Laboratorio 7:

Salida por un LCD

Programación de sistemas y dispositivos

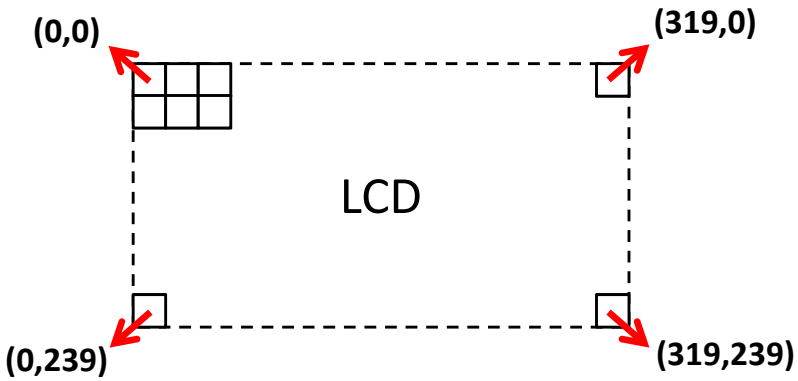
José Manuel Mendías Cuadros
Dpto. Arquitectura de Computadores y Automática
Universidad Complutense de Madrid

Presentación

- Desarrollar una capa de firmware para escribir en un LCD
  - Implementaremos 19 funciones.
    - Inicialización: **`lcd_init`**
    - Encendido/apagado/consulta de estado del LCD: **`lcd_on / lcd_off / lcd_status`**
    - Borrado completo del LCD: **`lcd_clear`**
    - Dibujo de un pixel: **`lcd_putpixel`**
    - Consulta del estado de un pixel: **`lcd_getpixel`**
    - Dibujo de líneas y rectángulos: **`lcd_draw_hline / lcd_draw_vline / lcd_draw_box`**
    - Visualización de un BMP: **`lcd_putWallPaper`**
    - Escritura de caracteres: **`lcd_putchar / lcd_putchar_x2`**
    - Escritura de cadenas de caracteres: **`lcd_puts / lcd_puts_x2`**
    - Escritura de números decimales/hexadecimales como cadenas de caracteres: **`lcd_putint / lcd_puthex / lcd_putint_x2 / lcd_puthex_x2`**

Presentación

- La placa S3CEV40 dispone de un LCD B/N (LRH9J515XA) conectado al controlador de LCD del microcontrolador a través del puerto D:
  - Resolución: 320 × 240 píxeles
  - Profundidad de color: 16 niveles de gris (4b/px).
  - Modo de barrido: 4 bit/single scan
- En modo 4 bit/single scan, el controlador de LCD:
  - Refresca el display pixel a pixel, de izquierda a derecha, de arriba a abajo
  - El color de cada pixel lo lee por DMA de una región de memoria (buffer de vídeo) de ubicación programable:
    - El buffer está compactado, es decir cada byte almacena 2 píxeles
    - El tamaño del buffer es: (320×240) / (2px/B) = 38400B



Presentación

- Para el programador, el buffer de vídeo es un array de bytes
  - Podría declararse de otro tipo pero su gestión es más difícil.

```
uint8 lcd_buffer[320*240/2];
```

cada posición del array contiene 2 píxeles

	pixel (x,y)
lcd_buffer[0]	(0,0)(1,0)
lcd_buffer[1]	(2,0)(3,0)
lcd_buffer[1]	(4,0)(5,0)
lcd_buffer[3]	(6,0)(7,0)
lcd_buffer[4]	(8,0)(9,0)

- Así un pixel (x,y):
  - Estará en el byte que ocupa la posición del array: `x/2 + y*(320/2)`
    - *px/byte*: `2`
    - *bytes/line*: `(320/2)`
  - Dentro del byte estará ubicado a partir del bit: `(1-x%2)*4`
    - *bits/px*: `4`
    - Podrá estar solo a partir del bit 0 ó 4

Salida por un LCD

configuración del controlador de LCD (i)

- Resolución: 320 × 240 píxeles
  - LCDCON2[20:10] = 79 — HOZVAL = (320 / 4) – 1
  - LCDCON2[9:0] = 239 — LINEVAL = 240 – 1
- Modo de barrido: 4 bit single scan
  - LCDCON1[6:5] = 1
- Modo del LCD: 16 niveles de gris (4b/píxel)
  - LCDSADDR1[28:27] = 2
- Pantalla virtual: no
  - LCDSADDR3[19:9] = 0 — OFFSIZE = 0
  - LCDSADDR3[8:0] = 80 — PAGEWIDTH = HOZVAL + 1 = 320 / 4
- Blanking horizontal: mínimo
  - LCDCON2[31:21] = 0 — LINEBLANK = 0 ciclos
- Sincronismo horizontal: anchura de pulso y retardo mínimos
  - LCDCON1[11:10] = 0 — WLH = 4 ciclos
  - LCDCON1[9:8] = 0 — WDLY = 4 ciclos

Salida por un LCD

configuración del controlador de LCD (ii)

- Frecuencia de refresco: como mínimo 60 Hz (aprox.)
  - LCDCON1[21:12] = 28 — CLKVAL = 28 $\Rightarrow$ t_{FRAME} = 59,4 Hz
    - t_{FRAME} = LINEVAL×(t_{VCLK}×HOZVAL + t_{MCLK}×(WLH+WDLY+LINEBLANK))
    - t_{FRAME} = LINEVAL×(2×CLKVAL×t_{MCLK}×HOZVAL + t_{MCLK}×(WLH+WDLY+LINEBLANK))
    - t_{FRAME} = t_{MCLK}×LINEVAL×(2×CLKVAL×HOZVAL + (WLH+WDLY+LINEBLANK))
    - 1/60 = (1/64000000)×240×(2×CLKVAL×80 + (4+4+0))
  - La frecuencia puede ser mayor pero aumentaría la tasa de transferencias en el bus del sistema sin ventajas en la visualización

- Polaridad de las señales de sincronismo: normales
  - LCDCON1[1]= 0 — no invierte señales de vídeo
  - LCDCON1[2] = 0 — no invierte la señal VFRAME
  - LCDCON1[3] = 0 — no invierte la señal VLINE
  - LCDCON1[4] = 0 — las señales de vídeo se muestrean a flanco de bajada
- Conmutación de VM: en cada frame
  - LCDCON1[7] = 0
  - LCDSADDR2[28:21] = X — MVAL no aplica

Salida por un LCD

configuración del controlador de LCD (iii)

- Configuración de paletas: no es necesaria
  - REDLUT = 0 — solo se usa en LCD color
  - GREENLUT = 0 — solo se usa en LCD color
  - BLUELUT = 0 — solo se usa en LCD color o LCD 4 niveles de gris

- Parámetros de dithering: los recomendados
  - DITHMODE = 0x12210
  - DP1_2 = 0xA5A5
  - DP4_7 = 0xBA5DA65
  - DP3_5 = 0xA5A5F
  - DP2_3 = 0xD6B
  - DP5_7 = 0xEB7B5ED
  - DP3_4 = 0x7DBE
  - DP4_5 = 0x7EBDF
  - DP6_7 = 0x7FDFBFE

Salida por un LCD

configuración del controlador de LCD (iv)

- Ubicación del buffer de vídeo: asumiendo que comienza en la dirección de memoria *DIR* y ocupa 38400B
  - LCDSADDR1[26:0] = *DIR*[27:1] — dirección de inicio
  - LCDSADDR2[20:0] = (*DIR*+38400)[21:1] — dirección de fin
- Reordenado de bytes: activado
  - LCDSADDR2[29] = 1
  - El DMA lee memoria en palabras de 4B en ordenación little-endian, sin embargo escribiremos los bytes en el buffer de vídeo con ordenación big-endian.
- Self-refresh: desactivado
  - LCDCON3[0] = 0
- Salida vídeo: inicialmente desactivada
  - LCDCON1[0] = 0

Salida por un LCD

configuración del controlador de LCD (v)

- Resumen:
  - LCDCON1 = 0x1C020 (XXXX.XXXX.XX00.0001.1100.0000.0010.0000)
  - LCDCON2 = 0x13CEF (0000.0000.0000.0001.0011.1100.1110.1111)
  - LCDSADDR1 = según la dirección del buffer (1.0XXX.XXXX.XXXX.XXXX.XXXX.XXXX.XXXX)
  - LCDSADDR2 = según la dirección del buffer (1X.XXXX.XXXX.XXXX.XXXX.XXXX.XXXX.XXXX)
  - LCDSADDR3 = 0x50 (0000.0000.0000.0101.0000)
  - LCDCON3 = 0x0
  - REDLUT = 0x0
  - GREENLUT = 0x0
  - BLUELUT = 0x0
  - DITHMODE = 0x12210
  - DP1_2 = 0xA5A5
  - DP4_7 = 0xBA5DA65
  - DP3_5 = 0xA5A5F
  - DP2_3 = 0xD6B
  - DP5_7 = 0xEB7B5ED
  - DP3_4 = 0x7DBE
  - DP4_5 = 0x7EBDF
  - DP6_7 = 0x7FDFBFE

Salida por un LCD

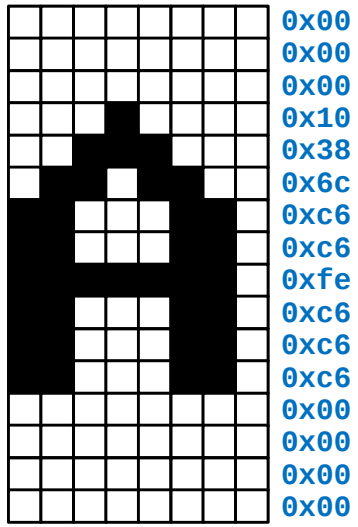
acceso a píxeles

- Para pintar un pixel en la posición (x,y) del LCD
  - Será necesario sobrescribir el nibble que corresponda en el buffer de vídeo.

```
static uint8 lcd_buffer[320*240/2];
...

void lcd_putpixel( uint16 x, uint16 y, uint8 c )
{
  uint8 byte, bit;
  uint16 i;
        px/byte          bytes/line
  i = x/ 2 + y * (320/2);        ····· calcula la posición que ocupa el pixel en el array

  bit = (1-x%2)* 4;              ····· calcula la posición que ocupa el pixel en el byte
                                        (numerado comenzando en el 0 de izq. a der.)
                 bits/px

  byte = lcd_buffer[i];          ····· lee el byte que contiene el pixel
  byte &= ~(0xF << bit );        ····· borra el nibble correspondiente al pixel
  byte |= c << bit;              ····· escribe el nibble correspondiente al pixel
  lcd_buffer[i] = byte;          ····· escribe el byte modificada que contiene el pixel
}
```

Fuentes

- El fichero font8x16.c define una fuente tipo serif de 8x16 px/carácter
  - Para cada carácter define su mapa de bits como una matriz de 8x16 bits almacenada en 16 posiciones consecutivas de un array de uint8.
  - La fuente completa es una matriz de 256x16 de uint8.
  - En cada mapa representa con 1 los foreground pixels y con 0 los background pixels, almacenándolos de izquierda a derecha y de arriba a abajo.




```
const uint8 font[256*16] =
{
  ...
  0x00,0x00,0x00,0x10,0x38,0x6c,0xc6,0xc6,0xfe,0xc6,0xc6,0xc6,0x00,0x00,0x00,0x00
  ...
}
```

- Dado un caracter de código ch:
  - El comienzo de su mapa está en: `(uint8 *)(font + ch*16)`
  - La línea y-ésima de su mapa es: `(unit8)font[ch*16 + y]`

Fuentes

- Para visualizar un carácter en una posición (x,y) del LCD:
  - Será necesario pintar fila a fila y columna a columna los pixeles definidos en el mapa de bits a partir de la posición indicada.

```
void lcd_putchar( uint16 x, uint16 y, uint8 color, char ch )
{
  uint8 line, row;
  uint8 *bitmap;

  bitmap = font + ch*16;
  for( line=0; line<16; line++ )
    for( row=0; row<8; row++ )
    if( bitmap[line] & (0x80 >> row) )
      lcd_putpixel( x+row, y+line, color );
    else
      lcd_putpixel( x+row, y+line, WHITE );
}
```

posición de comienzo del mapa de bits del caracter

Recorre el mapa de bits, línea a línea (de arriba abajo) y columna a columna (de izquierda a derecha)

foreground pixel

background pixel

Bitmaps

- Un fichero BMP convencional:
  - Dispone de una cabecera con información diversa
    - El inicio (relativo al comienzo) de los pixeles lo indica en los en los bytes 10..13
  - Los píxeles se ordenan por filas de izquierda a derecha de abajo a arriba
  - En la escala de gris, el 0 representa el color negro. Sucesivos valores representan tonalidades de gris cada vez más claras hasta alcanzar el color blanco
    - Si el BMP es monocromo: 0 es negro y 1 es blanco

- En el buffer de vídeo
  - No existe cabecera alguna.
  - Los píxeles se ordenan por filas de izquierda a derecha de arriba a abajo
  - En la escala de gris, el 1 representa el color blanco. Sucesivos valores representan tonalidades de gris cada vez más oscuras hasta alcanzar el color negro
    - Si el LCD se configura como monocromo: 0 es blanco y 1 es negro

Bitmaps

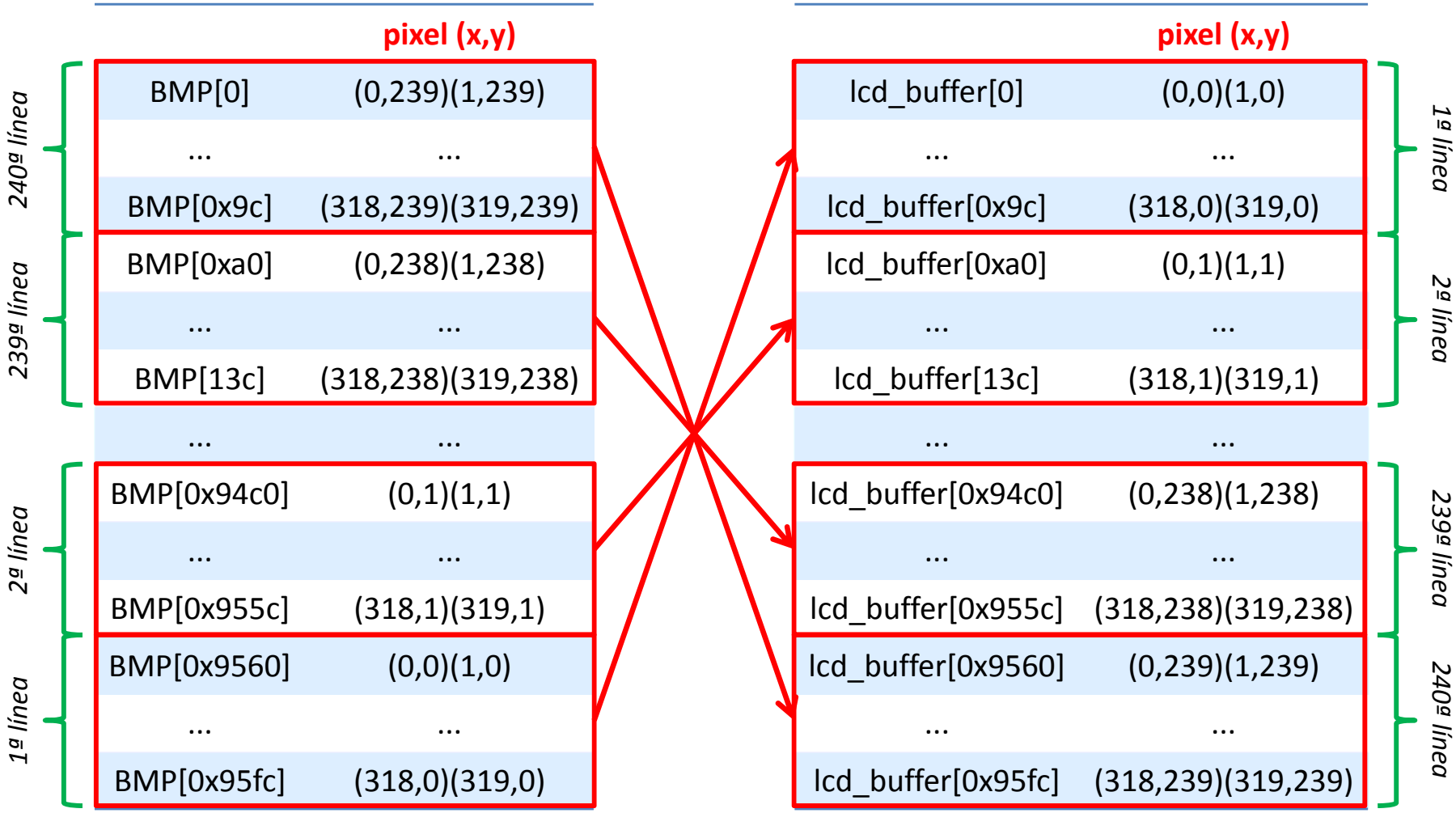
- Para visualizar un BMP a pantalla completa en el LCD
  - Deberá estar precargado en memoria
  - Tendrá una resolución de 320x240 píxeles con 16 niveles de gris
  - Deberá copiarse sobre el buffer de vídeo haciendo las siguientes transformaciones:
    - Ignorar la cabecera
    - Invertir el color
    - Voltear la imagen verticalmente (volteando las 240 filas respecto al centro de la imagen)

- Para cargar un BMP en memoria
  - Desde la consola del depurador teclear el comando

```
restore <fichero> binary <dir_memoria>
```

Bitmaps

- Voltear las líneas de una imagen significa:



Bitmaps

```
void lcd_putWallpaper( uint8 *bmp )
{
  uint32 headerSize;

  uint16 x, ySrc, yDst;
  uint16 offsetSrc, offsetDst;
```

Extrae de la cabecera del BMP la posición del comienzo de la imagen

```
  headerSize = bmp[10] + (bmp[11]<<8) + (bmp[12]<<16) + (bmp[13]<<24);

  bmp = bmp + headerSize;
```

Salta cabecera

Recorre fila a fila la imagen, llevando doble índice para facilitar el volteo de filas

```
  for( ySrc=0, yDst=240-1; ySrc<240; ySrc++, yDst-- )
  {
    offsetDst = yDst*320/2;
    offsetSrc = ySrc*320/2;
```

Calcula las posiciones en los arrays del primer byte de cada fila

```
    for( x=0; x<320/2; x++ )
```

Recorre los bytes de la fila

```
      lcd_buffer[offsetDst+x] = ~bmp[offsetSrc+x];
  }
}
```

Invierte color

Driver de LCD

lcd.h

```
#ifndef __LCD_H__
#define __LCD_H__

#include <common_types.h>

#define LCD_WIDTH   (320)
#define LCD_HEIGHT  (240)

#define LCD_BUFFER_SIZE (LCD_WIDTH*LCD_HEIGHT/2)

#define BLACK       (0xf)
#define WHITE       (0x0)
#define LIGHTGRAY   (0x5)
#define DARKGRAY    (0xa)

...

#endif
```

Dimensiones del LCD

Tamaño en bytes del buffer

Declara macros para identificar algunos colores

Driver de LCD

lcd.c

```
extern uint8 font[];
static uint8 lcd_buffer[LCD_BUFFER_SIZE];

static uint8 state;

void lcd_init( void )
{
  DITHMODE = ...;
  DP1_2    = ...;
  DP4_7    = ...;
  DP3_5    = ...;
  DP2_3    = ...;
  DP5_7    = ...;
  DP3_4    = ...;
  DP4_5    = ...;
  DP6_7    = ...;
  REDLUT   = ...;
  GREENLUT = ...;
  BLUELUT  = ...;
  LCDCON1  = ...;
  LCDCON2  = ...;
  LCDCON3  = ...;
  LCDSADDR1 = (2 << 27) | ((uint32)lcd_buffer >> 1);
  LCDSADDR2 = (1 << 29) | (((uint32)lcd_buffer + LCD_BUFFER_SIZE) & 0x3FFFFF) >> 1;
  LCDSADDR3 = ...;
  lcd_off();
}
```

El identificador de un array es un puntero a su inicio, es decir, contiene la dirección de inicio del buffer

Tareas

1. Crear el proyecto **`lab7`** a partir de una copia de uno anterior.
2. Descargar en el directorio **`lab7`** el fichero **`lab7.c`**
3. Descargar en el directorio **`BSP/include`** el fichero **`lcd.h`**
4. Descargar en el directorio **`BSP/source`** el fichero **`font8x16.c`**
5. Codificar en **`BSP/source`** el fichero **`lcd.c`**
6. Refrescar los proyectos **`BSP`** y **`lab7`**
7. Compilar primero el proyecto **`BSP`** y después el proyecto **`lab7`**.
8. Crear una configuración de depuración **`lab7`** a partir de una anterior.
9. Crear un directorio **`bmp320x240`** y descargar en él los ficheros:
   - **`*.bmp`** y **`load_bmp.txt`**
10. Conectar la placa y encenderla.
11. Arrancar OpenOCD.
12. Cargar en memoria los BMP ejecutando el script **`load_bmp.txt`**
13. Arrancar la configuración de depuración **`lab7`**