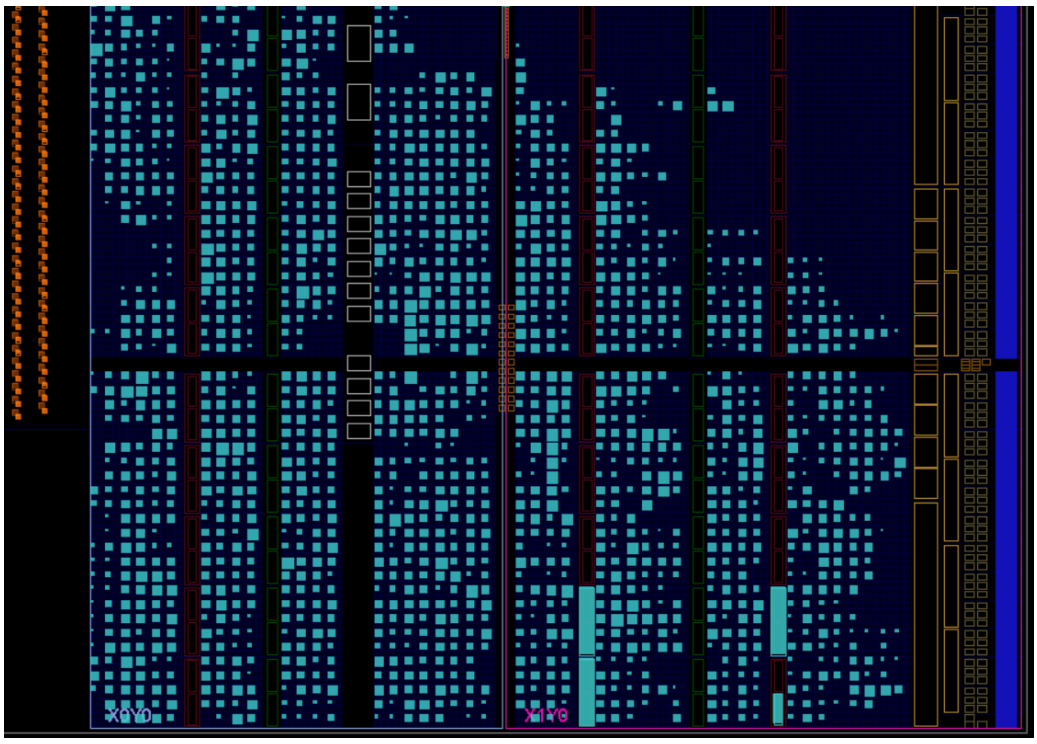
Dossier de soutenance de projet

P2 Datalogger

2015/2016 Hidéo Alexis VINOT

IMA5-SC



Thomas VANTROYS **Alexandre BOE**

Sommaire

I. Remerciements

Je tiens tout d'abord à remercier l'**équipe pédagogique du département I.M.A.** de m'avoir permis de réaliser ce projet et d'acquérir de nouvelles compétences dans les divers domaines abordés.

Je souhaite particulièrement remercier **Monsieur Alexandre BOE** et **Monsieur Thomas VANTROYS**, encadrants du projet, qui m'ont soutenu et appuyé lors des différentes étapes.

Je souhaite également remercier **Monsieur Mageshwaran SEKAR** pour m'avoir éclairé sur des sujets tant pratiques que théoriques.

II. Introduction

Dans le cadre de mon projet de fin d'étude de cinquième année en Informatique, Micro-électronique et Automatique, j'ai été amené à travailler sur le projet « Data Logger ». Durant plusieurs semaines, j'ai travaillé selon un cahier des charges défini par M. Thomas VANTROYS et M. Alexandre BOE, mes encadrants.

L'objectif est de créer un système à l'aide d'une board de développement FPGA, un dispositif capable de stocker des données à haute vitesse dans une carte de RAM.

Dans un premier temps, j'expliquerai le cahier des charges et les réflexions qui ont permis de dresser la liste du matériel nécessaire. Dans un second temps, je développerai les recherches effectuées puis, je préciserai les résultats obtenus lors de mes tentatives. Enfin, je terminerai mon mémoire en présentant une analyse et une critique de mon travail.

III. Cahier des charges

A) Contexte du projet

Le laboratoire IRCICA de M. Boé et M.Vantroys utilise un dispositif basé sur un FPGA pour traiter les données à une vitesse d'une centaine de MHz. Ce dispositif reçoit en parallèle plusieurs bits issus d'un C.A.N. qui informent sur la consommation mesurée d'un appareil électronique. Ce dispositif a été conçu par une entreprise et constitue donc une boite noire non évolutive. Ainsi, développer une nouvelle carte permettant l'acquisition des données avec un code connu pourrait permettre d'améliorer en amont le processus de collecte des données.

B) Dénomination du sujet par les enseignants

L'objectif est de réaliser une plate-forme de mesure de la consommation d'équipement électroniques : la consommation des équipements électroniques est devenue un enjeu majeur puisque les équipements sont très souvent utilisés en situation de mobilité. La réduction de la consommation passe par la compréhension des sources de consommation que ce soit par le logiciel ou par le matériel.

Afin de comprendre les mécanismes de consommation, il est nécessaire de pouvoir mesurer le courant des différents composants d'un système. Dans ce projet, nous proposons la réalisation d'une plate-forme permettant la mesure de courant.

Cette plate-forme devra être ouverte et facilement reconfigurable (ajout / retrait de voies de mesures, filtrage des mesures, ...)

La plate-forme sera constituée de :

- Un FPGA permettant de contrôler le système
- Des voies de mesures de courant embarquant un capteur et un convertisseur analogique / numérique
- Une mémoire RAM de stockage des données
- Une interface permettant de gérer la plate-forme et de récupérer les données

C) Choix matériel et logiciel

a) Logiciels

Xilinx Vivado

C'est un logiciel des synthèse et d'analyse développé par Xilinx qui vient ajouter des fonctionnalités concernant les SoC (System on Chip) pour un développement haut niveau. En comparaison avec l'ISE, le logiciel permet une autre approche du flow de conception. On a notamment accès à des outils comme le SDK intégré (Oracle) qui permet de programmer le FPGA et de compiler du code en C/C++ pour lancer des applications et debugger depuis les processeurs intégrés.

ISE Xilinx :

Xilinx ISE (Integrated Synthesis Environment) est un outil logiciel produit par Xilinx pour la synthèse et l'analyse HDL, permettant aux développeurs de synthétiser ou compiler leurs designs, réaliser des analyses temporelles, examiner les diagrammes RTL, simuler la réaction aux stimuli de la conception et configurer le dispositif cible.

Altium Designer :

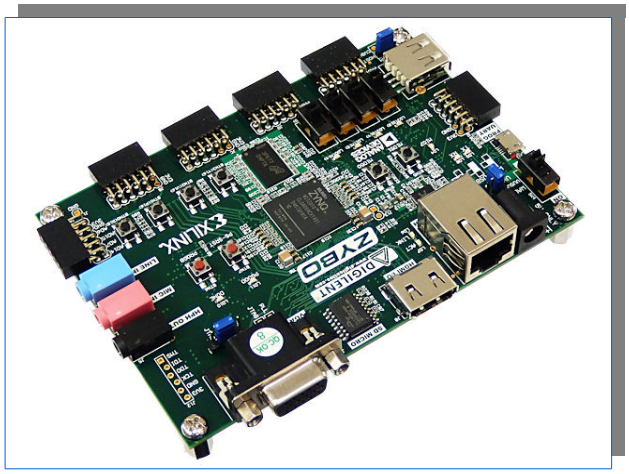
Altium Designer est une chaîne logicielle intégrée de conception assistée par ordinateur pour systèmes électronique. Altium Designer propose des fonctions de :

- saisie de schémas,
- simulation (basée sur Xspice2)
- conception/routage de circuits imprimés
- programmation de FPGA et microprocesseurs/microcontrôleurs

b) Matériel

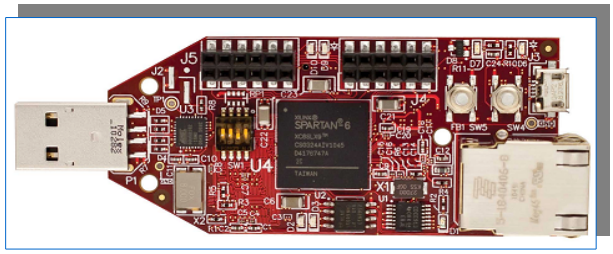
Vous pouvez trouver en **Annexe A** un accès au datasheet de chacune des board.

Plate-forme ZYNQ 7000 chez DILIGENT : ZYBO



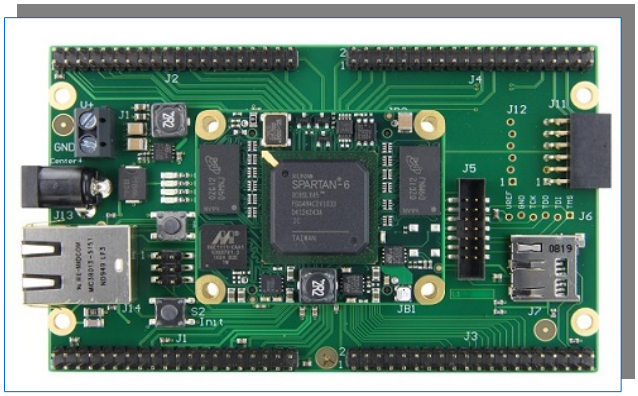
C'est une plate-forme de développement d'usage général. Celle-ci intègre un port HDMI, plusieurs ports USB dont un pour la programmation, plusieurs switch, plusieurs boutons poussoirs, un slot pour carte SD et surtout 512Mb de DDR3. Cette carte possède comme autre avantage qu'elle peut booter sur un linux via la carte SD. Ainsi, on peut se connecter en SSH/SCP sur la machine et interagir, émettre et recevoir des données en grande quantité. Je pense que le ZYNQ 7000 est un FPGA très avantageux pour faire du système embarqué.

Plate-forme SPARTAN 6 (lx9) chez AVNET : MicroBoard



C'est une plate-forme de développement à faible coût qui permet de travailler avec un SPARTAN 6 et MicroBlaze. Elle possède plusieurs entrées sorties (moins que la TE0600), des leds, un interrupteur et 4 switch très fragiles et un port Ethernet. Cette board vient de mes ressources personnelles mais le modèle étant simple et assez répandu, on trouve beaucoup de tutoriel explicatif sur internet.

Plate-forme SPARTAN 6 (lx45) chez TRENZ Electronic : TE0603 et TE0600 :



C'est une plate-forme FPGA qui intègre un micro module SPARTAN 6 lx45 sur une plate-forme de développement. L'avantage de cet équipement, est qu'il possède un connecteur PHY Ethernet, 4 leds et énormément de pins de sorties. Il sera facile, à l'aide de nappe filaire, de joindre la carte support de DDR et la carte FPGA. Ce module se programme par contre avec une liaison JTAG, une mémoire flash ou un connecteur B2B et non pas de USB.

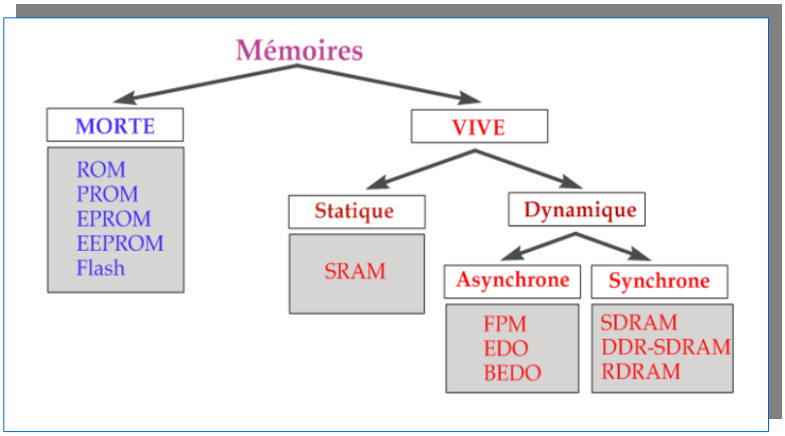
IV. De la théorie à son application

A) Recherche

Tout au long des séances de PFE, j'ai dû réaliser de nombreuses recherches pour réussir à répondre aux attentes. Je vais présenter dans cette partie les résultats des recherches que j'ai mené à propos de la mémoire. Cette partie aborde le sujet de manière générale pour se rapprocher de plus en plus du sujet qui nous concerne la mémoire DDR3 SDRAM.

a) Deux grandes familles de mémoires

Dans un ordinateur ou dans un système logique, on va classifier les mémoires de différentes manières. On distingue dans un premier temps la mémoire morte de la mémoire vive.



Mémoire morte

C'est une mémoire de type non volatile que l'on appelle aussi *Read Only Memory*. La modification de ces données est lente (ou impossible), on s'en sert pour stocker les firmwares, BIOS des ordinateurs et autres dispositifs électroniques pour la raison que sans alimentation, les données restent inscrites. L'ordre de grandeur du temps d'accès d'une mémoire morte est d'approximativement 150ns, tandis que comparativement, le temps d'accès d'une mémoire vive est de 10ns. On classe ensuite les mémoire mortes selon deux types, les réinscriptibles (*Erasable Programmable Read Only Memory, Electrically Erasable Programmable Read Only Memory, Ultra Violet Programmable Read Only Memory*) et les permanentes (*Read Only Memory, Programmable Read Only Memory*).

Pour l'avenir, on attend beaucoup des mémoires MRAM (*Magnetic Random Access Memory*). Ce sont des mémoires dont les données ne sont pas fixées par une charge électrique mais par une orientation magnétique. Cette nouvelle technologie permet un accès direct aux données et celles-ci n'ont pas besoins d'énergie pour être conservées.

Mémoire vive

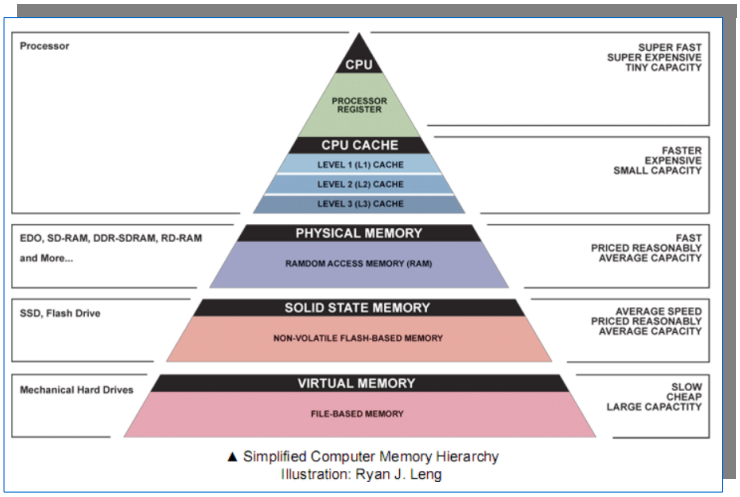
La mémoire vive (que l'on appelle *Random Access Memory*) dès son alimentation en électricité peut être modifiée à l'infini. Elle est qualifiée de volatile car les données inscrites sur

celle-ci n'ont pas lieu de persister sans énergie pour les maintenir. C'est une mémoire à accès direct dont la vitesse de lecture/écriture est plus grande que celle de la mémoire morte. On se sert de cette mémoire pour stocker temporairement les fichiers que le dispositif électronique utilise. Comme pour les mémoires mortes, on va distinguer deux grandes familles, les *Static Access Memory* (*Static Random Access Memory, Dual Ported Random Access Memory, Magnetic Random Access Memory, Phase-Change Random Access Memory)*, et les *Dynamic Random Access Memory (Synchronous Dynamic RAM, Double Data Rate Synchronous Dynamic RAM , Double Data Rate 2 SDRAM, Double Data Rate 3 SDRAM, Double Data Rate 4 SDRAM).*

On appelle « statique » les mémoires dont les données n'ont pas besoins de rafraîchissement pour se conserver. On appelle « dynamique » les mémoires dont il faut rafraîchir les données. Sans la SDRAM, un bit mémoire est symbolisé par l'association d'un picocondensateur et d'un transistor en commutation. Ainsi, à cause d'un courant de fuite dans le condensateur, il faut rafraîchir la valeur du bit en venant réécrire sa valeur. C'est une opération qu'il faut effectuer typiquement toutes les 64 ms.

b) Approche hiérarchique

On peut aussi avoir une approche hiérarchique de la mémoire, qui tient compte de l'éloignement du processeur.



Registres

C'est l'élément de mémoire le plus rapide, situé au niveau processeur ils servent au stockage des résultats intermédiaires et des opérandes.

Mémoire cache

C'est une mémoire rapide et de faible capacité. Son rôle est de stocker les données les plus utilisées par le processeur permettant ainsi l'accès à la mémoire centrale plus rapidement.

Mémoire centrale

La mémoire centrale est plus lente que la mémoire cache et que les registres. Elle contient les données et les programmes.

Mémoire d'appui

C'est l'équivalent de la mémoire cache mais pour la mémoire de masse.

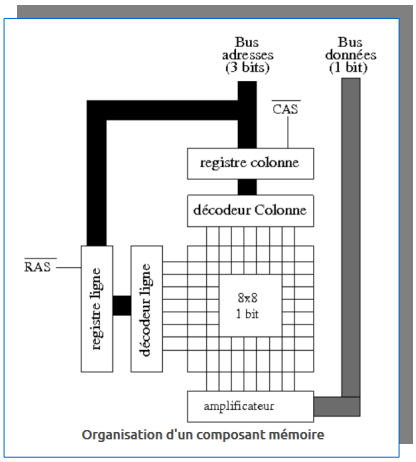
Mémoire de masse

C'est un support de stockage de grande capacité qui permet de stocker et d'archiver des informations.

c) Organisation de la mémoire

Le composant mémoire

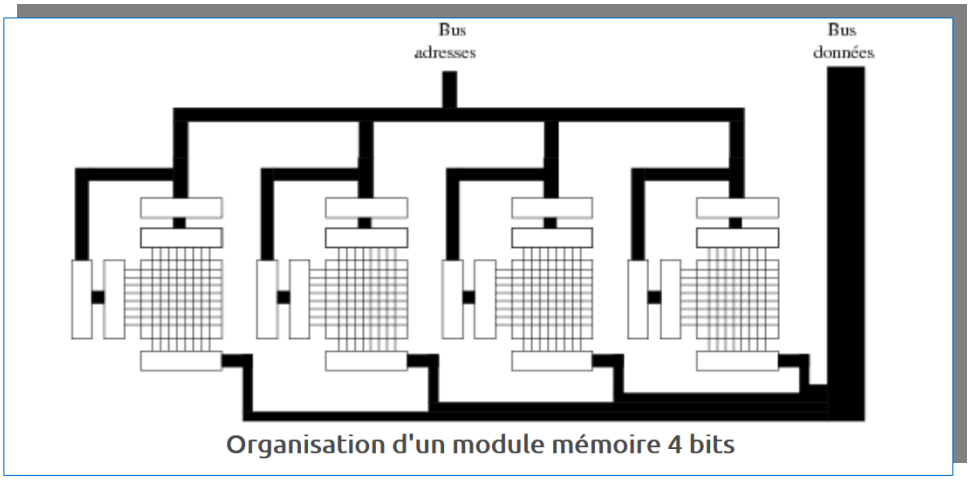
Un composant mémoire est composé de lignes L et de colonnes C. À chacune des intersections (ligne - colonne), on trouve des bascules pour le cas de la SRAM ou des associations condensateurs-transistors. On retrouve un bus adresse et un bus de données, lorsque l'utilisateur vient écrire une adresse, ou un masque adresse, l'adresse est décodé en ligne et en colonne pour changer le bit en question.



Organisation d'un composant mémoire

Le module mémoire

Voici un exemple d'un module mémoire à 4 bit.



Organisation d'un module mémoire 4 bits

d) Mémoire DDR-SDRAM

Il existe différent génération et modèle de DDR. Cette technologie a commencé sa route au début des années 2000 et n'a fait que évolué, la preuve en est qu'on est à la génération 4 pour les nouveaux PC. De plus on trouvera en annexe la descriptions des pins d'une barrette de DDR3, que qui peut être utile pour réaliser une carte électronique.

Annexe C : Caractéristiques des DDR 1234 SDRAM

Annexe D : Pin Description DDR3

Le débit mémoire

On parle souvent de « débit mémoire » lorsque l'on s'intéresse aux technologies des mémoires. Ainsi, le débit d'une mémoire dépend de trois facteurs.

- La fréquence du bus FSB (*Front Side Bus)* reliant la mémoire au CPU
- la largeur du bus et le nombre de bits
- un coefficient multiplicateur qui correspond au nombre de données transférées en un front d'horloge (normal = 1, DDR = 2, QDR = 4)

Pour effectuer le calcul, il suffit d'utiliser la formule suivante :

$$Débit = Fréquence \times Largeur \times Coefficient$$

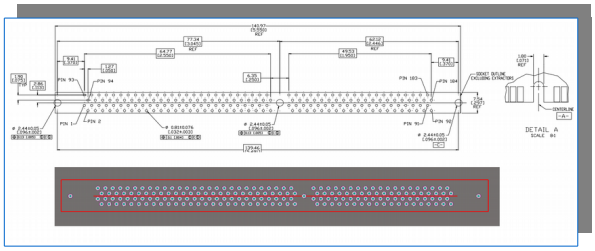
Package DDR SDRAM

Il existe plusieurs package pour la DDR, il est important de connaître les différences, car on ne parle pas des mêmes chose si on parle de DDR en boîtier DIP ou DDR en boîtier DIMM.

Annexe E : Package DDR3

Empreinte Altium Designer

On pourra trouver sur le wiki, le fichier de l'empreinte de DDR3 que j'ai réalisé sous Altium Designer. Celle ci peut être utile en cas de réalisation de typon utilisant la DDR3.



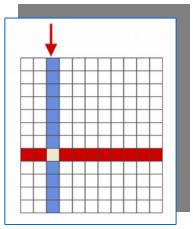
e) DDR3

J'ai voulu travailler avec de la DDR3. La raison est que développer une carte ou un code pour adresser de la DDR est aussi difficile que d'adresser de la DDR3. Du point de vue de la puissance, il me paraissait plus concret d'utiliser de la DDR3.

Timing DDR3

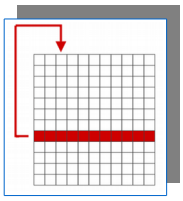
L'accès à un bit de mémoire se fait en plusieurs étapes. Chacune de ces étapes prend un certain temps. C'est pour ça qu'on parle de timing. Voici une liste des principaux :

- **CAS Latency** (tCL)



C'est le timing le plus important dans la mémoire RAM. CAS veut dire *Column Address Strobe*. Cette valeur nous renseigne sur le nombre de cycle d'horloge qu'il faut pour avoir un résultat une fois qu'une ligne est sélectionnée.

- **Row Address (RAS) to Column Address (CAS) Delay** (tRCD)

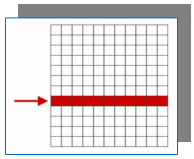


Une fois que l'on a envoyé au contrôleur de mémoire l'adresse d'une ligne, il faudra attendre tRCD cycle d'horloge pour accéder à l'une des colonnes. Si aucune des lignes n'a été sélectionnées, il faudra attendre tRCD+ tCL pour obtenir un résultat de la RAM

- **Row Precharge Time** (tRP)

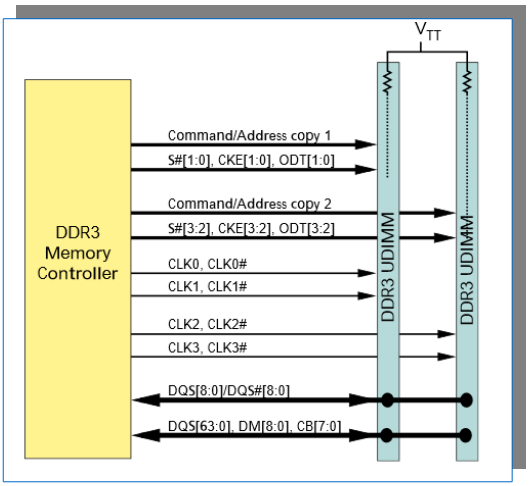
Si une ligne est déjà sélectionnée, il faudra attendre tRP nombre de cycle pour pouvoir sélectionner une autre ligne. Ce qui veut dire qu'il faut attendre tRP + tRCD + tCL pour accéder aux données d'une autre ligne.

- **Row Active Time** (tRAS)



C'est le nombre minimal de cycles qu'il faut attendre pour qu'une colonne devienne active et que l'on puisse accéder à la mémoire, on dit que cette valeur doit être égale à la somme des trois latences précédentes tRAS = tCL + tRCD + tRP .

Fly-By Architecture



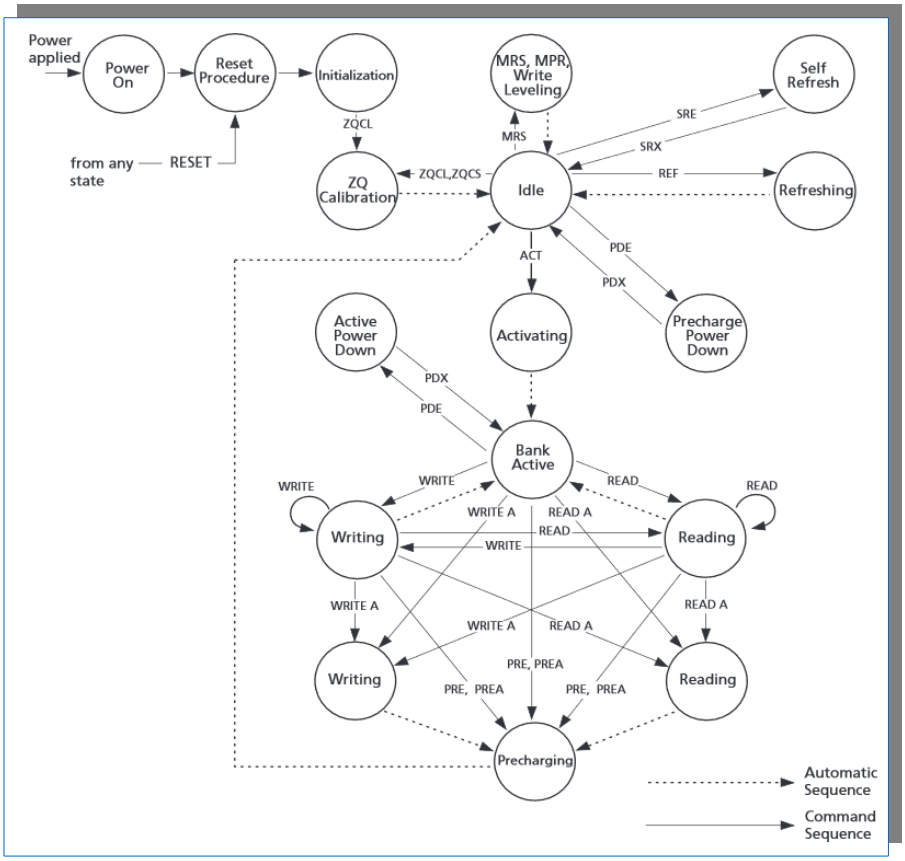
L'horloge de la DDR3 est plus rapide que les horloges des technologies DDR précédentes, rendant la qualité du signal encore plus importante. Pour améliorer la qualité du signal, l'horloge, la commande, les bus de données et d'adresses ont été routés en fly-by.

On peut diviser en quatre groupes les signaux qu'utilise la DDR3 pour fonctionner :

- Data group : Data strobe DQS[8:0], data strobe complement DQS#[8:0], data mask DM[8:0], data DQ[63:0], and check bits CB[7:0] (x72)
- Address and command group : Bank addresses BA[2:0]; addresses A[15:0]; and command inputs, including RAS#, CAS#, and WE#*
- Control group : Chip select S#[3:0], clock enable CKE[3:0], on-die termination ODT[3:0], and RESET#0
- Clock group : Differential clocks CK[3:0] and CK#[3:0]

Fonctionnement de DDR3

La DDR3 SDRAM est une mémoire vive à haute vitesse configurée comme huit banques de DRAM. La DDR3 SDRAM est utilisée comme une architecture dite 8n de pré extraction. Cette architecture de pré extraction est combinée avec une interface conçue pour transférer deux séries de données par front d'horloge sur les ports I/O.



Une opération de lecture ou d'écriture seule pour la DDRA SDRAM d'une donnée de 8n bits correspond à quatre fronts d'horloge pour le transfert de données jusqu'au cœur DRAM interne plus deux fois n (le nombre de fois 8bits des données) front d'horloge plus un demi cycle pour transférer jusqu'au I/O.

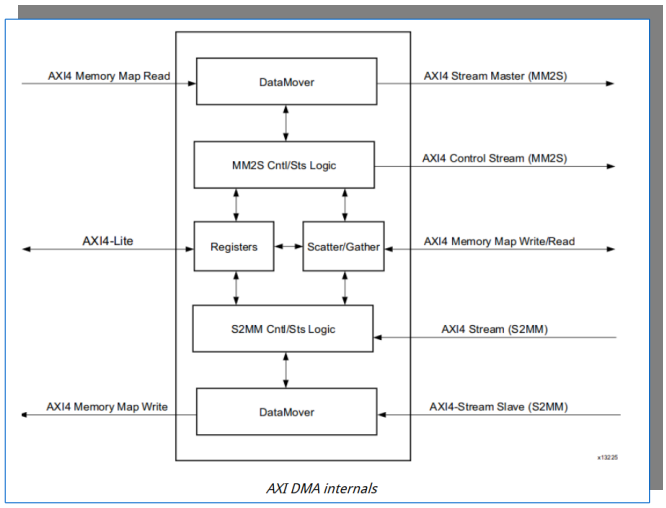
Une opération de lecture ou d'écriture commence par l'enregistrement d'une commande active qui est suivit par l'instruction de lecture ou d'écriture. L'adresse mémoire à laquelle on veut accéder dès le passage de la commande active est utilisée pour sélectionner la ligne et la colonne qui doit être activée (BA0-BA2 sélectionne la colonne et A0-A1 sélectionne la ligne voir Annexe D). Le registre d'adresse et l'opération de lecture et d'écriture doivent être à jour pour que la sélection de ligne et de colonne soit effective. De plus, l'opération de lecture ou d'écriture va permettre à ce moment-là de savoir si l'auto précharge est fonctionnelle ou non.

B) Réalisation

Dans cette partie, je vais aborder les quelques réalisations que j'ai effectué et qui ont apporté des résultats exploitables.

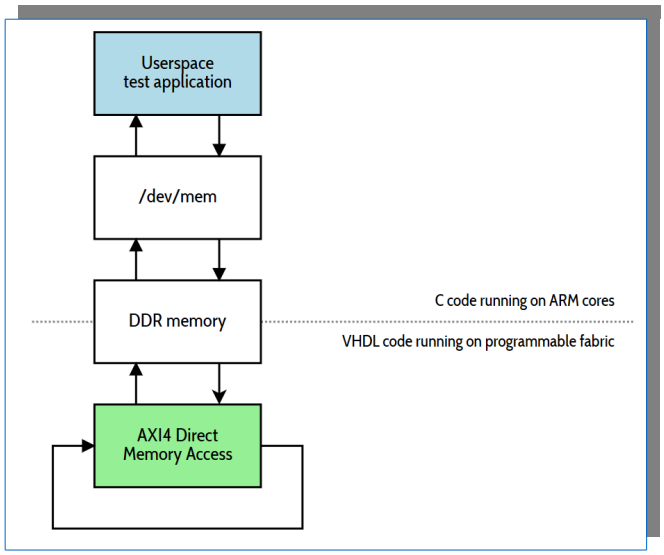
a) Direct Memory Acess

Tout d'abord, Xilinx marque une différence entre AXI DMA et AXI VDMA (*Video Direct Memory Acess*). AXI DMA fait réference à l'accès traditionnel à la mémoire DDR en transférant arbitrairement un flux de données depuis le FPGA vers la mémoire DDR. VDMA ajoute un mécanisme de stockage synchrone par frame en utilisant un ring buffer dans la DDR.



AXI DMA internals

AXI DMA distingue deux voies : MM2S (*Memory-Mapped to Stream*) qui transporte les données de la DDR au FPGA et S2MM (*Stream to Memory-Mapped*) qui transporte arbitrairement le flux de données à la DDR.

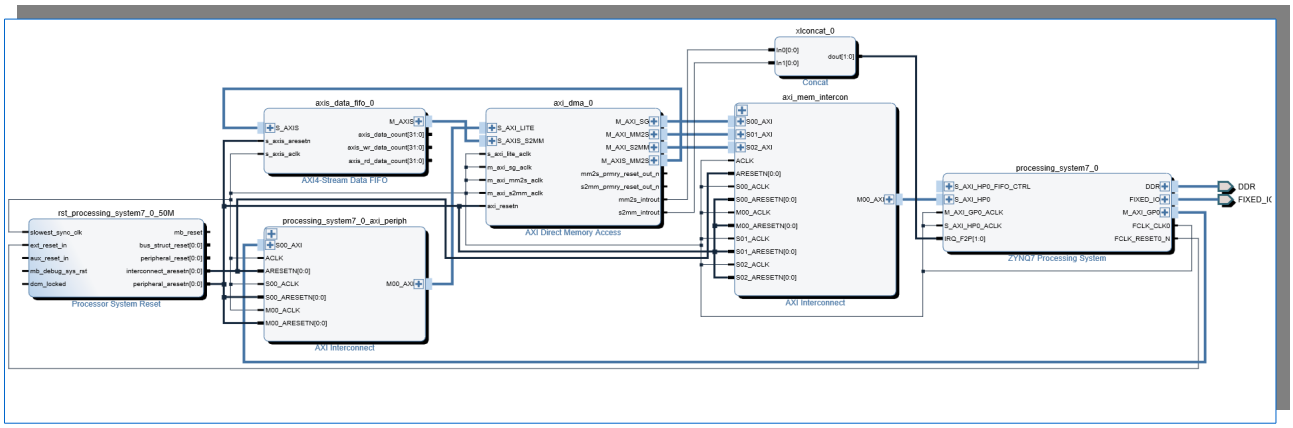


Utiliser la board Zybo permet de récupérer les données de manière avantageuse. Un simple script python permet d'accéder aux données et de les inscrire dans un fichier. L'avantage de l'architecture AXI DMA sur un ZYNQ 7000 est que l'architecture hybride facilite les choses. Permettant au FPGA de fonctionner avec les avantages de fonctionner en partie avec des lignes de commande. Ici la mémoire DDR va être partagée entre le processeur Cortex et le FPGA.

Dans le fichier RAR rendu en Annexe, vous trouverez un projet nommé AXI_DMA qui est le projet d'accès à la mémoire par DMA. Celui-ci fonctionne avec le SDK, il utilise un bloc FIFO pour le test. Il faut remplacer ce bloc par un bloc d'acquisition des GPIO. Pour accéder au debug, il faut exporter le bitstream qui est généré, générer un code en C/C++ et le lancer dans le processeur cortex. Pour l'instant, je n'arrive pas à debugger le programme que j'ai fait car je ne comprends pas

tous les éléments que me renvoie la liaison série.

Voici l'architecture du design que j'ai réalisé pour cette partie :



À droite on voit l'IP « processing_system7_0 » qui permet l'interaction entre les deux cœurs et le partage de certaines ressources comme les GPIOs et la DDR. Au centre on voit l'IP AXI_DMA qui est relié au bloc FIFO qu'il faut remplacer par un bloc GPIO que l'on peut trouver dans le projet « zybo_base_system.backup_bitstream_linux ». On voit aussi le bloc processing7_0_axi_periph qui permet un ordonnancement des périphériques entre les deux cœurs. Enfin, on trouve un bloc « xlconcat0 » qui permet d'utiliser le bloc DMA en mode « polling » grâce au déclenchement d'interruption sur le Zynq *Processing System* (PS).

Le mode « polling » consiste à faire attendre au PS que le FPGA lui dise de lire les données. C'est une méthode par interruption. Ainsi, il faut modifier l'IP de manière à ce que lorsque la mémoire DDR est pleine l'interruption se déclenche pour que le PS vienne inscrire les données dans un fichier.

Cette méthode peut fonctionner mais il faut bien maîtriser le flow de conception pour y arriver. Je conseille de commencer par la récupération et le stockage de données dans la DDR de la ZYBO.

b) Accès au GPIO par Xilinux

Voici une liste de matériel pour réaliser cette solution:

- Plateforme ZYBO
- Alimentation 5V 3A
- Carte microSD 4Gb (2Gb minimum)
- Cable RJ45
- Machine avec Linux type Debian

Voici les différentes étapes à réaliser pour récupérer et afficher les données venant des GPIOs:

Téléchargement et installation

a°)Télécharger et installer Vivado WebPack de la version la plus récente. (Xilinx)

B°) Télécharger depuis Zybo Resource Center le fichier zybo_base_system.zip.

C°) Télécharger depuis ce lien les ZYBO board files et placer le dossier source /new/zybo/ dans C:\Xilinx\Vivado\2015.4\data\boards\board_files

D°) Télécharger les fichiers xillinux-eval-zybo-1.3c.zip et n'en extraire que le dossier bootfiles dans le dossier de travail

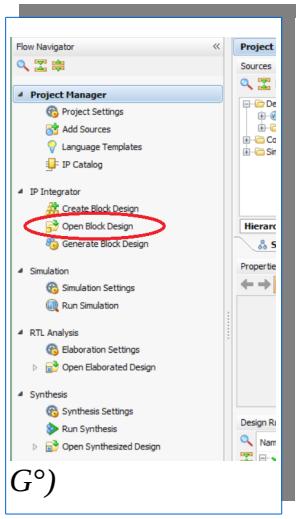
E°) Télécharger le fichier xillinux-1.3.img.gz et extraire son contenu dans le dossier de travail

Création d'une carte bootable sur ZYBO

Dans la suite du tutoriel, je vais présenter comment obtenir le fichier .bit qui permet d'obtenir un accès aux différents ports du FPGA comme les switchs, les leds, le HDMI etc. Si vous voulez prendre un fichier tout fait, je vous propose le mien qui est déjà compilé et fonctionnel. On le trouvera dans le dossier **Pack_bit** qui est dans l'archive RAR un imprim'écran qui montre les registres liés aux différentes interfaces. Ce fichier image est très important pour la partie "récupération" des données. Si vous utilisez ce fichier .zip, sautez directement au point P°).

F°) Ouvrir avec le logiciel Vivado zybo_base_system\source\vivado\hw\zybo_bsd\zybo_bsd.xpr

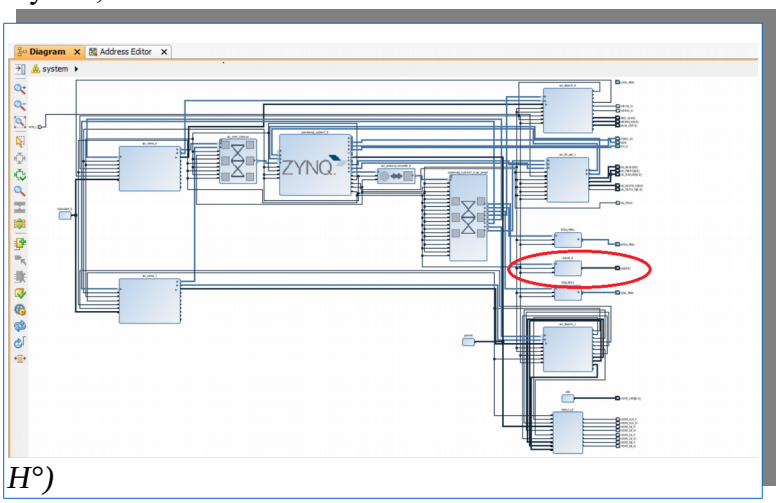
G°) Sur la gauche de l'écran, sous IP integrator, choisir Open Block Design



G°)

H°) Supprimer l'IP pour adresser les LEDS

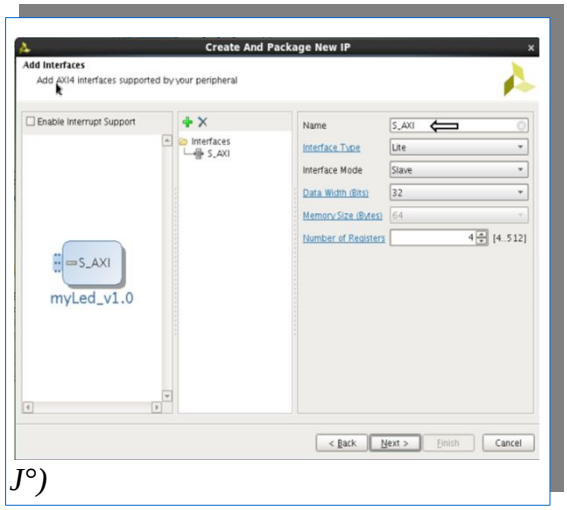
I°) On va créer un nouvel IP, tool->Create and Package IP...; next; name-> myLed; next



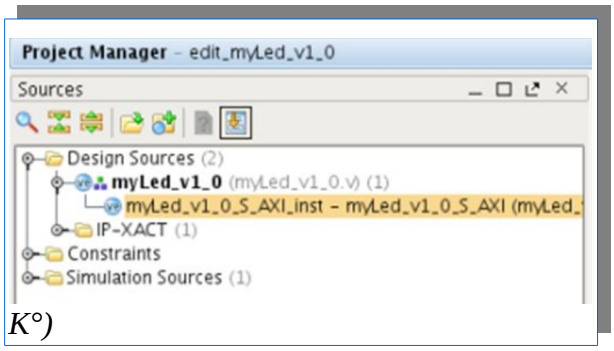
H°)

J°) La fenêtre suivante permet d'ajouter des interfaces. On va créer une interface AX14 pour le périphérique myLed.

- Interface type ->Lite
- Mode ->Slave data width ->32 bits
- number of registers ->4
- Name->S00_AXI



J°)



K°)

K°) La fenêtre suivante permet de choisir l'option "edit IP"

L°) Dans le project manager, ouvrir "myLed_V1..." et ajouter à la ligne 18 :

```
output wire [3:0] led,
```

et à la ligne 400 :

```
Assign led=slv_reg0[3:0];
```

J°) Maintenant dans le fichier myLed_v0, il faut ajouter à la ligne 18 :

```
output [3:0] led,
```

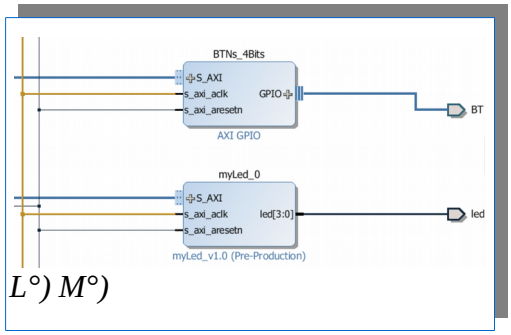
et à la ligne 51 :

```
.led(led),
```

K°) Dans l'onglet Package IP, cliquer su « Merge changes from Ip Custom », ainsi on obtient une IP qui a prit en compte les changements.

L°) Il faut maintenant ajouter l'IP au design. Cliquer sur Open Block Design, puis click droit → Add IP et choisir myLed0. Relier comme sur l'image le I/O de l'IP en cliquant sur « run connection automation ».

M°) Clique droit sur l'IP → create port, le nommer LED sur 4 bit et supprimer le précédent port. Relier le port de sortie à LED.



L°) M°)

N°) Dans les sources du Project manager, allez dans constraints, et ouvrir base.xdc et modifier comme sur l'image.

O°) Dans le flow navigator, cliquer sur create bitstream.

```
6
7 set_property PACKAGE_PIN M14 [get_ports {led[0]}]
8 set_property PACKAGE_PIN M15 [get_ports {led[1]}]
9 set_property PACKAGE_PIN G14 [get_ports {led[2]}]
10 set_property PACKAGE_PIN D18 [get_ports {led[3]}]
11 set_property IOSTANDARD LVCMOS33 [get_ports {led[*]}]
```
N°)

Préparation et utilisation de la carte µSD

P°) Dans notre dossier de travail, nous avons 4 fichiers :

- boot.bin
- devicetree.dtb
- uImage
- le fichier *.bit que l'on vient de générer et que l'on renomme xillydemo.bit

Q°) Monter la carte µSD dans le PC sous linux. Créer une partition de 36Mo en FAT16 et une partition qui prend tout le reste de la place en ext4. Personnellement après de nombreuse tentatives, je recommande GParted.

R°) Copier les quatre fichiers sur la partition de 36Mo.

S°) Éteindre la board avec le switch. Mettre le jumper de boot sur SD. Installer la carte SD dans le slot. Allumer la board.

Vous pouvez maintenant brancher un câble HDMI ou VGA à la board pour y connecter un écran, ainsi qu'un clavier pour faire la configuration et l'installation de Ubuntu. Le nom d'utilisateur est root et le mot de passe est toor. Le mieux est de faire un apt-get update, puis un apt-get upgrade pour mettre à jour les paquets. Il est maintenant nécessaire d'installer les paquets pour ssh si ce n'est pas déjà fait. Il est aussi nécessaire d'installer python.

Récupération des données des GPIOs

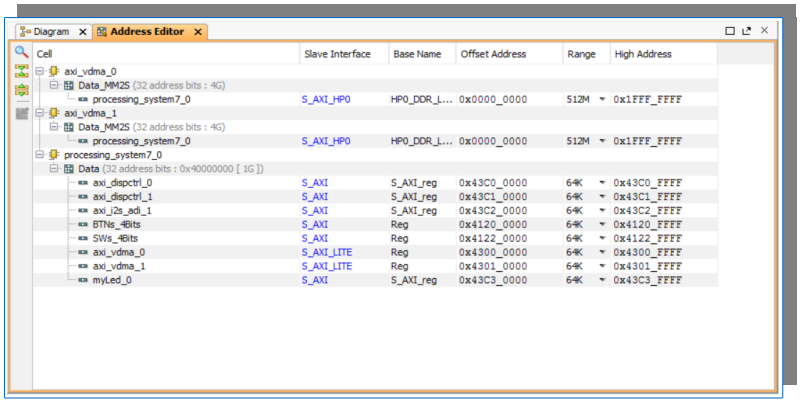
C'est là que le fichier d'adressage qui se situe dans **Pack_bit** si vous n'avez pas fait votre propre bitstream ou dans Vivado → Open Block Design → Adress Editor, va nous servir.

On peut utiliser nmap pour scanner son réseau et trouver l'adresse ip de la board ZYBO.

Une fois connecté, on va récupérer les données des switchs et écrire leur valeur dans un fichier. Voici un code en python qui récupère les données depuis des adresses mémoire dans /dev/mem. Le code est assez simple et efficace. L'astuce consiste à mettre en relation les offset que l'on voit avec les adresses que l'on a attribué avec l'Adress Editor dans Vivado :

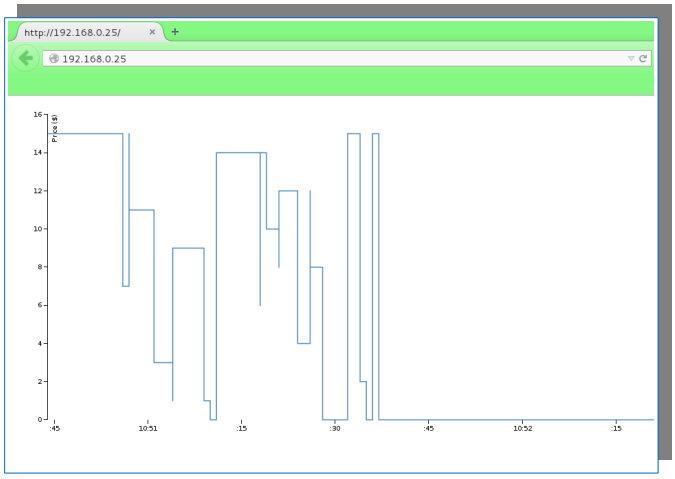
```python
from datetime import datetime
import time
import mmap
import sys

with open("/dev/mem", "r+b") as f:
    leds = mmap.mmap(f.fileno(), 4, offset=0x43C30000)
    switches = mmap.mmap(f.fileno(), 4, offset=0x41220000)
    f = open("/var/www/data.tsv", "w")
    f.write("date\tclose\n")
    while True:
            try:
            now=datetime.now()
            leds[0] = switches[0]
            f.write("%d-%02d-%02dT%02d:%02d:%02d\t%d\n" % (now.year, now.month, now.day, now.hour, now.minute, now.second, ord(switches[0])))
            except KeyboardInterrupt:
                    break
    leds.close()
    switches.close()
```

On voit que les offset des LED et des SWITCH correspondent.



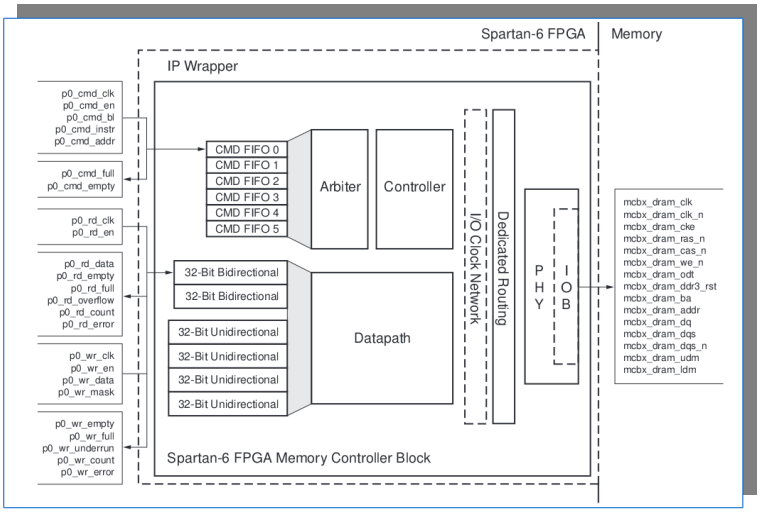
J'ai ensuite crée un serveur apache sur le linux du ZYBO et j'ai affiché les données que j'ai collécté sur un graphique que je peux atteindre en tapant l'adresse ip de la machine.

J'ai pu mesurer la vitesse d'acquisition des données, celle ci n'est pas très élevé. On obtient 180 Ko/s. Ce pendant cette méthode est utile si on veut récupérer les données qui sont dans la RAM. En effet, j'ai pu tester des codes en C qui permettent de récupérer directement les données qui sont dans la RAM à partir du linux embarqué.



c) Memory Controller Block

Si l'on travaille avec un FPGA Spartan 6, on peut utiliser les MCB. Ce sont des blocs qui sont préparé pour adresser facilement des mémoires.



Il y a trois fonctions principales du point de vue de l'utilisateur:

Lire le port (unidirectionnel)

Écrire sur le port(unidirectionnel)

Lire et écrire sur le port (bi directionnel)

On peut voir la représentation du fonctionnement d'un Memory Controller Block plusieurs blocks que l'on définit ainsi:

Arbiter ou arbitre

Il détermine quel port à la priorité pour accéder à la mémoire physique.

Controller ou contrôleur

Block de contrôle primaire qui convertit de simples requêtes utilisateur en de plus complexes instructions et séquences pour commmuniquer avec la mémoire.

Datapath

Ce block porte le flux de données écrite et lue entre la mémoire physique et l'utilisateur.

Physical Interface (PHY) ou interface physique

Convertit les instructions du contrôleur en des relations temporelles et signaux adapté à la mémoire DDR pour communiquer avec elle.

Calibration Logic

Clalibre l'interface physique (PHY) pour avoir le meilleur rendement.

Il existe un outil wizard sous Vivado pour réaliser des IP avec ce genre de block. J'en ais réalisé plusieurs, mais je n'ai jamais réussi à les faire fonctionner avec la board ZYBO.

V. Analyse, ressentis du projet

A) Méthodologie

Le choix matériel et logiciel a été une source de perte de temps énorme. Je sais maintenant avec le recul, qu'il aurait fallut travailler directement avec Vivado qui est un logiciel bien plus complet encore que l'ISE. Ce logiciel à une logique complète, que je pourrais comparer à Altium Designer (pour ce que j'en connais), d'utilisation d'IP. J'ai l'impression que le flow de conception est plus complexe, mais que les IP disponibles sur ce logiciel sont plus faciles d'accès que sur l'ISE. Au niveau de la board, je n'aurais pas du tenter de travailler avec la TE600. C'est une board professionnelle qui peut éventuellement servir contrôler une carte de DDR fille, (quoique le nombre de pins me semble insuffisant), mais tout en restant au stade de carte de développement sans options. De plus il est difficile de trouver des ressources préétablies comme des tutoriels ou des démos. Même trouver les fichiers de contraintes sur internet est difficile. Par ailleurs, j'ai pris du temps pour installer un système linux sur la ZYBO, et c'était une bonne option. En effet, le fait de pouvoir se connecter en ssh permet une autre approche notamment pour le débogage. Pour ceux qui ne maîtrise pas le FPGA comme un PC, c'est une option intéressante pour progresser. Je pense que le ZYNQ 7000 est un FPGA qui a un bon avenir devant lui pour beaucoup d'application temps réelle-système embarqué. Avec cette technologie, on peut notamment traiter des données dont les tailles serai différente de 32 ou 64 bits tout en restant temps réel, je pense à de la cryptographie par exemple.

Ma source de renseignement à principalement été internet. J'ai utilisé de nombreux forums comme *forums.xilinx.com, forum.digilentinc.com, fpgadeveloper.com.* Ce sont des pages qui sont intéressantes lorsque l'on a une erreur spécifique, par exemple une compilation qui ne fonctionne pas ou encore une configuration logiciel qui n'est pas comme il faudrait. Cependant, j'ai eu des difficultés à trouver des solutions concernant la méthode qu'il fallait que j'applique pour coder, et debugger mes programmes. En effet que ce soit sous Vivado ou ISE, le flow de conception est en grande partie graphique (encore plus à l'avenir ça le sera pour le FPGA). Ainsi, j'ai senti un grand vide de compréhension en comparaison à l'habituel codage en C sur PC, où l'on peut s'aider de printf, ou de leds pour un programme sur microcontrôleur. Sur FPGA, je me suis retrouvé directement avec des outils qui sont capables de gérer tous les niveaux d'abstractions en partant du VHDL jusqu'au C++. C'est quelque chose qui m'a déstabilisé par rapport à ce que je connaissais, puisqu'il me manquait souvent l'étape suivante de conception pour connaître ce que je devais produire. Souvent j'avançais laborieusement, et après plusieurs heures j'arrivais à sortir d'impasse qui n'en était pas. Ce genre de projet nécessite l'avis d'experts sur la question, j'ai pu m'entretenir brièvement avec un ingénieur hardware que je connaissais il y a quelques temps. Son avis m'a fait progresser instantanément, supprimant une arborescence de solutions fausses et m'apprenant

concrètement la voie à suivre et pourquoi.

Ce genre d'expérience a confirmé ce que je savais déjà sur la forme d'apprentissage qui me correspond le plus. Dans beaucoup de projets, l'apprentissage s'est fait pour moi par mimétisme et via la compréhension au moins partielle du processus. Soit grâce à l'aide de professeurs soit du groupe de travail. J'ai réussi à contrer ce manque en m'appliquant à une autre méthode, dès que je réussissait à effectuer une opération, je créais une sauvegarde et je recommençais l'opération jusqu'à obtenir le même résultat.

Le sujet en question était difficile à traiter, dans le sens ou je ne savais pas comment présenter mon travail. C'est vraiment un travail de recherches théorique autant que pratique. Il y a beaucoup de choses qui n'était pas clair pour que je puisse développer mon travail. Cependant, je pense avoir appris énormément sur les mémoires et le FPGA. Ce sont deux sujets qui m'intéressent et qui m'ont permis de mettre en relation un bagage technique et un bagage théorique.

B) Analyse du travail

Je l'expliquais plus haut, le choix du logiciel et de la plate-forme m'ont retardé dans mon travail pratique. Pourtant, la recherche quotidienne d'informations n'a fait que progresser tout au long du projet. Sans ça il aurait été difficile de réaliser un travail de programmation. Je ne regrette pas mon choix concernant le choix d'un sujet de recherche. La réussite de certains points apporte une réelle satisfaction lorsque l'on ne voit plus flou à travers les étapes. Je pense que, mon travail peut être perçu comme un débroussaillement méthodique qui permet d'appréhender un tel sujet.

C) Poursuite des travaux

Je pense que le projet aurait pu fonctionner si j'avais eu plus de temps ou si j'avais entrevu la bonne solution plus tôt. Ce projet peut être proposé dans les années suivantes à partir de la base que j'ai fourni. J'ai finalement bien apprécié travailler en autonomie sur la carte ZYBO que je maîtrise mieux. J'aimerai à l'avenir investir dans cette plate-forme qui n'est pas trop cher pour réaliser des projet personnels et finir d'accéder à la RAM car je ne pense pas être loin de réussir.

VI. Conclusion

Pendant toute la durée du PFE, j'ai pu m'intéresser au sujet de la mémoire SDRAM et aux FPGA. J'ai eu l'occasion de comparer différentes techniques de développement et de tester de nombreux programmes. J'ai pu faire des recherches approfondis sur le concept de mémoire pour pouvoir mettre en place une structure de réflexion autours de mon sujet. J'ai eu l'occasion de parler avec un ingénieur hardware en personne du sujet et de communiquer sur des forums pour trouver des solutions. J'ai eu la chance d'effectuer des recherches sur des sujets autant pratiques que théoriques ce qui m'a permis de mettre en relation les deux types de savoir. J'ai pu me familiarisé avec une board FPGA tel que la ZYBO, ce qui m'a donné l'envie de continuer, et d'investir dans ce type de plate-forme. J'aimerai à l'avenir que ce sujet soit poursuivit pour aboutir à sa réussite totale, pas seulement pour les besoins du laboratoire, mais aussi pour que les possibilités d'une telle fonctionnalités servent à d'autres projets.

VII. Sources

http://blog.elphel.com/2015/04/fpga-to-ddr3-memory-interface-step-by-step-timing-calibration-and-set-up/

http://www.hardwaresecrets.com/understanding-ram-timings/

http://mermaja.act.uji.es/docencia/is37/data/DDR3.pdf

http://lauri.võsandi.com/hdl/zynq/xilinx-dma.html

http://www.fpgadeveloper.com/2014/08/using-the-axi-dma-in-vivado.html

http://forum.trenz-electronic.de/index.php?topic=262.0

http://xillybus.com/downloads/doc/xillybus_getting_started_zynq.pdf

VIII. Annexe A : Cartes utilisées

Les cartes utilisées durant ce projet sont les suivantes :

Trenz Electronic TE0600 Series Spartan 6 :

http://www.trenz-electronic.de/de/produkte/fpga-boards/trenz-electronic/te0600.html

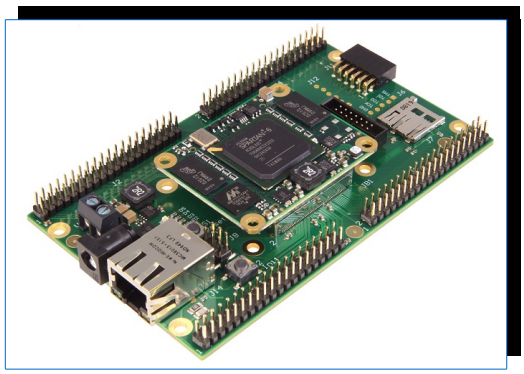
ZYBO Zynq-7000 Development Board :

https://www.digilentinc.com/Products/Detail.cfm?Prod=ZYBO

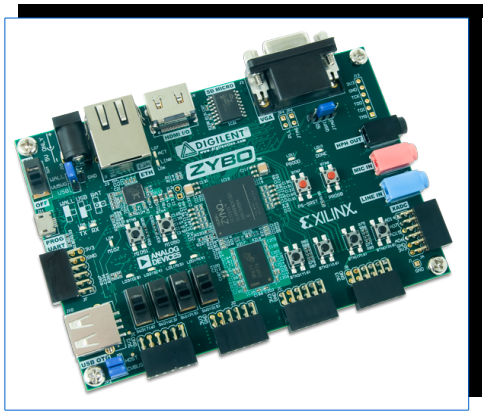
Xilinx Spartan-6 FPGA LX9 MicroBoard :

http://www.em.avnet.com/en-us/design/drc/Pages/Xilinx-Spartan-6-FPGA-LX9-MicroBoard.aspx

Photos des cartes :



TE0600 Series Spartan 6



ZYBO Zynq-7000

Spartan-6 FPGA LX9 MicroBoard

IX. Annexe B : Fichiers UCF des différentes boards (ISE uniquement)

TRENZ Electronic TE0600 :

https://github.com/Trenz-Electronic/TE060X-GigaBee-Reference-Designs/blob/master/Generic/TE0600.ucf

DILIGENT ZYBO :

https://www.digilentinc.com/Products/Detail.cfm?Prod=ZYBO

AVNET MicroBoard :

http://www.em.avnet.com/Support%20And%20Downloads/xlx_s9_lx9_fpga_microboard-ug072711.pdf

X.Annexe B bis : Fichiers XDC des différentes boards (Vivado uniquement)

TRENZ Electronic TE0600 :

http://www.trenz-electronic.de/download/d0/Trenz_Electronic/d1/TE0600-GigaBee_series/d2/TE0600/d3/reference_designs.html

DILIGENT ZYBO :

https://reference.digilentinc.com/_media/zybo:zybo_master_xdc.zip

AVNET MicroBoard :

https://www.em.avnet.com/en-us/design/drc/pages/supportanddownloads.aspx?RelatedId=90

XI. Annexe C : Caractéristiques des DDR 1234 SDRAM

Norme	Norme JEDEC	Fréquence Mémoire	Fréquence d'entrée/sortie	Bande Passante	Nombre de pins	Tension d'alimentation
DDR200	PC1600	100 MHz	100 MHz	1,6 Go/s	184	2.5 V
DDR266	PC2100	133 MHz	133 MHz	2,1 Go/s	184	2.5 V
DDR333	PC2700	166 MHz	166 MHz	2,7 Go/s	184	2.5 V
DDR400	PC3200	200 MHz	200 MHz	3,2 Go/s	184	2.5 V
DDR433	PC3500	217 MHz	217 MHz	3,5 Go/s	184	2.5 V
DDR466	PC3700	233 MHz	233 MHz	3,7 Go/s	184	2.5 V
DDR500	PC4000	250 MHz	250 MHz	4 Go/s	184	2.5 V
DDR533	PC4200	266 MHz	266 MHz	4,2 Go/s	184	2.5 V
DDR538	PC4300	269 MHz	269 MHz	4,3 Go/s	184	2.5 V
DDR550	PC4400	275 MHz	275 MHz	4,4 Go/s	184	2.5 V
DDR2-400	PC2-3200	100 MHz	200 MHz	3,2 Go/s	240	1.8 V
DDR2-533	PC2-4300	133 MHz	266 MHz	4,3 Go/s	240	1.8 V
DDR2-667	PC2-5300	166 MHz	333 MHz	5,3 Go/s	240	1.8 V
DDR2-675	PC2-5400	168 MHz	337 MHz	5,4 Go/s	240	1.8 V
DDR2-800	PC2-6400	200 MHz	400 MHz	6,4 Go/s	240	1.8 V
DDR2-1066	PC2-8500	266 MHz	533 MHz	8,5 Go/s	240	1.8 V
DDR2-1100	PC2-8800	280 MHz	560 MHz	8,8 Go/s	240	1.8 V
DDR2-1200	PC2-9600	300 MHz	600 MHz	9,6 Go/s	240	1.8 V

Norme	Norme JEDEC	Fréquence Mémoire	Fréquence d'entrée/sortie	Bande Passante	Nombre de pins (DIMM)	Tension d'alimentation
DDR3-800	PC3-6400	100 MHz	400 MHz	6,4 Go/s	240	1.5 V
DDR3-1066	PC3-8500	133 MHz	533 MHz	8,5 Go/s	240	1.5 V
DDR3-1333	PC3-10600	166 MHz	666 MHz	10,7 Go/s	240	1.5 V
DDR3-1600	PC3-12800	200 MHz	800 MHz	12,8 Go/s	240	1.5 V
DDR3-1800	PC3-14400	225 MHz	900 MHz	14,4 Go/s	240	1.5 V
DDR3-2000	PC3-16000	250 MHz	1000 MHz	16 Go/s	240	1.5 V
DDR3-2133	PC3-17000	266 MHz	1066 MHz	17 Go/s	240	1.5 V
DDR4-1600	---	---	800 MHz	12,8 Go/s	288	1.2 V
DDR4-1866	---	---	933 MHz	14,9 Go/s	288	1.2 V
DDR4-2133	---	---	1066 MHz	17 Go/s	288	1.2 V
DDR4-2400	---	---	1200 MHz	19,2 Go/s	288	1.2 V

Plus d'info sur la DDR4 :
http://www.kingston.com/fr/memory/ddr4
https://www.jedec.org/standards-documents/results/jesd79-4%20ddr4

XII. Annexe C : Assignation des Pins DDR3

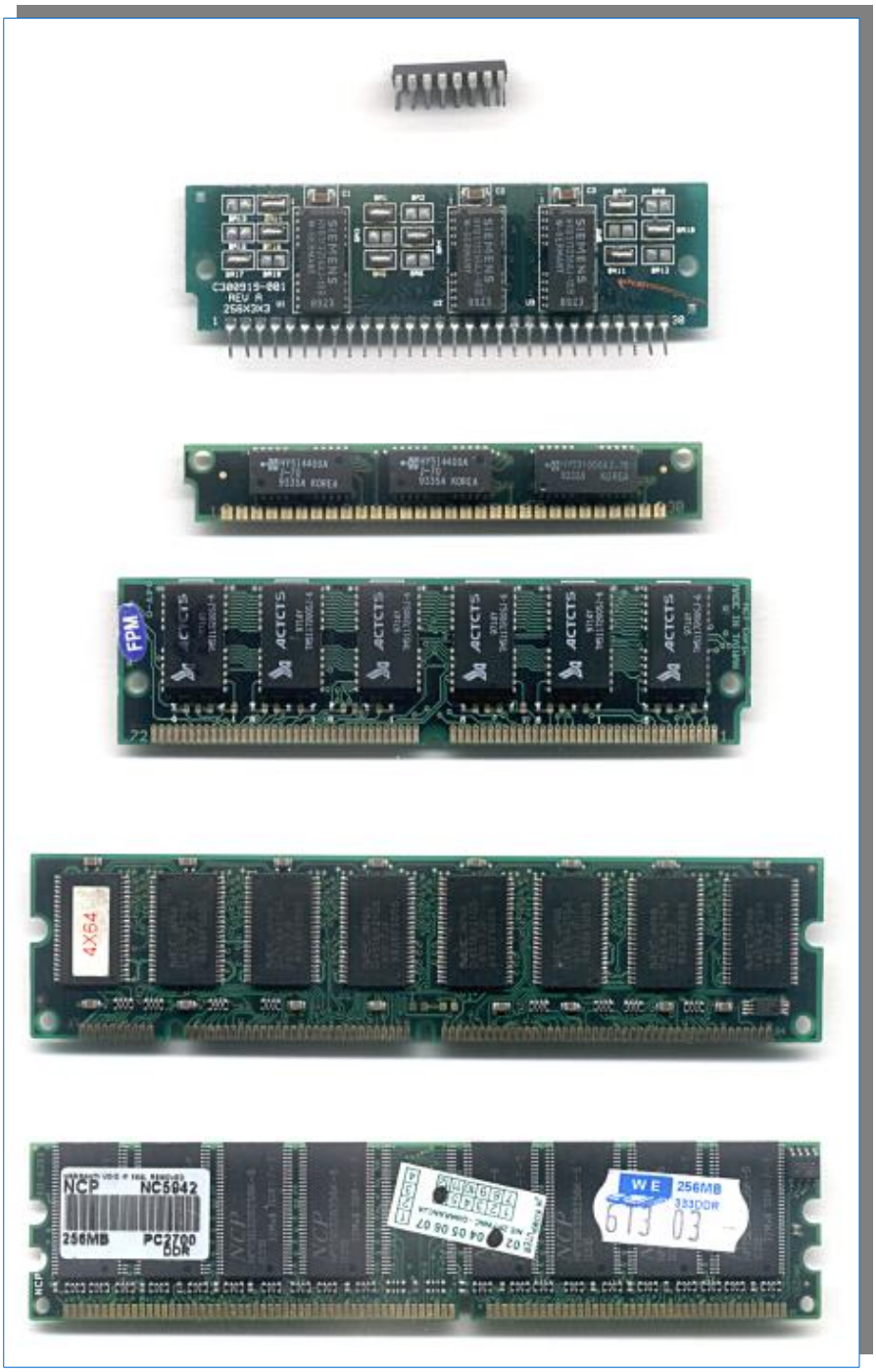
240-Pin DDR3 ECC DIMM Front							
Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
1	Vrefdq	31	DQ25	61	A2	91	DQ41
2	Vss	32	Vss	62	Vdd	92	Vss
3	DQ0	33	DQS3#	63	CK1/NC	93	DQS5#
4	DQ1	34	DQS3	64	CK1#/NC	94	DQS5
5	Vss	35	Vss	65	Vdd	95	Vss
6	DQS0#	36	DQ26	66	Vdd	96	DQ42
7	DQS0	37	DQ27	67	Vrefca	97	DQ43
8	Vss	38	Vss	68	NC	98	Vss
9	DQ2	39	CB0	69	Vdd	99	DQ48
10	DQ3	40	CB1	70	A10	100	DQ49
11	Vss	41	Vss	71	BA0	101	Vss
12	DQ8	42	DQS8#	72	Vdd	102	DQS6#
13	DQ9	43	DQS8	73	WE#	103	DQS6
14	Vss	44	Vss	74	CAS#	104	Vss
15	DQS1#	45	CB2	75	Vdd	105	DQ50
16	DQS1	46	CB3	76	S1#	106	DQ51
17	Vss	47	Vss	77	ODT1	107	Vss
18	DQ10	48	NC	78	Vdd	108	DQ56
19	DQ11	49	NC	79	NC	109	DQ57
20	Vss	50	CKE0	80	Vss	110	Vss
21	DQ16	51	Vdd	81	DQ32	111	DQS7#
22	DQ17	52	BA2	82	DQ33	112	DQS7
23	Vss	53	NC	83	Vss	113	Vss
24	DQS2#	54	Vdd	84	DQS4#	114	DQ58
25	DQS2	55	A11	85	DQS4	115	DQ59
26	Vss	56	A7	86	Vss	116	Vss
27	DQ18	57	Vdd	87	DQ34	117	SA0
28	DQ19	58	A5	88	DQ35	118	SCL
29	Vss	59	A4	89	Vss	119	SA2
30	DQ24	60	Vdd	90	DQ40	120	Vtt

240-Pin DDR3 ECC DIMM Back							
Pin	Symbol	Pin	Symbol	Pin	Symbol	Pin	Symbol
121	Vss	151	Vss	181	A1	211	Vss
122	DQ4	152	DM3	182	Vdd	212	DM5
123	DQ5	153	NC	183	Vdd	213	NC
124	Vss	154	Vss	184	CK0	214	Vss
125	DM0	155	DQ30	185	CK0#	215	DQ46
126	NC	156	DQ31	186	Vdd	216	DQ47
127	Vss	157	Vss	187	EVENT#	217	VSS
128	DQ6	158	CB4	188	A0	218	DQ52
129	DQ7	159	CB5	189	Vdd	219	DQ53
130	Vss	160	Vss	190	BA1	220	Vss
131	DQ12	161	DM8	191	Vdd	221	DM6
132	DQ13	162	NC	192	RAS#	222	NC
133	Vss	163	Vss	193	S0#	223	Vss
134	DM1	164	CB6	194	Vdd	224	DQ54
135	NC	165	CB7	195	ODT0	225	DQ55
136	Vss	166	Vss	196	A13	226	Vss
137	DQ14	167	NC	197	Vdd	227	DQ60
138	DQ15	168	RESET#	198	NC	228	DQ61
139	Vss	169	CKE1	199	Vss	229	Vss
140	DQ20	170	Vdd	200	DQ36	230	DM7
141	DQ21	171	A15	201	DQ37	231	NC
142	Vss	172	A14	202	Vss	232	Vss
143	DM2	173	Vdd	203	DM4	233	DQ62
144	NC	174	A12	204	NC	234	DQ63
145	Vss	175	A9	205	Vss	235	Vss
146	DQ22	176	Vdd	206	DQ38	236	Vddspd
147	DQ23	177	A8	207	DQ39	237	SA1
148	Vss	178	A6	208	Vss	238	SDA
149	DQ28	179	Vdd	209	DQ44	239	Vss
150	DQ29	180	A3	210	DQ45	240	Vtt

XIII. Annexe D : Pin Description DDR3

Symbol	Type	Description
A[15:0]	Input	**Address inputs:** Provide the row address for ACTIVATE commands, and the column address and auto precharge bit (A10) for READ/WRITE commands, to select one location out of the memory array in the respective bank. A10 is sampled during a PRECHARGE command to determine whether the PRECHARGE applies to one bank (A10 LOW, bank selected by BA[2:0]) or all banks (A10 HIGH). If only one bank is to be precharged, the bank is selected by BA. A12 is also used for BC4/BL8 identification as "BL on-the-fly" during CAS commands. The address inputs also provide the op-code during the mode register command set. A[13:0] address the 1Gb DDR3 devices. A[15:14] are needed to calculate parity on the command/address bus.
BA[2:0]	Input	**Bank address inputs:** BA[2:0] define the device bank to which an ACTIVATE, READ, WRITE, or PRECHARGE command is being applied. BA[2:0] define which mode register (MR0, MR1, MR2, and MR3) is loaded during the LOAD MODE command. BA[2:0] are used as part of the parity calculation.
CK0, CK0#	Input	**Clock:** CK and CK# are differential clock inputs. All control, command, and address input signals are sampled on the crossing of the positive edge of CK and the negative edge of CK#.
CKE[1:0]	Input	**Clock enable:** CKE enables (registered HIGH) and disables (registered LOW) internal circuitry and clocks on the DRAM.
DM[8:0] (TDQS[17:9], TDQS#[17:9])	Input	**Input data mask:** DM is an input mask signal for write data. Input data is masked when DM is sampled HIGH, along with the input data, during a write access. DM is sampled on both edges of the DQS. Although the DM pins are input-only, the DM loading is designed to match that of the DQ and DQS pins. When TDQS is enabled, DM is disabled and TDQS and TDQS# provide termination resistance, otherwise the TDQS# pins are no function.
ODT[1:0]	Input	**On-die termination:** ODT enables (registered HIGH) and disables (registered LOW) termination resistance internal to the DRAM. When enabled in normal operation, ODT is applied only to the following pins: DQ, DQS, DQS#, and DM. The ODT input will be ignored if disabled via the LOAD MODE command.
Par_In	Input	**Parity input:** Parity bit for the address, RAS#, CAS#, and WE#.
RAS#, CAS#, WE#	Input	**Command inputs:** RAS#, CAS#, and WE# (along with S#) define the command being entered.
RESET#	Input (LVCMOS)	**Reset:** RESET# is an active LOW CMOS input referenced to Vss. The RESET# input receiver is a CMOS input defined as a rail-to-rail signal with DC HIGH ≥ 0.8 × Vdd and DC LOW ≤ 0.2 × Vdd.
S#[1:0]	Input	**Chip select:** S# enables (registered LOW) and disables (registered HIGH) the command decoder.
SA[2:0]	Input	**Serial address inputs:** These pins are used to configure the temperature sensor/SPD EEPROM address range on the I2C bus.
SCL	Input	**Serial clock for temperature sensor/SPD EEPROM:** SCL is used to synchronize communication to and from the temperature sensor/SPD EEPROM.
CB[7:0]	I/O	**Check bits:** Data used for ECC.
DQ[63:0]	I/O	**Data input/output:** Bidirectional data bus.
DQS[8:0], DQS#[8:0]	I/O	**Data strobe:** DQS and DQS# are differential data strobes. Output with read data. Edge-aligned with read data. Input with write data. Center-aligned with write data. DQS# is used only when the differential data strobe mode is enabled via the LOAD MODE command.
SDA	I/O	**Serial data:** SDA is a bidirectional pin used to transfer addresses and data into and out of the temperature sensor/SPD EEPROM on the module on the I2C bus.
Err_Out#	Output (open-drain)	**Parity error output:** Parity error found on the command and address bus.
EVENT#	Output (open-drain)	**Temperature event:** The EVENT# pin is asserted by the temperature sensor when critical temperature thresholds have been exceeded.
Vdd	Supply	**Power supply:** 1.5V ±0.075V. The component Vdd and Vddq are connected to the module Vdd.
Vddspd	Supply	**Temperature sensor/SPD EEPROM power supply:** +3.0V to +3.6V.
Vrefca	Supply	**Reference voltage:** Control, command, and address (Vdd/2).
Vrefdq	Supply	**Reference voltage:** DQ, DM (Vdd/2).
Vss	Supply	Ground.
Vtt	Supply	**Termination voltage:** Used for control, command, and address (Vdd/2).
NC	–	**No connect:** These pins are not connected on the module.
NU	–	**Not used:** These pins are not used in specific module configuration/operations.

XIV. Annexe E : Packages DDR3



De haut en bas : DIP, SIPP, SIMM 30 broches, SIMM 72 broches, DIMM (168 broches), DDR DIMM (184 broches).