Description en VHDL d'un microcontrôleur 16 bits

Les constructeurs de FPGA propose avec leurs dernières générations de composants d'inclure un microcontrôleur (voir même un processeur de signal) au sein du circuit configurable. Deux stratégies sont possibles :

- le microcontrôleur est déjà implanté dans le silicium, ce qui permet une optimisation en terme de vitesse et d'intégration ;
- le microcontrôleur est un concept logiciel (optimisé pour le circuit cible) et sera synthétisé dans le FPGA. L'utilisateur peut alors définir les paramètres tels que la largeur des bus, le nombre de registres etc… avec l'aide du logiciel de synthèse au moyen d'un « mega wizard ». Dans le cas où on ne souhaiterait pas implanter de microcontrôleur, les ressources matérielles restent disponibles pour le reste de la conception. Cette solution est donc plus souple (mais moins performante) que la précédente.

Les machines d'état permettent de décrire très simplement des systèmes complexes tel qu'un microcontrôleur. Il vous est proposé au cours de cette séance la synthèse VHDL d'un microcontrôleur 16 bits, avec un jeu d'instruction relativement réduit dans un premier temps ; de manière très simple il sera possible d'enrichir ce jeu, instruction par instruction sans avoir à repenser la structure du composant tout entier (mis à part pour des instructions très complexes).

Sans prétendre rivaliser avec les solutions proposées par les constructeurs, notre application permettra éventuellement de résoudre de petits problèmes simples d'automatisme, bien que la conception de la machine d'état pour le problème lui-même sera toujours plus performante.
D'un point de vue pédagogique, elle peut permettre une bonne approche du fonctionnement d'un microcontrôleur, avec la possibilité pour l'étudiant d'apporter lui-même ses propres instructions, ce qui implique de bien comprendre le fonctionnement de l'ensemble.
Il convient de noter également, la présence sur internet d'un site développement, dans l'esprit du logiciel libre, d'un microprocesseur en VHDL : http://www.f-cpu.org .

Le logiciel de synthèse utilisé sera l'outil Altera Max+plus II, la description pouvant être implantée dans le Flex10K20 de la carte de développement UP1.

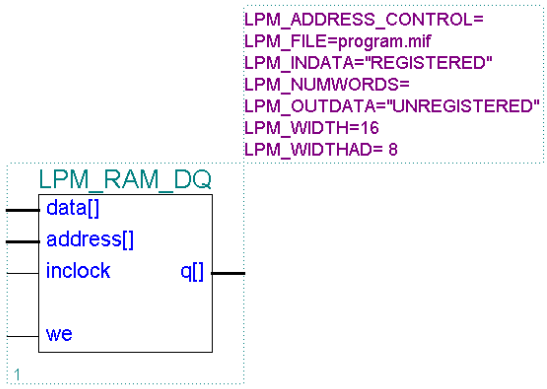
Structure et description

Notre microcontrôleur comprend un microprocesseur 16 bits de données, 8 bits d'adresse et une mémoire RAM de 256 mots de 16 bits. Pour la synthèse de cette dernière nous utilisons les composants prédéfinis du logiciel, en particulier le composant lpm_ram_dq de la bibliothèque maxplus2\maxlib\méga_lpm. Cette mémoire étant une mémoire à port de lecture et écriture différenciés (plus facile à implanter au sein des FPGA), elle conditionnera beaucoup l'architecture du microcontrôleur.

La description de ce dernier pourra se faire :

- de manière graphique en implantant d'un côté la RAM et de l'autre un composant correspondant au microprocesseur ; cette solution présente l'avantage d'être plus visuelle ;
- entièrement textuelle en déclarant l'utilisation (au moyen d'un paquetage) et l'instanciation du composant RAM au sein du fichier VHDL, cette solution étant plus concise.

Quelle que soit la solution retenue, on peut obtenir de l'aide sur le fonctionnement du composant RAM et allant chercher l'icône « ? » et en cliquant sur le symbole graphique (cas 1) ou sur l'instantiation (cas 2).



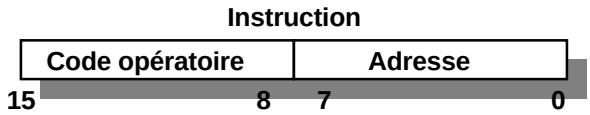
Symbole graphique de la mémoire

```
-- déclaration
LIBRARY lpm;
USE lpm.lpm_components.ALL;

-- instanciation
memory: lpm_ram_dq
    GENERIC MAP (
        lpm_widthad     => 8,
        lpm_outdata     => "UNREGISTERED",
        lpm_indata      => "REGISTERED",
        lpm_address_control=> UNREGISTERED",
        lpm_file        => "program.mif",
        lpm_width               => 16 )

        PORT MAP (  data    =>  Accumulateur,
address         =>      Registre_adresse_memoire,
we => WR, inclock => H, q => Reg_donnee_mem );
```

Déclaration du paquetage et instanciation

La RAM est configurée de la manière suivante :
- fonctionnement asynchrone en lecture : si « we » est au niveau logique 0, la donnée correspondant à l'adresse placée sur « address » apparaît sur le port « q » (données en sortie) ;
- fonctionnement synchrone en écriture : si « we » est au niveau logique 1, la donnée envoyée sur « data » est écrite à l'adresse placée sur « address » au front montant de l'horloge « inclock » ; l'adresse doit être stable avant le front actif de l'horloge ;
- le programme écrit en mémoire est donné par un fichier texte, d'extension « .mif » pour « Memory Input File » dont on précise le nom dans l'instanciation.

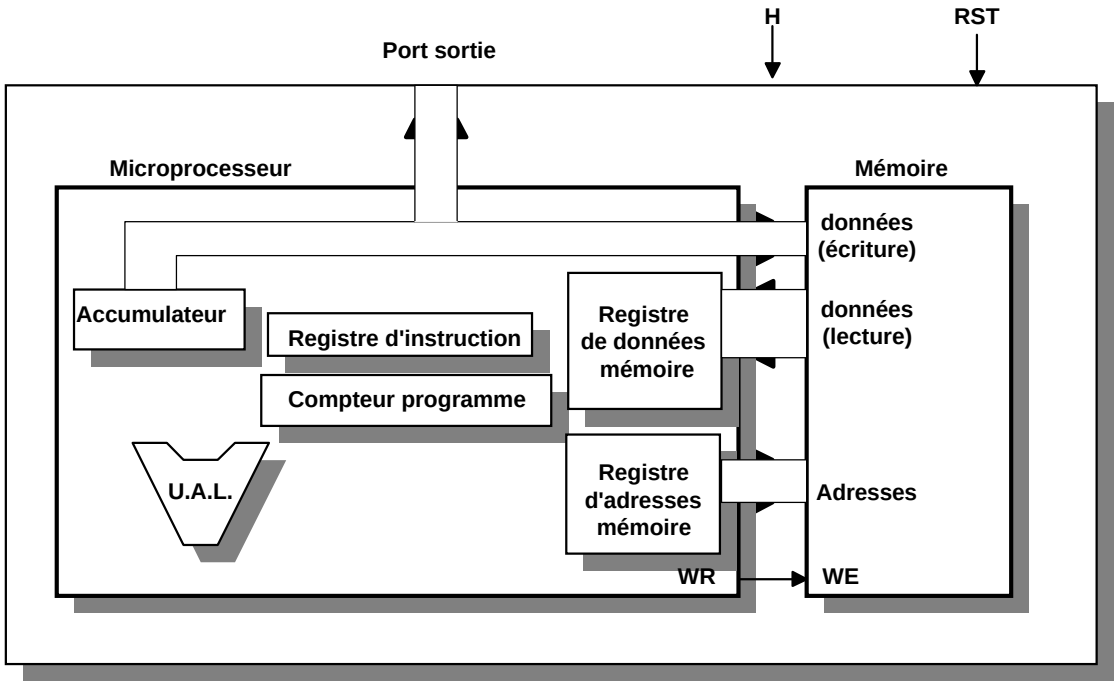
La structure de notre microcontrôleur est alors la suivante :



Les opérations se déroulent en trois temps :
- une phase de recherche de l'instruction nommée « fecth » ;
- une phase de décodage de l'opération nommée « décode » ;
- une phase d'exécution, dont le nom dépend de l'opération à réaliser.

Le compteur de programme (8 bits) précise, lors de la première phase « fetch » de recherche d'instruction, les adresses à envoyer à la mémoire pour recevoir les instructions ; celles-ci sont chargées via le registre de donnée mémoire dans le registre d'instruction.

Chaque instruction (reçue depuis la mémoire) est un mot de 16 bits dont les 8 bits de poids fort donnent le code de l'opération à effectuer et les 8 bits de poids faible une adresse de 8 bits.



Celle-ci est directement placée dans le registre d'adresse mémoire, à la fin de la seconde phase « décode » de décodage de l'instruction ; la valeur à utiliser est alors pointée en mémoire par le registre d'adresse et disponible sur le bus de sortie de données de la mémoire pour être utilisée lors de l'instruction suivante (lecture asynchrone de la mémoire).

La phase d'exécution utilise ensuite éventuellement l'unité arithmétique et logique qui reste pour notre description une simple opération VHDL. Si le résultat de cette opération est une donnée, elle se trouve généralement placée dans l'accumulateur, qui est relié directement au port d'entrée de la mémoire.

L'écriture en mémoire, nécessitant la mise au niveau 1, puis de nouveau à 0 de « we », prendra trois cycles d'horloge.

Notre description propose un jeu de quatre instructions résumées dans le tableau suivant :

Mnémonique possible	Opération réalisée	Code machine (en hexadécimal)
ADD XX	Addition du contenu de l'adresse XX et de l'accumulateur, résultat dans l'accumulateur	00XX
ST XX	Placement du contenu de l'accumulateur dans l'emplacement mémoire XX	01XX
LDA XX	Chargement de l'accumulateur par le contenu de l'adresse XX	02XX
JUMP XX	Saut à l'adresse XX du programme, adresse alors chargée dans le compteur programme	03XX

Le programme ci-après donne une description VHDL possible de ce microcontrôleur.

```
LIBRARY IEEE;
USE  IEEE.STD_LOGIC_1164.ALL;
USE  IEEE.STD_LOGIC_ARITH.ALL;
USE  IEEE.STD_LOGIC_UNSIGNED.ALL;

-- déclacration pour l'utilisation d'un composant prédéfini :
-- la mémoire, dans la bibliothèque lpm
LIBRARY lpm;
USE lpm.lpm_components.ALL;



ENTITY MicroC IS
PORT(  H, RST                          : IN STD_LOGIC;
                Port_sortie            : OUT STD_LOGIC_VECTOR(15 DOWNTO 0 ));
END MicroC;


ARCHITECTURE ARCH OF MicroC IS


TYPE TYPE_ETAT IS ( reset_pc, fetch, decode, execute_add, execute_load, execute_store,
                execute_store3, execute_store2, execute_jump );
SIGNAL ETAT: TYPE_ETAT;
SIGNAL Registre_instruction                : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL Reg_donnee_mem                      : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL Accumulateur                        : STD_LOGIC_VECTOR(15 DOWNTO 0 );
SIGNAL Compteur_programme                  : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL Registre_adresse_memoire            : STD_LOGIC_VECTOR( 7 DOWNTO 0 );
SIGNAL WR                                  : STD_LOGIC;

BEGIN

-- instantiation de la mémoire de 256 mots de 16-bit, double port, écriture synchrone
  memory: lpm_ram_dq
        GENERIC MAP (
                lpm_widthad     => 8,
                lpm_outdata     => "UNREGISTERED",
                lpm_indata              => "REGISTERED",
                lpm_address_control => "UNREGISTERED",
                lpm_file                => "program.mif",-- fichier programme à placer en mémoire
                lpm_width               => 16 )
        PORT MAP (     data => Accumulateur, address => Registre_adresse_memoire,
           we => WR, inclock => H, q => Reg_donnee_mem );

-- affectation des sorties
        Port_sortie                             <= Accumulateur;

PROCESS ( H, RST )
        BEGIN
        IF RST = '1' THEN
                ETAT <= reset_pc;
        ELSIF H'EVENT AND H = '1' THEN
                CASE ETAT IS

                                        -- initialisation
                WHEN reset_pc =>
                        Compteur_programme              <= "00000000";
```

```
                Registre_adresse_memoire    <= "00000000";
                Accumulateur                <= "0000000000000000";
                WR                          <= '0';
                ETAT                        <= fetch;
                        -- recherche d'instruction dans la mémoire programme
        WHEN fetch =>
                Registre_instruction    <= Reg_donnee_mem ;
                Compteur_programme      <= Compteur_programme + 1;
                WR                      <= '0';
                ETAT                    <= decode;
                        -- Decodage des instructions
        WHEN decode =>
                Registre_adresse_memoire <= Registre_instruction( 7 DOWNTO 0);
                CASE Registre_instruction( 15 DOWNTO 8 ) IS
                        WHEN "00000000" =>
                           ETAT         <= execute_add;
                        WHEN "00000001" =>
                           ETAT         <= execute_store;
                        WHEN "00000010" =>
                           ETAT         <= execute_load;
                        WHEN "00000011" =>
                           ETAT         <= execute_jump;
                        WHEN OTHERS =>
                           ETAT         <= fetch;
                END CASE;

                        -- Addition
        WHEN execute_add =>
                Accumulateur            <=      Accumulateur    +
Reg_donnee_mem ;
                Registre_adresse_memoire    <= Compteur_programme;
                ETAT                        <= fetch;

                        -- Ecriture en mémoire en 3 cycle d'horloge
                        -- le adresses ne doivent pas bouger lorsque WR=1
        WHEN execute_store =>
                WR                      <= '1';
                ETAT                    <= execute_store2;

        WHEN execute_store2 =>
                WR                      <= '0';
                ETAT                    <= execute_store3;

        WHEN execute_store3 =>
                Registre_adresse_memoire    <= Compteur_programme;
                ETAT                        <= fetch;

                        -- Chargement d'un donnée depuis la mémoire
        WHEN execute_load =>
                Accumulateur                <= Reg_donnee_mem ;
                Registre_adresse_memoire    <= Compteur_programme;
                ETAT <= fetch;


                        -- Saut
        WHEN execute_jump =>
                Registre_adresse_memoire    <= Registre_instruction( 7 DOWNTO 0 );
                Compteur_programme          <= Registre_instruction( 7 DOWNTO 0 );
                ETAT                        <= fetch;
        WHEN OTHERS =>
```

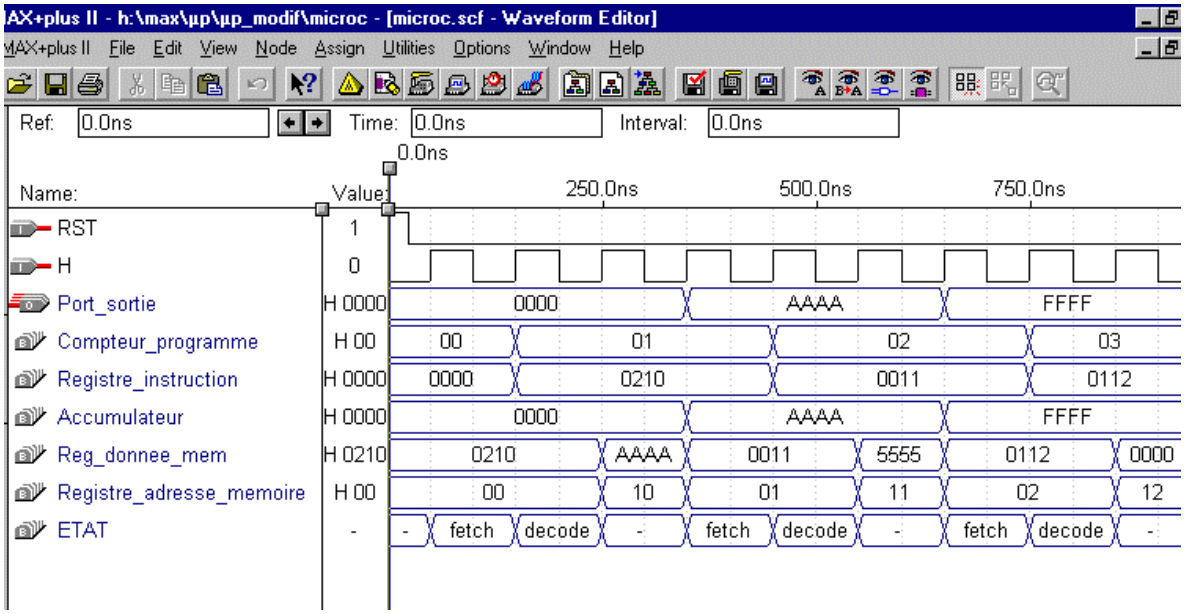
```
                        Registre_adresse_memoire      <= Compteur_programme;
                        ETAT <= fetch;
                END CASE;
        END IF;
  END PROCESS;
END ARCH;
```

On se propose alors d'exécuter le programme suivant, chargé dans « program.mif ». Celui-ci pourra être modifier à volonté avec un éditeur de texte quelconque.

```
DEPTH = 256;            % nombre de mots de la mémoire (en décimal)  %
WIDTH = 16;             % largeur d'un mot (en décimal)%

ADDRESS_RADIX = HEX;                % format des adresses en hexadécimal %
DATA_RADIX = HEX;                   % format des données en hexadécimal  %
                                    % hexadécimal par défaut        %

-- spécification des données à chaque adresse
        CONTENT
        BEGIN
        [00..FF]:       0000;   % initialisation de la mémoire à  0000    %
        00      :       0210;   % LDA 10, chargement dans A avec MEM(10) %
        01      :       0011;   % ADD 11, addition de A avec MEM(11), resultat dans A %
        02      :       0112;   % ST 12, chargement de          A à  MEM(12) %
        03      :       0212;   % LDA12, chargement dans A avec MEM(12), pour vérification %
        04      :       0304;   % JUMP04, saut à 04, boucle infinie %
        10      :       AAAA;   % Donnée %
        11      :       5555;   % Donnée %
        12      :       0000;   % Donnée %
END ;
```

Télécharger et compiler la description et le programme à partir de la disquette ou du cdrom fourni, puis effectuer compilation et simulation fonctionnelle.

On doit normalement obtenir des chronogrammes proches de la figure ci-après :



Etudier ces chronogrammes et vérifier qu'il s'agit bien du résultat attendu.

Modification

Tester maintenant un autre programme, effectuant l'addition de trois nombres stockés en mémoire. Evaluer les performances en terme de vitesse de notre microcontrôleur, implanté dans le Flex10K20.

Ajouter des instructions, par exemple saut conditionnel, opération arithmétiques et logiques etc…

Tester le fonctionnement sur des programmes plus complexes. On rappelle que le Flex10K20 de la carte est relié à deux afficheurs, un connecteur VGA, un connecteur pour souris ou clavier.

Bibliographie

Rapid Prototyping of Digital System par J.O. Hamblen et M.D. Furman chez Kluwer Academic Publishers.
Logique programmable par L. Dutrieux et D. Demigny chez Eyrolles