```
/*
 * File:       TFT_test_BRL4.c
 * Author:     Bruce Land
 * Adapted from:
 *             main.c by
 * Author:     Syed Tahmid Mahbub
 * Target PIC:  PIC32MX250F128B
 */

// graphics libraries
#define _SUPPRESS_PLIB_WARNING 1
#include "config.h"
#include "plib.h"
#include "tft_master.h"
#include "tft_gfx.h"
#include <string.h>
#include "LUFA.h"

// need for rand function
#include <stdlib.h>

// threading library
// config.h sets 40 MHz
#define           SYS_FREQ 40000000
#define DESIRED_BAUDRATE         (9600)      // The desired BaudRate
#define PB_FREQ SYS_FREQ
#define UNITS_IN 0
#define UNITS_MM 1
#include "pt_cornell_TFT.h"

/* Demo code for interfacing TFT (ILI9340 controller) to PIC32
 * The library has been modified from a similar Adafruit library
 */
// Adafruit data:
/***************************************************
  This is an example sketch for the Adafruit 2.2" SPI display.
  This library works with the Adafruit 2.2" TFT Breakout w/SD card
  ----> http://www.adafruit.com/products/1480

  Check out the links above for our tutorials and wiring diagrams
  These displays use SPI to communicate, 4 or 5 pins are required to
  interface (RST is optional)
  Adafruit invests time and resources providing this open source code,
  please support Adafruit and open-source hardware by purchasing
  products from Adafruit!
```

```
 Written by Limor Fried/Ladyada for Adafruit Industries.
 MIT license, all text above must be included in any redistribution
 ****************************************************/



// string buffer
char buffer[60];

// === thread structures ============================================
// thread control structs
// note that UART input and output are threads
static struct pt pt_cap_n_screen, pt_blink, pt_pwm,pt_read,pt_ring_in,pt_movey;

// system 1 second interval tick
int sys_time_seconds, sys_time_micro;

int alternator_x,alternator_y,motor_start_x,motor_start_y, accum;
int motor_step_x = 4064;
int motor_step_y = 4964;

unsigned int C1OUT = 0b1;
unsigned short pwm_speed = 0;
int capture1;
float fan_rpm;
int pwm_step;

//PID Values
float speed_target = 1000;
float pid_response;
float current_error,previous_error,difference_error,integral_error=0;
float kp = 0.01;
float ki = 0.0002;
float kd = 0.01;
float arg1f;
float arg2f;
char   units;
float step_set_x;
float step_set_y;
int dur_x=10000;
int dur_y=10000;
int motor_speed_x=10000;
int motor_speed_y=10000;
char done_x,done_y;
int speed_set_x = 1;
int speed_set_y=1;
```

```
float x_abs=0;
float y_abs=0;



static char read_buf[32];
static int char_num = 0;
static int read_complete =0;
static char RX_buf;
Receive_RingBuff_t rx_queue;

void move_x(){
   if(step_set_x < 0){
      step_set_x*=-1;
      mPORTBClearBits(BIT_8);
   }else{
      mPORTBSetBits(BIT_8);
   }
   motor_step_x = step_set_x * 4064;
   motor_speed_x = speed_set_x;
}

void move_y(){
   if(step_set_y < 0){
      step_set_y*=-1;
      mPORTBClearBits(BIT_7);
   }else{
      mPORTBSetBits(BIT_7);
   }
   motor_step_y = step_set_y * 27;
   motor_speed_y = speed_set_y;
}


static PT_THREAD (protothread_read(struct pt *pt))
{
   static int i;
   PT_BEGIN(pt);
   while(1)
   {

      while(DataRdyUART2() && !read_complete)
      {
         RX_buf = (char) ReadUART2();
         if(char_num==32 || RX_buf=='\r')
         {
```

```
            read_buf[char_num] = RX_buf;
            char_num++;
            for(i=char_num;i<32;i++)
            {
               read_buf[i]=0;
            }
            read_complete= 1;
         }
         else if (RX_buf==backspace)
         {

            putcUART2(backspace);
            putcUART2(' ');
            putcUART2(backspace);
            read_buf[char_num] = 0;
            if(char_num)
            {
               char_num--;
            }
         }
         else
         {
            read_buf[char_num] = RX_buf;
            putcUART2(read_buf[char_num]);
            char_num++;
         }
      }
      PT_YIELD_TIME_msec(100);
   }
   PT_END(pt);
}

static PT_THREAD (protothread_movey(struct pt *pt)){
   PT_BEGIN(pt);
   while(1){
      if(motor_step_y)
      {
         mPORTBSetBits(BIT_0);
         mPORTBSetBits(BIT_5);
         motor_step_y--;
         if(motor_step_y<=0){
            putsUART2("JOPLIN ");
            done_y=1;
            if(motor_step_x<=0){
               done_x=1;
```

```
            }
          }
        }

        PT_YIELD_TIME_msec(motor_speed_y);
        mPORTBClearBits(BIT_5);
        PT_YIELD_TIME_msec(motor_speed_y);
    }
    PT_END(pt);
}

// LED BLINK THREAD
static PT_THREAD (protothread_blink(struct pt *pt))
{
    static int diskstate = 0;
    PT_BEGIN(pt);
    while(1)
    {
        if(motor_step_x)
        {
            mPORTASetBits(BIT_0);
            mPORTBSetBits(BIT_4);
            motor_step_x--;
            if(motor_step_x<=0){
                putsUART2("SCOTT ");
                done_x=1;
                if(motor_step_y<=0){
                    done_y = 1;
                }

            }

        }


        if(done_x && done_y){
            mPORTAClearBits(BIT_0);
            mPORTBClearBits(BIT_0);
            done_x = 0;
            done_y = 0;
            putsUART2("ACK");
        }
        PT_YIELD_TIME_msec(motor_speed_x);
        mPORTBClearBits(BIT_4);
        PT_YIELD_TIME_msec(motor_speed_x);
```

```
   }
   PT_END(pt);
}

static PT_THREAD (protothread_ring_in(struct pt *pt)){
   PT_BEGIN(pt);

   if(pwm_step<=1){
      pwm_step = 1;
   }else if(pwm_step>=2){
      pwm_step = 2;
   }
   mPORTBSetBits(BIT_9);
   PT_YIELD_TIME_msec(pwm_step);
   mPORTBClearBits(BIT_9);
   PT_YIELD_TIME_msec(20);

   PT_END(pt);
}

// CHECK RPM AND UPDATE SCREEN
static PT_THREAD (protothread_cap_n_screen(struct pt *pt))
{
   PT_BEGIN(pt);
    tft_setCursor(0, 30);
    tft_setTextColor(ILI9340_WHITE);  tft_setTextSize(2);
    tft_writeString("The fan speed is: \n");
      while(1) {

         if(read_complete)
         {
            int read_done = 0;
            int i = 0;
            int state=0; //0 is command; 1 is arg1; 2 is arg2; 3 is end seq
            char command[8]="       ";
            int commctr=0;
            int arg1ctr=0;
            int arg2ctr=0;
            int rec_succ=0;
            char arg1[8]="\0      ";
            char arg2[8]="\0      ";
            float value;

            while(!read_done){
               switch(state){
```

```
      case 0:
        putsUART2(" IN0 ");
        if(read_buf[i] == ' ' || (commctr>=7)){
          state = 1;
          command[commctr]='\0';
        }else if( (read_buf[i] == '\r') || (read_buf[i] == '\n')){
          state = 3;
          command[commctr]='\0';
        }else{
          command[commctr] = read_buf[i];
          commctr++;

        }
        break;
      case 1:
        putsUART2(" IN1 ");
        if(read_buf[i] == ' ' || (arg1ctr>=7)){
          state = 2;
          arg1[arg1ctr] = '\0';
        }else if( (read_buf[i] == '\r') || (read_buf[i] == '\n')){
          state = 3;
          arg1[arg1ctr] = '\0';
        }else{
          arg1[arg1ctr] = read_buf[i];
          arg1ctr++;
        }
        break;
      case 2:
        putsUART2(" IN2 ");
        if( (read_buf[i] == '\r') || (read_buf[i] == '\n') || (arg2ctr>=7)){
          state = 3;
          arg2[arg2ctr] = '\0';
        }else{
          arg2[arg2ctr] = read_buf[i];
          arg2ctr++;
        }
        break;
      case 3:
        putsUART2(" IN3 ");
        read_done = 1;
        rec_succ = 1;
        break;
    }
    i++;
  }
```

```
if(!rec_succ){
   putsUART2("NACK");
}else{
   putsUART2(" PARSE GOOD ");
}

if(!strcmp("M",command)){
   putsUART2("EFFECSTING CHANGE ");
   step_set_x = atof(arg1);
   step_set_y = atof(arg2);
   x_abs += step_set_x;
   y_abs += step_set_y;
   if(!step_set_x && !step_set_y){
      putsUART2("ACK");
   }
   move_x();
   move_y();
}else if(!strcmp("S",command)){
   putsUART2("SPEED CHANGE");
   speed_set_x = atoi(arg1);
   speed_set_y = atoi(arg2);
   putsUART2("ACK");
}else if(!strcmp("PING",command)){
   putsUART2("PONG");
}else if(!strcmp("G",command)){
   putsUART2("G Comm");
   step_set_x = atof(arg1) - x_abs;
   step_set_y = atof(arg2) - y_abs;
   x_abs += step_set_x;
   y_abs += step_set_y;
   if(!((int)step_set_x*4064) && !((int)step_set_y*27)){
      putsUART2("ACK");
   }
   move_x();
   move_y();
}else if(!strcmp("A",command)){
   putsUART2("ANGLES");
   pwm_step = atoi(arg1);
   PT_YIELD_TIME_msec(100);
   putsUART2("ACK");
}else{
   putsUART2("ACK");
}

char_num=0;
read_complete = 0;
```

```
        }
        PT_YIELD_TIME_msec(200) ;
      }
  PT_END(pt);
}
// === Main  ======================================================
void main(void) {

    // === config threads ==========
  // turns OFF UART support and debugger pin
  PT_setup();

  mPORTBSetPinsDigitalOut(BIT_4);
  mPORTBSetBits(BIT_4);
  mPORTBSetPinsDigitalOut(BIT_5);
  mPORTBSetBits(BIT_5);
  mPORTBSetPinsDigitalOut(BIT_8);
  mPORTBSetBits(BIT_8);
  mPORTBSetPinsDigitalOut(BIT_7);
  mPORTBSetBits(BIT_7);
  mPORTBSetPinsDigitalOut(BIT_9);
  mPORTBSetBits(BIT_9);
  mPORTASetPinsDigitalOut(BIT_0);
  mPORTAClearBits(BIT_0);
  mPORTBSetPinsDigitalOut(BIT_0);
  mPORTBClearBits(BIT_0);



 SYSTEMConfigPerformance(PBCLK);

  ANSELA = 0; ANSELB = 0; CM1CON = 0; CM2CON = 0;

  // === setup system wide interrupts  ========
  INTEnableSystemMultiVectoredInt();

  // init the threads
  PT_INIT(&pt_cap_n_screen);
  PT_INIT(&pt_blink);
  PT_INIT(&pt_read);
  PT_INIT(&pt_ring_in);
  PT_INIT(&pt_movey);


  // init the display
  tft_init_hw();
```

```
 tft_begin();
 tft_fillScreen(ILI9340_BLACK);
 //240x320 vertical display
 tft_setRotation(0); // Use tft_setRotation(1) for 320x240

 while (1){
    PT_SCHEDULE(protothread_movey(&pt_movey));
    PT_SCHEDULE(protothread_ring_in(&pt_ring_in));
    PT_SCHEDULE(protothread_cap_n_screen(&pt_cap_n_screen));
    PT_SCHEDULE(protothread_blink(&pt_blink));
    PT_SCHEDULE(protothread_read(&pt_read));
    }
 } // main

// === end  ======================================================
```