Final project microprocessors

Santiago Arenas Martín

Introduction

Our project tries to be an alternative to playing chess on a TFT screen, using switches to indicate the moves. In the game it is also very important to note the movements made, which is why the UART acts as a guide for the players, change of the turn of the players, movements made, selected pieces, captures of pieces, as well as illegal movements.

Software

Because of the creation of a new type of variable, part, all the functions that deal with it are located below the main. The creation of such a variable simplifies simultaneous operations on the same part at the same time.

Main.c

It contains the while(1), the screen representation function, the state machine, and the functions that obtain the data to send information over the UART. To render the screen in a way that is comfortable for the players, which in turn is in accordance with the coordinates of chess, an offset is defined[1].

UART.c

It contains the UART initializer at 9600 baud, putsUart to send various strings, UART_Mov to report each movement or capture, UART_Info to inform of the selected piece and UART_Indica to tell the user what to do at all times.

TFTDriver.c

It defines the colors for the screen, the function to erase it in its entirety (something that must be done every time a new position has to be represented), the fillRect function to paint the squares, fillCircle() to paint pawns and print() to put the letters that indicate the position of the pieces. This, along with spi.h, is available in the Moodle of the subject, our only change is in the annotations, since it does not use the pins indicated by spi.h.

States

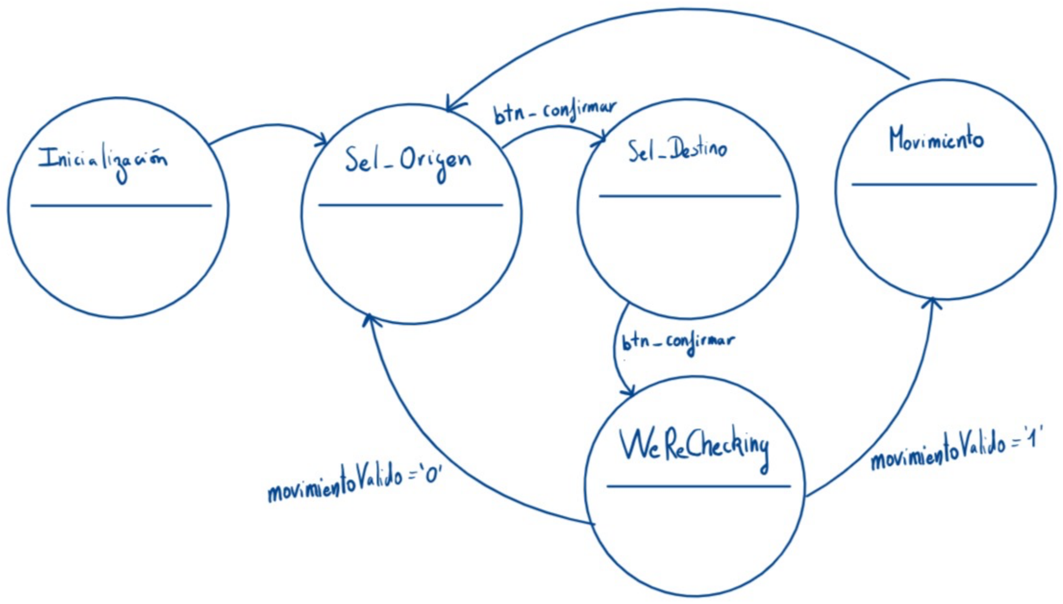
1. Initialization: Indicates a message that the source needs to select. It also lights up the first of the indicator LEDs, which is never seen with the human eye.
2. Sel_Origen: Wait until the player indicates that the switches indicate the position of the piece they want to move. Indicates the part and position to the UART. The second indicator LED lights up.
3. Sel_Destino: the destination of the piece is indicated in the same way. It is indicated by UART. The third indicator LED lights up.
4. WeReChecking: it is checked that the move is legal, both that the player who touches him moves a piece and that the move is correct for the piece he has selected. If it is not,

[1] The start of coordinates on the screen is located in square h1, where the white rook on the right would go. To artificially change this start of coordinates, it is determined that the column must be subtracted from 7, in this way if you enter a 0 (you would be selecting row a) row 7 is selected, which would be the one on the left of the whole, which is where the start of coordinates is in chess.

the reason (shift or illegality of the movement) is indicated by UART, and it is returned to the Sel_Origen status.

5. Movement: the movement is performed and taught on the screen. It is back to the Sel_Origen state.

The operation of the state machine is summarized in the following image:

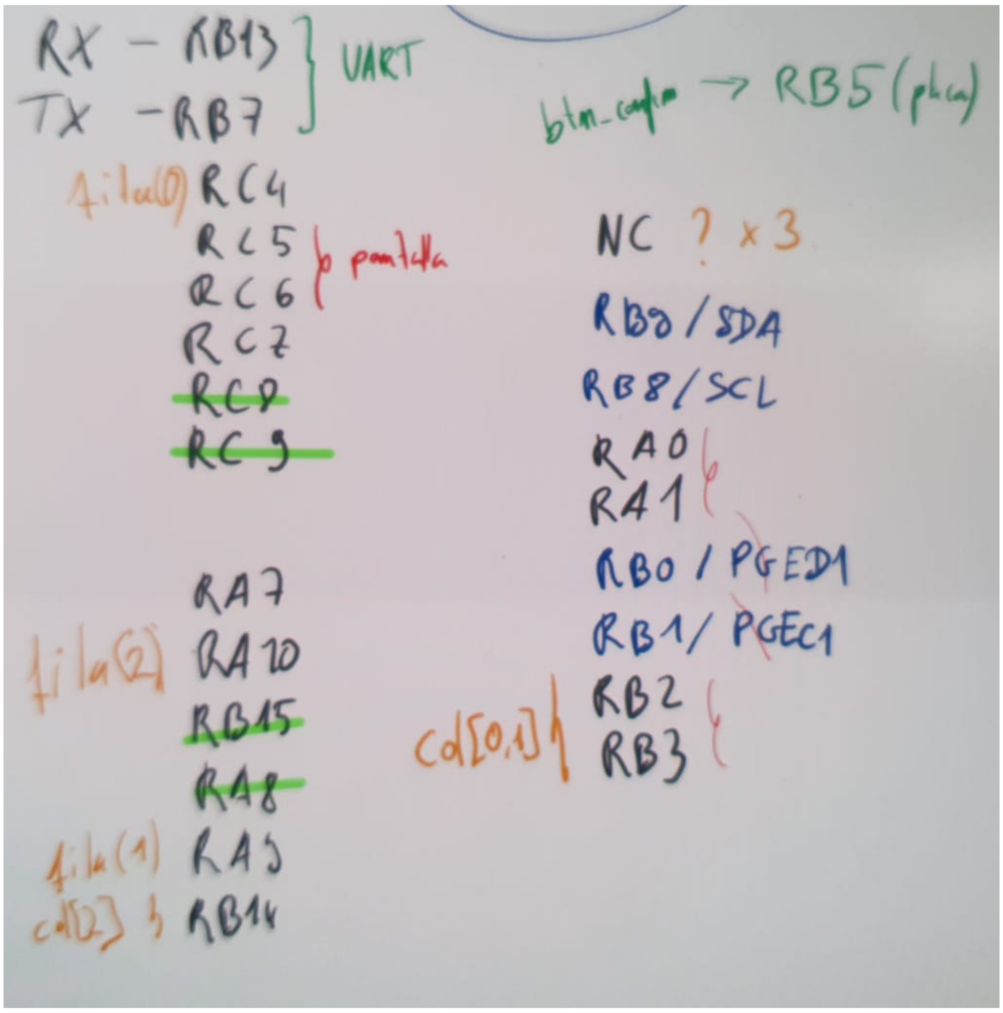


An example of a code interaction would be the following:
Interruptores —> main.c <—> UART. C <—> main.c —> TFTDriver.h

Hardware

This was, without a doubt, the worst part of our project, the one that gave the most headaches to the members. Initially we thought of making a board of 64 buttons, which quickly evolved to 16 buttons, an idea that we rejected after seeing the difficulty of the physical needs of so many circuits. A multitude of hours were spent trying to make such a design work, but it proved impossible. For the final design, it was decided to use two sets of 6 switches, one to indicate the row and the other to indicate the column. You could really use one set alone, but for convenience we decided to use the second one, because they are quite small and it is more accurate to have them separate.

The layout, with the pins used by all the circuits, can be seen in the following image:



All pushbuttons are powered by 3.3V, they all use the same GND. The 5V output is used by the UART power supply, and is used to the other GND of the board. We forcibly learned that you couldn't use entries that appear in yellow. RC5 and RC6 are reserved for the internal workings of the display, also requiring a bridge between RC8, RC9, RB15 and RA8. The rows and columns are read following the scheme represented in the orange image.

Conclusion

This has been a project that we considered very ambitious from the beginning, it proposed a series of problems, and we were changing and adapting until we reached a final version with which we are more than satisfied. Both of us are very interested in the world of chess, and having been able to do a project about it makes us proud in a way that is not possible to put into words. After many weeks of effort, seeing that it finally worked is very, very gratifying.

A video of how the project works can be seen here, with all the functionalities implemented.

Annex

Main.c

```
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <string.h>
#include <xc.h>
#include "Pic32Ini.h"
#include "InitTimer2.h"
#include "UART.h"
//#include "ChessSupport.h"
#include "Screen.h"
#include "TftDriver.h"

#define OFFSET 7

#define ROWS 8
#define 8 COLUMNS
#define PEON   1
#define BISHOP 2
#define KNIGHT 3
#define TOWER 4
#define LADY 5
#define KING 6
#define EMPTY 0
#define SQUARE_SIZE 16

#define BTN_F 4
#define BTN_B2 2
#define BTN_B14 14
#define BTN_CONFIRM 5
#define LED_ROJO 0

typedef struct pieza {
    char name; What piece is it
    char color; white = 1, quarter note = 0
    Sir Kalamana;
    Sir Phila;
    char movida; if it has been moved = 0
} piece;


static piece boardHeading[COLUMNS][ROWS];  Board with part structure
static uint8_t sel  = 0;   6 selection bits, 3 least significant for
columns, 3 most significant for rows
static piece PartMove;
static piece PieceFood;
static uint8_t turnoBlancas = 1; // 1 para blancas, 0 para negras


void RepresentarTab(void);
void Lectura(uint8_t *columna, uint8_t *fila);
void drawChessBoard(pieza board[8][8]);
char getName(char name);
Sir Gatkulamana (Sir Kalamana);
Four getphilae (four phyla);
```

```
uint8_t  movimiento(uint8_t  columna_orig,  uint8_t  fila_orig,  uint8_t
columna_dest, uint8_t fila_dest);
uint8_t   movimientoValido(uint8_t   columna_orig,   uint8_t   fila_orig,
uint8_t columna_dest, uint8_t fila_dest);

int main(void) {
    ANSELA = ~((1 << 9) | (1 << 10));
    ANSELB = ~((1 << 2) | (1 << 3) | (1 << 14) | (1 << 5));
    ANSELC = ~((1 << 4));

 TRISA = ((1 << 9) | (1 << 10);
    TRISB = ((1 << 2) | (1 << 3) | (1 << 14) | (1 << 5));
    TRISC = (1 << 4);

    CAN = 0;  At boot we leave all unused outputs at 0
    LATB = 0;
    LATC = 0xF;  I turn off the LEDs on the board


    initialize TFT(1);

    uint8_t btn_confirmar_act = 0, btn_confirmar_ant = 0;

    enum {
        Initialization, Sel_Origen, Sel_Destino, WeReChecking, Movement
    } status;
    state = Initialization;

    InitializeUART1(9600);
    RepresentTab();

    INTCONbits.MVEC = 1;  Modo multivector
    asm("ei");  Enabled Interr.

    Initialize the screen
    extern uint8_t SmallFont[]; //Esta definido en DefaultFonts.c

    // Initialize the LCD

    setFont(SmallFont);
    drawChessBoard(tableroPartida);

    uint8_t columna_orig = 0, fila_orig = 0, columna_dest = 0, fila_dest
= 0;


    while (1) {

        Reading the confirmation button
        btn_confirmar_act = (PORTB >> BTN_CONFIRM) & 0x1; // BotÃ□Â³n
de confirmaciÃ□Â³n

        Transition of states ( delta function )
        switch (estado) {
            case Initialization:
                UART_Indica(1);
                state = Sel_Origen;
                break;
            case Sel_Origen:
                if (btn_confirmar_act < btn_confirmar_ant) {
                    Reading(&columna_orig,  &fila_orig);    The  buttons
that give the column and row are read
                    UART_Info(columna_orig, fila_orig);
                    UART_Indica(0);
                    state = Sel_Destino;
                }
                break;
            case Sel_Destino:
                if (btn_confirmar_act < btn_confirmar_ant) {
                    destination = 0;
                    Reading(&columna_dest,  &fila_dest);    The  buttons
that give the column and row are read
                    UART_Info(columna_dest, fila_dest);
                    estado = WeReChecking;
                }
                break;
            case WeReChecking:
                if   (Valid(columna_orig,   fila_orig,   columna_dest,
fila_dest)) {
                    state = Movement;
                } else {
                    state = Sel_Origen;
                }
                break;
            Case Movement:
                state = Sel_Origen;

        }


        // FunciÃ¯Â¿Â½n Lambda
        if (state == Movement) {
            if   (movement(columna_orig,   fila_orig,   columna_dest,
fila_dest)) {
                Eaten a piece
                UART_Mov(1,                     getColumn(columna_dest),
getRow(fila_dest),getColumn(columna_orig),
getRow(fila_orig),MovePiece.name, FoodPiece.name);
            } else {
                Not eaten any pieces
                UART_Mov(0, getColumn(columna_dest), getRow(fila_dest),
getColumn(columna_orig),      getRow(fila_orig),      MovePiece.name,
FoodPiece.name);
            }
            drawChessBoard(SplitBoard);   Representation  of  the  board
after the move
            White's turn ^= 1;  Change of shift

        }

        The output bit is updated
        if (state == Initialization) {
            LATCCLR  =  (1  <<  0);   To  indicate  that  the  game  has  not
started
```

```
        } else {
            LATCSET = (1 << 0);
        }
        if (estado == Sel_Origen) {
            LATCCLR = (1 << 1);  To indicate that the game has not
started
        } else {
            LATCSET = (1 << 1);
        }
        if (estado == Sel_Destino) {
            LATCCLR = (1 << 2);  To indicate that the game has not
started
        } else {
            LATCHET = (1 << 2);
        }
        if (state == Movement) {
            LATCCLR = (1 << 3);  To indicate that the game has not
started
        } else {
            LATCSET = (1 << 3);
        }

        btn_confirmar_ant = btn_confirmar_act;
    }

    return 0;
}

Function that places the initial values for the pieces in their squares
void RerepresentTab() {
    for (int c = 0; c < 8; c++) {
        for (int f = 0; f < 8; f++) {
            piece p;
            p.columna = c;
            p.row = f;
            p.movida = 1;

            if (f == 1 || f == 6) {
                p.name = PEON;
                p.color = (f == 1) ? 1 : 0;  1 for white, 0 for black
            } else if (f == 0 || f == 7) {
                p.color = (f == 0) ? 1 : 0;  1 for white, 0 for black
                if (c == 0 || c == 7) {
                    p.name = TOWER;
                } else if (c == 1 || c == 6) {
                    p.name = HORSE;
                } else if (c == 2 || c == 5) {
                    p.name = BISHOP;
                } else if (c == 3) {
                    p.name = KING;
                } else if (c == 4) {
                    p.name = DAMA;
                }
            } else {
                p.noun = VOID;  Empty box
                p.color = 2;  Color not applicable
            }
            tableroPartida[c][f] = p;
        }
    }
}

void Lectura(uint8_t *columna, uint8_t *fila) {
    Reading the value of the pushbuttons
    uint8_t columna_previa = 0;
    columna_previa = ((PORTB >> BTN_B2) & 0x3) | (((PORTB >> BTN_B14) &
0x1) << 2);
    *columna = OFFSET - columna_previa;
    *fila = ((PORTC >> 4) & 0x1) | (((PORTA >> 9) & 0x1) << 1) | (((PORTA
>> 10) & 0x1) << 2);
}

uint8_t  movimiento(uint8_t  columna_orig,  uint8_t  fila_orig,  uint8_t
columna_dest, uint8_t fila_dest) {
    uint8_t comida = 0;

    MovePiece= Gameboard[columna_orig][fila_orig];   You  get  the  piece
to move
    boardGame[columna_orig][fila_orig].name = VOID;
    tableroPartida[columna_orig][fila_orig].color = 2;
    Eaten a piece
    if (boardHeading[columna_dest][fila_dest].name != VOID) {
        PieceFood = BoardGame[columna_dest][fila_dest];
        food = 1;
    }

    Corona
    if (PiezaMover.nombre == PEON && (fila_dest == 0 || fila_dest == 7))
{
        MovePiece.name = LADY;
    }
    MovePart.moved = 0;
    boardGame[columna_dest][fila_dest].name = MovePiece.name;
    boardMatch[columna_dest][fila_dest].color = PieceMove.color;
    boardGame[columna_dest][fila_dest].moved = MovePiece.moved;
    return food;  If he has eaten, he returns 1, if not 0
}

void drawChessBoard(pieza board[COLUMNAS][FILAS]) {
    clrScr();  Clear the screen

    int color = 1;

    for (int i = 0; i < COLUMNAS; i++) {
        for (int j = 0; j < FILAS; j++) {
            if (color) {
                setColor(VGA_WHITE);
            } else {
                setColor(VGA_BLACK);
            }
            fillRect(i  *  SQUARE_SIZE,  j  *  SQUARE_SIZE,  (i  +  1)  *
SQUARE_SIZE, (j + 1) * SQUARE_SIZE);
            color = !color;
        }
        color = !color;
    }

    Represent the pieces
    for (int i = 0; i < COLUMNAS; i++) {
        for (int j = 0; j < FILAS; j++) {
            if (board[i][j].number == PEON) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                fillCircle(i * SQUARE_SIZE + SQUARE_SIZE / 2, j *
SQUARE_SIZE + SQUARE_SIZE / 2, SQUARE_SIZE / 2 - 2);
            }
            if (board[i][j].number == ALFIL) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                print("A", i * SQUARE_SIZE + 9, j * SQUARE_SIZE + 14,
180);
            }
            if (board[i][j].name == KNIGHT) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                print("C", i * SQUARE_SIZE + 9, j * SQUARE_SIZE + 14,
180);
            }
            if (board[i][j].nombre == TORRE) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                print("T", i * SQUARE_SIZE + 9, j * SQUARE_SIZE + 14,
180);
            }
            if ( board[i][j].number == DAMA) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                print("D", i * SQUARE_SIZE + 9, j * SQUARE_SIZE + 14,
180);
            }
            if (board[i][j].nombre == REY) {
                if (board[i][j].color == 0) {
                    setColor(VGA_RED);
                } else {
                    setColor(VGA_OLIVE);
                }
                print("R", i * SQUARE_SIZE + 9, j * SQUARE_SIZE + 14,
180);
            }
        }
    }
}

uint8_t  movimientoValido(uint8_t  columna_orig,  uint8_t  fila_orig,
uint8_t columna_dest, uint8_t fila_dest) {

    Check if they have stayed in the same place
    if (columna_orig == columna_dest && fila_orig == fila_dest) {
        putsUART("You must move the piece\n");
        return 0;
    }

    Check if the movement is inside the board (in 8x8, this would not
be necessary, the buttons allow it)
    if (columna_orig < 0 || columna_orig >= COLUMNAS || fila_orig < 0
|| fila_orig >= FILAS || columna_dest < 0 || columna_dest >= COLUMNAS
|| fila_dest < 0 || fila_dest >= FILAS) {
        putsUART("Move off the board\n");
        return 0;
    }

    Check if it's their turn
    if (boardMatch[columna_orig][fila_orig].color != turnWhite) {
        putsUART("It's not your turn\n");
        return 0;
    }

    Check if the piece you capture is the same color
    if          (GameBoard[columna_dest][fila_dest].color          ==
GameBoard[columna_orig][fila_orig].color) {
        putsUART("You can't capture a piece of your color\n");
        return 0;
    }

    Check if the movement is according to the piece
    switch (gameboard[columna_orig][fila_orig].name) {
        case PEON:
            if (tableroPartida[columna_orig][fila_orig].color == 1) {
                White pawn moves forward two squares (first move only)
                if    ((GameBoard[columna_orig][fila_orig].moved)    &
((fila_dest  ==  fila_orig  +  2  &&  columna_dest  ==  columna_orig)  &&
(GameBoard[columna_dest][fila_dest].name == EMPTY))) {
                    return 1;
                }                     Real Madrid advances
                else if ((fila_dest == fila_orig + 1 && columna_dest ==
columna_orig) && (boardGame[columna_dest][fila_dest].name == EMPTY)) {
                    return 1;
                }
                    A piece is captured with a peel
                else if (fila_dest == fila_orig + 1 && (columna_dest ==
columna_orig  +  1  ||  columna_dest  ==  columna_orig  -  1)  &&
tableroPartida[columna_dest][fila_dest].nombre != VACIO) {
```

```
                    return 1;
                } else {
                    putsUART("Movement of peÃ³n no valid\n");
                }
            } else {
                Black pawn advances two squares (first move only)
                if    ((GameBoard[columna_orig][fila_orig].moved)    &
((fila_dest  ==  fila_orig  -  2  &&  columna_dest  ==  columna_orig)  &&
(GameBoard[columna_dest][fila_dest].name == EMPTY))) {
                    return 1;
                }                      Black pawn advances
                else if ((fila_dest == fila_orig - 1 && columna_dest ==
columna_orig) && (GameBoard[columna_dest][fila_dest].name == EMPTY)) {
                    return 1;
                    A piece is captured with a peel
                } else if (fila_dest == fila_orig - 1 && (columna_dest
==   columna_orig   +   1   ||   columna_dest   ==   columna_orig   -   1)   &&
tableroPartida[columna_dest][fila_dest].nombre != VACIO) {
                    return 1;
                } else {
                    putsUART("Movement of peÃ³n no valid\n");
                }
            }
            break;
        case ALFIL:
            if  (abs(columna_dest  -  columna_orig)  ==  abs(fila_dest  -
fila_orig)) {
                return 1;
            } else {
 putsUART("Remember that bishops move diagonally");
            }
            break;
        case CABALLO:
            if ((abs(columna_dest - columna_orig) == 2 && abs(fila_dest
-  fila_orig)  ==  1)  ||  (abs(columna_dest  -  columna_orig)  ==  1  &&
abs(fila_dest - fila_orig) == 2)) {
                return 1;
            } else {
                putsUART("Remember that horses move in the shape of an
L\n");
            }
            break;
        case TORRE:
            if (columna_dest == columna_orig || fila_dest == fila_orig)
{
                return 1;
            } else {
 putsUART("Remember that towers move vertically and horizontally");
            }
            break;
        case DAMA:
            if (columna_dest == columna_orig || fila_dest == fila_orig
|| abs(columna_dest - columna_orig) == abs(fila_dest - fila_orig)) {
                return 1;
            } else {
 putsUART("Remember  that  ladies  move  in  all  directions,  but  not
everywhere");
```

```
                }
                break;
            case REY:
                if (abs(columna_dest - columna_orig) <= 1 && abs(fila_dest
- fila_orig) <= 1) {
                    return 1;
                } else {
 putsUART("Remember that the king moves in all directions, but only 1
square away\n");
                }
                break;
            default:
            {
 putsUART("The piece in that box is not recognized");
            }
    }
    return 0;
}

Sir Gatklamana (Sir Kalamana) {
    switch (columna) {
        case 7:
            return 'a';
        case 6:
            return 'b';
        case 5:
            return 'c';
        case 4:
            return 'd';
        case 3:
            return 'e';
        case 2:
            return 'f';
        case 1:
            return 'g';
        case 0:
            return 'h';
    }
}

Four Getphila (Four Phila) {
    switch (fila) {
        case 0:
            return '1';
        case 1:
            return '2';
        case 2:
            return '3';
        case 3:
            return '4';
        case 4:
            return '5';
        case 5:
            return '6';
        case 6:
            return '7';
        case 7:
            return '8';
    }
}

char getName(char name) {
    switch {
        case PEON:
            return 'P';
        case ALFIL:
            return 'A';
        case CABALLO:
            return 'C';
        case TORRE:
            return 'T';
        case DAMA:
            return 'D';
        case REY:
            return 'R';
        case VACIO:
            return '0';
    }
}
```

Uart.c

```
#include <xc.h>
#include "Pic32Ini.h"
#include <string.h>
#include <stdio.h>
//#include "ChessSupport.h"
//#include "UART.h"
//#include <math.h>

static int divisor = 16; static to not overwrite the value (Ask)

void InitializeUART1(int baud){
    InitializeReloj();
    ANSELB &= ~((1<<13)|(1<<7));  RB13 and RB7 are configured as digital
    //ANSELC &= ~(1<<5);
    TRISB |= 1<<13;  and as an entrance
    LATB |= 1<<7;  A 1 if the transmitter is disabled.
    SYSKEY=0xAA996655;  Records are unlocked
    SYSKEY=0x556699AA;  de configuraci?n.
    U1RXR = 3;  U1RX connected to RB13.
    RPB7R = 1;  U1TX connected to your RB7.
    SYSKEY=0x1CA11CA1;  They are blocked again.
    if(baudios > 38400){
        divisor = 4;
    }
    U1BRG = (5000000/(splitter*baud)) - 1;  I calculate the baud just
in case
    ceil of the result if necessary
    IFS1bits.U1RXIF = 0;  Borro flag by itself
    IEC1bits.U1RXIE = 1;  Enable interrupts
    IFS1bits.U1TXIF = 0;  I delete flag from the transmitter just in
case
    IPC8bits.U1IP = 3;  Priority 3
    IPC8bits.U1IS = 1; // Subprioridad 1
    U1STAbits.URXISEL = 0;  Interruption when 1 talk arrives.
    U1STAbits.UTXISEL = 2;  Interruption when FIFO empties.
    U1STAbits.URXEN = 1;  The receiver is enabled.
    U1STAbits.UTXEN = 1;  And the transmitter, force a 1st send
    U1MODE = 0x8000;  Se arranca la UART
}

#define TAM_COLA 100
static int icab_rx = 0, icol_rx = 0, icab_tx = 0, icol_tx = 0;
static char cola_rx[TAM_COLA], cola_tx[TAM_COLA];

__attribute__((vector(32),   interrupt(IPL3SOFT),   nomips16))   void
InterrupcionUART1(void) {
    if (IFS1bits.U1RXIF == 1) {  Interrupted the receiver
        if ((icab_rx + 1 == icol_rx) ||
                (icab_rx + 1 == TAM_COLA && icol_rx == 0)) {
            The tail is there? Full
        } else {
            cola_rx[icab_rx] = U1RXREG; // Lee car?cter de la UART
            icab_rx++;
            if (icab_rx == TAM_COLA) {
                icab_rx = 0;
            }
        }
        IFS1bits.U1RXIF = 0;  And to finish the flag is deleted
    }
    if (IFS1bits.U1TXIF == 1) {  You have interrupted the transmitter
        if (icol_tx != icab_tx) {  New data
            U1TXREG = cola_tx[icol_tx];
            icol_tx++;
            if (icol_tx == TAM_COLA) {
                icol_tx = 0;
            }
        } else {  The queue has been emptied.
            IEC1bits.U1TXIE = 0;  To avoid endless looping
        }
        IFS1bits.U1TXIF = 0;  And to finish the flag is deleted
    }
}

char getcUART(void) {
    char c;
    if (icol_rx != icab_rx) {  New data
        c = cola_rx[icol_rx];
        icol_rx++;
        if (icol_rx == TAM_COLA) {
            icol_rx = 0;
        }
    } else {
        c = '\0';  Nothing has arrived
    }
    return c;
}

void putsUART(char *ps) {
    char *ps = s; We initialize a pointer at the beginning of the chain
    while (*ps != '\0') {  We iterate until we find the null terminator
        if ((icab_tx + 1 == icol_tx) ||
                (icab_tx + 1 == TAM_COLA && icol_tx == 0)) {
            break;   The queue is full. The sending of the remaining
characters is aborted
        } else {
            cola_tx[icab_tx] = *ps;  Copy the character into the queue
            ps++;
            icab_tx++;
            if (icab_tx == TAM_COLA) {
                icab_tx = 0;
            }
        }
    }
    Transmitter interrupts are enabled to start sending
    IEC1bits.U1TXIE = 1;
}
Sends the movement or capture that has been made by UART
void UART_Mov(uint8_t comida, uint8_t columna_dest, uint8_t fila_dest,
uint8_t columna_orig, uint8_t fila_orig ,char nombre_PiezaMover, char
nombre_PiezaComida) {
    char respuesta[100];

    if (comida) {
```

```
 sprintf(response, "The %c part was eaten with the %c part in %c%c\n",
getName(nombre_PiezaMover),                 getName(nombre_PiezaComida),
getColumn(columna_dest), getRow(fila_dest));
    }
    else {
 sprintf(response, "The %c part has been moved from %c%c to %c%c\n",
getName(nombre_PiezaMover), getColumn(columna_orig), getRow(fila_orig),
getColumn(columna_dest), getRow(fila_dest));
    }
    putsUART(answer);
}

Informs the user you have selected
void UART_Info(uint8_t columna, uint8_t fila) {
    char respuesta[100];
 sprintf(response,   "The   %c%c\n   box   has   been   selected",
getColumn(column), getRow(row));
    putsUART(answer);
}

Instructs the user to select
void UART_Indica(uint8_t origen) {
    char orig_dest[100];
    if (origen) {
        sprintf(orig_dest, "Seleccione pieza origen \n");
    } else {
        sprintf(orig_dest, "Destino de la pieza \n");
    }
    sprintf(orig_dest, "Select \n coordinates");
    putsUART(orig_dest);
}
```

TFTDriver.c

```
/**
 * @file     TftDriver.c
 *
 * @author José Daniel Muñoz Frías
 *
 * @version 1.0.0. Initial version
 *
 * @date     16/11/2016
 *
 * @brief Module in charge of managing the TFT1.8SP shield of
 * Elecfreaks (ref. EF02005). The shield is based on a display with
 * The controller ST7735S.
 * This driver is based on the UTFT driver, although it has been ported
to C
 * and the support for the rest of the TFTs that included the
 *          driver original.
 */

#include <xc.h>
#include <stdint.h>
#include <stdbool.h>
#include <math.h>
#include <string.h>

#include "TftDriver.h"
#include "spi.h"

Internal documentation (Driver developers only)
/// @cond INTERNAL
Constant

Pins on the arduino connector to which the shield display connects
TFT1.8SP by Elecfreaks. For documentation only
//#define PIN_SCK 10
#define PIN_SDO 11
//#define PIN_SS  9
#define PIN_CMD_DATO 7 // 0 command, 1 data.
//#define PIN_RST 6 // Reset del display
Non-SPI TFT pins
#define PIN_CMD_DAT 6     Port C (RC6), which is connected to pin 4 of
the arduino
#define PIN_RST 5     Port C (RC5), which is connected to pin 3 of the
arduino

// Macros
#define swap(type, i, j) {type t = i; i = j; j = t;}

/// @cond INTERNAL
Private Roles
void Retardo(unsigned int ms);
void setXY(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2);
void clrXY(void);
void drawHLine(int x, int y, int l);
void drawVLine(int x, int y, int l);
void setPixel(uint16_t color);
void printChar(uint8_t c, int x, int y);
void rotateChar(uint8_t c, int x, int y, int pos, int deg);
void LCD_Write_COM(uint8_t cmd);
void LCD_Write_DATA(uint8_t data);

Global variables private to the module
static int _orientacion; // PORTRAIT o LANDSCAPE

static  uint8_t  fch,  fcl,  bch,  bcl;  //  Foreground  color  high/low,
background color high/low

struct _current_font{
        uint8_t *font;
        uint8_t x_size;
        uint8_t y_size;
        uint8_t offset;
        uint8_t numchars;
} cfont;

static bool    _transparent;
/// @endcond

/**
 * Initializes the TFT display. Select black for the background and
white for
 *to write.
 *
 * @param orientation Display orientation: LANDSCAPE or PORTRAIT
 */
void initializeTFT(int orientation)
{
        First, SPI2 is initialized, which is the one that is connected
to the
  display card. A frequency of 1 MHz is used.
        Initialize SPI2(1000000);

        We set pins cmd_dato and rst as outputs and as digital pins
        TRISC &= ~( (1<<PIN_CMD_DAT)|(1<<PIN_RST) );
        ANSELC &= ~( (1<<PIN_CMD_DAT)|(1<<PIN_RST) );

        I keep the orientation in a global variable, as it is used by
other variables.
        Module Functions
        _orientacion = orientacion;

        We reset the display
        PORTCSET = 1<<PIN_RST;
        Delay(5);
        PORTCCLR = 1<<PIN_RST;
        Delay(15);
        PORTCSET = 1<<PIN_RST;
        Delay(15);

        Now we initialize the display

        LCD_Write_COM(0x11);//Sleep exit
        Delay(12);
```

```
//ST7735R Frame Rate
LCD_Write_COM(0xB1);
LCD_Write_DATA(0x05);
LCD_Write_DATA(0x3C);
LCD_Write_DATA(0x3C);
LCD_Write_COM(0xB2);
LCD_Write_DATA(0x05);
LCD_Write_DATA(0x3C);
LCD_Write_DATA(0x3C);
LCD_Write_COM(0xB3);
LCD_Write_DATA(0x05);
LCD_Write_DATA(0x3C);
LCD_Write_DATA(0x3C);
LCD_Write_DATA(0x05);
LCD_Write_DATA(0x3C);
LCD_Write_DATA(0x3C);

LCD_Write_COM(0xB4); //Column inversion
LCD_Write_DATA(0x03);

//ST7735R Power Sequence
LCD_Write_COM(0xC0);
LCD_Write_DATA(0x28);
LCD_Write_DATA(0x08);
LCD_Write_DATA(0x04);
LCD_Write_COM(0xC1);
LCD_Write_DATA(0xC0);
LCD_Write_COM(0xC2);
LCD_Write_DATA(0x0D);
LCD_Write_DATA(0x00);
LCD_Write_COM(0xC3);
LCD_Write_DATA(0x8D);
LCD_Write_DATA(0x2A);
LCD_Write_COM(0xC4);
LCD_Write_DATA(0x8D);
LCD_Write_DATA(0xEE);

LCD_Write_COM(0xC5); VCOM
LCD_Write_DATA(0x1A);

LCD_Write_COM(0x36); //MX, MY, RGB mode
LCD_Write_DATA(0xC0);

//ST7735R Gamma Sequence
LCD_Write_COM(0xE0);
LCD_Write_DATA(0x03);
LCD_Write_DATA(0x22);
LCD_Write_DATA(0x07);
LCD_Write_DATA(0x0A);
LCD_Write_DATA(0x2E);
LCD_Write_DATA(0x30);
LCD_Write_DATA(0x25);
LCD_Write_DATA(0x2A);
LCD_Write_DATA(0x28);
LCD_Write_DATA(0x26);
LCD_Write_DATA(0x2E);
LCD_Write_DATA(0x3A);
```

```
        LCD_Write_DATA(0x00);
        LCD_Write_DATA(0x01);
        LCD_Write_DATA(0x03);
        LCD_Write_DATA(0x13);
        LCD_Write_COM(0xE1);
        LCD_Write_DATA(0x04);
        LCD_Write_DATA(0x16);
        LCD_Write_DATA(0x06);
        LCD_Write_DATA(0x0D);
        LCD_Write_DATA(0x2D);
        LCD_Write_DATA(0x26);
        LCD_Write_DATA(0x23);
        LCD_Write_DATA(0x27);
        LCD_Write_DATA(0x27);
        LCD_Write_DATA(0x25);
        LCD_Write_DATA(0x2D);
        LCD_Write_DATA(0x3B);
        LCD_Write_DATA(0x00);
        LCD_Write_DATA(0x01);
        LCD_Write_DATA(0x04);
        LCD_Write_DATA(0x13);

        //LCD_Write_COM(0x2A);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x7F);
        //LCD_Write_COM(0x2B);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x00);
        //LCD_Write_DATA(0x9F);

        LCD_Write_COM(0x3A); //65k mode
        LCD_Write_DATA(0x05);
        LCD_Write_COM(0x29);//Display on

        setColorRGB(255, 255, 255);
        setBackColorRGB(0, 0, 0);
        cfont.font=0; NOTE: This is a pointer. I guess it initializes it
to give some sort of error if you try to use it without initializing.
        _transparent = false;
}

/**
 * Erase the screen by turning it all black.
 */
void clrScr(void)
{
        long i;

        clrXY();
        for (i=0; i<((DISP_X_SIZE+1)*(DISP_Y_SIZE+1)); i++){
                LCD_Write_DATA(0);
                LCD_Write_DATA(0);
        }
}
```

```
/**
 * Select the color to write. All the functions that draw in the
 * display (drawXX, fillXX y print) usan este color.
 *
 * @param r Percentage of red color (0-255). The 5 most significant bits
are used.
 * @param g Percentage of green color (0-255). The most significant 6
bits are used.
 * @param b Percentage of blue color (0-255). The 5 most significant
bits are used.
 */
void setColorRGB(uint8_t r, uint8_t g, uint8_t b)
{
        fch=((r&248)|g>>5);
        fcl=((g&28)<<3|b>>3);
}

/**
 * Select the color to write. All the functions that draw in the
 * Display (draw, fill and print) use this color. Color is defined by a
 * uint16_t with RGB565 format. Predefined constants can be used with
 * the colors used by VGA cards: VGA_WHITE, VGA_RED, etc. (see
TftDriver.h).
 *
 * @param Color color in RGB565 format (5 bits red, 6 bits green, 5 bits
blue).
 */
void setColor(uint16_t color)
{
        fch = (uint8_t)(color>>8);
        fcl = (uint8_t)(color & 0xFF);
}

/**
 * Select the background color of the display.
 *
 * @param r Percentage of red color (0-255). The 5 most significant bits
are used.
 * @param g Percentage of green color (0-255). The most significant 6
bits are used.
 * @param b Percentage of blue color (0-255). The 5 most significant
bits are used.
 */
void setBackColorRGB(uint8_t r, uint8_t g, uint8_t b)
{
        bch=((r&248)|g>>5);
        bcl=((g&28)<<3|b>>3);
        _transparent=false;
}

/**
 * Select the background color used to print the characters. The plot
 * is the color in the RGB565 format. Predefined constants can be used
with
 * the colors used by VGA cards: VGA_WHITE, VGA_RED, etc. (see
TftDriver.h).
```

```
 * In addition, you can select the VGA_TRANSPARENT color so that the
background of the
 * Characters be transparent.
 *
 * @param Color color in RGB565 format (5 bits red, 6 bits green, 5 bits
blue).
 * Use 0xFFFFFFFF (VGA_TRANSPARENT) to make the background of the
 * Characters be transparent.
 */
void setBackColor(uint32_t color)
{
        if (color==VGA_TRANSPARENT)
                _transparent=true;
        else
        {
                bch = (uint8_t)(color>>8);
                bcl = (uint8_t)(color & 0xFF);
                _transparent=false;
        }
}
/**
 * Draw a rectangle defined by its two corners.
 *
 * @param x1 X Coordinate of the first corner.
 * @param y1 Y coordinate of the first corner.
 * @param x2 X Coordinate of the second corner.
 * @param y2 Y coordinate of the second corner.
 */
void drawRect(int x1, int y1, int x2, int y2)
{
        if (x1>x2)
        {
                swap(int, x1, x2);
        }
        if (y1>y2)
        {
                swap(int, y1, y2);
        }

        drawHLine(x1, y1, x2-x1);
        drawHLine(x1, y2, x2-x1);
        drawVLine(x1, y1, y2-y1);
        drawVLine(x2, y1, y2-y1);
}

/**
 * Draw a rectangle with rounded edges defined by its two corners.
 *
 * @param x1 X Coordinate of the first corner.
 * @param y1 Y coordinate of the first corner.
 * @param x2 X Coordinate of the second corner.
 * @param y2 Y coordinate of the second corner.
 */

void drawRoundRect(int x1, int y1, int x2, int y2)
{
        if (x1>x2)
        {
                swap(int, x1, x2);
        }
        if (y1>y2)
        {
                swap(int, y1, y2);
        }
        if ((x2-x1)>4 && (y2-y1)>4)
        {
                drawPixel(x1+1,y1+1);
                drawPixel(x2-1,y1+1);
                drawPixel(x1+1,y2-1);
                drawPixel(x2-1,y2-1);
                drawHLine(x1+2, y1, x2-x1-4);
                drawHLine(x1+2, y2, x2-x1-4);
                drawVLine(x1, y1+2, y2-y1-4);
                drawVLine(x2, y1+2, y2-y1-4);
        }
}

/**
 * Draw a filled rectangle defined by its two corners.
 *
 * @param x1 X Coordinate of the first corner.
 * @param y1 Y coordinate of the first corner.
 * @param x2 X Coordinate of the second corner.
 * @param y2 Y coordinate of the second corner.
 */
void fillRect(int x1, int y1, int x2, int y2)
{
        int i;

        if (x1>x2){
                swap(int, x1, x2);
        }
        if (y1>y2){
                swap(int, y1, y2);
        }

        if (_orientacion==PORTRAIT){
                for (i=0; i<((y2-y1)/2)+1; i++){
                        drawHLine(x1, y1+i, x2-x1);
                        drawHLine(x1, y2-i, x2-x1);
                }
        }else{
                for (i=0; i<((x2-x1)/2)+1; i++){
                                drawVLine(x1+i, y1, y2-y1);
                                drawVLine(x2-i, y1, y2-y1);
                }
        }
}

/**
 * Draw a filled rectangle with rounded edges defined by its two corners.
 *
 * @param x1 X Coordinate of the first corner.
 * @param y1 Y coordinate of the first corner.
```

```c
 * @param x2 X Coordinate of the second corner.
 * @param y2 Y coordinate of the second corner.
 */
void fillRoundRect(int x1, int y1, int x2, int y2)
{
        int i;

        if (x1>x2){
                swap(int, x1, x2);
        }
        if (y1>y2){
                swap(int, y1, y2);
        }

        if ((x2-x1)>4 && (y2-y1)>4){
                for (i=0; i<((y2-y1)/2)+1; i++){
                        switch(i){
                        case 0:
                                drawHLine(x1+2, y1+i, x2-x1-4);
                                drawHLine(x1+2, y2-i, x2-x1-4);
                                break;
                        case 1:
                                drawHLine(x1+1, y1+i, x2-x1-2);
                                drawHLine(x1+1, y2-i, x2-x1-2);
                                break;
                        default:
                                drawHLine(x1, y1+i, x2-x1);
                                drawHLine(x1, y2-i, x2-x1);
                        }
                }
        }
}

/**
 * Draw a circle defined by its center and radius.
 *
 * @param x X coordinate of the center.
 * @param and Y Coordinate inside.
 * @param radius Circle radius.
 */
void drawCircle(int x, int y, int radius)
{
        int f = 1 - radius;
        int ddF_x = 1;
        int ddF_y = -2 * radius;
        int x1 = 0;
        int y1 = radius;

        setXY(x, y + radius, x, y + radius);
        LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
        setXY(x, y - radius, x, y - radius);
        LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
        setXY(x + radius, y, x + radius, y);
        LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
        setXY(x - radius, y, x - radius, y);
        LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
        while(x1 < y1)
        {
                if(f >= 0)
                {
                        y1--;
                        ddF_y += 2;
                        f += ddF_y;
                }
                x1++;
                ddF_x += 2;
                f += ddF_x;
                setXY(x + x1, y + y1, x + x1, y + y1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x - x1, y + y1, x - x1, y + y1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x + x1, y - y1, x + x1, y - y1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x - x1, y - y1, x - x1, y - y1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x + y1, y + x1, x + y1, y + x1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x - y1, y + x1, x - y1, y + x1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x + y1, y - x1, x + y1, y - x1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
                setXY(x - y1, y - x1, x - y1, y - x1);
                LCD_Write_DATA(fch); LCD_Write_DATA(fcl);
        }
        clrXY();
}

/**
 * Draw a filled circle defined by its center and radius.
 *
 * @param x X coordinate of the center.
 * @param and Y Coordinate inside.
 * @param radius Circle radius.
 */
void fillCircle(int x, int y, int radius)
{
        int x1, y1;

        for(y1=-radius; y1<=0; y1++){
                for(x1=-radius; x1<=0; x1++){
                        if(x1*x1+y1*y1 <= radius*radius){
                                drawHLine(x+x1, y+y1, 2*(-x1));
                                drawHLine(x+x1, y-y1, 2*(-x1));
                                break;
                        }
                }
        }
}

/**
 * Fills the screen with a defined color using its RGB components.
 *
 * @param r Percentage of red color (0-255).
 * @param g Percentage of green color (0-255).
 * @param b Percentage of blue color (0-255).
 */
void fillScrRGB(uint8_t r, uint8_t g, uint8_t b)
{
        uint16_t color = ((r&248)<<8 | (g&252)<<3 | (b&248)>>3);
        fillScr(color);
}

/**
 * Fills the screen with a defined color using a 16-bit value with
 * RGB565 format
 *
 * @param Color color in RGB565 format (5 bits red, 6 bits green, 5 bits
blue).
 */
void fillScr(uint16_t color)
{
        long i;
        char ch, cl;

        ch=(uint8_t)(color>>8);
        cl=(uint8_t)(color & 0xFF);

        clrXY();
        for (i=0; i<((DISP_X_SIZE+1)*(DISP_Y_SIZE+1)); i++){
                LCD_Write_DATA(ch);
                LCD_Write_DATA(cl);
        }
}

/**
 * Draw a line between the dots (x1,y1) and (x2,y2)
 *
 * @param x1 X coordinate of point 1.
 * @param y1 Y coordinate of point 1.
 * @param x2 X coordinate of point 2.
 * @param y2 Y coordinate of point 2.
 */
void drawLine(int x1, int y1, int x2, int y2)
{
        if (y1==y2)
                drawHLine(x1, y1, x2-x1);
        else if (x1==x2)
                drawVLine(x1, y1, y2-y1);
        else
        {
                unsigned int    dx = (x2 > x1 ? x2 - x1 : x1 - x2);
                short                   xstep =  x2 > x1 ? 1 : -1;
                unsigned int    dy = (y2 > y1 ? y2 - y1 : y1 - y2);
                short                   ystep = y2 > y1 ? 1 : -1;
                int                                     col = x1, row = y1;

                if (dx < dy){
                        int t = - (dy >> 1);
                        while (true){
                                setXY (col, row, col, row);
```

```c
                                LCD_Write_DATA(fch);
                                LCD_Write_DATA(FCL);
                                if (row == y2)
                                        return;
                                row += ystep;
                                t += dx;
                                if (t >= 0){
                                        col += xstep;
                                        t -= two;
                                }
                        }
                }else{
                        int t = - (dx >> 1);
                        while (true){
                                setXY (col, row, col, row);
                                LCD_Write_DATA(fch);
                                LCD_Write_DATA(FCL);
                                if (col == x2)
                                        return;
                                col += xstep;
                                t += 2;
                                if (t >= 0){
                                        row += ystep;
                                        t   -= dx;
                                }
                        }
                }
        }
        clrXY();
}

/**
 * Draw a pixel at the coordinates (x,y).
 *
 * @param x X coordinate of the pixel.
 * @param and Y coordinate of the pixel.
 */
void drawPixel(int x, int y)
{
        setXY(x, y, x, y);
        setPixel((fch<<8)|fcl);
        clrXY();
}

/**
 * Select the font to use by the text functions. For example, to
 * select the SmallFont font defined in DefaultFonts.c just do:
 *
 * @code
 * extern uint8_t SmallFont[];
 * ...
 * SetFont(SmallFont[];
 * @endcode
 *
 * @param font The name of the vector that contains the font definition.
 *
 */
```

```
void setFont(uint8_t* font)
{
        cfont.font=font;
        cfont.x_size=font[0];
        cfont.y_size=font[1];
        cfont.offset=font[2];
        cfont.numchars=font[3];
}

/**
 * Gets the vector address of the font used.
 *
 * @return Address of the font used.
 */
uint8_t* getFont(void)
{
        return cfont.font;
}

/**
 * Returns the width (in pixels) of the font in use.
 *
 * @return Width in pixels of the font in use.
 */
uint8_t getFontXsize(void)
{
        return cfont.x_size;
}

/**
 * Returns the height (in pixels) of the font in use.
 *
 * @return High pixels of the font in use.
 */
uint8_t getFontYsize(void)
{
        return cfont.y_size;
}

/**
 * Prints a string of characters from the coordinates (x,y) with a
 * Set angle. The coordinates define the upper left corner of the
 * first character of the string.
 *
 * @param st String.
 * @param x X coordinate of the upper left squein of the first character.
 *  @param and Y coordinate of the upper left squein of the first
character.
 * @param deg Angle at which the string is printed (0 horizontal, 90
vertical, etc.)
 */
void print(char *st, int x, int y, int deg)
{
        int stl, i;

        stl = strlen(st);
        if (_orientacion==PORTRAIT)
        {
        if (x==RIGHT)
                x=(DISP_X_SIZE+1)-(stl*cfont.x_size);
        if (x==CENTER)
                x=((DISP_X_SIZE+1)-(stl*cfont.x_size))/2;
        }
        else
        {
        if (x==RIGHT)
                x=(DISP_Y_SIZE+1)-(stl*cfont.x_size);
        if (x==CENTER)
                x=((DISP_Y_SIZE+1)-(stl*cfont.x_size))/2;
        }

        for (i=0; i<stl; i++)
                if (deg==0)
                        printChar(*st++, x + (i*(cfont.x_size)), y);
                else
                        rotateChar(*st++, x, y, i, deg);
}

/**
 * Draws an image (bitmap) on the screen. The bitmap must be defined
 * as a vector with the colors of all pixels in RGB565 format.
 * The easiest way to generate it is by using an available web application
 * on the original driver's author's page:
 *   http://www.rinkydinkelectronics.com/t_imageconverter565.php
 *
 * @param x X Coordinate in the upper right corner of the bitmap.
 * @param and Y-Coordinate in the upper right corner of the bitmap.
 * @param sx Horizontal bitmap size in pixels.
 * @param s and Vertical bitmap size in pixels.
 * @param data Address of the vector containing the bitmap.
 * @param scale Scale factor to draw the bitmap.
 */
void drawBitmap(int x, int y, int sx, int sy, uint16_t data[], int scale)
{
        unsigned int cabbage;
        int tx, ty, tc, tsx, tsy;

        if (scale==1)
        {
                if (_orientacion==PORTRAIT){
                        setXY(x, y, x+sx-1, y+sy-1);
                        for (tc=0; tc<(sx*sy); tc++){
                                col = data[tc];
                                LCD_Write_DATA(col>>8);    LCD_Write_DATA
(cervix & 0xff);
                        }
                }else{
                        for (ty=0; ty<sy; ty++){
                                setXY(x, y+ty, x+sx-1, y+ty);
                                for (tx=sx-1; tx>=0; tx--){
                                        col=data[(ty*sx)+tx];
                                        LCD_Write_DATA(col>>8);
LCD_Write_DATA (cervix & 0xff);
```

```
                                }
                        }
                }
        }else{
                if (_orientacion==PORTRAIT){
                        for (ty=0; ty<sy; ty++){
                                setXY(x, y+(ty*scale), x+((sx*scale)-1),
y+(ty*scale)+scale);
                                for (tsy=0; tsy<scale; tsy++)
                                        for (tx=0; tx<sx; tx++)
                                        {
                                                col=data[(ty*sx)+tx];
                                                for   (tsx=0;   tsx<scale;
tsx++)

        LCD_Write_DATA(col>>8); LCD_Write_DATA (cervix & 0xff);
                                        }
                        }

                }
                else
                {

                        for (ty=0; ty<sy; ty++)
                        {
                                for (tsy=0; tsy<scale; tsy++)
                                {
                                        setXY(x,        y+(ty*scale)+tsy,
x+((sx*scale)-1), y+(ty*scale)+tsy);
                                        for (tx=sx-1; tx>=0; tx--)
                                        {
                                                col=data[(ty*sx)+tx];
                                                for   (tsx=0;   tsx<scale;
tsx++)

        LCD_Write_DATA(col>>8); LCD_Write_DATA (cervix & 0xff);
                                        }
                                }
                        }

                }
        }
        clrXY();
}

/*********************************************************************
******/
/**************************             Private             Functions
*****************************/
/*********************************************************************
******/
/// @cond INTERNAL
/**
 * Generates an n millisecond delay using timer 1 in polling mode. The
 * Function saves the state of Timer 1 for as little disturbance as
possible.
 *
 * @param n_ms Number of milliseconds to wait
 *
 */

void Retardo(unsigned int n_ms)
{
        int pr1_bak, t1con_bak;

        I save the timer state 1.
        pr1_bak = PR1;
        t1con_bak = T1CON;

        I initialize the timer
  TMR1 = 0;
  PR1 = 5000;  Timer to 1 millisecond. Since PBCLK is 5 MHz, you have
to
              5000 count
  IFS0bits.T1IF = 0; // Borra el flag
  T1CON = 0x8000; // Timer on, Prescaler = 0 while(IFS0bits.T1IF == 0)
  while(n_ms != 0){
    while(IFS0bits.T1IF == 0)
      ;  Wait for the end of the timer
    IFS0bits.T1IF = 0; // Borra el flag
    n_ms--;
  }
        We left timer 1 as it was
        T1CON = t1con_bak;
        PR1 = pr1_bak;
}

/**
 * Select the frame of the display's graphics memory in which they are
going to
 * Write data. To do this, it sends the command 0x2A (Column Address
Set) that
 * Select which two columns to type between. This command receives
 * as argumantos the x coordinates of the frame. The following is sent
the
 * 0x2B command (Row Address Set) to select which two rows to go between
 *to write. The command receives the two coordinates and the frame as
arguments.
 * Finally, the 0x2C (Memory Write) command is sent to send to memory
 * the data to be written to the frame. This data will be sent by another
function.
 *
 * @param x1 x-coordinate of the first point of the frame.
 * @param y1 Coordinate and of the first point of the frame.
 * @param x2 x-coordinate of the second point of the frame.
 * @param y2 Coordinate y of the second point of the frame.
 */
void setXY(uint16_t x1, uint16_t y1, uint16_t x2, uint16_t y2)
{
        if (_orientacion==LANDSCAPE)
        {
                swap(uint16_t, x1, y1);
                swap(uint16_t, x2, y2)
                y1=DISP_Y_SIZE-y1;
                y2=DISP_Y_SIZE-y2;
                swap(uint16_t, y1, y2)
        }
        LCD_Write_COM(0x2a);
        LCD_Write_DATA(x1>>8);
        LCD_Write_DATA(x1);
        LCD_Write_DATA(x2>>8);
        LCD_Write_DATA(x2);
        LCD_Write_COM(0x2b);
        LCD_Write_DATA(y1>>8);
        LCD_Write_DATA(y1);
        LCD_Write_DATA(y2>>8);
        LCD_Write_DATA(y2);
        LCD_Write_COM(0x2c);
}

/**
 * Select the entire screen as the frame to write to the graphics memory.
 */
void clrXY(void)
{
        if (_orientacion==PORTRAIT)
                setXY(0,0,DISP_X_SIZE,DISP_Y_SIZE);
        else
                setXY(0,0,DISP_Y_SIZE,DISP_X_SIZE);
}

/**
 * Draw a horizontal line from the point (x,y) with the length l.
 *
 * @param x X Coordinate of the line's origin.
 * @param and Y Coordinate of the line's origin.
 * @param l Line length.
 */
void drawHLine(int x, int y, int l)
{
        int i;

        if (l<0){
                l = -l;
                x -= l;
        }

        setXY(x, y, x+l, y);
        for (i=0; i<l+1; i++){
                LCD_Write_DATA(fch);
                LCD_Write_DATA(FCL);
        }
        clrXY();
}

/**
 * Draw a vertical line from the point (x,y) with the length l.
 *
 * @param x X Coordinate of the line's origin.
 * @param and Y Coordinate of the line's origin.
 * @param l Line length.
 */

void drawVLine(int x, int y, int l)
{
        int i;

        if (l<0){
                l = -l;
                y -= l;
        }

        setXY(x, y, x, y+l);
        for (i=0; i<l+1; i++){
                LCD_Write_DATA(fch);
                LCD_Write_DATA(FCL);
        }
        clrXY();
}

/**
 * Writes the color of a pixel to the graphics memory. The address must
have
 * has been pre-set by a call to setXY().
 *
 * @param color Pixel color.
 */
void setPixel(uint16_t color)
{
        color viene en el formato rrrrrggggggbbbbb
        LCD_Write_DATA(color>>8);      The most significant 8 bits are
sent first
        LCD_Write_DATA(color&0xFF);    And  then  the  less  significant
ones
}

/**
 * Prints a character at the coordinates (x,y). The coordinates define
the
 * upper left corner of the character to be printed.
 *
 * @param c Character to print
 * @param x Coordenada X
 * @param and Y Coordinate
 */

void printChar(uint8_t c, int x, int y)
{
        uint8_t i,ch;
        uint16_t j;
        uint16_t temp;
        int zz;

        if (!_transparent){
                if (_orientacion==PORTRAIT){
                        setXY(x,y,x+cfont.x_size-1,and+cfont.y_size-1);
                        temp=((c-
cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
                        for(j=0;      j<((cfont.x_size/8)*cfont.y_size);
j++){
                                ch=cfont.font[temp];
                                for(i=0; i<8; i++){
                                        if((ch&(1<<(7)-i)))!=0){
                                                setPixel((fch<<8)|fcl);
                                        }else{
                                                setPixel((bch<<8)|bcl);
                                        }
                                }
                                temp++;
                        }
                }else{
                        temp=((c-
cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;

                        for(j=0;      j<((cfont.x_size/8)*cfont.y_size);
j+=(cfont.x_size/8)){

        setXY(x,y+(j/(cfont.x_size/8)),x+cfont.x_size-
1,y+(j/(cfont.x_size/8)));
                                for  (zz=(cfont.x_size/8)-1;  zz>=0;  zz--
){
                                        ch=cfont.font[temp+zz];
                                        for(i=0; i<8; i++){
                                                if((ch&(1<<i))!=0){

        setPixel((fch<<8)|fcl);
                                                }else{

        setPixel((bch<<8)|bcl);
                                                }
                                        }
                                }
                                temp+=(cfont.x_size/8);
                        }
                }
        }else{
                temp=((c-
cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
                for(j=0; j<cfont.y_size; j++){
                        for (zz=0; zz<(cfont.x_size/8); zz++){
                                ch=cfont.font[temp+zz];
                                for(i=0; i<8; i++){
                                        if((ch&(1<<(7)-i)))!=0){

        setXY(x+i+(zz*8),y+j,x+i+(zz*8)+1,y+j+1);
                                                setPixel((fch<<8)|fcl);
                                        }
                                }
                        }
                        temp+=(cfont.x_size/8);
                }
        }
        clrXY();
}

/**
 * Draw a character with the rotation given by deg.
 *
 * @param c
 * @param x
 * @param and
 * @param pos
 * @param you
 */
void rotateChar(uint8_t c, int x, int y, int pos, int deg)
{
        uint8_t i,j,ch;
        uint16_t temp;
        int newx,newy;
        double radian;
        int zz;

        radian=deg*0.0175;

        temp=((c-cfont.offset)*((cfont.x_size/8)*cfont.y_size))+4;
        for(j=0; j<cfont.y_size; j++){
                for(zz=0; zz<(cfont.x_size/8); zz++){
                        ch=cfont.font[temp+zz];
                        for(i=0; i<8; i++){

        newx=x+(((i+(zz*8)+(pos*cfont.x_size))*cos(radian))-
((j)*sin(radian)));

        newy=y+(((j)*cos(radian))+((i+(zz*8)+(pos*cfont.x_size))*sin(ra
dian)));

                                setXY(newx,newy,newx+1,newy+1);

                                if((ch&(1<<(7)-i)))!=0){
                                        setPixel((fch<<8)|fcl);
                                }else{
                                        if (!_transparent)
                                                setPixel((bch<<8)|bcl);
                                }
                        }
                }
                temp+=(cfont.x_size/8);
        }
        clrXY();
}
/**
 * Send a command to the display
 *
 * @param cmd Command to send to display
 */

void LCD_Write_COM(uint8_t cmd)
{
        LATCCLR = 1<<PIN_CMD_DAT; I'm going to send a command
        SPI_SendFrame(cmd); I send it
}

/**
 * Send a piece of information to the display
 *
 * @param cmd Command to send to display
 */

void LCD_Write_DATA(uint8_t data)
{
        LATCSET  =  (1<<PIN_CMD_DAT);  I'm  going  to  send  a  piece  of
information
        SPI_SendFrame(data); I send it
}
/// @endcond
```