Université Lille 1

Département Informatique Micro-éléctronique et Automatique, 5éme année.

Composant d'audit des accès cache mémoire sur un microprocesseur LEON3 simulé en F.P.G.A., rapport de mi-soutenance de PFE

Auteur :

Jérôme VAESSEN

Encadrants école :

Julien Cartigny

Pierrick Buret

Table des matières

Chapitre 1

Remerciements

Je tiens à remercier l'ensemble du personnel du laboratoire de l'Ircica, ainsi les deux encadrants-école qui m'ont accueilli au sein de l'équipe 2XS, à savoir Pierrick Buret et Julien Cartigny.

Chapitre 2

Introduction

Étant en 5e année au sein de l'école Polytech Lille, dans le département I.M.A. (Informatique Microélétronique et Automatique), nous avons eu à faire un choix entre plusieurs projets pour le semestre S9.

C'est ainsi que nous avons choisi de travailler sur la problématique qui suit ; en système temps réel, il arrive que certaine tâche consomme trop d'accès mémoire et ainsi met en défaut l'exécution ou le bon déroulement d'autre tâche.

C'est ainsi qu'il nous a était demande de faire un composant capable de suivre les accès mémoire d'un processeur (LEON3 simulé sur un F.P.G.A..

Chapitre 3

Contexte général

Dans le cadre des systèmes embarqués, plus particulièrement les systèmes temps réel, il arrive que certaine tâche, soit critique. Prenons l'exemple d'une régulation pour le contrôle d'un réacteur de fusée, qui doit être faite dans un temps imparti, celle-ci peu allez jusqu'à une explosion ou même un incendie, dans le cas ou le systéme n'est plus capable de répondre dans un temps bornée.

Or ce genre d'incident pourrait se reproduire non pas de manière directe, mais si le système d'un point de vue, électronique n'est pas capable de satisfaire toutes les demandes d'accès en lecture et en écriture à la mémoire d'un ou plusieurs processeurs. Il est possible qu'une tâche prenne temporairement trop de ressources et ainsi perturbe d'autres tâches.

C'est ainsi que le P.F.E. (Projet de Fin d'Études) vient s'inscrire, ainsi que le sujet proposé était le suivant :

Certains processeurs sont aujourd'hui simulés sur FPGA avant d'être produits. Lors d'un audit de sécurité d'un processeur, des faiblesses ont été observées sur l'accès à la mémoire et aux caches.

L'objectif de ce projet se décompose en 2 parties : l'identification du problème et la localisation de la partie défaillante du processeur (repérage des lignes de code VHDL incriminées). La modification du code VHDL afin d'intégrer un nouveau composant innovant réalisant la sécurité de cette partie du processeur. Un bon niveau de VHDL est nécessaire ainsi qu'une autonomie et une prédisposition (motivation) à la recherche de problèmes de sécurité sur systèmes.

Chapitre 4

Cahier des charges

L'architecture du Leon 3 est visible sur le schéma suivant :

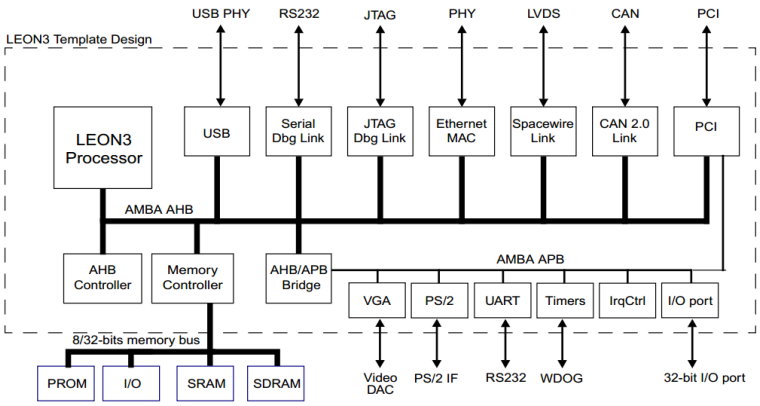

Schéma du bus AHB avec les différents composants

Ce processeur utilise un bus AMBA [1] notamment un bus AHB [2]. Le composant sera implanté dans le contrôleur mémoire. Notre composant étant destiné à compter le nombre d'accès mémoire pour chaque type de mémoire voir figure 4.1, certaines fonctions y sont déjà implémenter.

Notre composant s'appuie sur le même principe de fonctionnement que celui de l'arbitre de bus.

Les contraintes qui ont était établi sont :

Le composant

– doit pouvoir être adressable par le processeur à une adresse défini auparavant

– doit pouvoir être contrôlé ou forcé à un état [3]

– doit être réglable pour choisir le composant surveillé

– doit pouvoir faire la différence entre un accés en lecture ou écriture

Si le temps le permet, le composant devra permettre d'envoyer des signaux d'interruptions à un ou plusieurs processeurs.

1. Lien Amba v2 : `https://silver.arm.com/download/download.tm?pv=1062760`
2. Advanced High-performance Bus
3. l'accès en lecture ou en écriture par le processeur à l'adresse du composant, donc il doit être adressable

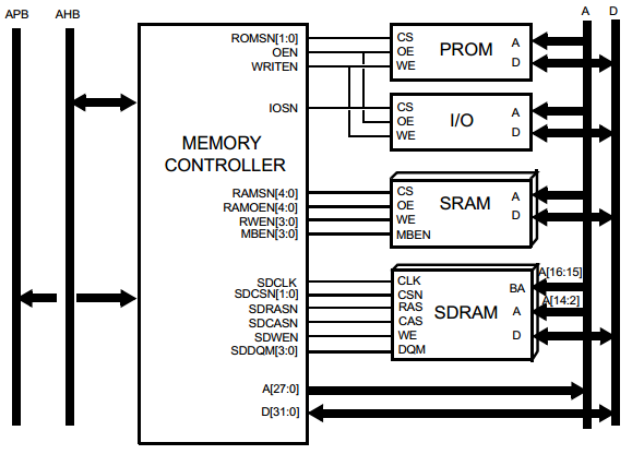

FIGURE 4.1 – Composants connectés au contrôleur mémoire

Chapitre 5

Avancés réalisées

Les sources de la GRLIB/LEON3 sont disponibles sur le site de Gaisler/Aeroflex. Nous avons mis en place un environnement de développement et de test. Les outils fournis par Gaisler/Aeroflex sont utilisables avec des environnements disposant de quelques utilitaires connus sous Linux (Unix). Nous avons mis en place MSYS [1] afin de disposer de l'utilitaire Make et aussi un environnement TCL/TK [2].

Nous utilisons l'environnement de développement ISE Xilinx Vivado car nous développons en VHDL afin qu'il soit testé sur une carte gr-pci-xc5v. Grmon (fournit par Gaisler/Aeroflex) permet de réaliser une communication série entre l'ordinateur et un core leon3 implanté sur le F.P.G.A. de la carte. Celui-ci permet d'accéder à des fonctionnalités de débogage [3].

Pour réaliser des fichiers binaires pouvant être exécutés sur le leon3, on utilise un cross-compiler (BCC) [4]

5.1 Configuration du leon3

Pour configurer le leon3, une interface est fournie par Aeroflex [5] (c'est cette interface graphique qui est écrite en langage TCL/TK) :

Nous avons réalisé une architecture comportant plusieurs leon3 en utilisant l'interface [6].

Après des étapes de synthétisation [7], de génération du bitfile ("make ise") et enfin de la programmation du F.P.G.A. [8] [9].

Nous n'utilisons pas le module Ethernet de la carte ni l'interface PCI, car celle-ci n'est pas

1. `http://www.mingw.org/wiki/MSYS`
2. on a utilisé la version 8.4 :`http://www.activestate.com/activetcl/downloads`, plus informations : `http://www.tcl.tk/about/`
3. si le core est configuré en conséquence
4. Bare-C Cross-Compiler System for LEON3/4 gcc-3.4.4 and gcc 4.4.2 `http://www.gaisler.com/index.php/downloads/compilers?task=view&id=161`
5. dans MSYS on fait un "make xconfig", dans le dossier du design souhaité
6. À noter qu'un message apparait, "config.vhd" après la configuration
7. commande "make ise-map"
8. "make ise-prog-prom"
9. La synthétisation et la génération du bitfile peuvent être aussi faites de manière interactive dans Xilinx ISE.

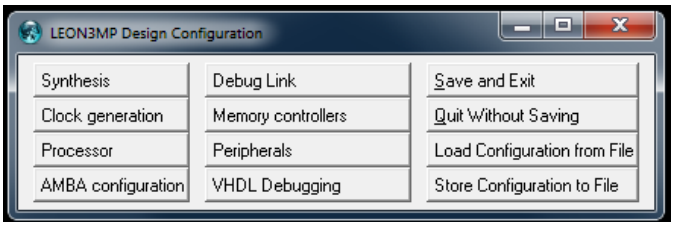

FIGURE 5.1 – Options de l'interface de configuration du leon3.

FIGURE 5.2 – Options des liens de débogage

dans un ordinateur. Nous utilisons la connexion série (UART) afin d'assurer les communications entre un ordinateur (avec grmon) et la carte. Nous avons implanté le Debug System Unit afin de contrôler l'état de la carte grâce à l'UART.

Cette UART est un composant maitre sur le BUS AHB [10].

Au travers l'UART [11] nous pouvons peut générer un accès en lecture ou en écriture sur le bus AMBA AHB. Et celui nous donne accès aux fonctionnalités offertes par grmon afin de faire fonctionner un programme de type "hello world" voir en annexe "Compilation d'un "hello world"".

5.2 Composant VHDL réalisé

Afin d'optimiser l'écriture de notre composant en utilisant le manuel des inférences fourni par Xilinx [12] et un cours détaillé sur le VHDL [13].

Nous avons réalisé un compteur asynchrone grâce aux inférences fournies par Xilinx. Nous avons rendu ce compteur configurable en fixant sa largeur de bus en VHDL voir en annexe "Compteur asynchrone configurable".

10. les processeurs et le Debug System Unit sont maitres sur le bus AHB
11. d'après "AHBUART-AMBA AHB Serial Debug Interface" page 106, leon3 gr-xc3s-1500 template design, based on grlib, october 2006
12. http ://www.xilinx.com/itp/xilinx10/books/docs/xst/xst.pdf
13. `http://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf`

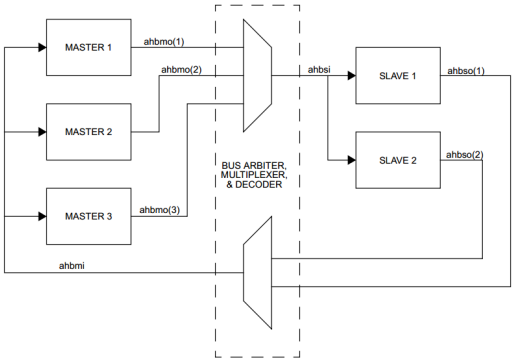

Figure 6. AHB inter-connection view

FIGURE 5.3 – Schéma d'interconnexions esclaves/maitres

14

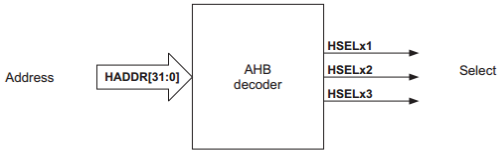

Figure 3-35 AHB decoder interface diagram

FIGURE 5.4 – Interface de décodage selon la norme AMBA 2.0

16

5.3 Compréhension du fonctionnement du BUS AMBA

La compréhension est importante, car celle-ci nous permet de ne pas nous jeter sur le VHDL et "refaire la roue". Ainsi parmi les composants qui nous auxquels nous apportons une attention particulière, il y a l'arbriteur de BUS AHB et le contrôleur mémoire.

5.3.1 Arbriter de bus

Intéressons-nous au rôle de l'arbitre de bus, car celui-ci choisit le prochain maitre qui va s'adresser à un esclave, voir la figure 5.3 :

Nous utilisons une des fonctionnalités qui est le plug&play (littéralement : branché, cela fonctionne), cette fonction est assurée par l'arbitre de bus avec un signal (hconfig [15]) qui est fourni par les composants sur le bus. Elle permet à l'arbitre de bus de faire le décodage d'adresse (voir 5.4) adéquate pour sélectionner le composant visé (voir 5.5).

15. annexe "Record ahb_slv_out_type"

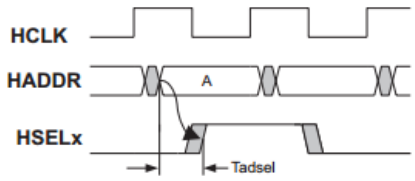

Figure 3-36 AHB decoder timing parameter

FIGURE 5.5 – Signaux du décodeur selon la norme AMBA 2.0

17

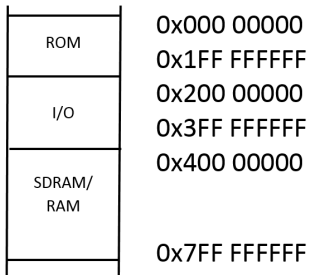

FIGURE 5.6 – Plan mémoire obtenu

Plan mémoire

Dans notre cas on modifiera le hconfig [18] du contrôleur mémoire, si on compare le résultat par lecture du source par rapport à la documentation [19] voir en annexe "Exemple de plan mémoire".

Le décodeur fait une comparaison entre l'adresse haddr présente sur le record (annexe "Record ahb_mst_out_type") du maitre sélectionné, il sélectionne les 12 bits de poids le plus fort et les comparent avec celui des masques [20] de chaque composant esclave.

Cette arbitre de bus implémente une fonction de décodage mémoire qui permet de sélectionner l'esclave ciblé, voir figure 5.5.

Nous obtenons le plan mémoire en figure 5.6 :

En s'appuyant de l'exemple en annexe "Controlleur mémoire - code d'origine", la déclaration d'une plage d'adressage d'un composant requiert plusieurs informations :

– l'adresse du composant

– son masque

Pour les autres paramètres pour ahb_membar() nous n'avons pas d'explications précises à fournir pour le moment.

La dernière ligne ("others => zero32") doit servir à combler le vide restant sur l'espace des possibilités attribuables, de la même manière qu'une ligne "others => '0'" sur un bus.

18. annexe "Record ahb_slv_out_type"
19. page 8, 2.17 Memory map, "LEON3 GR-XC3S-1500 Template Design"
20. page 50-51, 5.3.3 Address Decoding, GRLIB IP Library User's Manual, Version 1.3.7 - B4144, April 2014

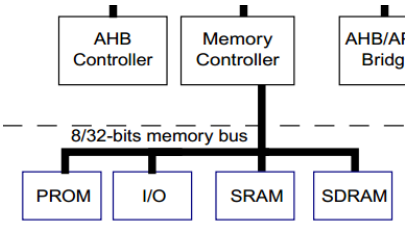

FIGURE 5.7 – page 4 LEON3 GR-XC3S-1500 Template Design", base on GRLIB, October 2006

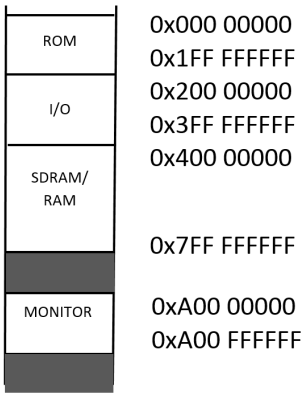

FIGURE 5.8 – Plan mémoire avec composant

5.3.2 Contrôleur mémoire

Le contrôleur mémoire est un esclave[21] qui reçoit un signal de type ahb_slv_in_type en entrée et fournit une sortie de type ahb_slv_out_type[22].

Nous avons procédé à la modification du contrôleur mémoire.

Contrôleur mémoire modifié

Les modifications sont visibles en annexe "Contrôleur Mémoire - Code modifié".

L'objectif est d'obtenir le plan mémoire visible en figure 5.8 :

Nous avons fait attention à ne pas utiliser des plages d'adresse occupées par d'autres composants[23].

21. page 60 "LEON3 GR-XC3S-1500 Template Design, base on GRLIB, October 2006
22. comme définit page 45 dans GRLIB IP Library User's Manual Version 1.3.7 - B4144, April 2014
23. page 8, 2.17 Memory map, "LEON3 GR-XC3S-1500 Template Design"

Chapitre 6

Bilan et objectif à venir

En bilan, on remarquera que nous avons avancé en ce qui concerne la compréhension de la GRLIB et le fonctionnement du bus AMBA AHB. Nous commençons a se projeter dans la construction du composant voir annexe "Branchement du composant".

Le prochain objectif est la mise en place de moyen de débogage pour faciliter le développement du composant. Cela consistera en une vérification de l'allure réelle des signaux pour adapter le VHDL du composant en conséquence.

Chapitre 7

Planning prévisionnel

Voici le planning de développement du projet :

Semaine 12 et 13 :

- Repérer les signaux intéressants et utiles au composant pour son branchement, déduction de la forme des signaux.
- Écrire/modifier le VHDL afin de récupérer les signaux pertinents sur un GPIO pour ensuite le visualiser sur un oscilloscope pour vérifier leurs formes. Adapter si besoin est, la simulation puis le composant.
- Implémenter le composant sur la carte.
- Tester le composant sur la carte dès que possible.

Semaine 14-15 :

- Tester le composant dès que possible et le déboguer.

Semaine 17 :

- Tester le composant dès que possible et le déboguer.
- Écrire le script pour la vidéo.

Semaine 18

- Tourner la vidéo, écrire le rapport.
- Tester le composant et déboguage.

Semaine 19 :

- Tester le composant.
- Vérifier le rapport PFE.

Semaine 20 :

- Rendre la vidéo finale.
- Rendre rapport de PFE.
- Préparer la soutenance.

Chapitre 8

Conclusion

Ce projet m'a jusqu'à présent appris l'un des aspects très importants de la recherche et le fait de mettre en place, consigné, synthétisé les informations et leurs multiples origines et enfin leur date à la laquelle elles ont étaient consultées.

Pour conclure, nous avions eu au cours de notre formation une base technique qui s'est avérée utile. Nous avons découvert une nouvelle façon de générer un code VHDL et de le configurer rapidement (TCL/TK), de compiler un exécutable spécifique à son processeur et de lancer un exécutable spécialement compilé pour notre processeur.

Annexe A

Carte utilisée

La carte utilisée durant ce projet est la suivante :

Des documentations sont disponibles sur le site de gaisler :

`http://www.pender.ch/docs/GR-PCI-XC5V_assy_drawing.pdf`, décembre 2014

`http://www.pender.ch/docs/GR-PCI-XC5V_user_manual.pdf`, décembre 2014

`http://www.gaisler.com/doc/GR-PCI-XC5V_product_sheet.pdf`, décembre 2014

FIGURE A.1 – Carte GR-PCI-XC5V

Annexe B

Compteur configurable

B.1 Compteur asynchrone configurable

```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    02:00:32 12/11/2014
-- Design Name:
-- Module Name:    tcmbj_counter - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
--library IEEE;
--use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--use IEEE.NUMERIC_STD.ALL;
```

```
-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

--
-- 4-bit unsigned up counter with an asynchronous reset.
--
--extrait page 45 de "xst user guide"
--
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity tcmbj_counter is
generic (--constant bus_width integer := 16 ?
    tcmbj_counter_width    : integer := 16
 );
port(
Clk, CLR : in std_logic;
Q : out std_logic_vector(tcmbj_counter_width-1 downto 0)
);
end tcmbj_counter;

architecture archi of tcmbj_counter is
signal tmp: std_logic_vector(tcmbj_counter_width-1 downto 0);
begin
process (Clk, CLR)
begin
if (CLR='1') then
--tmp <= "0000";
tmp <= (others => '0');
elsif (Clk'event and Clk='1') then
tmp <= tmp + 1;
end if;
end process;
Q <= tmp;
end archi;
```

B.2 Testbench associé

```
--------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:   14:32:04 12/11/2014
-- Design Name:
-- Module Name:   D:/PFE/LEON3 and GRLIB IP Library/grlib-gpl-1.3.7-b4144/designs/test_composant/t
-- Project Name:  test_composant
-- Target Device:
-- Tool versions:
-- Description:
--
-- VHDL Test Bench Created by ISE for module: tcmbj_counter
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
-- Notes:
-- This testbench has been automatically generated using types std_logic and
-- std_logic_vector for the ports of the unit under test.  Xilinx recommends
-- that these types always be used for the top-level I/O of a design in order
-- to guarantee that the testbench will bind correctly to the post-implementation
-- simulation model.
--------------------------------------------------------------------------------
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--USE ieee.numeric_std.ALL;

ENTITY testbench_test_composant IS
END testbench_test_composant;
```

```
ARCHITECTURE behavior OF testbench_test_composant IS

    -- Component Declaration for the Unit Under Test (UUT)
 constant largeur : integer := 6;

    COMPONENT tcmbj_counter
 GENERIC(
tcmbj_counter_width    : integer := largeur
);
    PORT(
         Clk : IN  std_logic;
         CLR : IN  std_logic;
         Q : OUT  std_logic_vector(largeur-1 downto 0)
        );
    END COMPONENT;


   --Inputs
   signal Clk : std_logic := '0';
   signal CLR : std_logic := '0';

  --Outputs
   signal Q : std_logic_vector(largeur-1 downto 0);

   -- Clock period definitions
   constant Clk_period : time := 10 ns;

BEGIN

-- Instantiate the Unit Under Test (UUT)
   uut: tcmbj_counter PORT MAP (
          Clk => Clk,
          CLR => CLR,
          Q => Q
        );

   -- Clock process definitions
   Clk_process :process
   begin
```

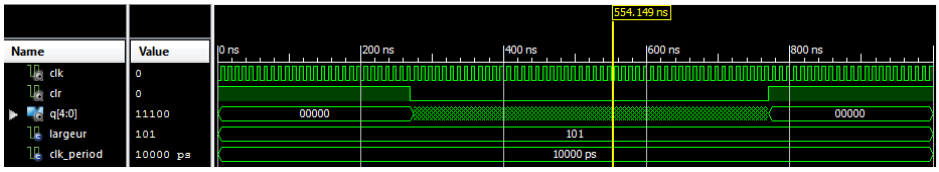

FIGURE B.1 – Simulation Xilinx du composant réalisé

```
Clk <= '0';
wait for Clk_period/2;
Clk <= '1';
wait for Clk_period/2;
   end process;


   -- Stimulus process
   stim_proc: process
   begin
      -- hold reset state for 100 ns.
CLR <= '1';
      wait for 100 ns;
CLR <= '1';
      wait for Clk_period*17;
CLR <= '0';
      wait for Clk_period*50;
CLR <= '1';
      wait for Clk_period*10;

      -- insert stimulus here

      wait;
   end process;

END;
```

B.3 Résultat de simulation

Après une simulation comportementale du composant avec en paramètre une largeur de 5 de bus donc un compteur modulo 32 (2^5), voir 21

Le composant à donc le comportement souhaité (un compteur ayant la possibilité de remise à

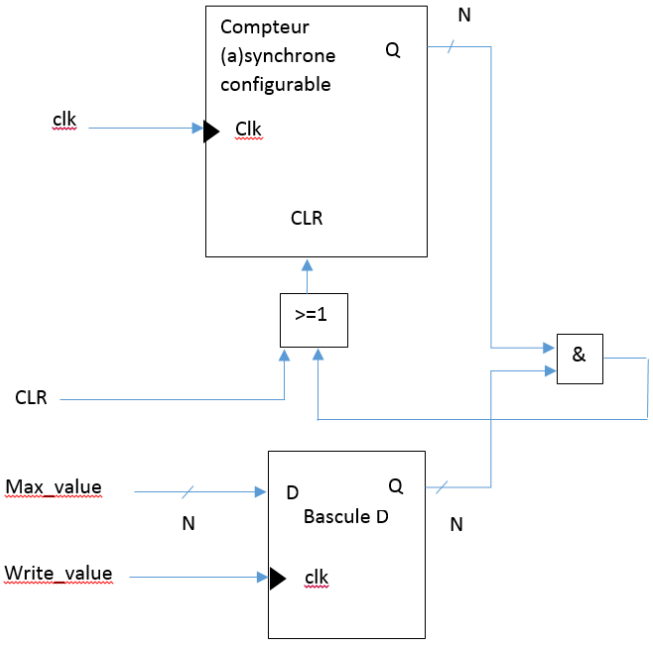

FIGURE B.2 – Proposition schématique

zéro et configurable de maniére matérielle et asynchrone).

Maintenant nous souhaitons que celui soit configurable en modulo de manière logiciel. Pour cela voici une proposition de schéma à faire voir B.2. Celui-ci sera implémenté et ajusté après avoir défini précisément comment connecter le composant.

Ce schéma fait abstraction de contenu du contrôleur mémoire.

Annexe C

Branchement du composant

Nous utilisons les signaux [1] provenant directement de l'entité (du contrôleur mémoire).

Rappel : un esclave est déclaré de la manière voir "déclarations d'un esclave" en annexe :

Rst (entrée)

Clk (entrée, horloge bus AMBA)

Ahbsi (entrée)

- Hsel (selection esclave)
- Haddr (adresse opération)
- Hwrite (lecture/écriture)
- Hwdata (donnée, si écriture)
- Hmaster (maitre actuel, compter pour un maitre spécifique)

Ahbso (sortie)

- Hready (transfert fini)
- Hresp (type de réponse)
- Hrdata (donnée, si lecture)
- Hirq (bus d'interruption, remontée d'interruption)

Nous fixons des génériques VHDL permettant de définir l'adresse (sniffed_addr) et le masque (sniffed_mask) du composant à surveillé. Nous définissons un générique VHDL writeaddr qui fixe l'adresse d'écriture du compteur accessible par les maitres.

Remarquons que pour permettre l'écriture writteaddr doit être dans l'espace adressable définit plus tôt dans le hconfig du contrôleur mémoire (mon_addr) et (mon_mask).

Ce schéma n'est pas définitif et est assujetti à des modifications/améliorations est n'est qu'une esquisse brute, des signaux sont manquants clairement pour le moment voir figure C.1.

Par convention on met une majuscule aux signaux et une minuscule aux génériques.

Notre composant est donc implanté dans le contrôleur mémoire voir 24.

1. ahbsi est de type ahb_slv_in_type et ahbso est de type ahb_slv_out_type

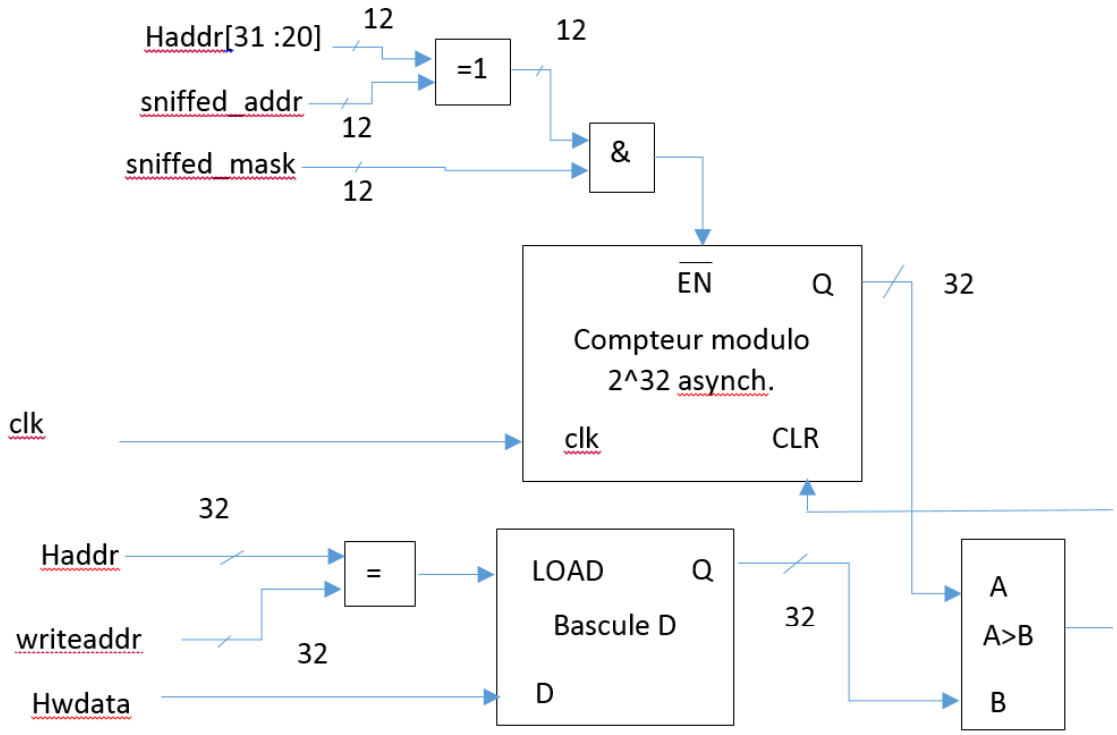

FIGURE C.1 – Proposition de schéma

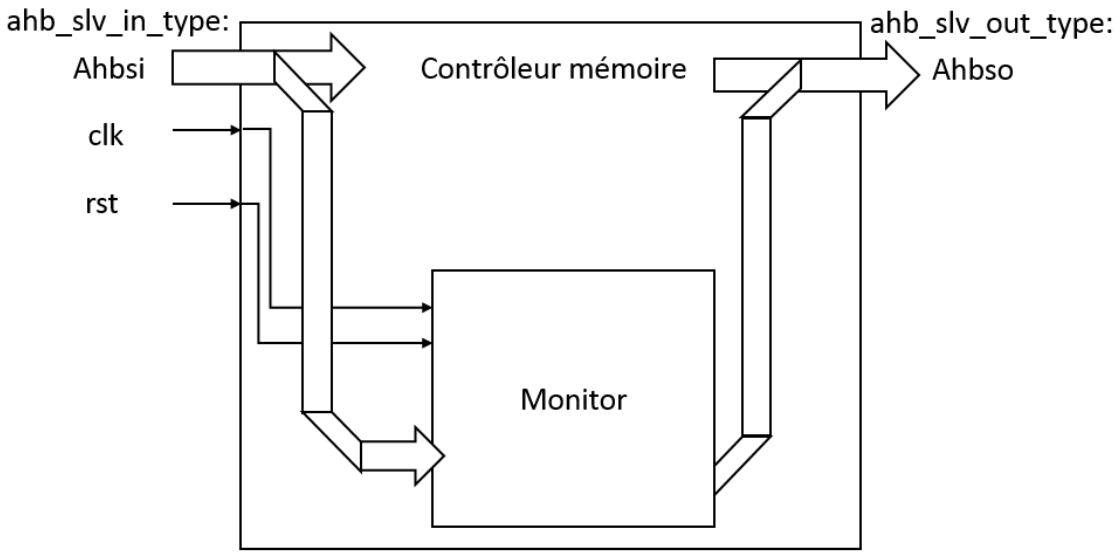

FIGURE C.2 – Schéma de l'implantation dans le contrôleur mémoire

Annexe D

Extrait de codes sources

D.1 hello.c

```
#include <stdio.h>
main()
{
printf("Hello World\n");
}
```

D.2 Compilation d'un "hello world"

Nous avons utilisé des programmes fournis dans les fichiers sources d'Aeroflex [1].

La compilation est effectuée avec le compileur BCC [2] [3]. Nous avons donc fait une compilation d'un fichier C fournit, voir hello.c page 25 nous n'arrivons pas réussi pour le moment à lancer l'exécutable sur la carte au travers de grmon.

Le code contenu dans l'archive (./software/leon3) de l'IP Core [4] n'est pas le seul code que l'ont peut testé, d'autres exemples sont fournis pour les différents compileurs [5].

Pour la partie logiciel, rien n'est fini, il reste à faire fonctionner les exécutables sur le leon3.

D.3 Types

D.3.1 Record ahb_slv_in_type

```
-- AHB slave inputs
```

1. dans les codes sources LEON3/GRLIB
2. Bare-C cross-compiler system, http://www.gaisler.com/index.php/downloads/compilers?task=view&id=161, décembre 2014
3. on compile grâce à une commande fournit dans le manuel comme documenté à la page 5, http://www.gaisler.com/doc/bcc.pdf, décembre 2014
4. à l'adresse, http://www.gaisler.com/products/grlib/grlib-gpl-1.3.7-b4144.tar.gz
5. http://www.gaisler.com/anonftp/bcc/src/examples-1.0.31.tar.gz

```
type ahb_slv_in_type is record
hsel : std_logic_vector(0 to NAHBSLV-1); -- slave select
haddr : std_logic_vector(31 downto 0); -- address bus (byte)
hwrite : std_ulogic; -- read/write
htrans : std_logic_vector(1 downto 0); -- transfer type
hsize : std_logic_vector(2 downto 0); -- transfer size
hburst : std_logic_vector(2 downto 0); -- burst type
hwdata : std_logic_vector(31 downto 0); -- write data bus
hprot : std_logic_vector(3 downto 0); -- protection control
hready : std_ulogic; -- transfer done
hmaster : std_logic_vector(3 downto 0); -- current master
hmastlock : std_ulogic; -- locked access
hbsel : std_logic_vector(0 to NAHBCFG-1); -- bank select
hirq : std_logic_vector(NAHBIRQ-1 downto 0); -- interrupt result bus
end record;
```

D.3.2 Record ahb_slv_out_type

```
-- AHB slave outputs
type ahb_slv_out_type is record
hready : std_ulogic; -- transfer done
hresp : std_logic_vector(1 downto 0); -- response type
hrdata : std_logic_vector(31 downto 0); -- read data bus
hsplit : std_logic_vector(15 downto 0); -- split completion
hirq : std_logic_vector(NAHBIRQ-1 downto 0); -- interrupt bus
hconfig : ahb_config_type; -- memory access reg.
hindex : integer range 0 to NAHBSLV-1; -- diagnostic use only
end record;
```

D.3.3 Record ahb_mst_out_type

```
-- AHB master outputs
type ahb_mst_out_type is record
hbusreq : std_ulogic; -- bus request
hlock : std_ulogic; -- lock request
htrans : std_logic_vector(1 downto 0); -- transfer type
haddr : std_logic_vector(31 downto 0); -- address bus (byte)
hwrite : std_ulogic; -- read/write
hsize : std_logic_vector(2 downto 0); -- transfer size
hburst : std_logic_vector(2 downto 0); -- burst type
```

```
hprot : std_logic_vector(3 downto 0); -- protection control
hwdata : std_logic_vector(31 downto 0); -- write data bus
hirq : std_logic_vector(NAHBIRQ-1 downto 0);-- interrupt bus
hconfig : ahb_config_type; -- memory access reg.
hindex : integer range 0 to NAHBMST-1; -- diagnostic use only
end record;
```

D.3.4 Record ahb_mst_in_type

```
-- AHB slave inputs
type ahb_slv_in_type is record
hsel : std_logic_vector(0 to NAHBSLV-1); -- slave select
haddr : std_logic_vector(31 downto 0); -- address bus (byte)
hwrite : std_ulogic; -- read/write
htrans : std_logic_vector(1 downto 0); -- transfer type
hsize : std_logic_vector(2 downto 0); -- transfer size
hburst : std_logic_vector(2 downto 0); -- burst type
hwdata : std_logic_vector(31 downto 0); -- write data bus
hprot : std_logic_vector(3 downto 0); -- protection control
hready : std_ulogic; -- transfer done
hmaster : std_logic_vector(3 downto 0); -- current master
hmastlock : std_ulogic; -- locked access
hbsel : std_logic_vector(0 to NAHBCFG-1); -- bank select
hirq : std_logic_vector(NAHBIRQ-1 downto 0); -- interrupt result bus
end record;
```

D.4 Déclaration d'un esclave

```
component ahbslave
generic (
hindex : integer := 0; -- slave index
hirq : integer := 0); -- interrupt index
port (
rst : in std_ulogic;
clk : in std_ulogic;
hslvi : in ahb_slv_in_type; -- AHB slave inputs
hslvo : out ahb_slv_out_type); -- AHB slave outputs
end component;
```

D.5 Contrôleur mémoire

D.5.1 Code d'origine

Voici un extrait du VHDL du contrôleur mémoire.

```
[...]
    romaddr   : integer := 16#000#;
    rommask   : integer := 16#E00#;
    ioaddr    : integer := 16#200#;
    iomask    : integer := 16#E00#;
    ramaddr   : integer := 16#400#;
    rammask   : integer := 16#C00#;
[...]
constant hconfig : ahb_config_type := (
  0 => ahb_device_reg ( VENDOR_ESA, ESA_MCTRL, 0, REVISION, 0),
  4 => ahb_membar(romaddr, '1', '1', rommask),
  5 => ahb_membar(ioaddr,  '0', '0', iomask),
  6 => ahb_membar(ramaddr, '1', '1', rammask),
  others => zero32);
[...]
```

D.5.2 Exemple de plan mémoire

Le B.A.R. (Bank Access Registry) est "l'inventaire" de tout l'espace adressable/accessible des composants, il est justement configuré avec les hconfig et c'est un "espace mémoire" accessible en lecture seulement.

La sélection d'un esclave est faite si la condition suivante est remplie (valable pour la RAM/S-DRAM et la ROM, des règles spécifiques s'applique aux I/Os) :

((BAR.ADDR xor HADDR[31 :20]) and BAR.MASK) = 0

Détails :

– BAR.ADDR sous-entend que l'on parle de l'adresse d'un esclave en particulier.

– BAR.MASK sous-entend que l'on parle du masque d'un esclave en particulier.

Le hconfig contenu dans le contrôleur mémoire est disponible en annexe "Controlleur mémoire - code d'origine".

Nous obtientenons en plage adressable pour : -le code source :
0x000 00000 -0x1FF FFFFFF : ROM
0x200 00000 -0x3FF FFFFFF : I/O
0x400 00000 -0x7FF FFFFFF : SDRAM/RAM
-Extrait de la documentation [6] :

6. page 8, 2.17 Memory map, "LEON3 GR-XC3S-1500 Template Design"

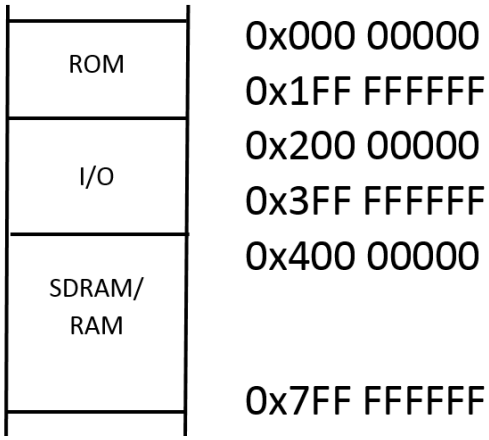

FIGURE D.1 – Plan mémoire obtenu

0x000 00000 - 0x200 00000 : PROM area
0x200 00000 - 0x400 00000 : I/O area
0x400 00000 - 0x800 00000 : SRAM/SDRAM area

Nous obtientenons le même plan mémoire entre la documentation et le code VHDL du contrôleur mémoire.

D.5.3 Code modifié

Voici un code modifié pour permettre l'ajout du composant :

```
[...]
    romaddr   : integer := 16#000#;
    rommask   : integer := 16#E00#;
    ioaddr    : integer := 16#200#;
    iomask    : integer := 16#E00#;
    ramaddr   : integer := 16#400#;
    rammask   : integer := 16#C00#;
    mon_addr   : integer := 16#A00#;
    mon_mask : integer := 16#FFF#;
[...]
constant hconfig : ahb_config_type := (
  0 => ahb_device_reg ( VENDOR_ESA, ESA_MCTRL, 0, REVISION, 0),
  4 => ahb_membar(romaddr, '1', '1', rommask),
  5 => ahb_membar(ioaddr,  '0', '0', iomask),
  6 => ahb_membar(ramaddr, '1', '1', rammask),
  7 => ahb_membar(mon_addr, '1', '1', mon_mask),
  others => zero32);
```

```
[...]
```