SOLUTION EN VHDL

Jorge Luiz MAYORQUIM

Cergy, 24 Juin 2003
EISTI

SOLUTION EN VHDL

Jorge Luiz MAYORQUIM

Cergy, France

1

SOLUTION DE L'EXAMEN EN VHDL

ANALYSE

L'objectif de l'examen est de vérifier les connaissances élémentaires du langage *VHDL*. Les questions ont été séparées en deux groupes : - Algorithmes nouveaux et travaux pratiques. Le niveau de difficulté des questions d'algorithmes nouveaux a été attaché aux concepts du langage *VHDL* et de l'algorithmique, et la pratique des autres langages, comme par exemple *C, C++, ADA, BASIC*.

— La première question a été basée sur la conception de base de la communication de données i.e., les ordinateurs travaillent avec un bus de données en parallèle et pour la communication les données sont sérielles. Ce problème, les élevés le rencontre tous les jours, et la grande difficulté de la question est d'élaborer un algorithme et ensuite de faire le script du problème.

— La deuxième question a été basée sur des connaissances élémentaires de la logique et un niveau de difficulté en algorithme très bas.

— La troisième question a été basée sur des connaissances rudimentaires d'architecture de systèmes informatiques, algorithme et circuit numériques (complément à 2 et complément à 1).

Pour les questions des travaux pratiques, il n'y a pas besoin de faire un commentaire.

1.1 Question 1. (3 points)

Écrire un programme en *VHDL* d'un système qui fait la conversion de données sérielles vers parallèle avec les caractéristiques suivantes :

— Conversion de 8 bits synchrones ;

— Reset asynchrone ;

— Le flot de données d'entrée doit être sériel ;

— La sortie sera présentée par un byte (8 bits).

Script 1.1.1 *Réponse de la question 1.*

```
01.  Library IEEE;
02.  USE IEEE.STD_LOGIC_1164.ALL;
03.  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
04.  USE IEEE.STD_LOGIC_ARITH.ALL;
05.
06.  ENTITY SERIE_PARALLELE IS
07.     PORT
08.          (Reset          : IN  std_logic;
09.           Clock          : IN  std_logic;
10.           Data_entree    : IN  std_logic;
11.           Data_sortie    : OUT std_logic_vector (7 downto 0)
12.           );
13.
14.  END SERIE_PARALLELE;
15.
16.  ARCHITECTURE ARCH OF SERIE_PARALLELE IS
17.  signal s_reg    : std_logic_vector (7 downto 0);
18.  signal s_compte : std_logic_vector (2 downto 0);
19.  Begin
20.   Deplacement : Process (Clock, Reset)
21.                 Begin
22.                  If (Reset = '1') Then
23.                    s_reg <= "00000000";
24.                  Elsif (Clock'event and Clock = '1') Then
25.                    s_reg <= Data_entree & s_reg (7 downto 1);
26.                  End If;
27.                 End Process Deplacement;
28.
29.   Compter :     Process (Clock, Reset)
30.                 Begin
31.                  If (Reset = '1') Then
32.                    s_compte <= "000";
33.                  Elsif (Clock'event and Clock = '1') Then
34.                    s_compte <= s_compte +1;
35.                  End If;
36.                 End Process Compter;
37.                 Data_sortie <= s_reg When s_compte = "111"
38.                                     Else "00000000";
39.  End ARCH;
```

1.2 Question 2. (3 points)

Écrire un programme en *VHDL* d'un compteur avec les caractéristiques suivantes :

— 8 bits synchrones ;

— Reset asynchrone ;

— Sélection entre le comptage *Up* et *Down* :

(1) Sélection par le signal *Dir* ;
(2) Incrément si $Dir = 1$;
(3) Décrément si $Dir = 0$.

Script 1.2.1 *Réponse de la question 2.*

```
01.  Library IEEE;
02.  USE IEEE.STD_LOGIC_1164.ALL;
03.  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
04.  USE IEEE.STD_LOGIC_ARITH.ALL;
05.
06.  ENTITY COMPTEUR IS
07.     PORT
08.         (Reset          : IN  std_logic;
09.          Clock          : IN  std_logic;
10.          Dir            : IN  std_logic;
11.          Compteur_out   : OUT std_logic_vector (7 downto 0)
12.          );
13.
14.  END COMPTEUR;
15.
16.  ARCHITECTURE ARCH OF COMPTEUR IS
17.  signal s_compte : std_logic_vector (7 downto 0);
18.
19.  Begin
20.   Compteur_Reg : Process (Clock, Reset)
21.                  Begin
22.                   If (Reset = '1') Then
23.                     s_compte <= "00000000";
24.                   Elsif (Clock'event and Clock = '1') Then
25.                     If dir = '1' Then
26.                       s_compte <= s_compte +1;
27.                     Else
28.                       s_compte <= s_compte - 1;
29.                     End If;
30.                   End If;
31.                  End Process Compteur_Reg;
32.                  Compteur_out <= s_compte;
33.  End ARCH;
```

1.3 Question 3. (4 points)

Écrire un programme en *VHDL* d'un accimulateur avec les caractéristiques suivantes :

— Accumulateur de 8 bits ;

— Reset synchrone ;

— Les opérations sur la donnée d'entrée peuvent être les suivantes :

(1) Si l'entrée $Ctl =$ "00" l'opération est l'addition ;

(2) Si l'entrée Ctl = "01" l'opération est la soustraction ;
(3) Si l'entrée Ctl = "10" l'opération est le complément à 2 de l'entrée ;
(4) Si l'entrée Ctl = "11" l'opération est le complément à 1 de l'entrée ;

Script 1.3.1 *Réponse de la question 3.*

```
01.  Library IEEE;
02.  USE IEEE.STD_LOGIC_1164.ALL;
03.  USE IEEE.STD_LOGIC_UNSIGNED.ALL;
04.  USE IEEE.STD_LOGIC_ARITH.ALL;
05.
06.  ENTITY ACCUMULATEUR IS
07.     PORT
08.          (Reset          : IN  std_logic;
09.           Clock          : IN  std_logic;
10.           Ctl            : IN  std_logic_vector (1 downto 0);
11.           Data_entree    : IN  std_logic_vector (3 downto 0);
12.           Data_sortie    : OUT std_logic_vector (7 downto 0)
13.           );
14.  END ACCUMULATEUR;
15.
16.  ARCHITECTURE ARCH OF ACCUMULATEUR IS
17.  signal s_acc : std_logic_vector (7 downto 0);
18.
19.  Begin
20.   P_ACCUMULATEUR : Process (Clock, Reset)
21.                    Begin
22.                     If (Reset = '1') Then
23.                       s_acc <= "00000000";
24.                     Elsif (Clock'event and Clock = '1') Then
25.                       Case Ctl IS
26.                         When "00"   => s_acc <= s_acc    +Data_entree;
27.                         When "01"   => s_acc <= s_acc - Data_entree;
28.                         When "10"   => s_acc <= Not(Data_entree) +1;
29.                         When "11"   => s_acc <= "0000" & Not(Data_entree);
30.                         When Others => s_acc <= "00000000";
31.                       End Case;
32.                     End If;
33.                    End Process P_ACCUMULATEUR;
34.                    Data_sortie <= s_acc;
35.  End ARCH;
```

1.4 Question 4. (2 points) - TP 1

Écrire un programme en *VHDl* avec le niveau d'abstraction structurelle d'un additionneur de 4 bits ave l'instruction *FOR GENERATE*. Il faut faire l'hypothèse que vous avez le component *add*_1comme l'illustre le Script 1.4.1.

Script 1.4.1 *Component add_1*

```
01. COMPONENT ADD_1
02. PORT
03.    (a            : IN  std_logic;
04.     b            : IN  std_logic;
05.     ret_entree   : IN  std_logic;
06.     sortie       : OUT std_logic;
07.     ret_sortie   : OUT std_logic
08.     );
09. END COMPONENT;
```

Script 1.4.2 *Réponse de la question 4.*

```
01. Library IEEE;
02. LIBRARY WORK;
03. USE IEEE.STD_LOGIC_1164.ALL;
04. USE WORK.ALL;
05.
06. ENTITY FULL_ADD IS
07.  PORT
08.     (
09.      a               : IN    STD_LOGIC_VECTOR (3 DOWNTO 0);
10.      b               : IN    STD_LOGIC_VECTOR (3 DOWNTO 0);
11.      ret_entree      : IN    STD_LOGIC;
12.      sortie          : OUT   STD_LOGIC_VECTOR (3 DOWNTO 0);
13.      ret_sortie      : OUT   STD_LOGIC
14.      );
15.  END FULL_ADD;
16.  ARCHITECTURE arch OF FULL_ADD IS
17.   COMPONENT ADD_1
18.   PORT
19.     (
20.      a               : IN    STD_LOGIC;
21.      b               : IN    STD_LOGIC;
22.      ret_entree      : IN    STD_LOGIC;
23.      sortie          : OUT   STD_LOGIC;
24.      ret_sortie      : OUT   STD_LOGIC
25.      );
26.   END COMPONENT;
27.
28.   FOR ALL: ADD_1 USE ENTITY WORK.ADD_1;
29.    SIGNAL temp  : std_logic_vector (0 to 2);
30.    BENGIN
31.     C0: ADD_1 PORT MAP(a(0), b(0), ret_entree, sortie(0), temp(0));
32.     C1TO2: FOR i IN 1 TO 2 GENERATE
33.              C: ADD_1 PORT MAP(a(i), b(i), temp(i-1), sortie(i), temp(i));
34.            END GENERATE;
35.     C3: ADD_1 PORT MAP(a(3), b(3), temp(2), sortie(3), ret_sortie);
36.  END arch;
```

1.5 Question 5. (2 points) - TP 2

Écrire un programme en *VHDL* qui génère la transition individuelle pour les registres (faux horloge) de la mémoire asynchrone.

Script 1.5.1 *Réponse de la question 5.*

```
-- Programme complet : mayo_mem.vhd (mémoire de 4 mots).
-- L'entrée a et web est définit dans l'entity.
-- Les signaux sont définits dans l'architecture.

a                       : in std_logic_vector (1 downto 0);
web                     : in std_logic;
signal w0, w1, w2, w3   : std_logic;
signal w                : std_logic_vector (3 downto 0);
Begin

 w <= "1110" WHEN ((a="00") AND (web='0')) ELSE
      "1101" WHEN ((a="01") AND (web='0')) ELSE
      "1011" WHEN ((a="10") AND (web='0')) ELSE
      "0111" WHEN ((a="11") AND (web='0')) ELSE
      "1111";

 w0 <= w(0);
 w1 <= w(1);
 w2 <= w(2);
 w3 <= w(3);
```

1.6 Question 6. (2 points) - TP 3

Écrire un programme en *VHDL* qui génère un package qui a l'élément suivant :

— Fonction pour déterminer la parité (mots) d'une entrée qui est du type std_logic_vector.

Script 1.6.1 *Réponse de la question 6.*

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

Package EXAME IS
   TYPE integers is array (0 to 255) of integer;
   Function test_parite (a: std_logic_vector) return std_logic;
End EXAME;

Package Body EXAME IS

Function test_parite (a : std_logic_vector) Return std_logic Is
 Variable temp   : std_logic := '0';
 Begin
  For i IN a'Range Loop
   temp := temp XOR a(i);
  End Loop;
 Return temp;
End test_parite;

 end EXAME;
```

1.7 Question 7. (4 points) - TP 4

Écrire le programme en *VHDL* pour le microprosseur *MP2500* qui exécute :

— Le process de l'ULA qui contient l'opération *XOR*.

Script 1.7.1 *Réponse de la question 7.*

```
01.  Programme complet : mayo_micro_h.vhd (microprocessuer MP2005).
02.  L_ALU : Process (alu_op, curr_zero, curr_retenue, curr_acc, curr_ir, sum)
03.  Begin
04.  sum          <= "00000";
05.  next_acc     <= "0000";
06.  next_retenue <= '0';
07.  next_zero    <= '0';
08.  Case alu_op IS
09.    When xor_op    =>  next_acc     <= curr_acc xor curr_ir(3 downto 0);
10.                       next_retenue <= curr_retenue;
11.                       next_zero    <= curr_zero;
12.
13.    When others    =>  next_acc     <= curr_acc;
14.                       next_retenue <= curr_retenue;
15.                       next_zero    <= curr_zero;
16.  End Case;
17.  End Process L_ALU;
```

8 † 1.7 Question 7. (4 points) - TP 4

TABLE DES MATIÈRES