1. Exercice 1 (5 pts)

Répondre aux questions suivantes en justifiant vos réponses :

1.1 C'est quoi la différence entre les coût fixes et les coûts non fixes pour la conception et la fabrication de circuits intégrés. Donner trois exemples de coûts fixes et trois exemples de coûts non fixes. (1 pt)
Coûts fixes : se sont les coûts qui ne dépendent pas du nombre de circuits fabriqués. Exemples : coût des masques, coût des logiciels de conception et le salaire des ingénieurs de conception.
Coûts non fixes : se sont les coûts qui dépendent du nombre de circuits fabriqués. Exemples : Coût de fabrication (surface), coût de test et coût d'encapsulation.

1.2 Quels sont les différents types de délais qui existe dans le VHDL ? Expliquer le rôle de chacun des types. (*1 pt*)
types de délai : delta-cycle, inertie et transport
Delta-cycle : n'a pas de signification physique et permet juste d'ordonnancer l'exécution des instructions VHDL car l'exécution dans les ordinateurs est séquentielle.
Transport : le signal de sortie est modifié après ce délai de transport par rapport au signal d'entrée.
Inertie : le signal de sortie ne sera modifié que si le signal d'entrée reste stable pendant ce délai d'inertie.

1.3 Pourquoi le coût d'utilisation d'un FPGA est plus faible pour les petites productions par rapport à l'utilisation d'un ASIC, alors que pour les grandes productions c'est le contraire ?. (*1 pt*)
Pour les petites productions, se sont les coûts fixes qui dominent, alors que pour les grandes productions, les coûts fixes ramenés sur le nombre de circuits devient négligeable et se sont les coûts non fixes qui dominent. Les coûts fixes sont plus élevés pour les ASICs, alors que les coûts non fixes sont moins élevés pour les ASICs que pour les FPGA car les ASICs nécessitent moins de surface de silicium que les FPGAs.

1.4 Quelles est la différence entre un signal et une variable en VHDL ?. Comment cette différence affecte-t-elle la simulation VHDL ?. Donner un exemple pour illustrer votre réponse. (*1 pt*)
Le signal n'est modifié qu'à la fin du cycle, alors que la variable est modifiée tout de suite. Le signal permet aussi de modéliser les délais physiques. Comme le signal n'est modifié qu'à la fin du cycle, l'utilisation des signaux augmente le nombre de delta-cycles alors que les variables n'affectent pas le nombre de delta-cycles.

1.5 Avec le VHDL, est-il possible de correspondre plusieurs architectures à une entity ? Si oui, donner les avantages, si non, donner les problèmes que ça engendre. (*1 pt*)
Oui. Ça permet d'avoir plusieurs descriptions pour le même module (comportementale, RTL et structurelle).

2. Exercice 2 (5 pts)

On souhaite utiliser le VHDL pour définir un nouveau type énuméré qui nous permettra de manipuler les couleurs suivantes : noire, blanche, grise et indéfinie.

2.1 Que veut-on dire par un type **énuméré** ? (*0,5 pt*)
Type contenant un ensemble fini de valeurs.

2.2 Donner la syntaxe VHDL pour définir le type **ucolor** décrit ci-haut. *(0,5pt)*
type ucolor is (blanche, noire, grise, indéfinie) ;

2.3 Est-ce que le type ucolor défini à la section 2.2 est un type résolu ou non ? Jusitifier votre réponse. *(0,5pt)*
Non, car il n'a pas de fonction de résolution

2.4 Expliquer à quoi sert et comment utilise-t-on une fonction de résolution. (*0,5 pt*)
La fonction de résolution permet de résoudre le problème de conflit quand il y a affectation multiple.
On définit un sous type du type de départ en utilisant la commande subtype.

2.5 Donner le code VHDL complet (avec la fonction de résolution) permettant de définir le type **color** comme étant le type résolu du type ucolor avec les conditions suivantes : une affectation multiple de n'importe quel couleur avec la couleur indéfinie donne la couleur indéfinie et noire avec blanche = noire avec grise = blanche avec grise = grise. *(3 pt)*

```
type ucolor is (blanche, noire, grise, indefinie) ;
type ucolor_vector is array ( natural range <> ) of ucolor ;
function resolved ( s : ucolor_vector ) return ucolor ;
subtype color is resolved color ;
constant resolution_table : color_table := (
 --| indefinie     | indefinie     | indefinie     | indefinie
( indefinie,       indefinie,      indefinie,      indefinie ),   -- | indefinie |
( indefinie,       grise,          grise,          grise ),       -- | grise |
( indefinie,       grise,          blanche,        grise ),       -- | blanche |
 ( indefinie,      grise,          grise,          noire )        -- | noire |
 );
 function resolved ( s : ucolor_vector ) return ucolor is
      variable result : ucolor := indefinie ;
begin
      if   (s'LENGTH = 1) then   return   s(s'LOW);
      else
        for i IN s'RANGE loop
          result := resolution_table(result, s(i));
        end loop;
      end if;
     return result;
   end resolved;
```

3. Exercice 3 (3 pts)

Soit le code VHDL suivant :

```
ENTITY EXO2 IS PORT(
    a       : IN    STD_LOGIC ;
    b       : IN    STD_LOGIC ;
    c       : OUT   STD_LOGIC ) ;
END EXO2 ;

ARCHITECTURE RTL OF EXO2 IS
    SIGNAL s1,s2,s3,s4  : STD_LOGIC ;

BEGIN
    P1 : PROCESS BEGIN
        s1 <= not a ;
        s2 <= not b ;
        s3 <= s1 and b ;
        s4 <= s2 and a ;
        c  <= s3 or s4 ;
        wait on a,b,s1,s2,s3,s4 ;
    END PROCESS P1 ;
END RTL;
```

3.1 Donner le chronogramme temporelle contenant a,b, c, s1, s2,s3 et s4 en considérant le testbench suivant : *(1pt)*

```
a <= '0', '1' after 5ns, '0' after 10ns, '1' after 15 ns, '0' after 20ns;
b <= '1', '0' after 10ns ; '1' after 20ns ;
```

Temps (ns)	*0 ns*	*5 ns*	*10 ns*	*15ns*	*20ns*
a	*0*	*1*	*0*	*1*	*0*
b	*1*	*1*	*0*	*0*	*1*
s1	*1*	*0*	*1*	*0*	*1*
s2	*0*	*0*	*1*	*1*	*0*
s3	*1*	*0*	*0*	*0*	*1*
s4	*0*	*0*	*0*	*1*	*0*
c	*1*	*0*	*0*	*1*	*1*

3.2 Combien de delta cycles y a t-il a à l'instant t=5 ns ? Donner les valeurs de chaque signal et de chaque variable pour chaque delta-cycle. *(1pt)*

Il y a 3 delta-cycles.

Delta -cycle	*-0*	*+0*	*1*	*2*	*3*
a	*0*	*1*			
b	*1*				
s1	*1*		*0*		
s2	*0*				
s3	*1*			*0*	
s4	*0*		*0*		
c	*1*			*1*	*0*

3.3 Même question pour t = 10 ns ?. *(1pt)*

Il y a 2 delta-cycles

Delta -cycle	*-0*	*+0*	*1*	*2*	*3*
a	*1*	*0*			
b	*1*	*0*			
s1	*0*		*1*		
s2	*0*		*1*		
s3	*0*		*0*	*0*	
s4	*0*		*0*	*0*	
c	*0*				

4. Exercice 4 (2 pts)

Soit l'architecture d'un PLA avec les connexions déjà programmées de la figure 4-1.

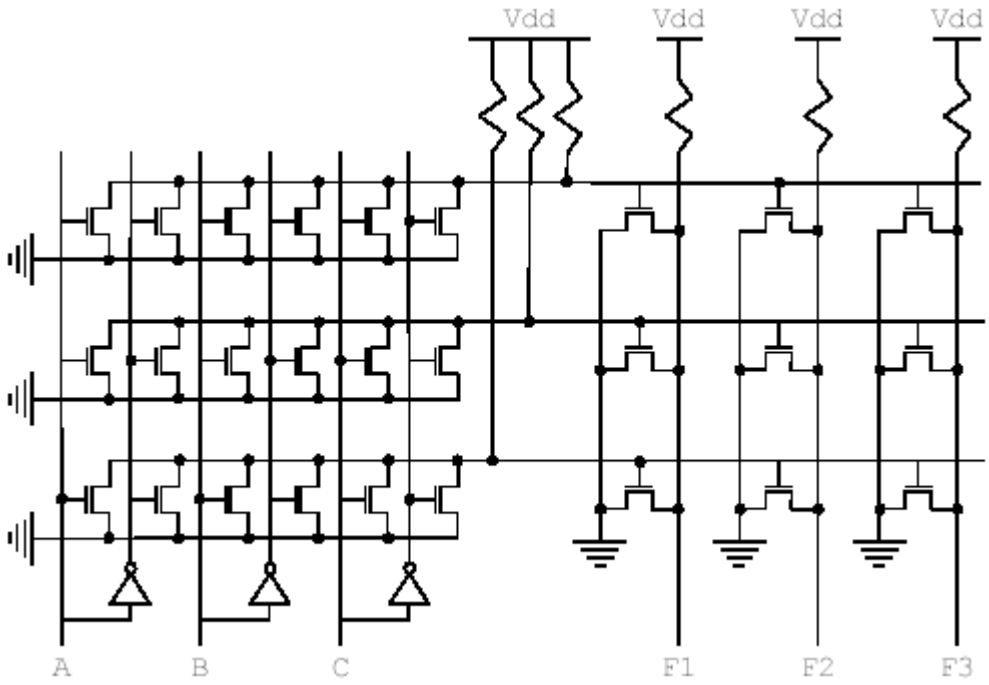


Figure 4-1

4.1 Donner les fonctions réalisées par ce PLA dans le cas ou on considère les entrées et les sorties en logique positive. Justifier votre réponse. *(0,75 pt)*

P1 = (A+B+C')'
P2 = (A'+ B'+C)'
P3 = (C')'=C
Le PLA réalise deux fonctions qui sont :
F1 = (P1+P2)' = P1'.P2' = (A+B+C').(A'+B'+C)
F2 =P3' = C'

4.2 Donner les fonctions réalisées par le PLA dans le cas ou on considère les entrées et les sorties en logique négative. Justifier votre réponse. *(0,75 pt)*

P1 = (A.B.C')'
P2 = (A'. B'.C)'
P3 = (C')'=C
Le PLA réalise deux fonctions qui sont :
F1 = (P1.P2)' = P1' + P2' = A.B.C' + A'.B'.C
F2 =P3' = C'

4.3 D'après les résultats des questions 4.1 et 4.2, est-il préférable de travailler avec la logique positive ou la logique négative ? Justifier votre réponse. *(0,5 pt)*
Si on travaille avec la forme normale disjonctive (ΣΠ), il est plus pratique d'utiliser la logique négative et si on travaille avec la forme normale conjonctive (ΠΣ) il est plus pratique d'utiliser la logique positive.

5. Exercice 4 (5 pts)

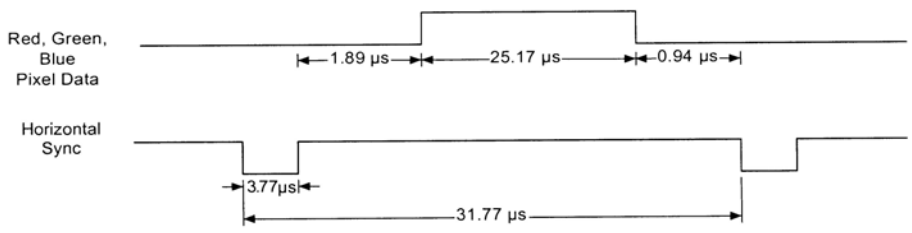
5.1 Quelle est la fréquence de l'horloge de référence d'un synchronisateur VGA possédant une résolution de 1280x1024 et un taux de rafraîchissement de 80Hz. Considérez un ajout de 10% pour VF et de 8% pour HF afin de ramener le rayon d'électron. *(1pt)*
VF=HF/(nbH(100%-ajout%))*
*HF=80Hz*1024*90%=73,728 kHz*

HF=clk/(nbpixel(100%-ajout%))*
*clk=73,728kHz*1280*92% = 86,82 MHz*

5.2 Voici une partie du code VHDL du synchonisateur VGA que vous avez réalisé au laboratoire :

```
--compteur horizontal
if (h_count = 799) then     --ajout du temps d'extra 640+extra=800
   h_count <= "0000000000";
else
   h_count <= h_count +1;
end if;
--génération de Hsync durant 97 x 1/25,175MHz secondes avec H_count
-- Hsync ------------------------_____--------
--      0                      640   659   755   799
if (h_count <= 755) and (h_count >= 659) then
   horiz_sync <= '0';
else
   horiz_sync <= '1';
end if;
```

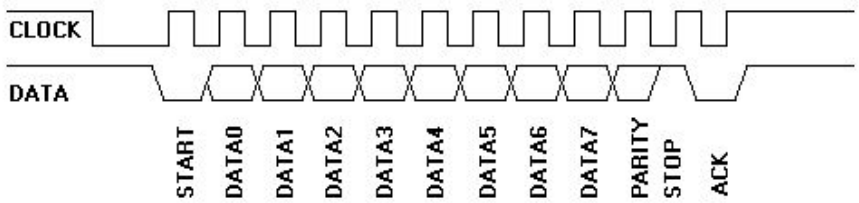
5.3 Modifiez-le afin de <u>mieux</u> respecter ce diagramme temporel : *(1pt)*



800=>31,78us (0-799)
640=>25,42us (0-639) ← 631=>25,1us (0-630)

24=>0,95us (640-664) ← (631-655)
95=>3,77us (665-760) ← (656-751)
47=>1,87us (761-808) ← (752-799)

5.4 Dessinez le diagramme temporel d'une communication PS/2 Host-to-Device. La commande à transmette est : Set all keys make. Expliquez chaque séquence du protocole. *(2 ps)*



Ecriture par le contrôleur : CLK=LOW
Lecture par l'interface : CLK=HIGH

Afin d'obtenir la génération de l'horloge par l'interface
Imposer CLK à un niveau bas (BUS Idle)
Ensuite mettre DATA à un niveau bas et relâcher CLK (niveau haut)

ACK : par l'interface

DATA0=LSB, DATA7=MSB,DATA= F9,PARITY= 1

5.5 Complétez le programme suivant : (*1pt*)

```
#include "excalibur.h"
void main(void)
{

int compteur=0;
                        --déclaration du pointeur timer
                        --initialisation du compteur 1 seconde


                        --démarrer le timer en mode continu

                        --boucle infinie

                        --lorsque le timer atteint zéro incrémenter le compteur



                        --lorsque le compteur atteint 60 arrêter le timer

}
```

```
#include "excalibur.h"
void main(void)
{
int compteur=0;

np_timer *timer = na_timer1;          --déclaration du pointeur timer
                                      --initialisation du compteur 1 seconde
timer->np_timerperiodl = (short)(nasys_clock_freq & 0x0000ffff);
timer->np_timerperiodh = (short)(nasys_clock_freq >> 16) &0x0000ffff;

                                      --démarrer le timer en mode continu
timer->np_timercontrol= np_timercontrol_start_mask + np_timercontrol_cont_mask;

                                      --boucle infini
while(1)
{

                                      --lorsque le timer atteint zéro incrémenter compteur
  if (timer->np_timerstatus & np_timerstatus_to_mask)
          compteur++;
                                      --lorsque le compteur atteint 60 arrêter le timer
  if (compteur==60)
          timer->np_timercontrol = np_timercontrol_stop_mask;
  }
}
```

Bon examen !