Valentin LABRUNE
Gabin Kalbusch
L3 2025 - Groupe B

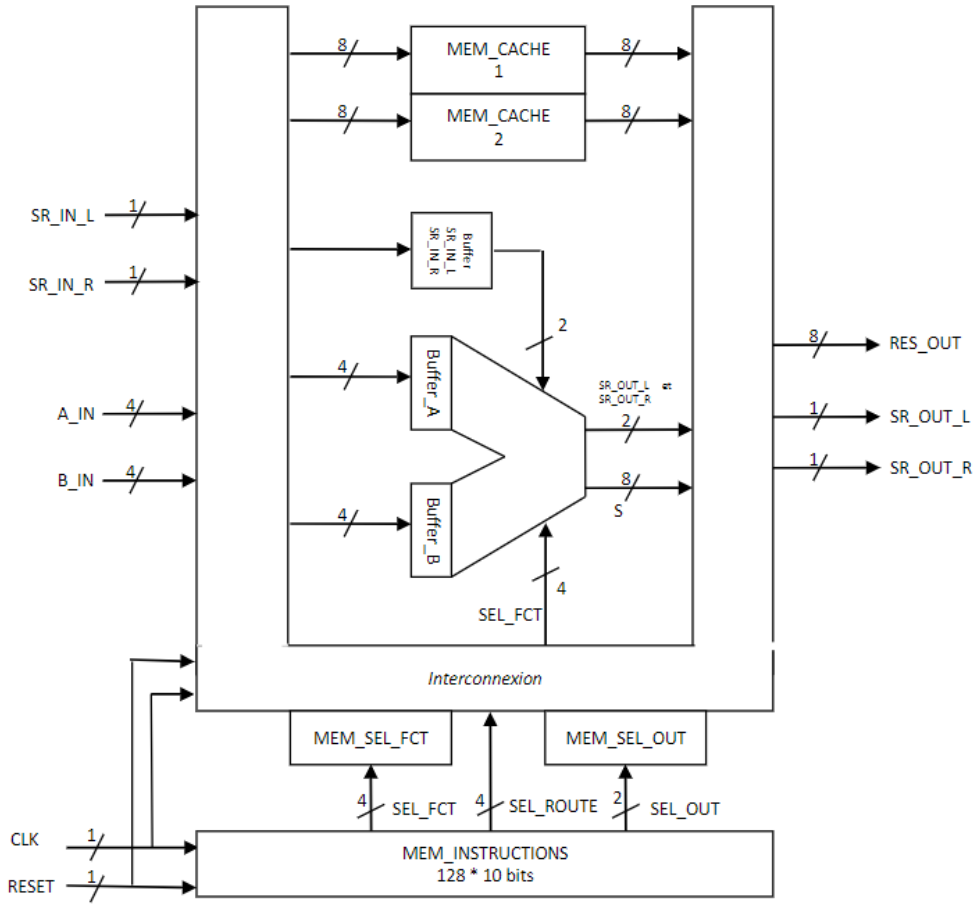
eFrei
PARIS PANTHÉON-ASSAS UNIVERSITÉ

Conception de circuit numérique
“Mini Projet”
Microcontrôleur en VHDL

Enseignant : Didier Meier

Présentation :

L'objectif de ce projet est de réaliser un coeur de microcontrôleur constitué d'une ALU (Unité de traitement arithmétique et logique ou Logic and Arithmetic Unit en anglais) , de mémoires internes utilisées pour les calculs et de mémoire utilisées pour les instructions. Par le biais de séances projet ainsi que de les notions vues durant le module.



Sur ce schéma, nous pouvons distinguer :

- L'ALU : c'est le sorte de V à l'horizontal
- les mémoires de calculs : BufferA, BufferB, Buffer SR_IN_L et SR_IN_R, Mem_cache1 Mem_cache2
- les mémoires d'instruction Mem_SEL_FCT, Mem_SEL_OUT et MEM_INSTRUCTIONS qui contient toutes les instructions que nous souhaitons effectuer

I-ALU :

Tout d'abord, intéressons- nous à L'ALU. Il correspond au fichier ALU Compo.vhd

Notre ALU possède 5 signaux d'entrées et 3 signaux de sorties.
Penchons nous vers les entrées :

- Les entrées A et B sont 2 entrées codées codées sur 4 bits. Notre ALU effectue ses calculs principalement sur ces 2 entrées
- Les entrées SR_IN_L et SR_IN_R sont les valeurs de retenues d'entrées, elles ne sont pas utilisés dans tous les calculs de l'ALU
- Enfin, l'entrée SEL_FCT codées sur 4 bits correspond à l'opération qui va être effectuée par l'ALU. En effet, selon la valeur du signal SEL_FCT, telle ou telle opération va être effectuée.

Maintenant, qu'en est il des sortie :

- S est le résultat de l'opération effectué par l'ALU
- SR_OUT_L et SR_OUT_R sont les retenues de sorties de l'opération

Dans la première partie du fichier ALU Compo.vhd, nous avons l'entité ALU Compo, dans celle-ci, nous définissons les ports d'entrées et de sorties de l'entité, ainsi que les types et la taille des ports. (voir image ci-dessous)

```
entity ALUCompo is
port(
    -- entrées
    A : in std_logic_vector(3 downto 0);
    B : in std_logic_vector(3 downto 0);
    SR_IN_L : in std_logic := '0';
    SR_IN_R : in std_logic := '0';

    -- sel fct
    SEL_FCT : in std_logic_vector(3 downto 0);

    -- sorties
    S : out std_logic_vector(7 downto 0);
    SR_OUT_L : out std_logic := '0';
    SR_OUT_R : out std_logic := '0'
);
end ALUCompo;
```

Ensuite, nous créons l'architecture de notre entité, on déclare d'abord tous les signaux internes.

```
signal My_A, My_B : std_logic_vector(3 downto 0);
signal My_S : std_logic_vector(7 downto   0);
signal My_SR_IN_L, My_SR_IN_R, My_SR_OUT_L, My_SR_OUT_R : std_lo
signal My_SEL_FCT : std_logic_vector(3 downto 0);
```

Ici, nous voyons qu'ils sont inutiles et ne sont pas utilisés puisqu'il n'y pas besoin de relier des composants au sein même de cette entité.

Ensuite, dans l'architecture toujours, nous définissons le process de l'ALU qui dépend donc de SEL_FCT

```
begin

    ALUCompo_Proc : process(A, B, SR_IN_L, SR_IN_R, SEL_FCT)
        variable S_var, A_var, B_var : std_logic_vector(7 downto 0);
    begin
        case SEL_FCT is
            when "0000" => -- no op : toutes les sorties a 0
                S <= "00000000";
                SR_OUT_L <= '0';
                SR_OUT_R <= '0';

            when "0001" => -- S = décallage à droite B sur 4 bits, SR_IN_L
                S(7 downto 4) <= (others => '0');
                S(3) <= SR_IN_L;
                S(2 downto 0) <= B(3 downto 1);
                SR_OUT_R <= B(0);
                SR_OUT_L <= '0';

            when "0010" => -- S = décallage à gauche B sur 4 bits, SR_IN_R
                S(7 downto 4) <= (others => '0');
                S(0) <= SR_IN_R;
```

Les tests de ce design sont réalisés dans le testbench ALU_testbench.vhd

Les Buffers :

Correspondent aux fichier bufferNbits.vhd et bufferNbitsWCE.vhd

Dans ce projet, nous avons utilisés 2 types de buffers : Un avec et l'autre sans CE (Chip enabler)
Le chip enabler est une sécurité, il n'est possible de modifier la valeur du buffer que lorsque le CE est activé. Ainsi, par défaut, les CE de tous les buffers sont nuls et seulement lorsque l'on veut assigner une valeur à un buffer à active son CE.
D'autres part, nous avons créé nos buffers avec l'attribut générique, qui nous permet facilement de déclarer des buffers de taille différentes

Pour ce qui est des ports de l'entité, rien de sorcier

```
entity bufferNbitsWCE is -- buffer N bits with chip enabler
    generic(
        N : integer
    );
    port (
        e : in std_logic_vector (N-1 downto 0);
        ce : in std_logic;
        reset : in std_logic;
        clock : in std_logic;
        s : out std_logic_vector (N-1 downto 0)
    );
end bufferNbitsWCE;
```

```vhdl
entity bufferNbits is
    generic(
        N : integer
    );
    port (
        e : in std_logic_vector (N-1 downto 0);
        reset : in std_logic;
        clock : in std_logic;
        s : out std_logic_vector (N-1 downto 0)
    );
end bufferNbits;
```

Chaque buffer possède une entrée et une sortie de la taille du buffer, une clock et un reset. On ajoute juste un port pour le CE dans le buffer avec le CE.

Pour le process des buffers :
Cela consiste seulement à copier l'entrée sur la sortie lors du front montant de la clock.

```vhdl
BufferNbitsProc : process (reset, clock)
begin
    if (reset = '1') then
        s <= (others => '0');
    elsif (rising_edge(clock)) then
        s <= e;
    end if;
end process;
```

La particularité du buffer avec le CE, c'est que le CE doit être activé lors du front montant pour que l'entrée soit copiée sur la sortie.

```vhdl
BufferNbitsProc : process (reset, clock)
begin
    if (reset = '1') then
        s <= (others => '0');
    elsif (rising_edge(clock)) then
        report "clock " & std_logic'image(clock);
        if (ce = '1') then
            report "ce here = " & std_logic'image(ce);
            s <= e;
        end if;
    end if;
end process;
```

Les tests de ce buffer sont dans les fichiers : bufferNbits_testbench.vhd et bufferNbitsWCE_testbench.chd

La Big Unit :

Correspond au fichier BigUnit.vhd

Attaquons nous maintenant à la partie que j'ai trouvé la plus dure du projet : La BigUnit.
Ce que j'entends par Big unit, c'est l'interconnexion entre l'ALU, les buffers de calculs, le signal de SEL_FCT, SEl_ROUTE et SEL_OUT.

Au niveau des ports de l'entité : (il y en a beaucoup)

- 4 entrés de valeurs : A_IN, B_IN, SR_IN_L_G et SR_IN_R_G
- 3 entrées d'intructions : SEL_FCT_IN, SEL_ROUTE_IN et SEL_OUT_IN
- 2 entrées clock et reset
- 3 sortie de valeurs : RES_OUT_G, SR_OUT_L_G et SR_OUT_R_G

```
entity BigUnit is
    port(
        -- entrées
        A_IN : in std_logic_vector (3 downto 0);
        B_IN : in std_logic_vector (3 downto 0);
        SR_IN_L_G : in std_logic;
        SR_IN_R_G : in std_logic;

        -- SEL
        SEL_FCT_IN : in std_logic_vector (3 downto 0);
        SEL_ROUTE_IN : in std_logic_vector (3 downto 0);
        SEL_OUT_IN : in std_logic_vector (1 downto 0);

        -- others
        clock_IN : in std_logic;
        reset_IN : in std_logic;

        -- sorties
        SR_OUT_L_G : out std_logic;
        SR_OUT_R_G : out std_logic;
        RES_OUT_G : out std_logic_vector (7 downto 0)
    );
end BigUnit;
```

Dans l'architecture, nous déclarons tous les composants utilisé dans l'entité BigUnit, c'est à dire l'ALU, et nos 2 différents types de buffers.

```
architecture BigUnit_Arch of BigUnit is

    -- components

    component ALUCompo is
        port(
            A : in std_logic_vector (3 downto 0);
            B : in std_logic_vector (3 downto 0);
            SR_IN_L : in std_logic;
            SR_IN_R : in std_logic;

            SEL_FCT : in std_logic_vector (3 downto 0);

            S : out std_logic_vector (7 downto 0);
            SR_OUT_L : out std_logic;
            SR_OUT_R : out std_logic
        );
    end component;

    component bufferNbits is
        generic(
            N : integer
        );
        port(
            e : in std_logic_vector(N-1 downto 0);
            reset : in std_logic;
            clock : in std_logic;
            s : out std_logic_vector(N-1 downto 0)
        );
    end component;

    component bufferNbitsWCE is
        generic(
            N : integer
        );
```

Puis, nous créons tous les signaux internes essentiels pour lier les composants de l'entité.

```
-- liste signaux
-- Buffers : A, B, SR_IN_L, SR_IN_R
signal My_BufferA_IN, My_BufferB_IN : std_logic_vector (3 downto 0);
signal My_BufferA_OUT, My_BufferB_OUT : std_logic_vector (3 downto 0);
signal My_BufferA_ce, My_BufferB_ce : std_logic;
signal My_BufferSR_IN_L_OUT, My_BufferSR_IN_R_OUT : std_logic_vector (0 downto 0);

-- Memoire : Mem_cache1, Mem_cache2
signal My_Mem_cache1_IN, My_Mem_cache2_IN : std_logic_vector (7 downto 0);
signal My_Mem_cache1_OUT, My_Mem_cache2_OUT : std_logic_vector (7 downto 0);
signal My_Mem_cache1_ce, My_Mem_cache2_ce : std_logic;

-- ALU
signal My_A, My_B : std_logic_vector (3 downto 0);
signal My_SR_IN_L, My_SR_IN_R : std_logic;
signal My_SR_OUT_L, My_SR_OUT_R : std_logic;
signal My_RES_OUT_G : std_logic_vector (7 downto 0);
```

Puis nous associons, tous les ports des composants aux ports de l'entité et aux signaux internes.

```
begin

    BufferA : bufferNbitsWCE
        generic map (N => 4)
        port map(
            e => My_BufferA_IN,
            ce => My_BufferA_ce,
            reset => reset_IN,
            clock => clock_IN,
            s => My_BufferA_OUT
        );

    BufferB : bufferNbitsWCE
        generic map (N => 4)
        port map(
            e => My_BufferB_IN,
            ce => My_BufferB_ce,
            reset => reset_IN,
            clock => clock_IN,
            s => My_BufferB_OUT
        );

    BufferSR_L : bufferNbits
        generic map (N => 1)
        port map(
            e(0) => SR_IN_L_G,
            reset => reset_IN,
            clock => clock_IN,
```

Enfin, dans le process de l'entité, nous effectuons le routage des signaux en fonction de SEL_ROUTE. En effet, comme pour SEL_FCT dans l'ALU, en fonction de SEL_ROUTE, les données seront dirigées dans tel ou tel composant. En outre, la sortie RES_OUT est déterminée par la fonction d'instruction SEL_OUT, qui comme SEL_ROUTE est appelé dans le process de l'entité.

```
routingProc : process(clock_IN, reset_IN)
begin
    if (rising_edge(clock_IN)) then
        case SEL_ROUTE_IN is
            when "0000" => -- stockage de S dans Mem_Cache1
                My_BufferA_ce <= '0';
                My_BufferB_ce <= '0';
                My_Mem_cache1_ce <= '1';
```

```
    end case;
end if;
if rising_edge(clock_IN) then
    case SEL_OUT_IN is
        when "00" => -- RES_OUT = 0
            RES_OUT_G <= (others => '0');

        when "01" => -- RES_OUT = Mem_cache1
            RES_OUT_G <= My_Mem_cache1_OUT;
```

Les tests ont été effectués dans le fichier BigUnit_testbench.vhd mais le résultat n'est pas concluant.
Pour être honnête, nous avons fait ce que nous avons pu pour cette partie, nous avons créé un design et un testbench mais a chaque test le résultat RES_OUT qu'on obtient a la fin était égal à 0.

```
[2023-04-13 11:22:43 EDT] ghdl -i '--ieee=synopsys' '-fexplicit' design.vhd testbench.vhd  && ghdl -m '--ieee=synops
design.vhd:95:9:warning: instance "buffera" of component "buffernbitswce" is not bound [-Wbinding]
design.vhd:30:14:warning: (in default configuration of bigunit(bigunit_arch))
design.vhd:105:9:warning: instance "bufferb" of component "buffernbitswce" is not bound [-Wbinding]
design.vhd:30:14:warning: (in default configuration of bigunit(bigunit_arch))
design.vhd:115:9:warning: instance "buffersr_l" of component "buffernbits" is not bound [-Wbinding]
design.vhd:30:14:warning: (in default configuration of bigunit(bigunit_arch))
design.vhd:124:9:warning: instance "buffersr_r" of component "buffernbits" is not bound [-Wbinding]
design.vhd:30:14:warning: (in default configuration of bigunit(bigunit_arch))
design.vhd:133:9:warning: instance "mem_cache1" of component "buffernbitswce" is not bound [-Wbinding]
design.vhd:30:14:warning: (in default configuration of bigunit(bigunit_arch))
design.vhd:143:9:warning: instance "mem_cache2" of component "buffernbitswce" is not bound [-Wbinding]
```

Comme vous nous l'avez dit, nous avons vérifié tous nos noms au niveau de la déclaration et de l'instanciation mais nous n'avons pas réussi à résoudre le problème de n'importe quelle façon que ce soit. De ce fait, Une metavalue est détectée et 0 est retourné pour le résultat.

```
testbench.vhd:75:13:@0ms:(report note):  entrée :
testbench.vhd:76:13:@0ms:(report note): A_IN = 2 , B_IN = 6 , SR_IN_L = '0' , SR_IN_R = '0'
testbench.vhd:77:13:@0ms:(report note):  sel :
testbench.vhd:78:13:@0ms:(report note): SEL_FCT_IN = 15 , SEL_ROUTE_IN = 7 , SEL_OUT_IN = 3
testbench.vhd:79:13:@0ms:(report note):  sortie :
../../src/ieee/v93/numeric_std-body.vhdl:2098:7:@0ms:(assertion warning): NUMERIC_STD.TO_INTEGER: metavalue detected, returning 0
testbench.vhd:80:13:@0ms:(report note): RES_OUT = 0 , SR_OUT_L = 'U' , SR_OUT_R = 'U'
Finding VCD file...
```

Mem_Instruction :

correspond au fichier MemInstruction.vhd

L'entité MemInstruction est composé de 5 ports :

- 2 entrées correspondants a reset et clock
- 3 sorties correspondant aux fonctions d'instructions

```vhdl
entity MemInstruction is
    port(
        clock_IN : in std_logic;
        reset_IN : in std_logic;

        SEL_FCT_OUT : out std_logic_vector (3 downto 0);
        SEL_ROUTE_OUT : out std_logic_vector (3 downto 0);
        SEL_OUT_OUT : out std_logic_vector (1 downto 0)
    );
end MemInstruction;
```

Pour faire simple Mem_Instruction est le bloc mémoire de notre microcontroleur.
En effet, nous déclarons un tableau de 128 éléments de 10bits chacun. Ainsi, qu'un pointeur (un entier compris entre 0 et 128, initialisé à 0)

```vhdl
-- memoire
type memory is array (0 to 127) of std_logic_vector (9 downto 0);
signal pointeur : integer range 0 to 127 := 0;

constant MemInstruction : memory := (
    ("0000000000"),  -- pointeur : 0

    -- operation 1      You, 5 hours ago • tout codé mais avec des bu
    ("0000011100"), -- pointeur : 1
    ("0000111100"), -- 2
    ("1101000011"), -- 3

    -- operation 2
    ("0000011100"), -- 4
    ("0000111100"), -- 5
    ("1011000000"), -- 6
    ("0000100000"), -- 7
    ("0000011100"), -- 8
    ("0101000000"), -- 9
    ("0000010100"), -- 10
    ("0000100100"), -- 11
    ("1001000011"), -- 12

    -- opération 3

    ("0000011100"), -- 13
    ("0000111100"), -- 14
    ("0111000000"), -- 15
    ("0000100000"), -- 16
    ("0000011100"), -- 17
```

Chaque élément du tableau correspond à un bloc d'instruction et le pointeur pointe un élément du tableau.
Dans le process de MemInstruction, à chaque coup de clock, le pointeur est incrémenté de 1 et l'élément du tableau est divisé en 3 instructions simultanées : les 4 bits de poids forts correspondent à l'instruction SEL_FCT, les 2 bits de poids faibles correspondent à l'instruction SEL_OUT et les 4 bits restants correspondent à l'instruction SEL_ROUTE.

Les tests ont été fait dans MemInstruction_testbench.vhd.

Le microcontrôleur :

correspond au fichier microcontroleur.vhd

Il s'agit de l'entité de plus haut niveau de notre programme. De ce fait les ports sont les même que ceux du schéma tout en haut de rapport.
Ainsi, nos ports sont :

- 4 entrées de valeurs : A_IN, B_IN, SR_IN_L et SR_IN_R
- 2 entrées clock et reset
- 3 sorties de valeurs : RES_OUT, SR_OUT_L, SR_OUT_R

```
entity Microcontroleur is
port(
    A_IN_G : in std_logic_vector (3 downto 0);
    B_IN_G : in std_logic_vector (3 downto 0);
    SR_IN_L : in std_logic;
    SR_IN_R : in std_logic;

    clock_G : in std_logic;
    reset_G : in std_logic;

    RES_OUT : out std_logic_vector (7 downto 0);
    SR_OUT_L : out std_logic;
    SR_OUT_R : out std_logic
);
end Microcontroleur;
```

Pour notre architecture, nous appelons donc les composants BigUnit, BufferNbits et MemInstruction. Nous créons également les signaux internes afin de relier les différents composants ensemble et enfin, nous assignons les ports des composants aux ports de l'entité et aux signaux internes (comme pour BigUnit).

```
architecture Microcontroleur_Arch of Microcontroleur is

    component MemInstruction is
        port (
            reset_IN : std_logic;
            clock_IN : std_logic;

            SEL_FCT_OUT : std_logic_vector (3 downto 0);
            SEL_ROUTE_OUT : std_logic_vector (3 downto 0);
            SEL_OUT_OUT : std_logic_vector (1 downto 0)
            );
    end component;

    component BigUnit is
        port(
            A_IN : in std_logic_vector (3 downto 0);
            B_IN : in std_logic_vector (3 downto 0);
            SR_IN_L_G : in std_logic;
            SR_IN_R_G : in std_logic;

            SEL_FCT_IN : in std_logic_vector (3 downto 0);
            SEL_ROUTE_IN : in std_logic_vector (3 downto 0);
            SEL_OUT_IN : in std_logic_vector (1 downto 0);
```

```
-- signaux
-- signaux sortant du mem instruction
signal SEL_FCT_OUT_MEM, SEL_ROUTE_OUT_MEM : std_logic_vector(3 downto 0);
signal SEL_OUT_OUT_MEM : std_logic_vector(1 downto 0);

-- signaux entrant et sortant des buffers Selfct et sel out
signal SEL_FCT_IN_BUF, SEL_FCT_OUT_BUF : std_logic_vector(3 downto 0);
signal SEL_OUT_IN_BUF, SEL_OUT_OUT_BUF : std_logic_vector(1 downto 0);

-- signaux entrant dans le big unit
signal SEL_FCT_IN_BU, SEL_ROUTE_IN_BU : std_logic_vector(3 downto 0);
signal SEL_OUT_IN_BU : std_logic_vector(1 downto 0);


begin

    Memoire : MemInstruction
        port map(
            reset_IN => reset_G ,
            clock_IN => clock_G,
            SEL_FCT_OUT => SEL_FCT_OUT_MEM,
            SEL_ROUTE_OUT => SEL_ROUTE_OUT_MEM,
            SEL_OUT_OUT => SEL_OUT_OUT_MEM
        );

    MEM_SEL_FCT : bufferNbits
        generic map (N => 4)
        port map(
            e => SEL_FCT_IN_BUFF,
            reset => reset_G,
            clock => clock_G,
            s => SEL_OUT_OUT_BUFF
        );

    MEM_SEL_OUT : bufferNbits
```

Nous n'avons donc pas pu tester notre design de microcontroleur puisque notre BigUnit ne sont pas concluants.
Cependant, nous avons tout de même créé un testbench "microcontroleur_testbench.vhd" qui nous semble fonctionnel

Les tests :

Tous les tests des entités de bases de marche : Buffers, ALU, MemInstruction. Malheureusement, dès que nous appelons un composant dans nos entités, cela ne marche plus (peut être (voire sûrement) une erreur dans l'appel de nos composants). Ce qui ne nous permet pas de tester les entités les plus intéressantes : Big Unit et Microcontroleur.
De ce fait, nous n'avons pas fait l'implémentation et l'intégration sur la carte de développement.

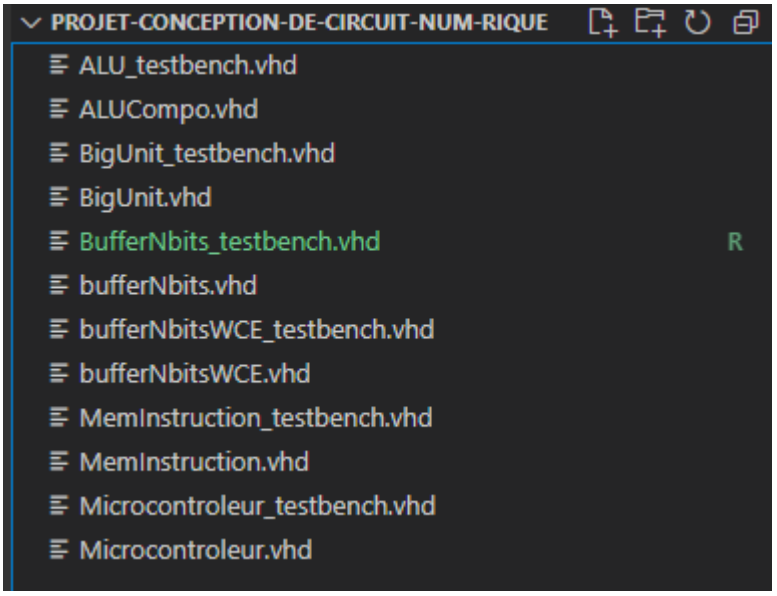
Conclusion :

Pour conclure, afin de réaliser notre microcontroleur en VHDL, nous avons divisé notre travail en 6 différentes entités, qui ont toutes un rôle spécifique dans le projet :

- L'ALU, qui est le coeur de calcul du microcontroleur
- les buffers avec ou sans Chip Enabler, qui permette de stocker les données sur lesquelles nous travaillons
- le MemInstrunction, qui est notre bloc mémoire d'instruction
- La BigUnit, qui est l'interconnexion entre les buffers et l'ALU
- et enfin le Microcontroleur qui est l'entité de plus haut niveau.

Ce projet nous aura permis de mieux comprendre et de nous exercer sur le langage VHDL. C'est un langage difficile et qui nécessite beaucoup de concentration, d'autant plus que certains messages d'erreur n'indiquent pas réellement quelle erreur a été commise.

Voici donc la liste de nos fichiers :



que vous pouvez retrouver sur le repository github suivant :
https://github.com/ValentinLabrune/Projet-Conception-de-circuit-num-rique

Valentin Labrune :

“Pour ma part, je trouve que la matière est très intéressante d’un point de vue technologique. Cependant, étant intéressé par la majeure Software Engineering, j’ai du mal à comprendre pourquoi elle n’est pas enseigné dans un cours électif, puisqu’elle s’addresse principalement aux personnes souhaitant faire du système embarqué. J’ai pris du plaisir à faire ce projet avec mon camarade, malgré la contrainte de temps qui pesait sur nos épaules.

D’autre part, j’aimerais vraiment remercié Didier Meier, pour le temps qu’il a passé à nous expliquer et nous aider. On sent qu’il est passionné par ce qu’il fait et cela se ressent dans sa manière d’enseigner (et ce n’est pas tous les jours qu’on voit ca).

Enfin, nous espérons que notre projet répondra un minimum à vos attentes malgré ses petits défauts.”