Arquitecturas Reconfigurables

Tutorial 2

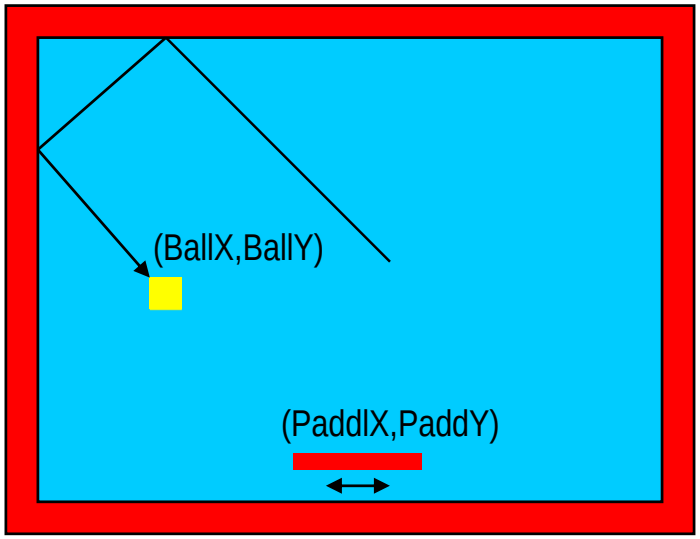
Profesores: Sergio Cuenca y Antonio Martínez

sergio@dtic.ua.es

Dept. Tecnología Informática y Computación

Universidad de Alicante

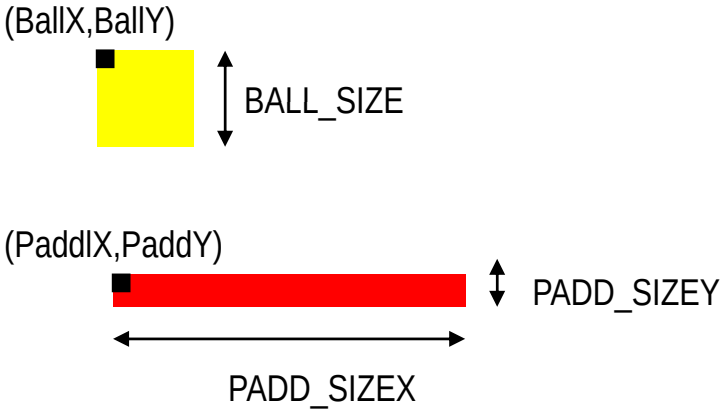
1. Raqueta



- Posición Vertical fija:

```
PaddY = VisibleLines-24;
```

- Control de rebote bola-raqueta:



Raqueta: Dibujo

```
macro proc Display(Video,GamePixel,BallX,BallY,PaddX,PaddY)
 InPaddX=(sx==0@PaddX)?1:((sx==0@(PaddX+PADD_SIZEX))?0:InPaddX);
 InPaddY=(sy==0@PaddY)?1:((sy==0@(PaddY+PADD_SIZEY))?0:InPaddY);
 if (Video.Visible==1)   // if we're in the visible range
   {
    if (InBallX==1 && InBallY==1)   // ball
      {GamePixel = logo;}
    else if (InPaddX==1 && InPaddY==1  // paddle
      {GamePixel = Red;}
    else if (InAreaX==1 && InAreaY==1)  // main area
      {GamePixel = Cyan;}
    else              // border
      {GamePixel = Red;}
    }
  else    // In the blanking period
   {GamePixel = Black;}
```

Verificar antes que "main area"

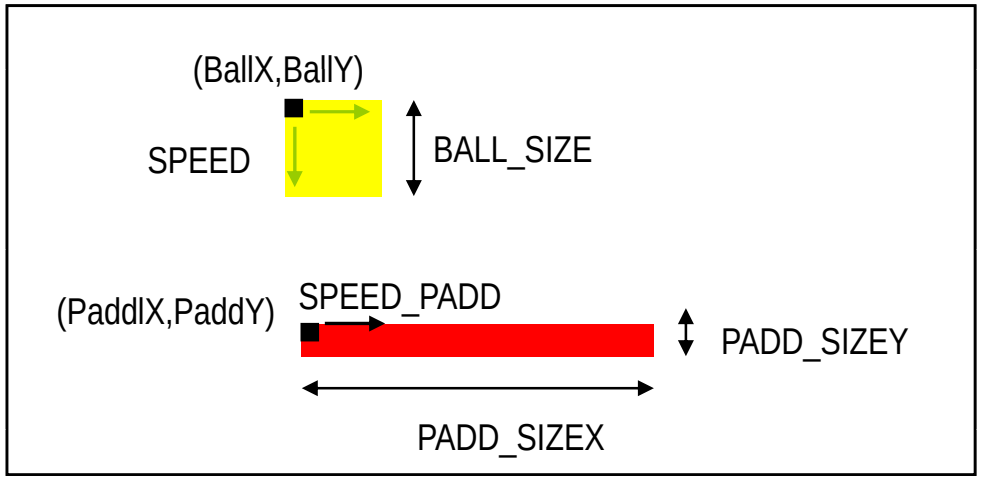
Control de rebote bola-raqueta

```
macro proc PerFrameUpdate()

if (BallY < (BORDER_WIDTH+SPEED) ||
   (BallY > (PaddY-BALL_SIZE-SPEED) &&
   (BallX > (PaddX-SPEED+SPEED_PAD) &&
      BallX<(PaddX+PADD_SIZEX-SPEED+SPPED_PAD))))
    {
       dy = ~dy;
    }
```

Rebote borde horizontal superior

Rebote raqueta



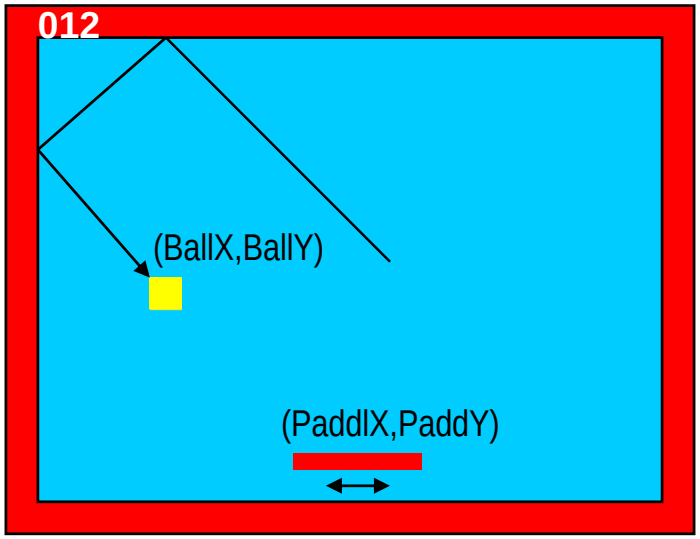
Movimiento raqueta

```
macro proc PerFrameUpdate()
  macro expr MoveRight= #BotónDerechoJoystick;
  macro expr MoveLeft= #BotónIzquierdoJoystick;
// Update paddle position (2cycles)
Colocar dentro del while {delay}
  if ((MoveRight) &&(PaddX<VisibleCols-PADD_SIZEX- BORDER_WIDTH))
    PaddX=PaddX+1;
  else
    delay;
  if ((MoveLeft) && (PaddX>BORDER_WIDTH))
    PaddX=PaddX-1;
  else
    delay;
```

Control fin de recorrido raqueta

```
// Update paddle position (1 cycle)
  if ((MoveRight) &&
     (PaddX<VisibleCols-PADD_SIZEX- BORDER_WIDTH))
    PaddX=PaddX+1;
  else if ((MoveLeft) && (PaddX>BORDER_WIDTH))
    PaddX=PaddX-1;
  else
    delay;
```

2. Marcador y contador de vidas



• Marcador: 3 dígitos, se incrementa cada vez que le damos a la bola

• Vidas: cada vez que la bola toca el borde horz. Inferior pierde una vida y se reinicializa el juego

Main ( )

```
void main (void)
{
  XS40_VGA_DRIVER Video;     // Video driver structure
  unsigned 9 BallX,PaddX;    // ball position variables
  unsigned 9 BallY,PaddY;    // paddle position variables
  unsigned 4 BCDScore[3];    // 3 digits Score
  unsigned 6 GamePixel;      // Pixel colour
  static signal unsigned 1 VideoSelect = 0;
  macro expr White   = 0b101010;
  par
    { XS40VGADriver(&Video); //sustituye a SyncGen
      Display(Video,GamePixel,BallX,BallY,PaddX,PaddY);
      PerFrameUpdate(Video,BallX,BallY,PaddX,PaddY,BCDScore);
      DisplayScore(Video, BCDScore, VideoSelect);
     // Assign video output
      do
        {
           Video.Output = VideoSelect ? White : GamePixel;
      }while(1);
    }
}
```

Modificaciones en PerFrameUpdate( )

```
macro proc PerFrameUpdate()
  unsigned 1 init;
  unsigned 9 Score, ScoreCopy;
  unsigned 4 BCDScoreCopy[3];
  unsigned 2 Life;

  while (1)
    {
   // Cycle 1: Detect bounce and update Score
        par {…… }
   // Cycle 2: Update ball position
        if …
   // Cycle 3: Update paddle position
        if …
   //Cycle 4: Update the copy
        par{… …}
   // n Cycles: Copv Score (binary) to BCD
        ScoreCopy=Score;
        BCDConvert(BCDScoreCopy,ScoreCopy);
    }
```

Mantiene una copia de Score y BCDScore

Convierte de binario a 3 dígitos BCD

Actualizar el Marcador

```
macro proc PerFrameUpdate()

    // Bounce off top horizontal border or paddle (update score)
    if (BallY < (BORDER_WIDTH+SPEED))
     { dy = ~dy;}
    else if  ((BallY > (PaddY-BALL_SIZE-SPEED) &&
              (BallX > (PaddX-SPEED) &&
                 BallX<(PaddX+PADD_SIZEX-SPEED-BALL_SIZE))))
     {
        par{ dy = ~dy; Score++;}
     }
   else
      delay;
```

Separa el rebote con borde superior del rebote con raqueta

Actualiza posición bola

```
macro proc PerFrameUpdate()

  while (1)
    {
   // Cycle 2: Update ball position
     if (init==0 && Life>0)
        par
            {
                BallX = dx ? BallX+SPEED : BallX-SPEED;
                BallY = dy ? BallY+SPEED : BallY-SPEED;
             }
     else
        par
            {
                BallX = VisibleCols/8;
                BallY = isibleLines/2;
                init=0;
             }
```

Solo hay movimiento si quedan vidas

Marcador

```
macro proc PerFrameUpdate()

  //Cycle 4: Update the copy
   par(i=0;i<3;i++)
        {
           BCDScoreCopy[i]=0;
           BCDScore[i] = BCDScoreCopy[i];
        }

  // n Cycles: Conv Score (binary) to BCD
   ScoreCopy=Score;
   BCDConvert(BCDScoreCopy,ScoreCopy);
```

Inicialización en paralelo de todo el array

Se actualiza el valor del marcador BCD

Se saca una copia del valor del marcador (bin) y se llama a la función de conversión

Las copias se utilizan para que no se visualicen valores intermedios durante el proceso de conversión

Display Score

```
macro proc DisplayScore(Video, BCDScore, VidSelect)
{  macro expr sx=Video.ScanX;
   macro expr sy=Video.ScanY;
   while(1)
     {                    // check the ScanY position.
      if(((sy)[3] == 1) && ((sy)\\4 == 0))  // lines 8 - 15
          par
           {
             if (sx == 8)    // check the ScanX position.
                Display3Digits(BCDScore,sy,VidSelect);
             else
               delay;
           }
      else
          delay;
    }
}
```

Visualiza los tres dígitos

```
macro proc Display3Digits(BCDScore,CurrentY,VidSelect)
{ unsigned 2 DigitCount;
  unsigned 4 RomIndex; // cuenta 0-7
  par
    {DigitCount = 2;RomIndex = 0;}

  while(DigitCount !=3 )
   { while(RomIndex != 8)    //each character is 8 pixels wide
        par{
          VidSelect =
numbers[BCDScore[DigitCount]@CurrentY[2:0]@RomIndex[2:0]];
          RomIndex++;
         }

     RomIndex = 0;
     DigitCount--;  //decrement gives 2,1,0 stops when ==3
   }
  VidSelect = 0;  //reset at end of display
}
```

Un bit más para verificar condición de fin de linea

Dígito a visualizar

Fila del dígito a visualizar

Columna a visualizar

Memoria ROM para los dígitos

```
rom unsigned 1 numbers [10*8*8] = {
    0,0,1,1,1,1,0,0,
    0,1,0,0,0,0,1,0,
    0,1,0,0,0,0,1,0,
    0,1,0,0,0,0,1,0,
    0,1,0,0,0,0,1,0,          //0
    0,1,0,0,0,0,1,0,
    0,1,0,0,0,0,1,0,
    0,0,1,1,1,1,0,0,

    0,0,0,0,1,0,0,0,
    0,0,0,1,1,0,0,0,
    0,0,1,0,1,0,0,0,
    0,0,0,0,1,0,0,0,
    0,0,0,0,1,0,0,0,          //1
    0,0,0,0,1,0,0,0,
    0,0,0,0,1,0,0,0,
    0,1,1,1,1,1,1,0,
        … … … …
```

3. Pantalla Game-over

• Cuando se agotan las tres vidas aparece en mensaje de finalización del juego.
• El mensaje está almacenado en memoria RAM o ROM interna (no existe memoria externa): 128lin x 128cols 8bits/pixel

Main ( )

```
void main (void)
{ unsigned 6 GamePixel, ImgPixel; static signal unsigned 1
VideoSelect0 = 0;
  static signal unsigned 1 VideoSelect1 = 0;
  par
    {XS40VGADriver(&Video);
     Display(Video,GamePixel,BallX,BallY,PaddX,PaddY);
     PerFrameUpdate(Video,BallX,BallY,PaddX,PaddY,BCDScore,Life);
     DisplayScore(Video,BCDScore,VideoSelect0);
     DisplayEnd(Video,ImgPixel,VideoSelect1,Life);
    // Assign video output
     do {
        switch (VideoSelect1@VideoSelect0)
        {case 0:
            Video.Output = GamePixel; break;
         case 1:
            Video.Output = White;  break;
         default:
            Video.Output = ImgPixel;  break;
        }
    }while(1);
```

Pantallazo final tiene preferencia sobre marcador

Visualiza pantalla final

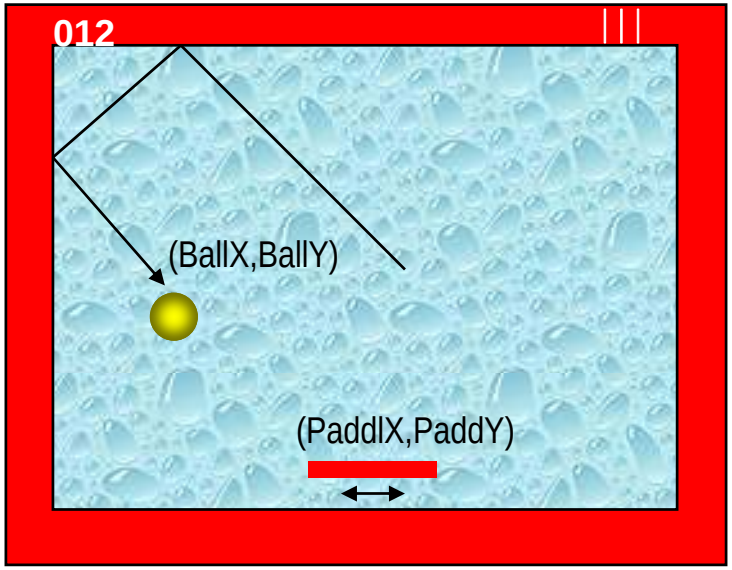
```
macro proc DisplayEnd(Video, ImgPixel, VidSelect,Life)
{ unsigned 8 ColIndex;
  macro expr sx=Video.ScanX;
  macro expr sy=Video.ScanY;
  macro expr addr=0@sy[6:0]@ColIndex;
  while(1)
    { if (Life==0)
       if(((sy)[7] == 1) && ((sy)\\8 == 0))  // lines 128- 256?
         if (sx == 32)    // check the ScanX position.
            while(ColIndex != 128)
                par{
                 VidSelect=1;
                 ReadSRAM(ImgPixel,addr);
                 ColIndex++;
                }
         else
            par{VidSelect=0;ColIndex=0;} // out of image line
       else
         delay; // out of image
     else
       delay; // still alive
}
```

Cálculo dirección memoria

0s....0_1000_0000
...
0s....0_1111_1111

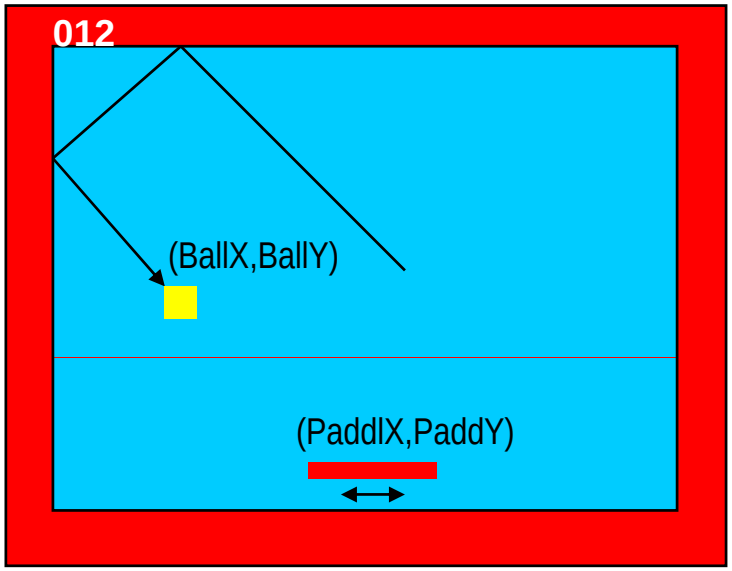
Un bit más para verificar condición de fin de linea

Propuesta I



- Dibujar la pelota mediante una imagen almacenada en RAM.
- Añadir una imagen de fondo (almacenada en RAM)
- Visualizar el número de vidas en la esquina superior derecha

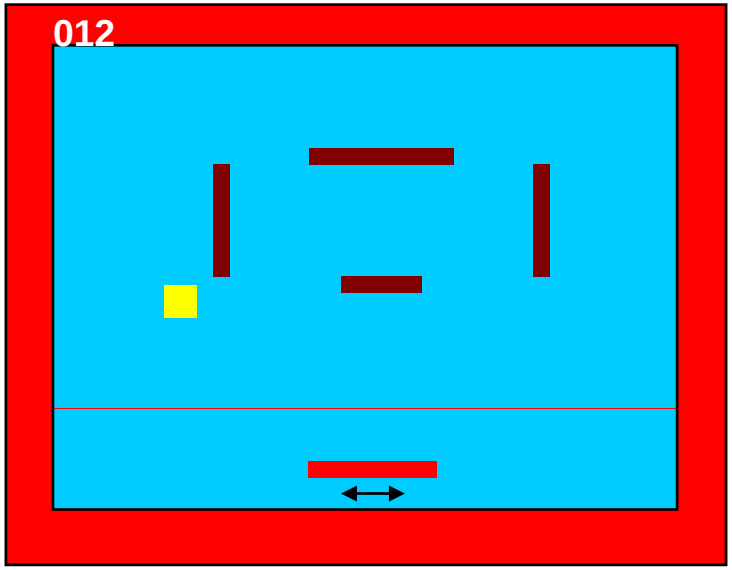
Propuesta II



- Añadir realismo en el rebote bola-raqueta:
  - Aumenatar/disminuir velocidad en función del movimiento de la raqueta al golpear la pelota.
  - Cambiar el ángulo de reflexión de la bola en función de la dirección del movimiento de ésta.

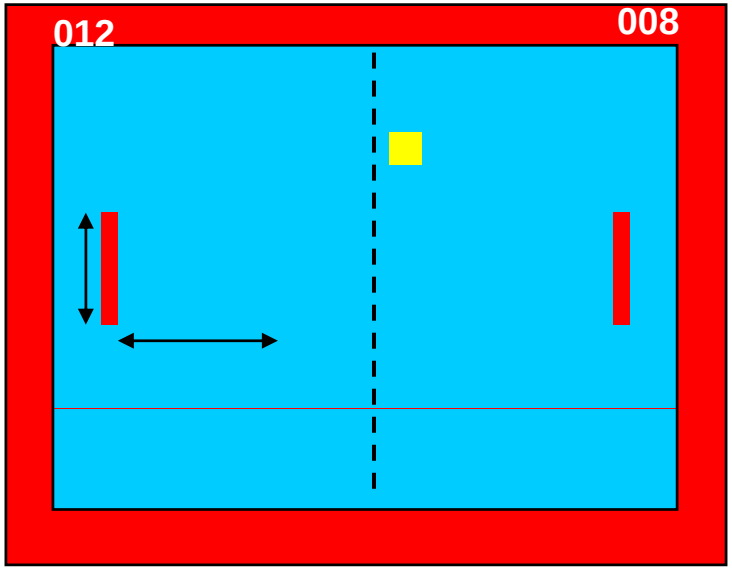
- Aumentar la velocidad de la bola conforme se alcance una puntuación determinada

Propuesta III



• Cambiar el escenario en función de las configuraciones guardadas en memoria

Propuesta IV



- Juego de Tenis:
  - 2 raquetas
  - 2 marcadores
  - Posibilidad de “subidas a la red”