CentraleSupélec

Reconfiguration Dynamique Partielle

Flow de conception : Reconfiguration dynamique partielle FPGA. Design suite : Vivado 2021.1 ou supérieur

---

Elèves :
Thomas BADTS
Gabriel BLANCHET
Hippolyte BOYER
Alexandre LEROYER
Ulysse VINCENTI
Johnny WAKIM

Professeur :
Amor Nafkha

5 avril 2023

Table des matières

1 Pré requis

Les outils utilisés sont :
— **Ressource :** Source du projet Vivado
— **Software :** Xilinx Vivado 2021.1 ou supérieur
— **Software :** Xilinx Vitis 2021.1 ou supérieur
— **Hardware :** Digilent Zybo Z7-20

2 Objectifs

L'objectif de cette procédure est de décrire le procédé de conception pour réaliser une ou plusieurs parties dynamiquement reconfigurable .
Nous décrirons deux implémtentations de la DPR (Dynamic Partial Reconfiguration) :

— par commande externe via le Hardware Manager
— avec l'utilisation de la PS Zynq-7000

Application test :

Traitements d'image câblés implémentés dans le module de traitement reconfigurable PU. Le module PU est connecté en entrée à une image d'entrée 128 x 128 et range le résultat du traitement dans une mémoire d'image de sortie de dimension 256 x 256.
Un contrôleur VGA génère les signaux nécessaires à l'affichage de cette image sur un écran VGA. **Plusieurs configurations de traitement sont envisagées :**

— **PU_A :** répétition de l'image 128x128 dans 4 quadrants de l'image 256x256.
— **PU_B :** upscaling d'un facteur 2 par simple duplication des pixels.
— **PU_C :** upscaling d'un facteur 2 par interpolation simple avec les 2 pixels voisins.
— **PU_D :** upscaling d'un facteur 2 suivant la norme H264.
— **PU_E :** rotation de l'image.

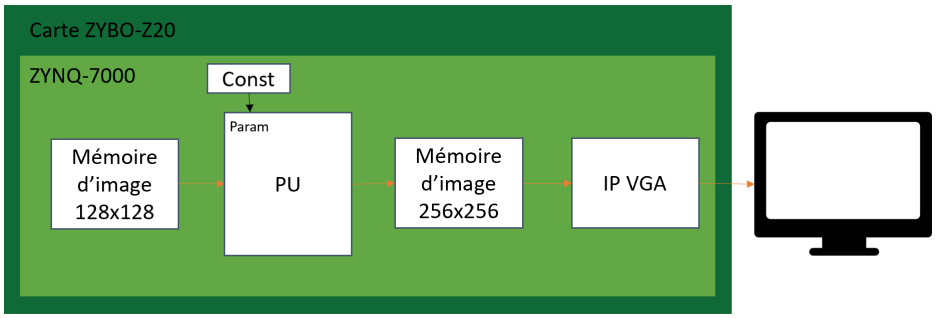
Concept de la DPR :

DPR, ou reconfiguration dynamique partielle, permet de programmer un FPGA avec une partie fixe et une partie reconfigurable. Cette dernière peut être composée de plusieurs partitions. Chaque partition a différentes configurations parmi lesquelles il est possible d'alterner. Chaque configuration d'une même partition doit avoir les mêmes sorties, même si celles-ci ne sont pas utilisées. Toutes ces configurations partagent les mêmes ressources physiques. Chaque configuration a son propre *bitstream* . Pour reprogrammer la partition avec la configuration souhaitée, nous pouvons utilisé le *bitstream* correspondant. On note que la partie fixe possède son propre *bitstream*. Celui-ci est fixe une fois téléversé sur le FPGA.

3 Design sans PS

3.1 Objectifs et description

Ce design, le plus simple, est composé d'une mémoire d'entrée (128x128) pré-chargé avec l'image (*image.coe*), d'un bloc **PU** pour traiter cette image, d'une mémoire de sortie (256x256), finalement un contrôleur VGA pour afficher notre image.
Ce design permet de réaliser de la DPR via le *hardware manager* en chargeant les *bitstream*s correspondants aux différents PU.



3.2 Réalisation

3.2.1 Organisation du projet sous Vivado

Pour réaliser une DPR de façon générale il est nécessaire de créer un projet vierge pour notre board (Zybo-Z20) si non disponible **Cliquer ICI**, et d'y créer un ***Block Design*** dans lequel on y retrouve :

— 2 x Block Memory Generator : True Dual Port RAM
— Custom IP_VGA
— Digilent IP RGB to DVI Video Encoder (Source)
— IP Clocking Wizard

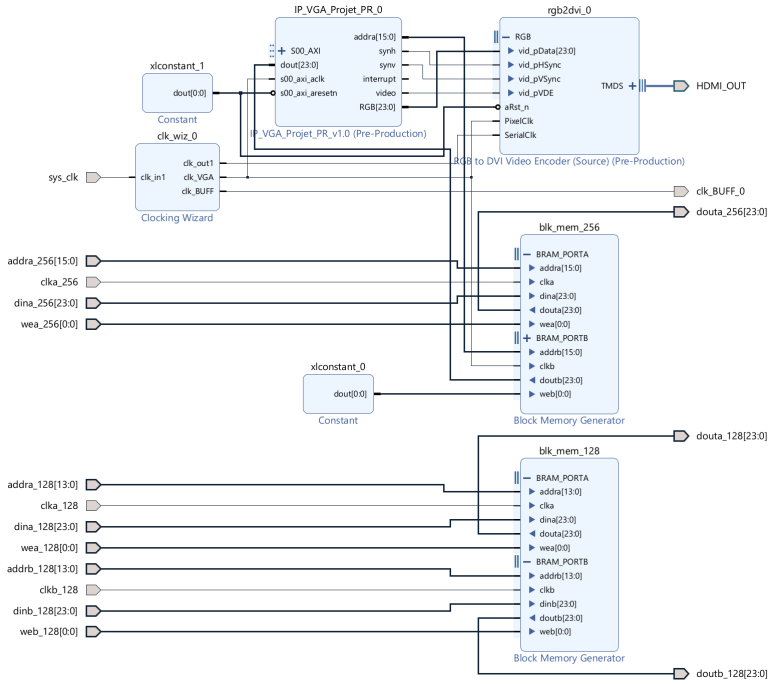


FIGURE 1 – Diagramme bloc du projet

3.2.2 Configuration des blocs

En prenant référence sur le diagramme bloc précédent, configurer les différentes IP comme suivant :

blk_mem_128 :
- Memory Type : True Dual Port RAM
- Port A Options :
  - Write Width = 24
  - Read Width = 24
  - Write Depth : 16384
  - Read Depth : 16384
  - Enable Port Type : Always Enabled
- Port B Options :
  - Write Width : 24
  - Read Width : 24
  - Write Depth : 16384
  - Read Depth : 16384
  - Enable Port Type : Always Enabled
- Other Options :
  - Load Init File : image.coe

blk_mem_256 :
- Memory Type : True Dual Port RAM
- Port A Options :
  - Write Width = 24
  - Read Width = 24
  - Write Depth : 65536
  - Read Depth : 65536
  - Enable Port Type : Always Enabled
- Port B Options :
  - Write Width : 24
  - Read Width : 24
  - Write Depth : 65536
  - Read Depth : 65536
  - Enable Port Type : Always Enabled

rgb2dvi :
- Reset Active High : unchecked
- Generate SerialClk internally from pixel clock : unchecked

clk_wiz :
- Clocking Options
  - PLL : Checked
  - Primary Clock Input Frequency : 125 MHz
- Output Clocks
  - clk_out1
    - Output Clock : Checked
    - Freq Requested : 125 MHz
  - clk_out2
    - Output Clock : Checked
    - Freq Requested : 25 MHz
  - clk_out1
    - Output Clock : Checked
    - Freq Requested : 25 MHz

  - Phase : 180 degrees
- reset : unchecked

Il est nécessaire de générer une seconde horloge à 25 MHz avec un décalage de phase de 180.0°. Cela nous permet de synchroniser la seconde mémoire ainsi que l'affichage pour éviter les conflits entre le traitement de l'image et son affichage.

Rajouter les IP : *Constantes* pour les ports *reset* et *write enable* des mémoires.

Passer tous les ports des mémoires en externe via l'option *external* pour pouvoir les connecter par la suite avec la PU.

3.2.3 Ajout de la PU au projet

Une fois le diagramme bloc réalisé, nous générons le *wrapper*. Puis nous rajoutons le fichier **top.vhd** Ainsi que les fichiers **pu_d.vhd, pu_d_h.vhd , pu_d_v.vhd**. Dans un premier temps, nous ajoutons uniquement la **pu_d**. Celle-ci est la plus contraignante en terme de ressource physique qu'elle nécessite.

Pour utiliser la pu_d, il faut que l'on crée une IP *block memory* : **img_ram_pu** :

- Memory Type : Simple Port RAM
- Port A Options :
  - Write Width = 24
  - Read Width = 24
  - Write Depth : 32768
  - Read Depth : 32768
  - Enable Port Type : Always Enabled

3.2.4 Configuration de la DPR

Avant d'aller plus loin, il faut faire attention de faire une sauvegarde du projet car les étapes suivantes sont irrémédiables. Pour cela : File/Project/Archive...

Une fois les différentes *PU* importées, nous devons activer l'option de DPR, en sélectionnant : *Tools/Dynamic Funcfuction eXchange Wizard....*
Puis, nous sélectionnons le fichier PU dans la source et on sélectionne l'option *Create Partition Definition....* Dans le menu qui vient de s'ouvrir, nous entrons un nom unique pour cette première version de la PU. Une fois le chargement terminé, nous ouvrons le menu *Dynamic Function eXchange Wizard* depuis le *Project Manager* qui permet de configurer cette partition.
Dans le menu *Dynamic Function eXchange Wizard*, appuyer sur *next*. Nous pouvons voir les différentes configurations de la partition. Pour en ajouter, appuyer sur le "+", entrer un nom unique pour chaque configuration de la partition ajoutée , et ajouter le ou les fichiers source de votre PU. Alors que les versions pu_a, pu_b,pu_c et pu_e ont chacun un seul fichier, la version pu_d nécessite l'importation de plusieurs fichiers ainsi que le bloc mémoire dont on a parlé précédemment.
Appuyer sur *next* et choisir l'option *automatically create configuration* puis appuyer sur *next* et *automatically create configuration run* puis *next* et dernièrement *finish*.
Après être sortit du *Dynamic Function eXchange Wizard*, nous pouvons lancer la synthèse

de notre design. Cette première synthèse concerne uniquement le *top* de notre projet, La synthèse des PU se fera ultérieurement après le *FloorPlanning*.

3.2.5 *FloorPlanning*

Création du *Pblock* : Une fois la synthèse finit, ouvrir le design synthetisé grâce au *flow navigator*. Ouvrir la fenêtre *device* à travers *"window/device"*. Dans la netlist, 2 nets apparaissent : U_pu et design. Il faut clique droit sur le PU_PR pour accéder au menu de *Floorplanning/Draw Pblock*. On dessine un bloc sur le FPGA en prenant en compte la taille requise.

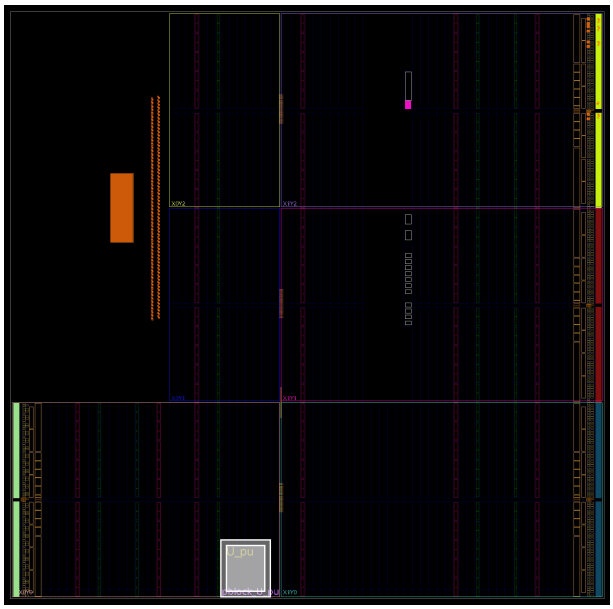


FIGURE 2 – Création d'un *Pblock* dans le *Floorplanning*

Changement des pramètres : En allant dans la fenêtre de propriétés du *Pbloc*, on change les paramètres :

1. RESET_AFTER_RECONFIG : sélectionné.
2. SNAPPING_MODE : ROUTING

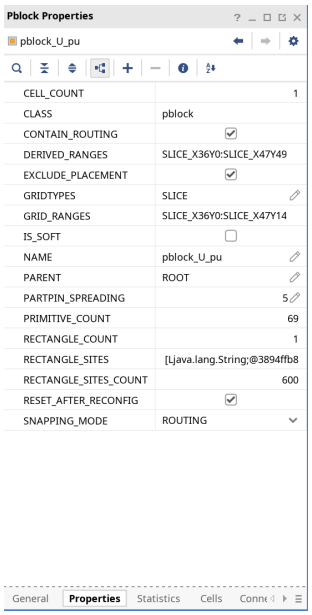


FIGURE 3 – Changement des propriétés du *Pblock*

Vous verrez alors que votre bloc se connecte maintenant aux bords haut et bas du super bloc du FPGA, c'est à ces endroits que sont connectés les *netlists* de *clock* et de *reset*. Il faut que chaque *Pblock* soit connecté à un *netlists* de *clock* et de *reset*, il faut donc qu'il fasse au minimum une colonne complète. Celà est fait grâce à l'option SNAPPING_MODE. L'option *reset after reconfig* permet de *reset* le *Pblock* juste après qu'il soit reprogrammé.

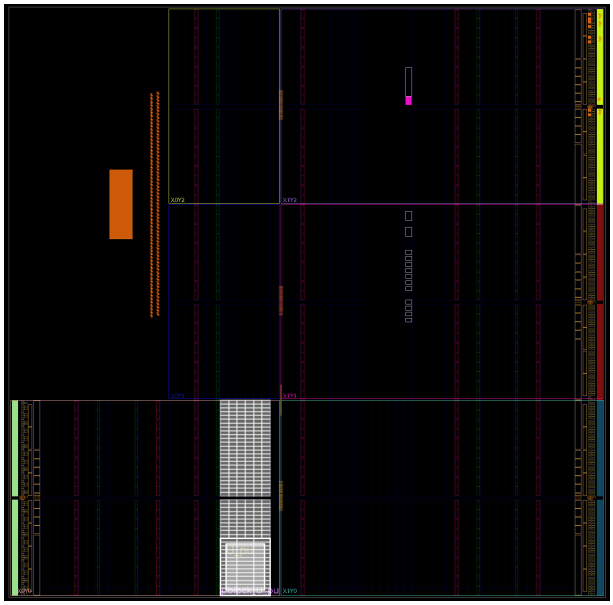


FIGURE 4 – Le *Pblock* change de taille automatiquement

Un *Pblock* adapté à vos besoins : On va maintenant regarder qu'elles sont les besoins de notre *Pblock* au niveau des ressources et allons accorder notre *Pblock* à nos besoins.

Les blocs vert sont les DSP, les blocs rouges sont la BRAM et enfin les blocs noirs sont les LUTs. En fonction de vos besoins intégrée plus ou moins de ressources. Si votre *Pblock* ne contient pas les ressources qu'il vous faut, la ligne qui aura un manque de ressource apparaîtrait rouge.

Physical Resource Estimates

Site Type	Available	Used	% Util
Slice LUTs	9776	766	7.84
LUT as Logic	9776	766	7.84
LUT as Memory	3196	0	0
Slice Registers	19552	554	2.83
Register as Flip Flop	19552	554	2.83
Register as Latch	19552	0	0
F7 Muxes	4888	36	0.74
F8 Muxes	2444	0	0
Block RAM Tile	24	22	91.67
RAMB36/FIFO	24	22	91.67
RAMB18	48	0	0
DSPs	32	0	0

FIGURE 5 – Ressources pour le *Pblock* de notre projet normalement

On observe que le *Pblock* demande de la BRAM, il faut donc faire attention à intégrer une ligne rouge.

Une fois optimisé pour le *Pblock*, le *Floorplanning* ressemble à l'image suivante, l'élément contraignant est la BRAM dans ce design qui provient du pu_d.

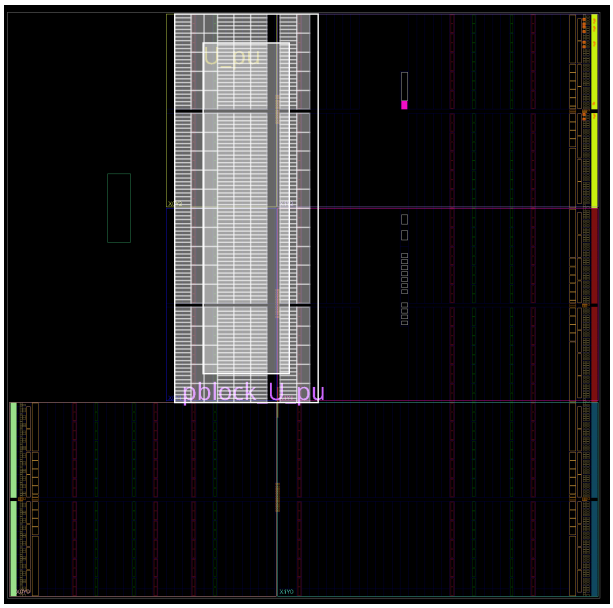


FIGURE 6 – Le *Pblock* est optimisé pour les ressources demandés

Par la suite on réalise un rapport de DRC en allant dans reports/report DRC. On sélectionne le rapport DRC du floorplanning. Si ce test ne donne pas d'erreur on peut refaire la synthèse puis l'implémentassions et la génération du *bitstream*. On observe alors que la génération du *bitstream* se fait d'abords sur le projet complet puis sur le *childs* qui sont les configurations PU.

3.2.6 Hardware Manager

Une fois les *bitstream* générés. Nous ouvrons le Hardware Manager et nous pouvons accéder à la carte via *Open Target* et téléverser le *bitstream* du *top* dans le FPGA à travers *Program Device*. Les images s'affichent alors comme dans la configuration avec la PU_5 qui est la PU par défaut. On peut alors téléverser le *bitstream* partiel d'un autre PU pour obtenir un autre résultat. Pour cela, nous pouvons clicker sur "Program Device" et choisir le *bitstream* de la configuration partielle voulue : pu_PR_pu_b_partial.bit pour choisir la pu_b, par exemple.

4 Design avec PS

4.1 Design flow avec ICAP

Nous n'avons pas réussi à réaliser la DPR avec la PS (Processing System), cependant nous avons étudié les différentes IP à mettre en jeu et compris le concept.
Il est nécessaire d'utiliser une mémoire partitionnée en N partitions de tailles identiques, ajouter un DFX controller et un ICAP.

4.1.1 DFX Controller

Le *Dynamic Functional eXchange Controller* est une IP fournie par Xilinx. Elle permet de réaliser l'interface entre la mémoire (où sont contenues les *bitstreams*) et l'ICAP. Elle joue un rôle crucial à l'ajout de la PS, elle nous permet en fonction des événements hardware et/ou software de définir le *bitstream* à sélectionner. (Grâce à la compatiblitée avec l'AXI4)
Cette IP nécessite de connaître au préalables les différents modules.
Lors de sa définition il est possible d'instancier de 1 à 32 *virtual sockets* (i.e. zones reconfigurables) et par *virtual socket* de 1 à 128 modules reconfigurables.

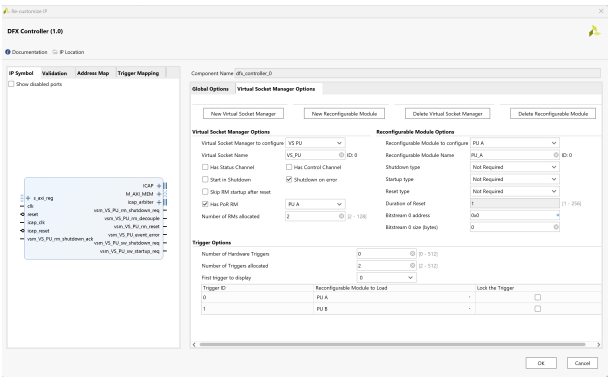


FIGURE 7 – Configuration des *virtual sockets* et modules

Pour configurer ce bloc il est nécessaire de se munir des informations suivantes :
— La taille des *bitstreams* (en octet)
— L'adresse mémoire de début de ces *bitstreams* (dans le bloc mémoire utilisée)
La taille de chacun des *bitstreams* est facilement accessible grâce à la partie précédente, dans notre cas nous avons **150k** octets par *bitstream*.

4.1.2 Mémoire externe

Nous avons donc besoin d'un espace mémoire assez important si par exemple nous souhaitons par la suite y ajouter de nouveaux modules à notre *virtual socket*, nous avons déterminé que la mémoire **RAJOUTER LE BLOC MEMOIRE identifié dans le HARDWARE MANAGER** serait adaptée à notre besoin.
Cette mémoire est externe à notre PL, elle est disponible sur la carte zybo-z20, il est donc nécessaire de l'interfacer pour y avoir accès. **a faire**.

4.1.3 ICAP

Nous utiliserons Advanced eXtensible Interface (AXI) HWICAP (Hardware Internal Configuration Access Port), qui nous permet d'écrire et de lire la mémoire de configuration du FPGA au travers de l'ICAP.

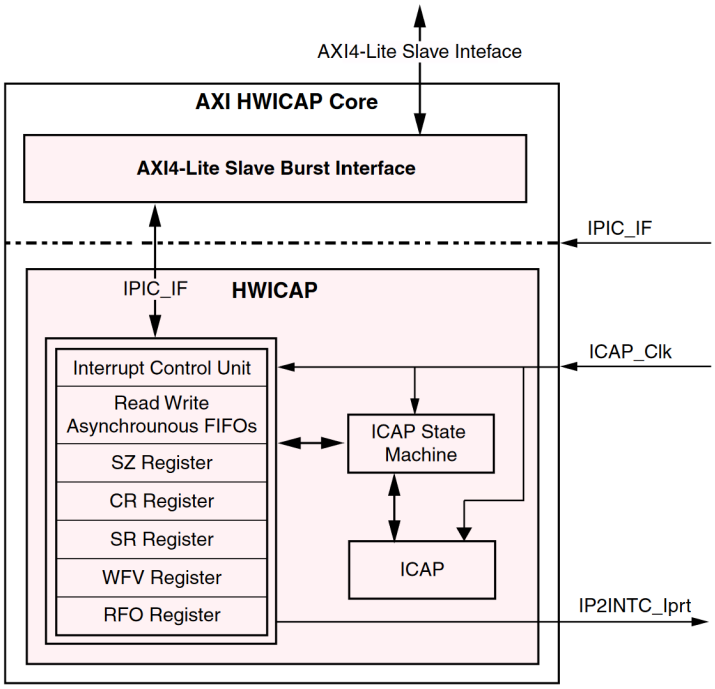


FIGURE 8 – Top Level Block Diagram for the AXI HWICAP core

4.2 Design flow PCAP

Afin de réaliser la DPR à l'aide de la PS nous allons utiliser la PCAP (Programmable Configuration Access Port). Pour cela, nous allons inclure la PS dans le diagramme bloc. Afin de réaliser les connections avec la PS et ajouter les blocs nécessaires à son utilisation, appuyer sur *run block automation* puis sur *run connexcion automation*. Après avoir ajouté et connecté la PS, lancé le *bitstream* afin de vérifier le bon fonctionnement du projet. Une fois le *bitstream* terminé, exporter le projet de Vivado via "*file->export...*". Nous exportons le projet en incluant le *bitstream*.

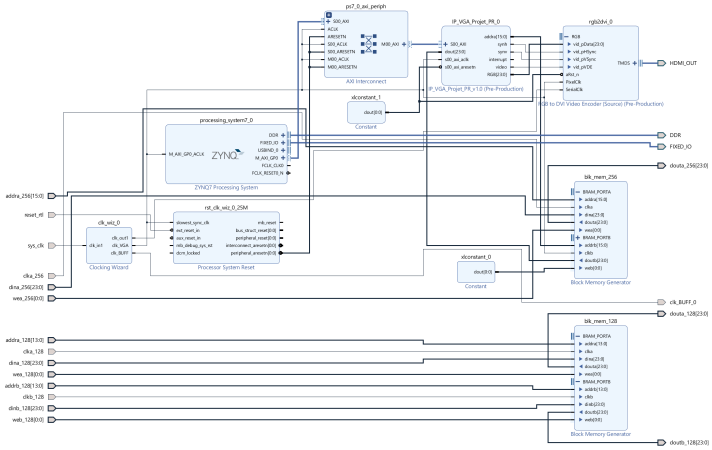


FIGURE 9 – Top Level Block Diagram for the AXI HWICAP core

4.2.1 PCAP

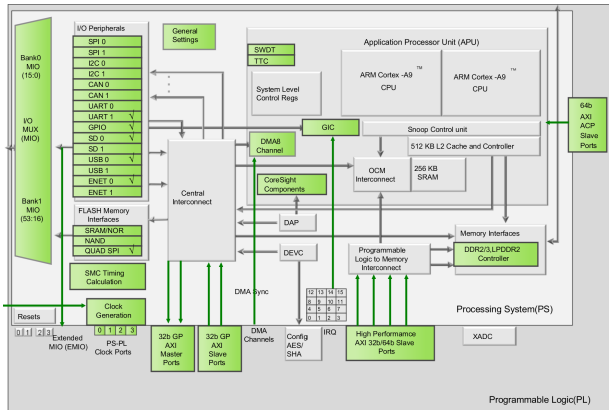


FIGURE 10 – Diagramme Interne de la PS

4.2.2 Configuration de la SD CARD

Nous allons, en parallèle, utiliser le script tcl : *create_prom_file.tcl*. Pour cela regrouper les différents *bitstreams* des PUs (ils sont localisés dans le dossier du projet : [noms_du_projet].runs dans les dossiers *child_0_impl_1* et dans le dossier *impl_1*). Pour ce projet nous en avons 5 :

- pu_1_pu_a_partial.bit
- pu_1_pu_b_partial.bit
- pu_1_pu_c_partial.bit
- pu_1_pu_d_partial.bit
- pu_1_pu_e_partial.bit

Dans le script tcl il faut nommer ces fichiers dans la variable "*partial*". Afin d'exécuter le script, lancer depuis d'un terminal de commande :

```
vivado2021 -mode tcl -source create_prom_file_kc705.tcl
```

Après l'exécution du script, nous obtenons 4 fichiers en .bin qu'il va falloir placer dans la carte SD. Les fichiers .bin pourront être placés à la racine d'une carte SD vierge. Ouvrir Vitis et créer une plateforme à partir du .xsa généré lors de l'exportation du hardware Vivado. Créer une *application*. Cette application permettra de paramétrer le PCAP et ainsi contrôler le téléversement du bitstream.

5 Annexe

Listing 1 – "code tcl permettant la création de .bin de chaque .bit donné ainsi qu'un .mem qui permet d'utiliser la flash comme espace de stockage du bitstream"

```
# Create a programming file for the PROM containing the static and the
# partial bitstreams
#
# vivado2021 -mode tcl -source create_prom_file_kc705.tcl

# ---------------------------------
# Options for the complete MCS
# ---------------------------------
set BITS 16
set final_target     "-format MCS"
#set options          "-force -checksum FF -size 32"
set bpi_options      "-interface BPIx$BITS"

set static  "top"
set partials  { \
                    pu1_pu_a_partial\
                    pu1_pu_b_partial\
              }


set sizetop [file size  ${static}.bit]


# Convert each partial bitfile into a bin file formatted for the ICAP port
foreach p $partials {
    set cmd "write_cfgmem -force -format BIN -interface SMAPx$BITS -disablebitswap -loadbit \"up 0
         $p.bit\" $p"
    eval $cmd
}

set sizepart1 [file size  [lindex $partials 0].bin]

# Now do the static and pack the partials as datafiles
set cmd "write_cfgmem -force $final_target $bpi_options -loadbit \"up 0 ${static}.bit \" -loaddata \""
#append cmd " up [format %x $sizetop] [lindex $partials 0].bin"
set cntr  $sizetop
foreach p $partials {
    set cntr [expr $sizepart1 + $cntr]
    append cmd " up [format %x $cntr] $p.bin"
}
append cmd "\" dfx_prom"

puts $cmd
eval $cmd


# Now create a report with the sizes
foreach p $partials {
    set ret [file size $p.bin]
    puts "$p : $ret bytes"
}

#exit
```