTOR(39 downto 0) | Séquence de row |
| IN | i\_NRO | STD\_LOGIC\_VECTOR(5 downto 0) | Longueur de la sequence à prendre en compte |
| OUT | o\_seq\_5MHz | STD\_LOGIC | Séquence lue à la fréquence du clock enable |

L’objectif de ce bloc est de lire chaque bit de la séquence un à un à une fréquence freq\_row (dans notre cas 5MHz). A partir du paramètre NRO, on lira la longueur de séquence voulue.

Signaux internes :

|  |  |  |
| --- | --- | --- |
| cmd\_int | STD\_LOGIC\_VECTOR(39 downto 0) | Signal de séquence interne |
| counter | UNSIGNED(5 downto 0) | Compteur interne |

Lorsque i\_rst\_n est égal à 0 c’est à dire lorsque mode = 0 ou que i\_rst = 1, on initialise le counter à 0 et cmd\_int est rempli avec la séquence i\_cmd.

Lorsque i\_rst\_n est égal à 1 c’est à dire lorsque mode = 1 et que i\_rst = 0, à chaque fois que i\_clk\_en\_5M est égal à 1 :

* Si NRO = 0 : le signal de sortie o\_seq\_5MHz est égal à 0
* Si counter < NRO -1 ET counter < 39 : le compteur est incrémenté, on lit le bit 0 de cmd\_int que l’on stocke dans o\_seq\_5MHz et on effectue une rotation vers la droite sur cmd\_int pour décaler le bit 0 à lire.
* Si counter >= NRO OU counter >=40 : on lit le bit 0 de cmd\_int que l’on stocke dans o\_seq\_5MHz, cmd\_int est initialisé avec la séquence i\_cmd et le counter est remis à 0.

## Shift\_register\_15b.vhd

Figure 5: Bloc shift\_register\_15b

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_seq\_5MHz | STD\_LOGIC | Séquence lue à la fréquence du clock enable |
| OUT | o\_sig\_late | STD\_LOGIC\_VECTOR(14 downto 0) | Signaux retardés |

L’objectif de ce bloc est de créer du retard sur le signal de la séquence à 5MHz. On veut créer en tout 15 coups d’horloge de retard (coup d’horloge toutes les 10 ns).

Signal interne :

|  |  |  |
| --- | --- | --- |
| sig\_late\_int | STD\_LOGIC\_VECTOR(14 downto 0) | Signaux retardés internes |

Lorsque i\_rst\_n est égal à 0 c’est à dire lorsque mode = 0 ou que i\_rst = 1, on initialise le signal sig\_late\_int à 0.

Lorsque i\_rst\_n est égal à 1 c’est à dire lorsque mode = 1 et que i\_rst = 0, à chaque coup d’horloge on donne la valeur de i\_seq\_5MHz au bit 0 de sig\_late\_int et on exécute un décalage vers la gauche ce qui créer au bout de 15 coups d’horloge un retard de 150 ns sur le signal de départ.

A tout instant o\_sig\_late prend la valeur de sig\_late\_int.

## Mux\_overlap\_neg.vhd



Figure 6: Bloc mux\_overlap\_neg

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_sig\_late | STD\_LOGIC\_VECTOR(14 downto 0) | Signaux retardés |
| IN | i\_REV | STD\_LOGIC\_VECTOR(3 downto 0) | Valeur de recouvrement |
| OUT | o\_sig | STD\_LOGIC | Signal retardé sélectionné |

Ce bloc est un multiplexeur, il permet de choisir le signal retardé voulu en fonction de la valeur de REV. On établit que le signal de référence (non retardé) est i\_sig\_late(7). On vient donc choisir dans les signaux en avance puisqu’on souhaite un retard négatif. Cela correspond aux signaux i\_sig\_late(6 downto 0).

- Si REV = -1 : o\_sig prend la valeur de i\_sig\_late(6)

- Si REV = -2 : o\_sig prend la valeur de i\_sig\_late(5)

- …

- Si REV = -7 : o\_sig prend la valeur de i\_sig\_late(0)

## Mux\_overlap\_pos.vhd

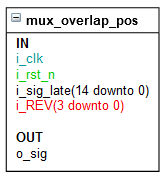


Figure 7: Bloc mux\_overlap\_pos

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_sig\_late | STD\_LOGIC\_VECTOR(14 downto 0) | Signaux retardés |
| IN | i\_REV | STD\_LOGIC\_VECTOR(3 downto 0) | Valeur de recouvrement |
| OUT | o\_sig | STD\_LOGIC | Signal retardé sélectionné |

Ce bloc est un multiplexeur, il permet de choisir le signal retardé voulu en fonction de la valeur de REV. On établit que le signal de référence (non retardé) est i\_sig\_late(7). On vient donc choisir dans les signaux en retard puisqu’on souhaite un retard positif. Cela correspond aux signaux i\_sig\_late(14 downto 8).

- Si REV = 0 : o\_sig prend la valeur de i\_sig\_late(7)

- Si REV = +1 : o\_sig prend la valeur de i\_sig\_late(8)

- Si REV = +2 : o\_sig prend la valeur de i\_sig\_late(9)

- …

- Si REV = +7 : o\_sig prend la valeur de i\_sig\_late(14)

## Mux\_overlap.vhd



Figure 8: Bloc mux\_overlap

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_REV | STD\_LOGIC\_VECTOR(3 downto 0) | Valeur de recouvrement |
| IN | i\_overlap\_pos | STD\_LOGIC | Signal retardé choisi |
| IN | i\_overlap\_neg | STD\_LOGIC | Signal en avance choisi |
| OUT | o\_sig\_overlap | STD\_LOGIC | Signal sélectionné |

Ce bloc est un multiplexeur, il permet de choisir entre le signal retardé et le signal en avance sélectionnés au niveau des multiplexeurs précédents. En fonction de la valeur de REV le signal correspondant est sélectionné :

* Si REV < 0 : o\_sig\_overlap prend la valeur de i\_sig\_overlap\_neg
* Si REV >= 0 : o\_sig\_overlap prend la valeur de i\_sig\_overlap\_pos

## Sequence\_treatment.vhd

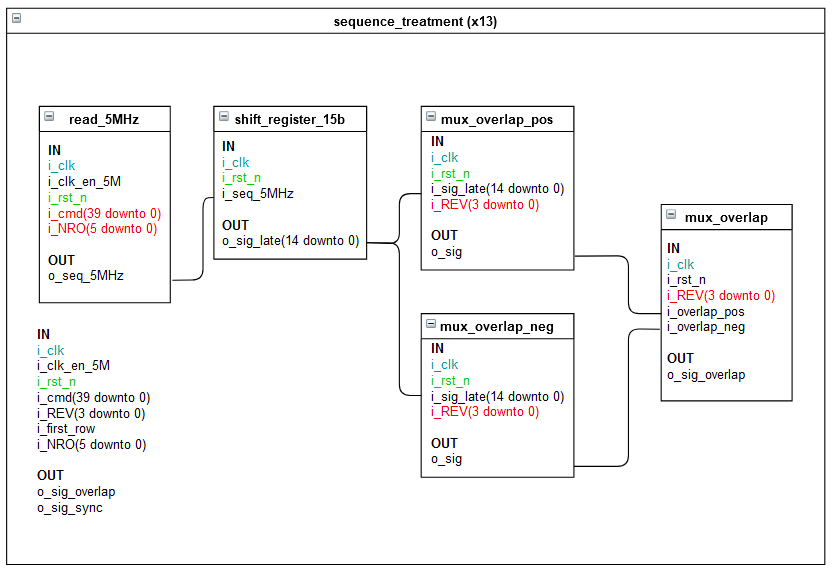


Figure 9: Bloc sequence\_treatment

|  |  |  |  |
| --- | --- | --- | --- |
| IN | i\_clk | STD\_LOGIC | Signal d’horloge 100 MHz |
| IN | i\_clk\_en\_5M | STD\_LOGIC | Clock enable |
| IN | i\_rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| IN | i\_cmd | STD\_LOGIC\_VECTOR(39 downto 0) | Séquence de row |
| IN | i\_REV | STD\_LOGIC\_VECTOR(3 downto 0) | Valeur de recouvrement |
| IN | i\_first\_row | STD\_LOGIC | i\_cmd(0) premier bit de la séquence |
| IN | i\_NRO | STD\_LOGIC\_VECTOR(5 downto 0) | Longueur de la séquence à prendre en compte |
| OUT | o\_sig\_overlap | STD\_LOGIC | Signal retardé sélectionné |
| OUT | o\_sig\_sync | STD\_LOGIC | Signal de synchro |

Ce bloc VHDL implémente les sous modules précédents et manipule des signaux internes afin de générer les signaux de sortie pour le pilotage des rows. Il a aussi pour objectif de générer un signal de synchronisation. Il y 13 blocs sequence\_treatment pour les 13 rows à piloter.

Signaux internes :

|  |  |  |
| --- | --- | --- |
| seq\_5MHz | STD\_LOGIC | Séquence interne lue à la fréquence du clock enable |
| sig\_late | STD\_LOGIC\_VECTOR(14 downto 0) | Signaux retardés |
| sig\_neg | STD\_LOGIC | Signal en avance sélectionné |
| sig\_pos | STD\_LOGIC | Signal en retard sélectionné |
| overlap\_neg | STD\_LOGIC | Signal avec recouvrement négatif |
| overlap\_pos | STD\_LOGIC | Signal avec recouvrement positif |
| sig\_t0 | STD\_LOGIC | Alias = sig\_late(8) |

Dans ce bloc on implémente les sous-modules suivants :

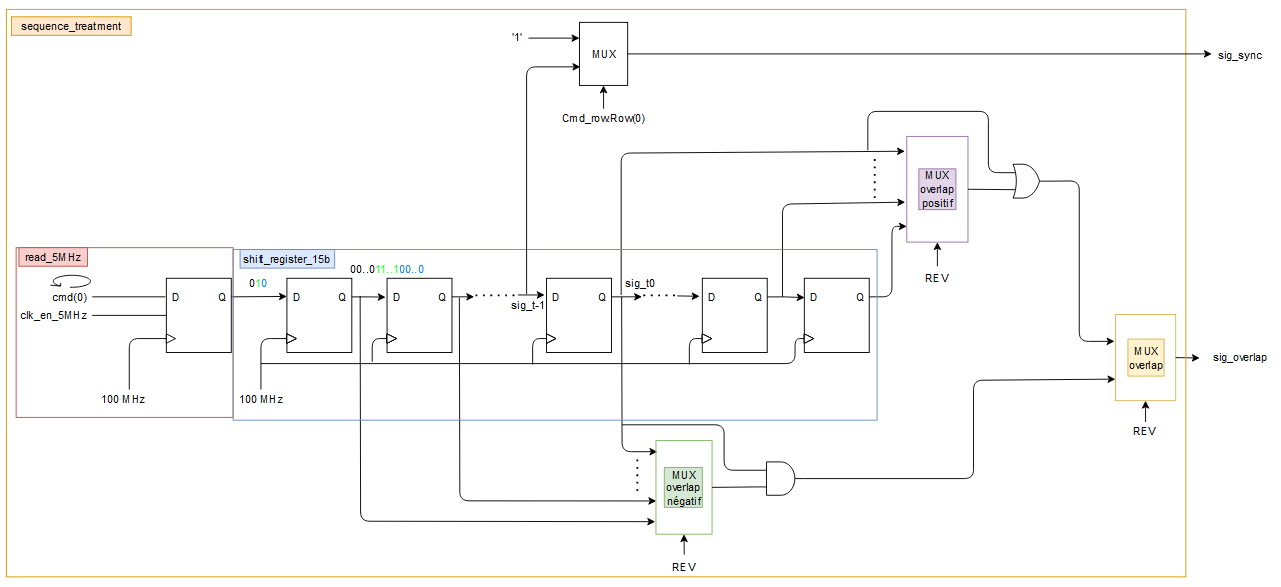
* Read\_5MHz
* Shift\_register\_15b
* Mux\_overlap\_neg
* Mux\_overlap\_pos
* Mux\_overlap

On applique également un ET logique entre sig\_t0 et sig\_neg pour donner overlap\_neg qui sera à l’entrée du bloc mux\_overlap et un OU logique entre sig\_t0 et sig\_pos pour donner overlap\_pos qui sera à l’entrée du bloc mux\_overlap également.

Le bloc présente aussi un process qui gère la génération d’un signal de synchro. Lorsque i\_rst\_n est égal à 0 c’est à dire lorsque mode = 0 ou que i\_rst = 1, on initialise le signal de synchro o\_sig\_sync est initialisé à 0.

Lorsque i\_rst\_n est égal à 1 c’est à dire lorsque mode = 1 et que i\_rst = 0, on a 2 cas de figure :

* Si i\_first\_row = 1, alors le signal o\_sig\_sync sera égal à sig\_t0
* Si i\_first\_row = 0, alors le signal o\_sig\_sync sera égal à 1



## Row\_addressing.vhd

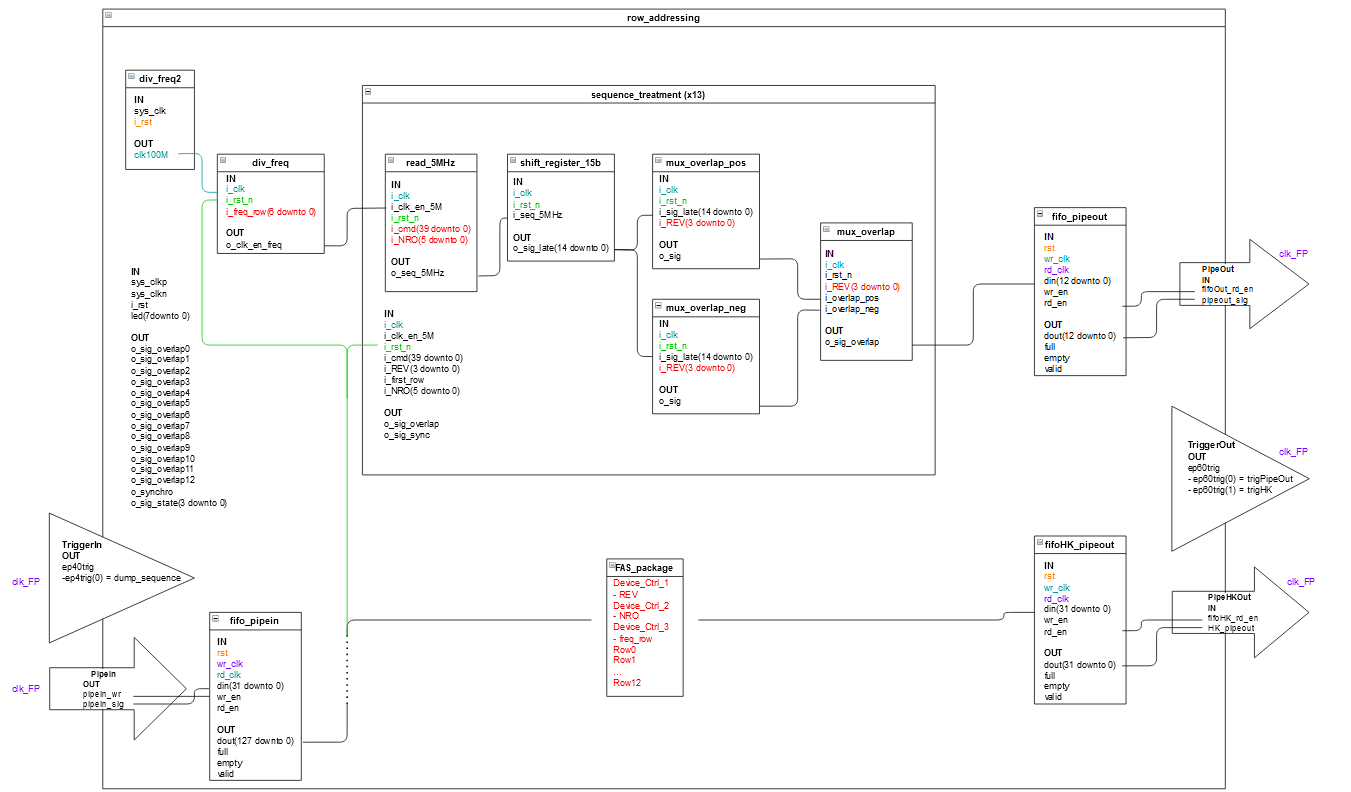


Figure 10: Bloc row\_addressing

|  |  |  |  |
| --- | --- | --- | --- |
| IN | okUH | STD\_LOGIC\_VECTOR(4 downto 0) | Signal FrontPanel |
| OUT | okHU | STD\_LOGIC\_VECTOR(2 downto 0) | Signal FrontPanel |
| INOUT | okUHU | STD\_LOGIC\_VECTOR(31 downto 0) | Signal FrontPanel |
| INOUT | okAA | STD\_LOGIC | Signal FrontPanel |
| IN | sys\_clkp | STD\_LOGIC | Signal d’horloge positif interne au FPGA |
| IN | sys\_clkn | STD\_LOGIC | Signal d’horloge négatif interne au FPGA |
| IN | i\_rst | STD\_LOGIC | Signal de reset du FPGA actif HAUT |
| IN | led | STD\_LOGIC\_VECTOR(7 downto 0) | Signaux des leds |
| OUT | o\_sig\_overlap0 | STD\_LOGIC | Signal de pilotage Row0 avec chevauchement |
| OUT | o\_sig\_overlap1 | STD\_LOGIC | Signal de pilotage Row1 avec chevauchement |
| OUT | o\_sig\_overlap2 | STD\_LOGIC | Signal de pilotage Row2 avec chevauchement |
| OUT | o\_sig\_overlap3 | STD\_LOGIC | Signal de pilotage Row3 avec chevauchement |
| OUT | o\_sig\_overlap4 | STD\_LOGIC | Signal de pilotage Row4 avec chevauchement |
| OUT | o\_sig\_overlap5 | STD\_LOGIC | Signal de pilotage Row5 avec chevauchement |
| OUT | o\_sig\_overlap6 | STD\_LOGIC | Signal de pilotage Row6 avec chevauchement |
| OUT | o\_sig\_overlap7 | STD\_LOGIC | Signal de pilotage Row7 avec chevauchement |
| OUT | o\_sig\_overlap8 | STD\_LOGIC | Signal de pilotage Row8 avec chevauchement |
| OUT | o\_sig\_overlap9 | STD\_LOGIC | Signal de pilotage Row9 avec chevauchement |
| OUT | o\_sig\_overlap10 | STD\_LOGIC | Signal de pilotage Row10 avec chevauchement |
| OUT | o\_sig\_overlap11 | STD\_LOGIC | Signal de pilotage Row11 avec chevauchement |
| OUT | o\_sig\_overlap12 | STD\_LOGIC | Signal de pilotage Row12 avec chevauchement |
| OUT | o\_synchro | STD\_LOGIC | Signal de synchronisation |
| OUT | o\_sig\_state | STD\_LOGIC\_VECTOR(3 downto 0) | Etat de la machine d’état |

L’objectif de ce bloc VHDL est d’implémenter l’ensemble des blocs VHDL précédents et de gérer l’interfaçage avec l’extérieur à l’aide des composants HDL du Front Panel de Opal Kelly.

Signaux internes :

|  |  |  |
| --- | --- | --- |
| sys\_clk | STD\_LOGIC | Signal d’horloge interne au FPGA 200 MHz |
| clk100M | STD\_LOGIC | Signal d’horloge 100 MHz |
| clk\_en\_freq | STD\_LOGIC | Signal clock enable |
| okClk | STD\_LOGIC | Signal Front Panel |
| okHE | STD\_LOGIC\_VECTOR(112 downto 0) | Signal Front Panel |
| okEH | STD\_LOGIC\_VECTOR(64 downto 0) | Signal Front Panel |
| okEHx | STD\_LOGIC\_VECTOR(259 downto 0) | Signal Front Panel |
| ep40trig | STD\_LOGIC\_VECTOR(31 downto 0) | Signal Front Panel TriggerIn |
| dump\_sequence | STD\_LOGIC | alias ep40trig(0) |
| ep60trig | STD\_LOGIC\_VECTOR(31 downto 0) | Signal Front Panel TriggerOut |
| trigPipeOut | STD\_LOGIC | alias ep60trig(0) |
| trigHK | STD\_LOGIC | alias ep60trig(1) |
| pipein\_wr | STD\_LOGIC | Signal écriture Pipe IN |
| pipein\_sig | STD\_LOGIC\_VECTOR(31 downto 0) | Signal données Pipe IN |
| pipeout\_sig | STD\_LOGIC\_VECTOR(31 downto 0) | Signal données Pipe OUT |
| pipeout\_sig\_13bit | STD\_LOGIC\_VECTOR(12 downto 0) | Alias pipeout\_sig(12 downto 0) |
| HK\_pipeout | STD\_LOGIC\_VECTOR(31 downto 0) | Signal données Pipe HK |
| fifoIn\_write\_en | STD\_LOGIC | Signal écriture fifo in |
| fifoIn\_read\_en | STD\_LOGIC | Signal lecture fifo in |
| fifoIn\_full | STD\_LOGIC | Signal fifo in pleine |
| fifoIn\_empty | STD\_LOGIC | Signal fifo in vide |
| fifoIn\_dout\_128b | STD\_LOGIC\_VECTOR(127 downto 0) | Signal données en sortie fifo in |
| fifoIn\_dout | STD\_LOGIC\_VECTOR(31 downto 0) | Alias fifoIn\_dout\_128b(127 downto 96) |
| fifoIn\_valid | STD\_LOGIC | Signal données valides fifo in |
| fifoOut\_write\_en | STD\_LOGIC | Signal écriture fifo out |
| fifoOut\_read\_en | STD\_LOGIC | Signal lecture fifo out |
| fifoOut\_full | STD\_LOGIC | Signal fifo out pleine |
| fifoOut\_full\_r | STD\_LOGIC | Not(fifoOut\_full) |
| fifoOut\_empty | STD\_LOGIC | Signal fifo out vide |
| fifoOut\_din | STD\_LOGIC\_VECTOR(12 downto 0) | Signal données en entrée fifo out |
| fifoHK\_write\_en | STD\_LOGIC | Signal écriture fifo HK |
| fifoHK\_read\_en | STD\_LOGIC | Signal lecture fifo HK |
| fifoHK\_full | STD\_LOGIC | Signal fifo HK pleine |
| fifoHK\_empty | STD\_LOGIC | Signal fifo HK vide |
| state | FSM\_state | Etat de la machine d’état |
| addr | unsigned(9 downto 0) | Adresse du registre à manipuler |
| num\_row | integer | Registre du row à manipuler |
| reception\_param | STD\_LOGIC\_VECTOR(95 downto 0) | Signal de réception du paramètre qui sera stocké |
| reception\_mode | STD\_LOGIC | Signal de réception du mode qui sera stocké |
| reception\_manual\_row | STD\_LOGIC\_VECTOR(39 downto 0) | Signal de réception du manual row qui sera stocké |
| reception\_cmd | t\_Array13bits(12 downto 0) | Signal de réception du row qui sera stocké |
| Cmd\_param\_1 | t\_Device\_Ctrl\_1 | Signal de stockage du paramètre qui sera stocké |
| Cmd\_param\_2 | t\_Device\_Ctrl\_2 | Signal de stockage du paramètre qui sera stocké |
| Cmd\_param\_3 | t\_Device\_Ctrl\_3 | Signal de stockage du paramètre qui sera stocké |
| Cmd\_manual\_row | t\_Manual\_Row | Signal de stockage du manual row qui sera stocké |
| Cmd\_row | t\_Row | Signal de stockage du row qui sera stocké |
| Version | t\_Version | Version du FW et de la carte HW |
| rst\_n | STD\_LOGIC | = not(i\_rst) ET mode |
| sig\_overlap0\_int | STD\_LOGIC | Signal interne de pilotage Row0 avec chevauchement |
| sig\_overlap1\_int | STD\_LOGIC | Signal interne de pilotage Row1 avec chevauchement |
| sig\_overlap2\_int | STD\_LOGIC | Signal interne de pilotage Row2 avec chevauchement |
| sig\_overlap3\_int | STD\_LOGIC | Signal interne de pilotage Row3 avec chevauchement |
| sig\_overlap4\_int | STD\_LOGIC | Signal interne de pilotage Row4 avec chevauchement |
| sig\_overlap5\_int | STD\_LOGIC | Signal interne de pilotage Row5 avec chevauchement |
| sig\_overlap6\_int | STD\_LOGIC | Signal interne de pilotage Row6 avec chevauchement |
| sig\_overlap7\_int | STD\_LOGIC | Signal interne de pilotage Row7 avec chevauchement |
| sig\_overlap8\_int | STD\_LOGIC | Signal interne de pilotage Row8 avec chevauchement |
| sig\_overlap9\_int | STD\_LOGIC | Signal interne de pilotage Row9 avec chevauchement |
| sig\_overlap10\_int | STD\_LOGIC | Signal interne de pilotage Row10 avec chevauchement |
| sig\_overlap11\_int | STD\_LOGIC | Signal interne de pilotage Row11 avec chevauchement |
| sig\_overlap12\_int | STD\_LOGIC | Signal interne de pilotage Row12 avec chevauchement |
| HK\_value | STD\_LOGIC\_VECTOR(31 downto 0) | Valeur du registre que l’on souhaite lire |
| sig\_sync | STD\_LOGIC\_VECTOR(12 downto 0) | Signaux de synchronisation de chaque bloc sequence\_treatment |
| test | STD\_LOGIC | Signal test |
| cmp\_trig | Unsigned(7 downto 0) | Compteur trigger led |
| led\_int | STD\_LOGIC\_VECTOR(7 downto 0) | Signaux internes des leds |
| cmp | Unsigned(25 downto 0) | Compteur |

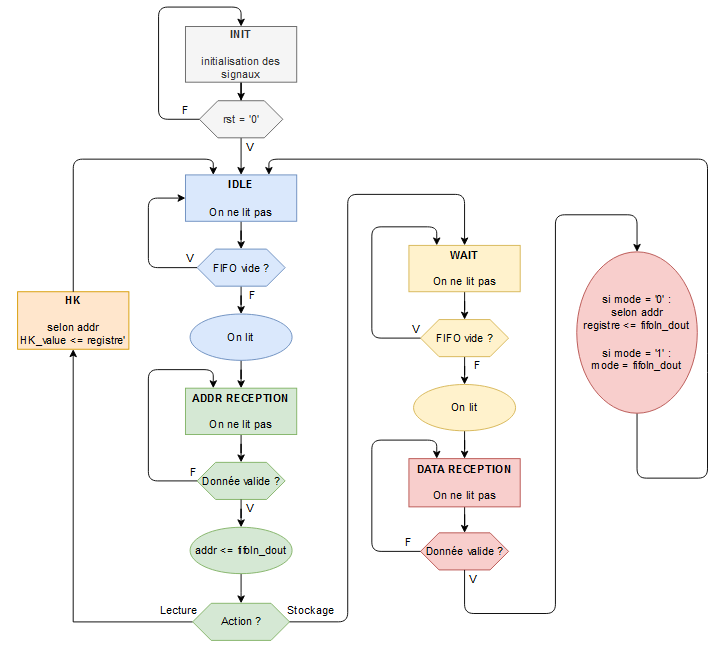
Description du bloc :

Ecriture dans les registres et stockage des paramètres et séquences

Processus IBUSDS\_inst de récupération du signal d’horloge interne au FPGA

Processus P\_Cmd\_reception

Ce processus permet la réception et la lecture ou le stockage des paramètres dans les différents registres. Voici le fonctionnement de la machine d’état :



Le processus de lecture des données dans le Pipe In est le suivant :

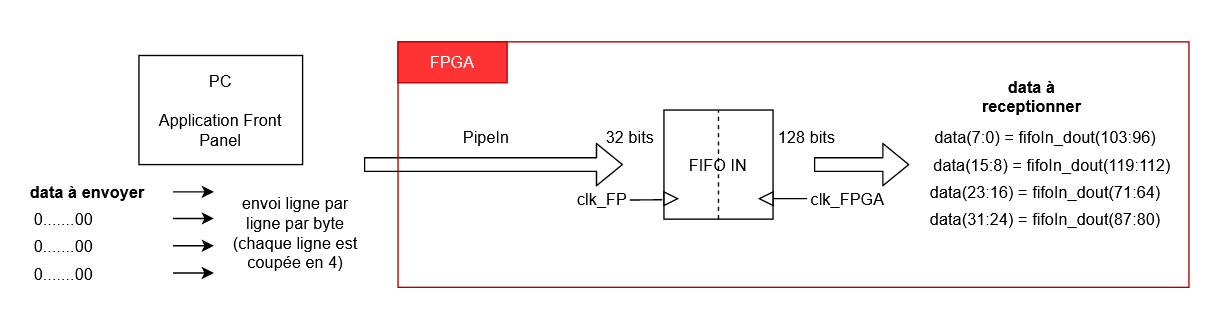
L'envoi des données dans le pipe est un peu particulier, le Front Panel ne nous laisse pas envoyer les données dans le format que l'on souhaite :

Pour une communication grâce à un USB2.0, le Pipe In pourra transporter des mots de 16 bits mais depuis le PC on ne pourra envoyer que des mots de 8 bits. Le premier mot envoyé sera le byte de poids faible (7:0) et le deuxième mot sera le byte de poids fort (15:8), le Pipe se charge de les recoller. Toutes les données envoyées doivent être de format binaire. Le transfert se fait à une vitesse de 38 MB/s.

Pour une communication grâce à un USB3.0, le Pipe In pourra transporter des mots de 32 bits mais on ne pourra envoyer que des mots de 8 bits depuis le PC. De la même façon que précédemment, le premier bytes envoyé sera de poids faible (7:0), le second (15:8),..., le Pipe se charge de les recoller. Toutes les données envoyées doivent être de format binaire. Le transfert se fait à une vitesse de 200 MB/s.

Il faut également prendre en compte le fait que le Pipe ne transporte qu'un nombre de bit égal ou multiple de 128 bits. Ainsi quand on utilise l'USB3.0, il faut au minimum envoyer 4 mots de 32 bits ou 4\*N mots pour avoir un nombre de bits multiple de 128 bits. Pour palier à cette contrainte, lorsque l'on envoie un mot de 32 bits on envoie systématiquement 3 mots de 32 bits égaux à '0'. On utilise ensuite une FIFO dans le Firmware avec une entrée sur 32 bits et une sortie sur 4\*32=128 bits et on ne récupère que le mot qui nous intéresse vraiment. Ainsi on respecte bien un transfert d'un nombre de bit multiple de 128 bits.

Exemple pour un transfert par USB3.0 :



Processus P\_pipeout\_process qui permet l’envoie des données de pilotage des rows lorsque l’utilisateur le souhaite (passage de dump\_sequence à ‘1’). Pour cela on charge la fifo OUT en autorisant l’écriture (fifoOut\_write\_en = ‘1’) et on écrit tant que la fifo OUT n’est pas. Une fois la fifo OUT pleine on stop l’écriture (fifoOut\_write\_en = ‘0’) et on sort un signal de trigger grâce à un Trigger OUT du Front Panel.

Implémentation des blocs VHDL :

* Div\_freq2
* Div\_freq
* Sequence\_treatment row0
* Sequence\_treatment row1
* Sequence\_treatment row2
* Sequence\_treatment row3
* Sequence\_treatment row4
* Sequence\_treatment row5
* Sequence\_treatment row6
* Sequence\_treatment row7
* Sequence\_treatment row8
* Sequence\_treatment row9
* Sequence\_treatment row10
* Sequence\_treatment row11
* Sequence\_treatment row12
* Fifo\_pipein
* Fifo\_pipeout
* fifoHK\_pipeout

Implémentation des composants HDL du Front Panel Opal Kelly :

* okHost
* okWireOR
* okTriggerIn
* okTriggerOut
* okPipeIn
* okPipeOut (signaux de pilotage)
* okPipeOut (signaux HK)

**DISTRIBUTION LIST**

|  |  |  |  |
| --- | --- | --- | --- |
| **INTERNAL Distribution** |  | **EXTERNAL Distribution** | |
| **(IRAP)** |  | **Name** | **Company** |
|  |  |  |  |
|  |  |  |  |
| **BARRET** Didier |  |  |  |
| **BERTRAND** Bernard |  |  |  |
| **CAMUS** Thierry |  |  |  |
| **COEUR-JOLY** Odile |  |  |  |
| **MURAT** David |  |  |  |
| **OZIOL** Christophe |  |  |  |
| **PAROT** Yann |  |  |  |
| **POINTECOUTEAU** Etienne |  |  |  |
| **RAVERA** Laurent |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |