UNIVERSITÉ SAVOIE MONT BLANC

Travaux pratiques

Processeurs reconfigurables

NIOS II – Carte Altera DE1

Dans les travaux pratiques, nous utiliserons plusieurs systèmes pour la partie matérielle :

- Soit un système prédéfini et fourni par Altera, décrivant un microprocesseur et ces périphériques : DE1_Basic_Computer, DE1_Media_Computer… etc Ces systèmes sont réalisés à bases d'un microprocesseur NIOSII.
- Soit un système que nous avons conçu spécifiquement pour notre application, aussi à base d'un microprocesseur NIOSII.

De plus, nous utiliserons plusieurs codes pour la partie logicielle :

- Soit des exemples (C et assembleur) fournis par Altera : Test_Media_Computer.s, getting_started.c, etc…
- Soit des codes que nous allons développer spécifiquement pour répondre au cahier des charges d'une application.

1 Préparation aux travaux pratiques

1.1 Préparation du Lab1

Le code assembleur suivant est proposé :

```
.equ LIST, 0x800        /* Starting address of the list   */

.global _start

_start:
  movia r4, LIST      /* r4 points to the start of the list    */
  ldw   r5, 4(r4)     /* r5 is a counter, initialize it with N */
  addi  r6, r4, 8     /* r6 points to the first number         */
  ldw   r7, (r6)      /* r7 is the largest number found so far */

LOOP:
  subi  r5, r5, 1         /* Decrement the counter             */
  beq   r5, r0, DONE      /* Finished if r5 is equal to 0      */
  addi  r6, r6, 4         /* Increment the list pointer        */
  ldw   r8, (r6)          /* Get the next number               */
```

```
    bge   r7, r8, LOOP     /* Check if larger number found       */
    add   r7, r8, r0       /* Update the largest number found    */
    br    LOOP
DONE:
    stw   r7, (r4)      /* Store the largest number into RESULT  */

STOP:
    br    STOP          /* Remain here if done                   */

.org    0x800
RESULT:
.skip  4                  /* Space for the largest number found
      */
N:
.word 7                   /* Number of entries in the list
      */
NUMBERS:
.word  4, 5, 3, 6, 1, 8, 2   /* Numbers in the list */

.end
```

- En vous référant aux instructions assembleur de la documentation « Introduction to the Altera NIOS II soft processeur », réaliser l'organigramme complet et précis de ce programme, et donner sa fonction.
- Ecrire un programme en langage C, réalisant la fonctionnalité équivalente.

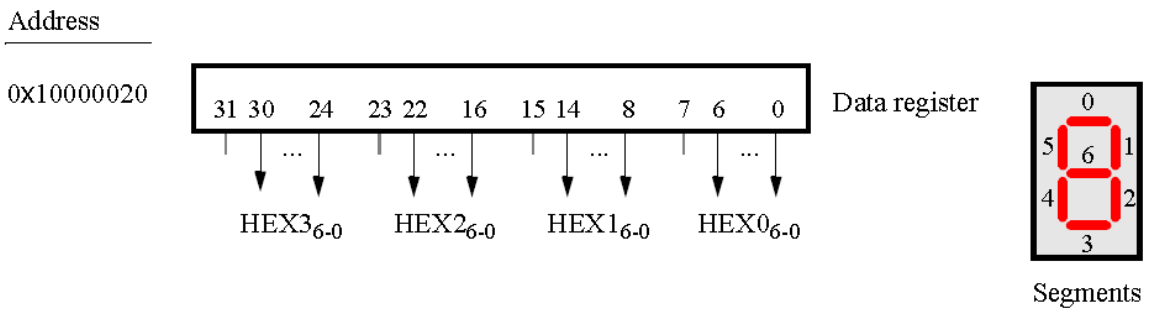
1.2 Préparation du Lab3

On cherche à afficher (en hexadécimal) un nombre entier (32 bits, non signé), sur un afficheur 7 segments. Pour cela, on déclare un tableau représentant les 7 segments de chacun des 16 digits (0 à F).

```
unsigned char table[16] = { 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d,
0x7d, 0x07, 0x7f, 0x67,0x77, 0x7c, 0x39, 0x5e, 0x79, 0x71 };
```

- A la case 0 du tableau se trouve les 7 segments du chiffre 0.
- A la case 1 du tableau se trouve les 7 segments du chiffre 1.
- …
- A la case 15 du tableau se trouve les 7 segments du chiffre F.

On rappelle l'organisation du périphérique «7 segments » du Media_Computer :



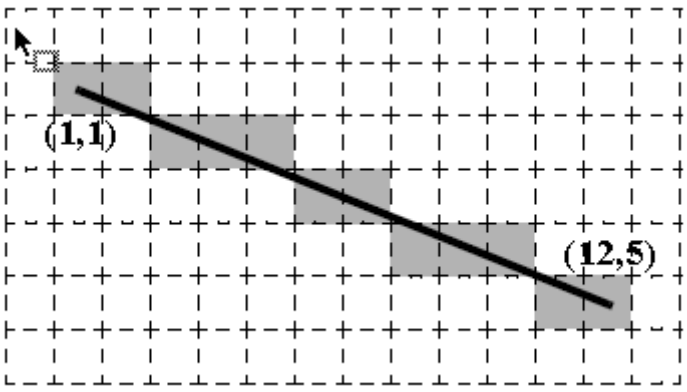
- Vérifier le code de quelques-uns des digits du tableau « table ».

Soit une variable « sum » de 32 bits représentant le nombre à afficher.

- Ecrire en langage C la partie du programme permettant d'afficher la variable sum sur l'afficheur 7 segments.

1.3 Préparation du Lab7

Représenter une ligne sur un écran requiert de colorier les pixels entre deux points (x1; y1) et (x2; y2), de tel manière qu'ils représentent une ligne la plus parfaite possible. Voici un exemple pour les deux points (1,1) et (12,5).



Les carrés grisés représentent les pixels qui devront être coloriés. Pour dessiner cette ligne, nous devons suivre la ligne pour chaque valeur de x et trouver la valeur de y correspondante. Pour cela, nous utilisons l'équation de la droite déduite des deux coordonnées précédentes.

Cette méthode possède un inconvénient majeur. En effet, pour des pentes raides de la droite, le nombre de lignes devant être coloriés est supérieur au nombre de colonne entre les deux points. Si nous suivons cette méthode, des trous dans la représentation seront observés.

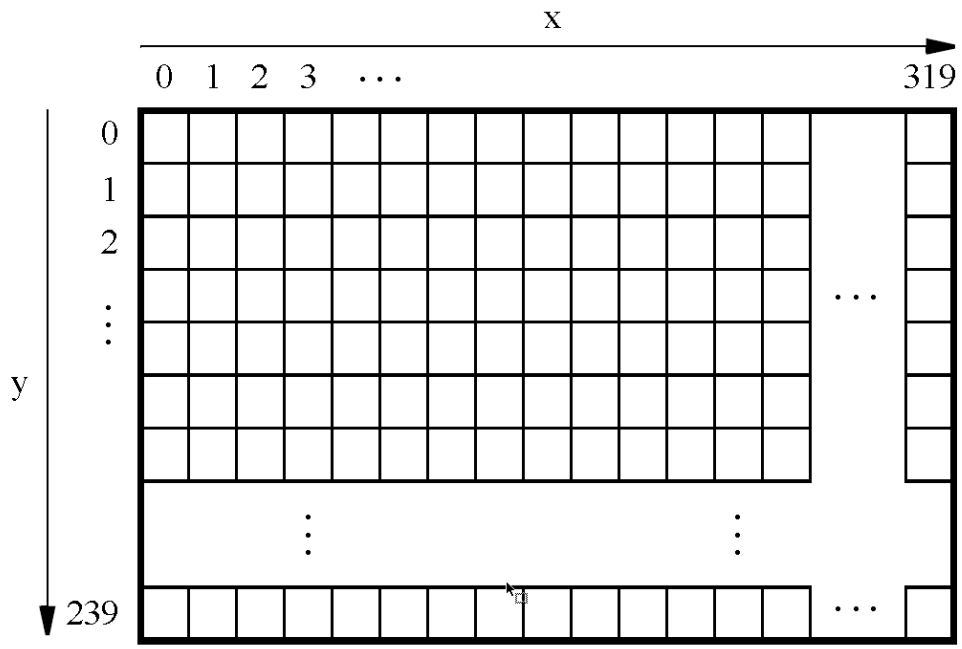
Pour contrecarré ce phénomène, il faut parcourir la droite selon les x (et chercher les y correspondants) lorsque la droite est de faible pente, et parcourir la droite selon y (et chercher les x correspondants) lorsque la droite est de faible pente.

L'algorithme de Bresenham tiens compte de ce phénomène, et permet de représenter une ligne à l'écran sans interruption dans l'affichage. Voici le pseudo code :

```
draw line(x0, x1, y0, y1)

    boolean is steep = abs(y1 - y0) > abs(x1 - x0)
    if is steep then
        swap(x0, y0)
        swap(x1, y1)
    if x0 > x1 then
        swap(x0, x1)
        swap(y0, y1)

    int deltax = x1 - x0
    int deltay = abs(y1 - y0)
    int error = -(deltax / 2)
    int y = y0
    if y0 < y1 then y step = 1 else y step = -1

    for x from x0 to x1
        if is steep then draw pixel(y,x) else draw pixel(x,y)
        error = error + deltay
        if error >= 0 then
            y = y + y step
            error = error - deltax
```

- Réaliser le code C complet de cette fonction draw_line().

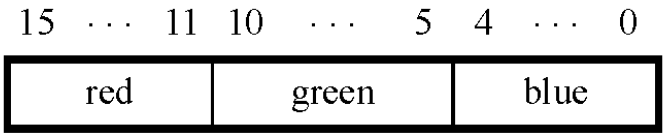
La représentation de l'écran sous forme de pixels est symbolisée par le schéma suivant :



Les périphériques VGA_Pixels_buffer et VGA_controler du DE1_Media_Computeur ont été conçus pour gérer l'affichage sur un écran de 320 x 240, nous allons étudier leur fonctionnement.

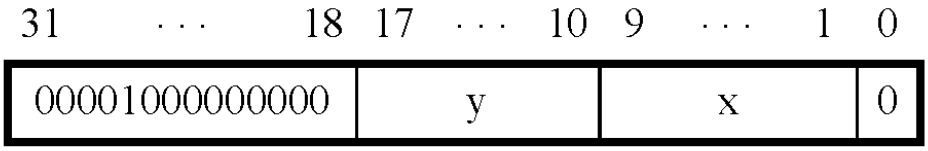
- Combien de bits sont nécessaires pour coder l'abscisse du pixel ?
- Combien de bits sont nécessaires pour coder l'ordonnée du pixel ?

Le pixel_buffer, va réserver une case mémoire par pixel, dans laquelle il stockera la valeur RGB de la couleur du pixel. La valeur RGB est un mot de 16 bits de la forme suivante :



(a) Pixel color

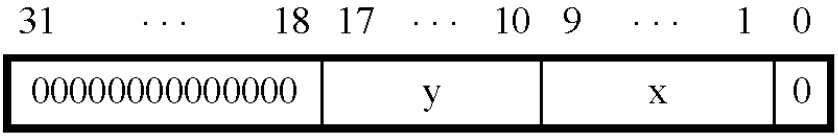
On cherche à faire la correspondance entre la coordonnée du pixel et son emplacement en mémoire. La première adresse utilisée est 0x0800 0000, c'est l'adresse du pixel (0,0). La solution envisagée par le pixel buffer est de définir l'adresse du pixel comme suit :



(b) Pixel buffer addresses

- Donner en hexadécimal, l'adresse du pixel de coordonnée (0,0), (0,1) et (0,2) ?
- Donner en hexadécimal l'adresse du pixel de coordonnée (0,319) et du pixel suivant (1,0) ? Que peut remarquer ? Quel inconvénient cela aura sur l'utilisation de la mémoire ?

La variable « offset » représente l'offset de l'adresse en partant de 0x 0800 0000. Elle est donc représentée par la forme suivante :



(b) Pixel (x,y) offset

- Ecrire en langage C la valeur de la variable offset en fonction de x et y.

2 Travaux pratiques

Les exercices sont réalisés sur des platines DE1 de chez Altera. Les sujets ont parfois été écrits pour des platines DE2 mais vous considérez toujours qu'ils sont valables pour votre carte DE1. Trois séances de 4h sont réservées pour ces Travaux Pratiques. Vous réaliserez uniquement les parties suivantes :

- Lab 1 : Part I, II, V et VI.
- Lab 3 : Part IV, V et VI.
- Lab 7 : Tout

Optionnel :

- Lab 10 : Tout