```c
//18111C_Master
/*--------------------------------------------------------*/
// DriverLCD.c
/*--------------------------------------------------------*/
//  Description : Utilitaire qui gère l'initialization de
//              l'écran ainsi que d'autres fonctions
//
//  Auteur      : Paulo Gomes
//  Version     : V1.0
//

/*--------------------------------------------------------*/
// Section: Included Files
/*--------------------------------------------------------*/

#include "DriverLcd.h"
#include "Mc32Delays.h"
#include "DefineLCD.h"
#include "glcdfont.c"
#include "app.h"
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#define D_C_low PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_C,PORTS_BIT_POS_4,false)
#define D_C_High PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_C,PORTS_BIT_POS_4,true)

#define TFT_CS_Low
PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_C,PORTS_BIT_POS_3,false)
#define TFT_CS_High
PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_C,PORTS_BIT_POS_3,true)

#define BCKL_ON PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_5,true)
#define BCKL_OFF PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_5,false)

#define RST_ON PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_6,false)
#define RST_OFF PLIB_PORTS_PinWrite(PORTS_ID_0,PORT_CHANNEL_B,PORTS_BIT_POS_6,true)
/*--------------------------------------------------------*/
// Section: User Functions
/*--------------------------------------------------------*/

// Fonction qui permet d'envoyer des commandes
void writecommand(char c)
{
    // Datacommand low
    D_C_low;
    // Chip Select low
    TFT_CS_Low;

    // Envoi des données avec le SPI

    PLIB_SPI_BufferWrite(SPI_ID_1, c);
    do {

    } while (PLIB_SPI_IsBusy(SPI_ID_1));

    // Chip select high
    TFT_CS_High;
}

// Fonction qui permet d'envoyer des datas
void writedata(char c)
{
    // Datacommand high
    D_C_High;

    // Chip Select low
    TFT_CS_Low;

    // Envoi des données avec le SPI
    PLIB_SPI_BufferWrite(SPI_ID_1, c);
    do {

    } while (PLIB_SPI_IsBusy(SPI_ID_1));

    // Chip select high
    TFT_CS_High;
}



// Initialisation de l'écran
void tft_begin(void) {

    //RST_ON;
    RST_OFF;
    BCKL_OFF;
    BCKL_ON;

    delay_ms(500); // delay 500 ms
    BCKL_OFF;
    //RST_OFF;
    TFT_CS_High;// Chip Select low
    D_C_High; // Datacommand high

    writecommand(0xEF);
    writedata(0x03);
    writedata(0x80);
    writedata(0x02);

    writecommand(0xCF);
    writedata(0x00);
    writedata(0XC1);
    writedata(0X30);

    writecommand(0xED);
    writedata(0x64);
    writedata(0x03);
    writedata(0X12);
    writedata(0X81);

    writecommand(0xE8);
    writedata(0x85);
    writedata(0x00);
    writedata(0x78);

    writecommand(0xCB);
    writedata(0x39);
    writedata(0x2C);
    writedata(0x00);
    writedata(0x34);
    writedata(0x02);

    writecommand(0xF7);
    writedata(0x20);

    writecommand(0xEA);
    writedata(0x00);
    writedata(0x00);

    writecommand(ILI9341_PWCTR1);   //Power control
    writedata(0x23);   //VRH[5:0]

    writecommand(ILI9341_PWCTR2);   //Power control
    writedata(0x10);   //SAP[2:0];BT[3:0]

    writecommand(ILI9341_VMCTR1);   //VCM control
    writedata(0x3e);
    writedata(0x28);

    writecommand(ILI9341_VMCTR2);   //VCM control2
    writedata(0x86);

    writecommand(ILI9341_MADCTL);   // Memory Access Control
    writedata(0x48);

    writecommand(ILI9341_PIXFMT);
    writedata(0x55);

    writecommand(ILI9341_FRMCTR1);
    writedata(0x00);
    writedata(0x18);

    writecommand(ILI9341_DFUNCTR);  // Display Function Control
    writedata(0x08);
    writedata(0x82);
    writedata(0x27);

    writecommand(0xF2);    // 3Gamma Function Disable
    writedata(0x00);

    writecommand(ILI9341_GAMMASET);  //Gamma curve selected
    writedata(0x01);

    writecommand(ILI9341_GMCTRP1);   //Set Gamma
    writedata(0x0F);
    writedata(0x31);
    writedata(0x2B);
    writedata(0x0C);
    writedata(0x0E);
    writedata(0x08);
    writedata(0x4E);
    writedata(0xF1);
    writedata(0x37);
    writedata(0x07);
    writedata(0x10);
    writedata(0x03);
    writedata(0x0E);
    writedata(0x09);
    writedata(0x00);

    writecommand(ILI9341_GMCTRN1);  //Set Gamma
    writedata(0x00);
    writedata(0x0E);
    writedata(0x14);
    writedata(0x03);
    writedata(0x11);
    writedata(0x07);
    writedata(0x31);
    writedata(0xC1);
    writedata(0x48);
    writedata(0x08);
    writedata(0x0F);
    writedata(0x0C);
    writedata(0x31);
    writedata(0x36);
    writedata(0x0F);
    writecommand(ILI9341_SLPOUT);  //Exit Sleep
    delay_ms(120);
    writecommand(ILI9341_DISPON);  //Display on


    _width  = ILI9341_TFTWIDTH;
    _height = ILI9341_TFTHEIGHT;


    // Backlight ON
    BCKL_ON;
}



// Remplissage de l'écran avec une couleur
void tft_fillScreen(unsigned short color) {
/* Rempli l'écran en entier avec une certaine couleur
 * Parameters: color: 16-bit color value
 * Returs:  Nothing
 */
    tft_fillRect(0, 0, _width , _height, color);
```

```c
}

// Dessine et rempli un rectangle avec une couleur
 void tft_fillRect(short x, short y, short w, short h,
                  unsigned short color) {
/* Desine un écran rempli avec une certaine couleur.
 * Commence top-left (en haut à gauche) et on peut définir
 * la taille et la hauteur
 * Parameters:
 *      x:  x-coordinate of top-left vertex; top left of screen is x=0
 *              and x increases to the right
 *      y:  y-coordinate of top-left vertex; top left of screen is y=0
 *              and y increases to the bottom
 *      w:  width of rectangle
 *      h:  height of rectangle
 *      color:  16-bit color value
 * Returns:     Nothing
 */

    // rudimentary clipping (drawChar w/big text requires this)
    if((x >= _width) || (y >= _height)) return;
    if((x + w - 1) >= _width)  w = _width  - x;
    if((y + h - 1) >= _height) h = _height - y;

    // Défini l'endroit ou écrire
    tft_setAddrWindow(x, y, x+w-1, y+h-1);

    // Masquage pour envoyer en 2x8bits
    uint8_t hi = color >> 8;
    uint8_t lo = color;

    //D_C_High;// Datacommand high
    D_C_low;
    TFT_CS_Low; // Chip Select low

    for(y=h; y>0; y--) {
      for(x=w; x>0; x--) {
        // MSB
        PLIB_SPI_BufferWrite(SPI_ID_1, hi);
        do {

        } while (PLIB_SPI_IsBusy(SPI_ID_1));

        // LSB
            PLIB_SPI_BufferWrite(SPI_ID_1, lo);
        do {

        } while (PLIB_SPI_IsBusy(SPI_ID_1));

        }
    }
    TFT_CS_High; // Chip Select high
}

 // Fonction qui choisi l'endroit ou on veut écrire ou dessiner
void tft_setAddrWindow(unsigned short x0, unsigned short y0, unsigned short x1,
 unsigned short y1) {

    // x
    writecommand(ILI9341_CASET); // Column addr set
    writedata(x0 >> 8);
    writedata(x0 & 0xFF);     // XSTART
    writedata(x1 >> 8);
    writedata(x1 & 0xFF);     // XEND

    // Y
    writecommand(ILI9341_PASET); // Row addr set
    writedata(y0>>8);
    writedata(y0);     // YSTART
    writedata(y1>>8);
    writedata(y1);     // YEND

    // Écrit dans la mémoire
    writecommand(ILI9341_RAMWR); // write to RAM
}

// Fonction qui permet de dessiner des Pixels
void tft_drawPixel(short x, short y, unsigned short color) {
/* Dessine un pixel dans la localization voulu (x,y) avec une certaine couleur
 * Parameters:
 *      x:  x-coordinate of pixel to draw; top left of screen is x=0
 *              and x increases to the right
 *      y:  y-coordinate of pixel to draw; top left of screen is y=0
 *              and y increases to the bottom
 *      color:  16-bit color value
 * Returns:     Nothing
 */

    if((x < 0) ||(x >= _width) || (y < 0) || (y >= _height)) return;

    // Défini l'endroit ou écrire
    tft_setAddrWindow(x,y,x+1,y+1);

    // Masquage pour envoyer en 2x8bits
    uint8_t hi = color >> 8;
    uint8_t lo = color;

    D_C_High;// Datacommand high
    TFT_CS_Low; // Chip Select low

    // MSB
    PLIB_SPI_BufferWrite(SPI_ID_1, hi);
    do {

    } while (PLIB_SPI_IsBusy(SPI_ID_1));

    // LSB
        PLIB_SPI_BufferWrite(SPI_ID_1, lo);
    do {

    } while (PLIB_SPI_IsBusy(SPI_ID_1));

    TFT_CS_High; // Chip Select high
}



/* DrawLine(Xa,Ya,Xb,Yb,color);
 * dessine une ligne entre 2 points A et B
 * http://www.brackeen.com/vga/shapes.html
 */
void DrawLine(short Xa, short Ya, short Xb, short Yb, unsigned short color)
{
    int dx,dy,sdx,sdy,px,py,dxabs,dyabs,i;
    float slope;

    dx=Xb-Xa;      /* the horizontal distance of the line */
    dy=Yb-Ya;      /* the vertical distance of the line */
    dxabs=abs(dx);
    dyabs=abs(dy);
    if(dx==0 ) {tft_drawFastVLine(Xa,Ya,dy,color);return;}
    if(dy==0 ) {tft_drawFastHLine(Xa,Ya,dx,color);return;}
    sdx=(dx>0)? 1:-1;
    sdy=(dy>0)? 1:-1;
    if (dxabs>=dyabs) /* the line is more horizontal than vertical */
    {
        slope=(float)dy / (float)dx;
        for(i=0;i!=dx;i+=sdx)
        {
          px=i+Xa;
          py=slope*i+Ya;
          tft_drawPixel(px,py,color);
        }
    }
    else /* the line is more vertical than horizontal */
    {
        slope=(float)dx / (float)dy;
        for(i=0;i!=dy;i+=sdy)
        {
          px=slope*i+Xa;
          py=i+Ya;
          tft_drawPixel(px,py,color);
        }
    }


}

// Fonction qui dessine des lignes horizantales
void tft_drawFastHLine(short x, short y, short w, unsigned short color)
{
    // Rudimentary clipping
    if((x >= _width) || (y >= _height)) return;
    if((x+w-1) >= _width)  w = _width-x;

    // Défini l'endroit ou écrire
    tft_setAddrWindow(x,y,x+w-1,y);

    D_C_High;// Datacommand high
    TFT_CS_Low; // Chip Select low

    // Masquage pour envoyer en 2x8bits
    uint8_t hi = color >> 8;
    uint8_t lo = color;

    while (w--) {
        writedata(hi); // MSB
        writedata(lo); // LSB
    }

    TFT_CS_High;// Chip Select high
}

// Fonction qui dessine des lignes verticales
void tft_drawFastVLine(short x, short y, short h, unsigned short color) {

    // Rudimentary clipping
    if((x >= _width) || (y >= _height)) return;
    if((y+h-1) >= _height)
     h = _height-y;

    // Défini l'endroit ou écrire
    tft_setAddrWindow(x, y, x, y+h-1);

    D_C_High;// Datacommand high
    TFT_CS_Low;// Chip Select low

    // Masquage pour envoyer en 2x8bits
    uint8_t hi = color >> 8;
    uint8_t lo = color;

    while (h--) {
        writedata(hi); // MSB
        writedata(lo); // LSB
    }

    TFT_CS_High;// Chip Select high
}

// Fonction qui permet de dessiner un caractère
void drawChar(short x, short y, unsigned char c,
        unsigned short fgcolor, unsigned short bgcolor, unsigned short size)
{
    if((x >= _width)            || // Clip right
       (y >= _height)           || // Clip bottom
       ((x + 6 * size - 1) < 0) || // Clip left  TODO: is this correct?
       ((y + 8 * size - 1) < 0))   // Clip top   TODO: is this correct?
        return;

```

```c
        if (fgcolor == bgcolor) {
            // This transparent approach is only about 20% faster
            if (size == 1) {
                uint8_t mask = 0x01;
                int16_t xoff, yoff;
                for (yoff=0; yoff < 8; yoff++) {
                    uint8_t line = 0;
                    for (xoff=0; xoff < 5; xoff++) {
                        if (font[c * 5 + xoff] & mask) line |= 1;
                        line <<= 1;
                    }
                    line >>= 1;
                    xoff = 0;
                    while (line) {
                        if (line == 0x1F) {
                            tft_drawFastHLine(x + xoff, y + yoff, 5, fgcolor);
                            break;
                        } else if (line == 0x1E) {
                            tft_drawFastHLine(x + xoff, y + yoff, 4, fgcolor);
                            break;
                        } else if ((line & 0x1C) == 0x1C) {
                            tft_drawFastHLine(x + xoff, y + yoff, 3, fgcolor);
                            line <<= 4;
                            xoff += 4;
                        } else if ((line & 0x18) == 0x18) {
                            tft_drawFastHLine(x + xoff, y + yoff, 2, fgcolor);
                            line <<= 3;
                            xoff += 3;
                        } else if ((line & 0x10) == 0x10) {
                            tft_drawPixel(x + xoff, y + yoff, fgcolor);
                            line <<= 2;
                            xoff += 2;
                        } else {
                            line <<= 1;
                            xoff += 1;
                        }
                    }
                    mask = mask << 1;
                }
            } else {
                uint8_t mask = 0x01;
                int16_t xoff, yoff;
                for (yoff=0; yoff < 8; yoff++) {
                    uint8_t line = 0;
                    for (xoff=0; xoff < 5; xoff++) {
                        if (font[c * 5 + xoff] & mask) line |= 1;
                        line <<= 1;
                    }
                    line >>= 1;
                    xoff = 0;
                    while (line) {
                        if (line == 0x1F) {
                            tft_fillRect(x + xoff * size, y + yoff * size,
                                5 * size, size, fgcolor);
                            break;
                        } else if (line == 0x1E) {
                            tft_fillRect(x + xoff * size, y + yoff * size,
                                4 * size, size, fgcolor);
                            break;
                        } else if ((line & 0x1C) == 0x1C) {
                            tft_fillRect(x + xoff * size, y + yoff * size,
                                3 * size, size, fgcolor);
                            line <<= 4;
                            xoff += 4;
                        } else if ((line & 0x18) == 0x18) {
                            tft_fillRect(x + xoff * size, y + yoff * size,
                                2 * size, size, fgcolor);
                            line <<= 3;
                            xoff += 3;
                        } else if ((line & 0x10) == 0x10) {
                            tft_fillRect(x + xoff * size, y + yoff * size,
                                size, size, fgcolor);
                            line <<= 2;
                            xoff += 2;
                        } else {
                            line <<= 1;
                            xoff += 1;
                        }
                    }
                    mask = mask << 1;
                }
            }
        } else {
            // This solid background approach is about 5 time faster
            tft_setAddrWindow(x, y, x + 6 * size - 1, y + 8 * size - 1);
            uint8_t xr, yr;
            uint8_t mask = 0x01;
            uint16_t color;
            uint8_t hi ;
            uint8_t lo ;
            uint8_t bhi = bgcolor >> 8;
            uint8_t blo = bgcolor;
            for (y=0; y < 8; y++) {
                for (yr=0; yr < size; yr++) {
                    for (x=0; x < 5; x++) {
                        if (font[c * 5 + x] & mask) {
                            color = fgcolor;
                        } else {
                            color = bgcolor;
                        }
                        hi = color >> 8;
                        lo = color;
                        for (xr=0; xr < size; xr++) {

                            writedata(hi);
                            writedata(lo);
                        }
                    }
                    for (xr=0; xr < size; xr++) {
                        writedata(bhi);
                        writedata(blo);
                    }
                }
                mask = mask << 1;
            }
        }
    }

    // Fonction qui permet de faire une rotation de 90° de l'écran
    void setRotation(short m) {

        writecommand(ILI9341_MADCTL);
        short rotation = m % 4; // can't be higher than 3
        switch (rotation) {
                case 0: // Écran initial
                writedata(MADCTL_MX | MADCTL_BGR);
                _width  = ILI9341_TFTWIDTH;
                _height = ILI9341_TFTHEIGHT;
            break;

                case 1: // Rotation de l'écran 90°
                writedata(MADCTL_MV | MADCTL_BGR);
                _width  = ILI9341_TFTHEIGHT;
                _height = ILI9341_TFTWIDTH;
            break;

                case 2: // Rotation de l'écran 180°
                writedata(MADCTL_MY | MADCTL_BGR);
                _width  = ILI9341_TFTWIDTH;
                _height = ILI9341_TFTHEIGHT;
            break;

                case 3: // Rotation de l'écran 270°
                writedata(MADCTL_MX | MADCTL_MY | MADCTL_MV | MADCTL_BGR);
                _width  = ILI9341_TFTHEIGHT;
                _height = ILI9341_TFTWIDTH;
            break;
        }
    }

    // Defini l'emplacement du curseur
    void tft_setCursor(short x, short y) {
    /* Set cursor for text to be printed
     * Parameters:
     *      x = x-coordinate of top-left of text starting
     *      y = y-coordinate of top-left of text starting
     * Returns: Nothing
     */

        cursor_x = x; // en haut à gauche
        cursor_y = y; // en haut à gauche
    }

    // Modifie la couleur de la police
    void tft_setTextColor(unsigned short c) {
        // For 'transparent' background, we'll set the bg
        // to the same as fg instead of using a flag
        textcolor = textbgcolor = c;
    }
    // Modifie la couleur ET le background de la police
    void tft_setTextColor_F(unsigned short fore, unsigned short Back)
    {

        textcolor =  fore;
        textbgcolor = Back;
    }

    // Fonction qui permet d'écrire une chaine de caractères
    void tft_writeString(char* str){
    /* Call tft_setCursor(), tft_setTextColor(), tft_setTextSize()
     * as necessary before printing
     */
        while (*str){
            tft_write(*str++);
        }
    }

    // Fonction qui permet d'écrire
    void tft_write(unsigned char c){
        if (c == '\n') {
            cursor_y += textsize*8;
            cursor_x  = 0;
            } else if (c == '\r') {
            // skip em
            } else if (c == '\t'){
            int new_x = cursor_x + 4;
            if (new_x < _width){
            cursor_x = new_x;
            }
            } else {
            drawChar(cursor_x, cursor_y, c, textcolor, textbgcolor, textsize);
            cursor_x += textsize*6;
            if (wrap && (cursor_x > (_width - textsize*6))) {
            cursor_y += textsize*8;
            cursor_x = 0;
            }
        }
    }

    // Modifie la taille du text à afficher
    void tft_setTextSize(unsigned char s) {
    /*Set size of text to be displayed
     * Parameters:
     *      s = text size (1 being smallest)
     * Returns: nothing
     */
      textsize = (s > 0) ? s : 1;
    }
```