Université Lille 1

Département Informatique Micro-éléctronique et Automatique.

Rapport de Présoutenance de Projet Fin d'Étude I.M.A.5

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 accueillit 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 que l'on nous a suggéré de faire un composant capable de faire un composant capable de suivre les accès mémoire d'un processeur (LEON3 voir figure 2.1) simulé sur un F.P.G.A..

FIGURE 2.1 – Gaisler/Aeroflex : origine code source LEON3/GRLIB

Chapitre 3

Contexte général

Dans le cadre des systèmes embarqués, plus particulièrement les systèmes temps réel, il arrive que certaine tâche, soit critique. C'est à dire que si l'on prend par exemple une régulation qui doit être faite dans un temps imparti, car celle-ci est chimique par exemple et peu mené à une explosion (réacteur de fusée) ou même des tâches de sécurité.

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

Tout d'abord présentons l'architecture du Leon 3, voir la figure le schéma suivant :

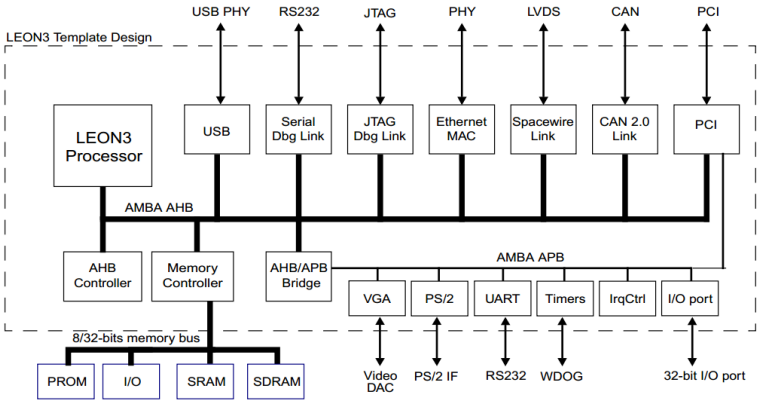

Schéma du bus AHB avec les différents composants

Ce processeur utilise un bus AMBA [1] notamment un bus AHB [2]. Nous avions évoqué la possibilité de faire un sniffeur sur le bus AHB qui serait passif, mais nous avons abandonné au profit dans composant directement intégré dans le contrôleur mémoire. Nous avons fais ce choix, car 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 devra s'appuyer sur le même principe de fonctionnement que celui de l'arbitreur de bus.

Les contraintes qui ont était établi sont :

– le composant doit pouvoir être adressable par le processeur à une adresse pré-définit auparavant

– le composant devra pouvoir être controlé ou forcé à un état [3]

A l'avenir 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. on aura l'accés en lecture ou en écriture par le processeur à l'adresse du composant

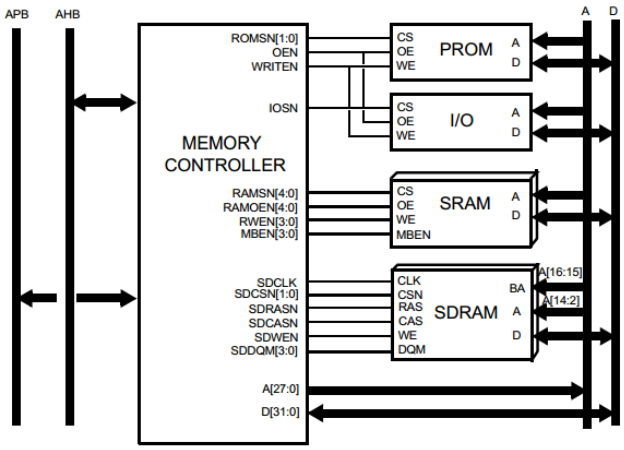

FIGURE 4.1 – Composants connectés au contrôleur mémoire

Chapitre 5

Avancés réalisées

Il faut savoir qu'avant d'avoir accès aux sources du leon3 dans une des configurations possibles, il faut pouvoir faire mettre en place l'environement de développement et de test. En effet les outils fourni par Gaisler/Aeroflex permettent d'être utilisé sur des environnements disposant des fonctionnalités équivalents à Linux (Unix). C'est pourquoi nous avons mis en place MSYS [1] afin de disposer de l'utilitaire Make.

Les utilitaires fournis par Gaisler fonctionnent avec des fonctionnalités écrites en TCL/TK [2].

On utilise l'environnement de développement ISE Xilinx Vivado car nous développons en VHDL afin qu'il soit tester sur une carte gr-pci-xc5v. Parmi les softs qui sont intéressants il y a grmon (fournit par Gaisler/Aeroflex) qui permet de réaliser une communication série entre l'ordinateur et un core leon3 implanté sur le F.P.G.A. de la carte. Et celui-ci aussi nous permet d'accéder à des fonctionnalités de débogage [3].

Afin de réaliser des fichiers binaires pouvant être éxecuter sur le leon3 on utilise un cross-compiler (BCC) [4]

5.1 Configuration du leon3

Afin de configurer le leon3 nous utilisons l'interface fournit par Aeroflex (dans MSYS on fait un "make xconfig"), une interface graphique écrite en langage TCL/TK démarre :

On nous a demandé de réaliser une architecture comportant plusieurs leon3 et c'est ainsi que nous avons aussi implanté rapidement plusieurs processeurs [5].

On poursuit ensuite par une étape de synthétisation (commande "make ise-map") et de génération du bitfile ("make ise") et enfin de programmation du F.P.G.A. ("make ise-prog-prom").

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 a été 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. A noter qu'un message apparait, "config.vhd" aprés la configuration

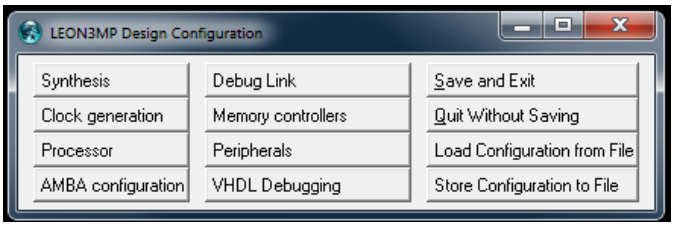

FIGURE 5.1 – Options de l'interface de configuration du leon3.

FIGURE 5.2 – Options des liens de déboggage

La synthétisation et la génération du bitfile peuvent être aussi faites de manière interactive dans Xilinx ISE.

Dans notre cas on ne souhaite pas utiliser le module Ethernet de la carte ni l'interface PCI, car celle-ci n'est pas dans un ordinateur. On utilisera la connexion série (UART) afin d'assurer les communications entre un ordinateur (avec grmon) et la carte. On y implante 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 [6].

Au travers l'UART [7] on peut générer un accés 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".

5.2 Compilation d'un "hello world"

Afin de compilé un hello world, nous nous sommes inspiré des programmes fournit dans les fichiers sources d'Aeroflex [8].

La compilation est effectué avec le compileur BCC [9] et on compile grâce à une commande

6. les processeurs sont maitres sur le bus AHB, le Debug System Unit, les autres composants sont des esclaves, mais nous allons explorer cette aspect plus tard.

7. d'après "AHBUART-AMBA AHB Serial Debug Interface" page 106, leon3 gr-xc3s-1500 template design, based on grlib, october 2006

8. dans les codes sources LEON3/GRLIB

9. Bare-C cross-compiler system, `http://www.gaisler.com/index.php/downloads/compilers?task=view&id=`

fournit dans le manuel [10]. Nous avons donc fait une compilation d'un fichier C fournit, voir hello.c page 26nous n'arrivons pas pour le moment à lancer l'exécutable sur la carte au travers de grmon.

On notera cependant que le code contenu dans l'archive (./software/leon3) de l'IP Core [11] n'est pas le seul code que l'ont pourra testé, il y aussi une série d'exemples fournit pour les différents compileurs [12].

En ce qui concerne la partie logiciel, rien n'est fini, il reste toujours à trouver un moyen de faire fonctionner ces exécutables sur le leon3.

5.3 Composant VHDL réalisé

Afin d'optimiser l'écriture de notre composant, nous avons utilisé le manuel des inférences fourni par Xilinx [13] et un cours détaillé sur le VHDL [14].

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".

5.4 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.

5.4.1 Arbriteur de BUS

Nous nous sommes intéressés au rôle tout d'abord à l'arbriteur, car celui-ci permet de choisir le prochain maitre qui va s'adressai à un esclave, voir la figure 5.3 :

Mais dans le cas de l'implantation de notre composant de manière optimale et propre de notre code. C'est pour cela que l'on utilisera une des fonctionnalités qui est le plug&play (littéralement : branché, cela fonctionne), cette fonction est assurée par l'arbitrer avec un signal (hconfig [16]) fournit automatiquement par les composants sur le bus. Elle permet à l'arbitrer de bus de faire le décodage d'adresse (voir 5.4) adéquate pour sélectionner le composant visé (voir 5.5).

161, décembre 2014

10. comme documenté à la page 5, http://www.gaisler.com/doc/bcc.pdf, décembre 2014
11. à l'adresse, http://www.gaisler.com/products/grlib/grlib-gpl-1.3.7-b4144.tar.gz
12. http://www.gaisler.com/anonftp/bcc/src/examples-1.0.31.tar.gz
13. http ://www.xilinx.com/itp/xilinx10/books/docs/xst/xst.pdf
14. http://www.ics.uci.edu/~alexv/154/VHDL-Cookbook.pdf
16. annexe "Record ahb_slv_out_type"

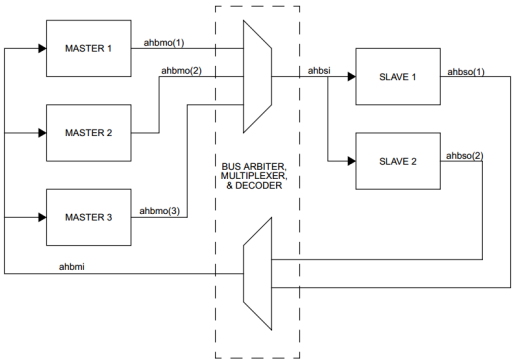

Figure 6. AHB inter-connection view

FIGURE 5.3 – Schéma d'interconnexions esclaves/maitres

15

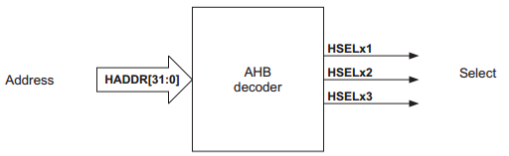

Figure 3-35 AHB decoder interface diagram

FIGURE 5.4 – Interface de décodage selon la norme AMBA 2.0

17

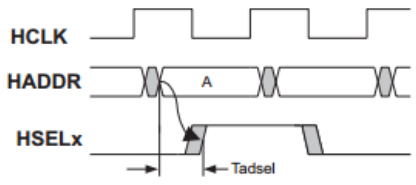

Figure 3-36 AHB decoder timing parameter

FIGURE 5.5 – Signaux du décodeur selon la norme AMBA 2.0

18

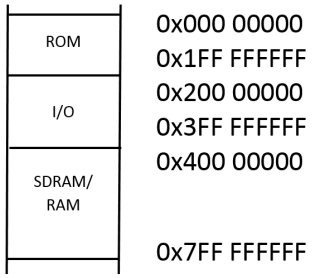

FIGURE 5.6 – Plan mémoire obtenu

Exemple d'adressage sur le code d'origine

Dans notre cas on modifiera le hconfig[19] du controleur mémoire, mais tout d'abord confrontons le source à la documentation[20].

Le décodeur fait une comparaison entre l'adresse haddr présente sur le record (annexe "Record ahb_mst_out_type") du maitre sélectionné, il utilise les 12 bit de poids le plus fort et les comparent avec celui des masques[21] de chaque composant esclave.

Un exemple de plan mémoire est fait en annexe "Exemple de plan mémoire".

On obtient le plan mémoire en figure 5.6 :

En s'appuyant de l'exemple en annexe "Controlleur mémoire - code d'origine". Ainsi comme on peut le voir, 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.

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

5.4.2 Controlleur mémoire

Les bibliothèques sur lesquelles on a travaillé sont les codes source du leon3 qui est disponible sur le site de Gaisler[22] et notamment sur les fichiers sources du Leon3[23].

On sait que le contrôleur mémoire est un esclave[24] et que donc notre composant reçoit un signal de type ahb_slv_in_type en entrée et fournit une sortie de type ahb_slv_out_type[25].

19. annexe "Record ahb_slv_out_type"
20. page 8, 2.17 Memory map, "LEON3 GR-XC3S-1500 Template Design"
21. page 50-51, 5.3.3 Address Decoding, GRLIB IP Library User's Manual, Version 1.3.7 - B4144, April 2014
22. `http://www.gaisler.com/index.php/downloads/leongrlib?task=view&id=156`
23. `http://www.gaisler.com/products/grlib/grlib-gpl-1.3.7-b4144.tar.gz`
24. page 60 "LEON3 GR-XC3S-1500 Template Design, base on GRLIB, October 2006
25. comme définit page 45 dans GRLIB IP Library User's Manual Version 1.3.7 - B4144, April 2014

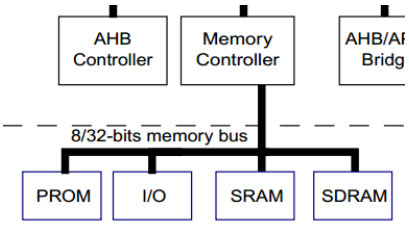

FIGURE 5.7 – page 4 LEON3 GR-XC3S-1500 Template Design", base on GRLIB, October 2006

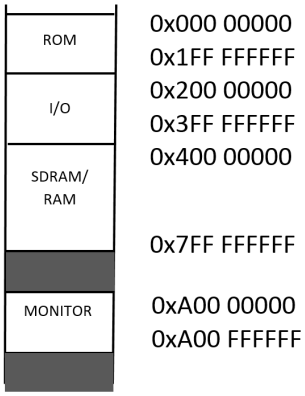

FIGURE 5.8 – Plan mémoire avec composant

Nous ne présentons pas l'intégralité du code du contrôleur mémoire, car celui-ci étant trop long cela n'aura pas d'intérêt ($\tilde{1}$000 lignes), seules les lignes pertinentes ont été extraites.

Modification du contrôleur mémoire

Pour cela on procède aux modifications visibles en annexe "Contrôleur Mémoire - Code modifié".

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

On a fait attention à ne pas utiliser des plages d'adresse occupées par d'autres composants [26].

26. 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. On commence aussi à avoir une idée de comment l'on va construire notre composant voir annexe "Branchement du composant".

Notre prochain objectif incontournable est la mise en place de moyen de débogage pour 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

On souhaite poursuivre le développement du projet de la manière suivante :

Semaine 12 et 13 :

- Repérage des signaux intéressant et utile au composant pour son branchement, déduction de la forme des signaux.
- Écriture/modification de VHDL afin de récupérer les signaux pertinents sur un GPIO pour ensuite le visualiser sur un oscilloscope pour vérifier leurs formes. Cela nous permettra d'adapter si besoin est, le composant et sa simulation.
- Une fois ces VHDL développés on commencera l'implémentation du composant avec le leon3 sur la carte.
- Test du composant sur la carte dès que possible.

Semaine 14-15 :

- Test du composant dès que possible et débogage.

Semaine 17 :

- Test du composant et débogage.
- Écriture du script pour la vidéo

Semaine 18

- Prise de la vidéo, écriture du rapport.
- Test du composant et débogage.

Semaine 19 :

- Test du composant
- Rapport PFE

Semaine 20 : Écriture finale du rapport de PFE et préparation à 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 les multiples origines et enfin leur date à la laquelle elles ont étaient consultées.

Pour conclure, il faut savoir que nous avions eu au cours de notre formation une base technique qui s'est avérée utile. Nous avons pu découvrir une nouvelle façon de générer un code VHDL et de le configuré rapidement (TCL/TK), de compiler un exécutable spécifique à son processeur et de faire une communication avec un FPGA (grmon) afin d'y lancer son exécutable spécialement compilé pour notre configuration.

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
```

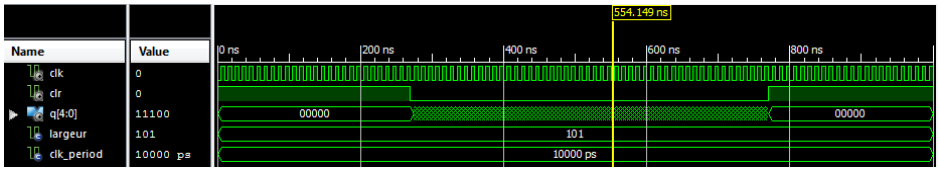

FIGURE B.1 – Simulation Xilinx du composant réalisé

```
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

Nous avons fait une simulation comportementale du composant avec en paramètre une largeur de 5 de bus donc un compteur modulo 32 (2^5).

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

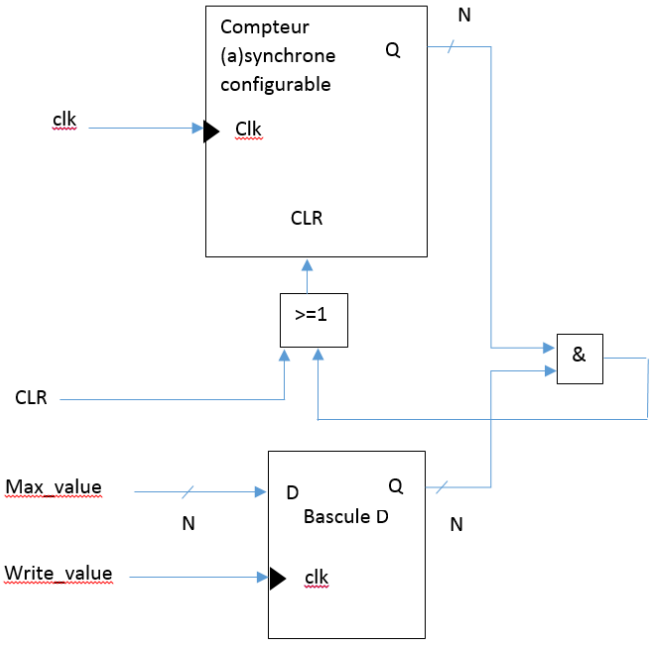

FIGURE B.2 – Proposition schématique

de remise à zéro et configurable de maniére matériel).

Maintenant on souhaite que celui soit configurable en modulo de maniére logiciel. Pour cela nous avons une proposition de schéma à faire mais celui sera implémenté et ajusté après avoir déterminée précisement comment connecté le composant.

Cela aura l'avantage de pouvoir être modifiable facilement, mais le fait que les composants soient synchrones ou non dépendra directement des signaux disponibles.

Annexe C

Branchement du composant

En étudiant la question des signaux disponibles dans le contrôleur mémoire, on peut obtenir les signaux [1] provenant directement de l'entité (du contrôleur mémoire).

On rappelle que tout esclave est déclaré de la manière suivante 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, si l'on souhaite 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)

On définit des generics VHDL permettant de définit l'adresse (sniffed_addr) et le masque (sniffed_mask) du composant à surveillé. On définit un generic 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 generics.

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

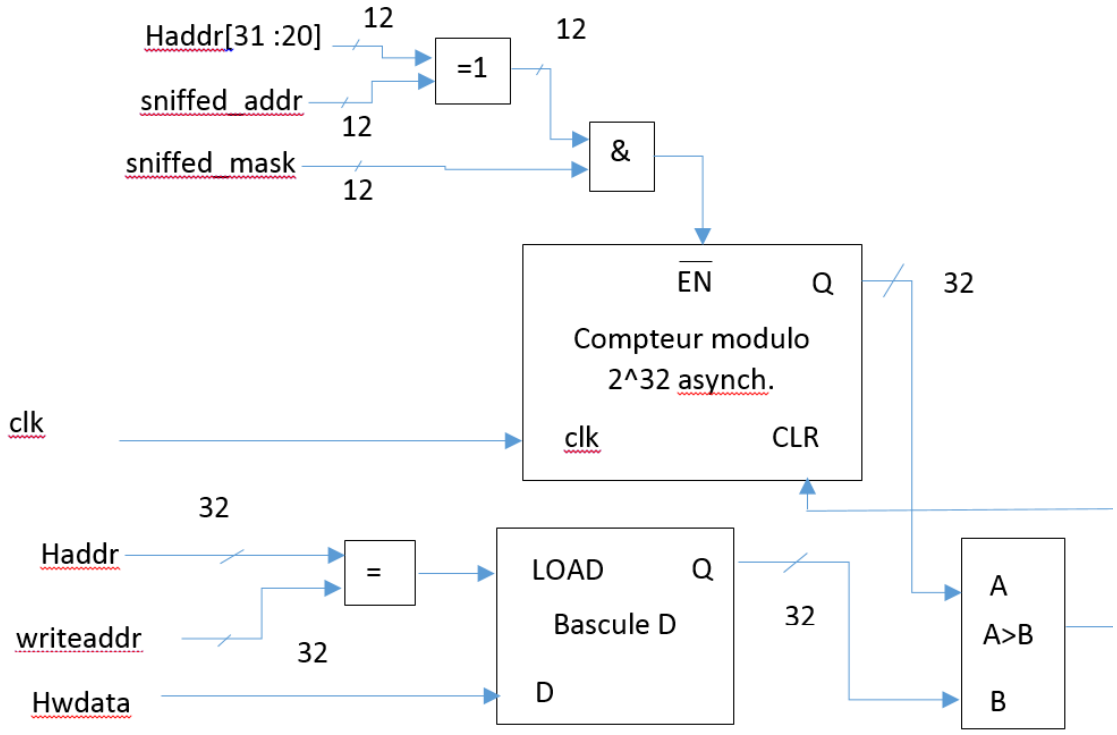

FIGURE C.1 – Proposition de schéma

Annexe D

Extrait de codes sources

D.1 hello.c

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

D.2 Types

D.2.1 Record ahb_slv_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.2.2 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.2.3 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.2.4 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.3 Déclaration d'un esclave

D.4 Contrôleur mémoire

D.4.1 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.4.2 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 aux I/Os) :

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

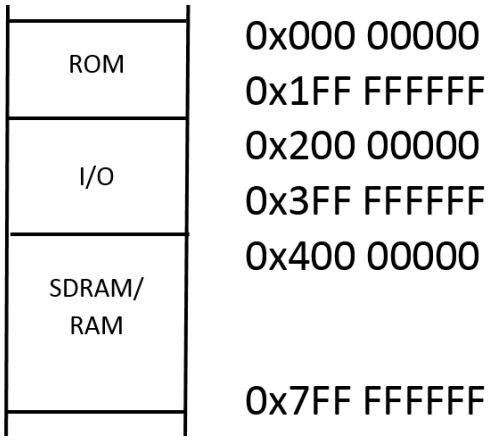

FIGURE D.1 – Plan mémoire obtenu

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.

Si l'on prends l'exemple du hconfig contenu dans le controleur mémoire voir en annexe "Controlleur mémoire - code d'origine".

On obtient en plage addressable 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 [1] :

0x000 00000 - 0x200 00000 : PROM area

0x200 00000 - 0x400 00000 : I/O area

0x400 00000 - 0x800 00000 : SRAM/SDRAM area

On obtient le même plan mémoire entre la documentation et le code VHDL du contrôleur mémoire.

D.4.3 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#;
```

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

```
    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);
[...]
```