IFT6223
Systèmes reconfigurables

Introduction

Jean Pierre DAVID

david@iro.umontreal.ca

Université de Montréal

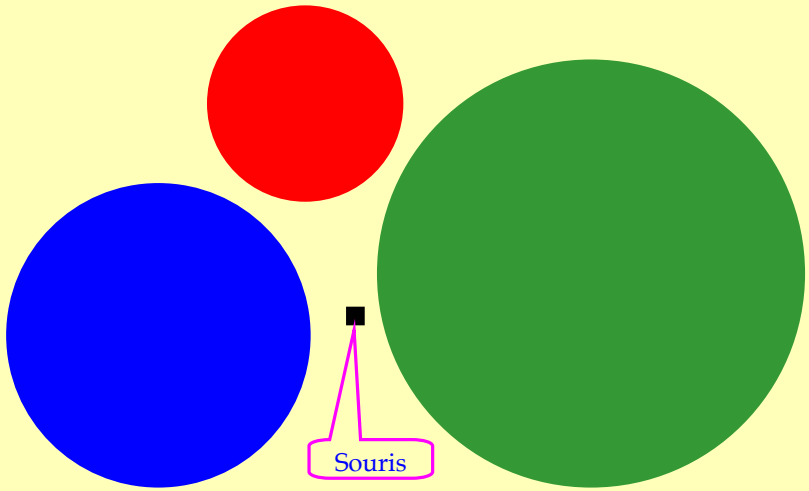
Plan

- Énoncé du problème
- Vue globale de l'architecture
- 1ère partie : l'interface VGA
  - Logique combinatoire et interface graphique
- 2ème partie : le coprocesseur
  - La racine carrée en C
  - ASM
  - Optimisation de l'ASM

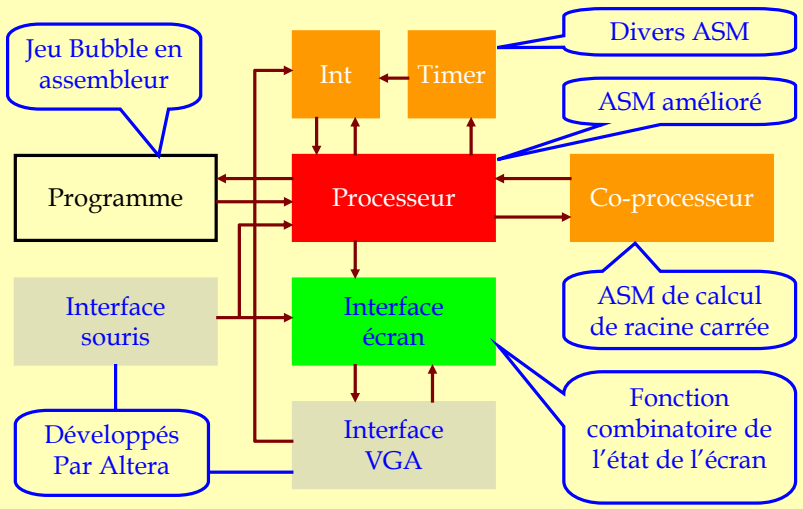
Énoncé du problème

- En début de partie, trois disques (rouge – vert – bleu) de rayon minimal (10 pixels) apparaissent à l'écran.
- Ces disques grossissent avec le temps jusqu'à ce qu'une des deux conditions suivantes soit remplie :
  - La souris passe sur le disque : le disque disparaît alors et réapparaît à un autre endroit avec un rayon minimal. Le joueur marque un nombre de point inversément proportionnel au rayon du disque.
  - Le disque sort de l'écran et la partie se termine.
- Les points sont affichés sur l'afficheur LCD de la carte.
- La vitesse de croissance des disques peut varier au cours du temps

Aperçu de l'affichage en cours de partie



Vue globale de l'architecture



Rôle des différentes parties

- Le processeur : calcule en temps réel les variables d'état de l'interface écran en fonction du balayage des lignes et de la position de la souris. C'est lui qui gère la taille et la position des disques.
- Le co-processeur : est capable de calculer une racine carrée en un temps minimal (et en parallèle avec le processeur).
- L'interface écran : calcule la couleur du point (x,y) de manière combinatoire (à 25.175MHz) en fonction de ses variables d'état.
- Les interruptions sont utilisées pour signaler un changement de ligne, d'image ou un timer.

1ère partie : L'interface VGA et l'interface écran

- L'interface VGA balaye les 480 lignes de haut en bas.
- Pour chaque ligne, l'interface VGA balaye les 640 colonnes de gauche à droite.
- Notre interface écran doit être capable de calculer la couleur de chaque point (x,y) à une cadence de 25.175 Mhz
  - Red(0/1), Green(0/1), Blue(0/1) soit 8 couleurs au total
- Nous travaillerons ligne par ligne avec les variables d'état suivantes :
  - Low_R et High_R (début et fin de la couleur rouge)
  - Low_G et High_G (début et fin de la couleur verte)
  - Low_B et High_B (début et fin de la couleur bleue)
  - Mouse_X et Mouse_Y : position (x,y) de la souris
- A chaque fin de ligne, les variables min et max doivent être recalculées (par le processeur)

Un balayage ligne par ligne

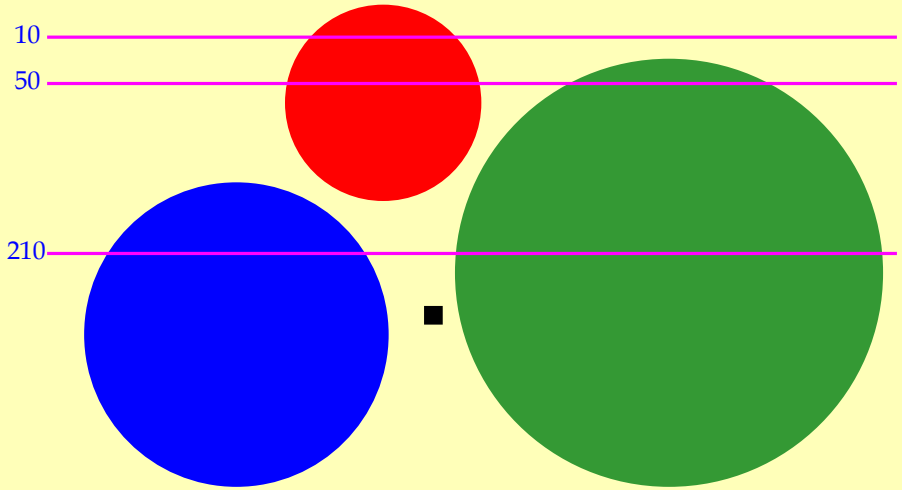
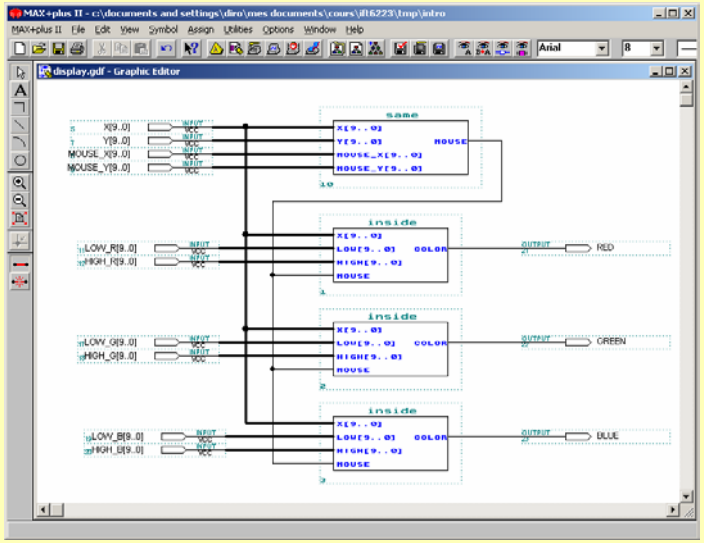
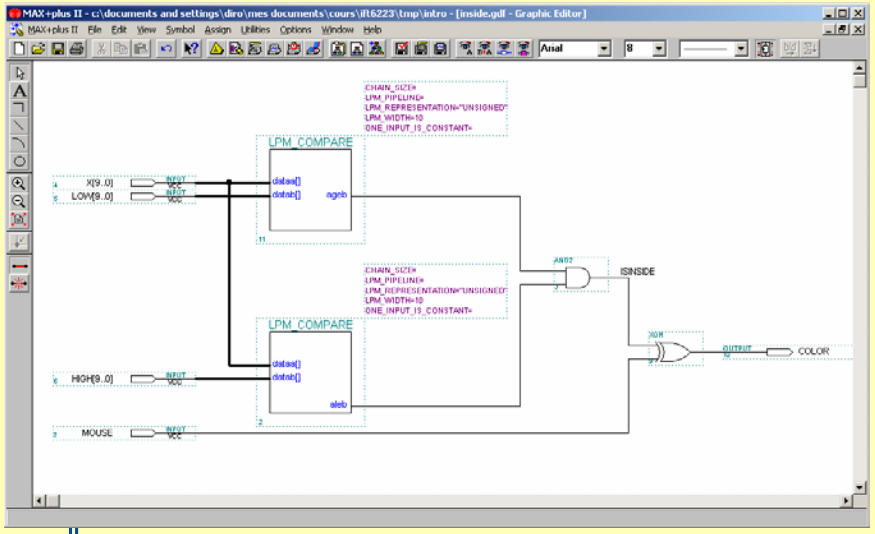


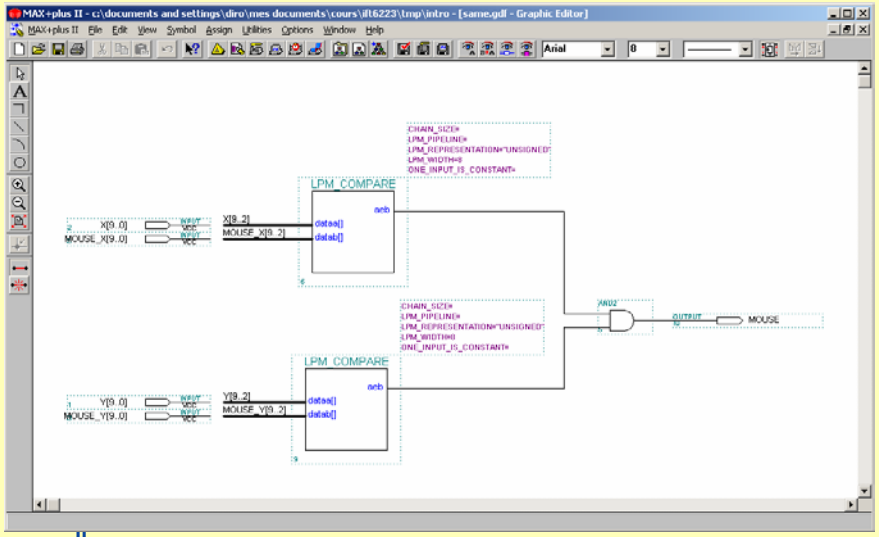
Schéma de l'interface écran



Le bloc « INSIDE »



Le bloc « SAME »



2ème partie : Le co-processeur

- L'équation d'un cercle centré à l'origine est :
  - $X^2+Y^2=R^2$
- Dans le cas qui nous concerne, Y est donné (la ligne), R est donné et X est l'inconnue (Low… et High…) Ceci implique donc le calcul d'une racine carrée.
  - *Remarque, il serait possible de s'en passer en utilisant une approche incrémentale mais nous l'utiliserons néanmoins dans un but pédagogique.*
- La racine carrée d'un nombre de N bits peut se calculer assez simplement en N/2 étapes (voir l'algorithme ci-après)
- Nous considérerons des nombres de 24 bits (car X,Y et R sont sur 10 bits).

L'algorithme C de la racine carrée

```
#define width 24
long Sqrt(long x) {/* Source: Ken Turkowski (Apple)*/
  unsigned long root, remHi, remLo, testDiv, count;
  root = 0; remHi = 0; remLo = x; count = width/2-1;
  do {
      remHi = (remHi<<2) | (remLo>>(width-2));
      remLo <<= 2;
      root <<= 1;
      testDiv = (root << 1) + 1; /* Test radical */
      if (remHi >= testDiv) {
      remHi -= testDiv;
      root++;}
  } while (count-- != 0);
  return (root);
}
```


Élimination des while

```
long Sqrt(long x)
{
      unsigned long root, remHi, remLo, testDiv, count;
      root = 0; remHi = 0; remLo = x; count = width/2-1;
N1:   remHi = (remHi<<2) | (remLo>>(width-2));
      remLo <<= 2;
      root <<= 1;
      testDiv = (root << 1) + 1;
      if (remHi >= testDiv) {
            remHi -= testDiv;
            root++;}
      if    (count-- != 0) goto N1;
      return (root);
}
```


Élimination des tests imbriqués

```
long  Sqrt(long x) {
      root = 0; remHi = 0; remLo = x; count = width/2-1;
N1:   remHi = (remHi<<2) | (remLo>>(width-2));
      remLo <<= 2;
      root <<= 1; testDiv = (root << 1) + 1;
N2:   if (remHi >= testDiv) goto N2i;
      else  goto N2e;
N2i:  remHi -= testDiv;
      root++; goto N2n
N2e:  goto  N2n
N2n:  N3:   if    (count-- != 0) goto N3i;
            else goto N3e;
N3i:  goto N1;
N3e:  return (root);
}
```


Augmenter le parallélisme

```
long  Sqrt(long x) {
N0:   if (true)          {root=0; remHi=0; remLo=x;
                          count=width/2-1; goto N1;}
N1:   if (true)          {remHi=(remHi<<2) |
                         (remLo>>(width-2));
                          remLo<<=2;root<<=1;
                          testDiv=(root<<2)+1; goto N2}
N2:   if (remHi >= testDiv) {
                          remHi-=testDiv; root++;
                          goto N3;}
      else               {goto N3;}
N3:   if (count != 0)    {count--; goto N1};
      else               {return root};
}
```

Anticiper le calcul des tests

```
long  Sqrt(long x) {
N0:   if (true)          {root=0; remHi=0; remLo=x;
                          count=width/2-1; goto N1;}
N1:   if (true)          {remHi=(remHi<<2) |
                         (remLo>>(width-2));
                          remLo<<=2;root<<=1;
                          testDiv=(root<<2)+1;
                          t1=((remHi<<2) | (remLo>>(width-
                               2))) >= ((root<<2)+1);
                         goto N2}
N2:   if (t1)            {remHi-=testDiv; root++;
                          t2=(count !=0); goto N3;}
      else               {t2=(count !=0); goto N3;}
N3:   if (t2)            {count--; goto N1};
      else               {return root};
}
```


Ajouter des signaux de synchronisation

```
long  Sqrt(long input_x) {
N0:   if (start)  {Ready=0; root=0; remHi=0;
                   remLo=Input_x; count=width/2-1;
                   goto N1;}
      else        {goto N0;}
N1:   if (true)   {remHi=(remHi<<2) |
                  (remLo>>(width-2));
                   remLo<<=2;root<<=1;
                   testDiv=(root<<2)+1;
                   t1=((remHi<<2) | (remLo>>(width-2)))
                   >= ((root<<2)+1); goto N2}
N2:   if (t1)     {remHi-=testDiv; root++;
                   t2=(count !=0); goto N3;}
      else        {t2=(count !=0); goto N3;}
N3:   if (t2)     {count--; goto N1};
      else        {Ready=1; Result=root; goto N0};}
```


Fusionner certains états

```
long   Sqrt(long input_x) {
N0:    if (start)    {Ready=0; root=0; remHi=0;
                      remLo=Input_x;count=width/2-1;
              goto N1;}
       else          {goto N0;}
N1:    if (true)     {remHi=(remHi<<2) | (remLo>>(width-2));
              remLo<<=2;root<<=1;
                      testDiv=(root<<2)+1;
                      t1=((remHi<<2) | (remLo>>(width-2))) >=
                      ((root<<2)+1);
                      t2=(count!=0); goto N2}
N2:    switch (t1t2) {
       case 00:      {Ready=1; Result=root; goto N0;}
       case 01:      {count--; goto N1;}
       case 10:      {Ready=1; Result=root+1; goto N0;}
       case 11:      {remHi-=testDiv; root++; count--;
                      goto N1;}}
```


Augmenter encore le parallélisme

```
long  Sqrt(long input_x) {
N0:   if (start)  {Ready=0; root=0;
                   remHi=Input_x>>(width-2);
                   remLo=Input_x<<2;
                   count=width/2-1;
                   testDiv=1;
                   t1=((Input_X>>(width-2))>=1);
                   t2=1; goto N2}
      else        {goto N0;}
N2:   switch (t1t2)     {
      case 00:    {Ready=1; Result=root; goto N0;}
      case 01:    {count--; SPEED1; goto N2;}
      case 10:    {Ready=1; Result=root+1; goto N0;}
      case 11:    {count--; SPEED2; goto N2;}}
```

SPEED1 et SPEED2

```
SPEED1
{remHi=(remHi<<2) | (remLo>>(width-2));
remLo<<=2;root<<=1;
testDiv=(root<<2)+1;
t1=((remHi<<2) | (remLo>>(width-2))) >= ((root<<2)+1);
t2=(count!=1); goto N2}

SPEED2
remHi_tmp=rem_Hi-testDiv; En logique combinatoire !!!
root_tmp=root+1; En logique combinatoire !!!
{remHi=(remHi_tmp<<2) | (remLo>>(width-2));
remLo<<=2;root=root_tmp<<1;
testDiv=(root_tmp<<2)+1;
t1=((remHi_tmp<<2) | (remLo>>(width-2))) >=
   ((root_tmp<<2)+1);
t2=(count!=1); goto N2}
```