UNIVERSITY OF LEEDS

ELEC5620M
Embedded Microprocessor System Design
Assignment 1

Alexander Bolton
200938078

March 2019

1 Abstract

In this assignment the task was to create a graphics library for the DE1-SoC using an LT24 LCD screen.

Contents

2 Introduction

3 Code Design and checking

4 Testing

5 Conclusion

6 Appendix

6.1 Graphics.c/.h

6.1.1 Graphics Header

```c
#ifndef GRAPHICS_H
#define GRAPHICS_H

#include "../DE1SoC_LT24/DE1SoC_LT24.h"
#include "../HPS_Watchdog/HPS_Watchdog.h"
#include "../sevenSeg/sevenSeg.h"

 void Graphics_initialise(unsigned volatile int lcd_pio_base,unsigned volatile int lcd_hw_base);

 void Graphics_drawBox(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned short colour,bool noFill,unsigned short fillColour);

 void Graphics_drawCircle(unsigned int x,unsigned int y,unsigned int r,unsigned short colour,bool noFill,unsigned short fillColour);

 void Graphics_drawLine(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned short colour);

 void Graphics_drawTriangle(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned int x3,unsigned int y3,unsigned short colour,bool noFill,unsigned short fillColour);

 void Graphics_fillTriangle(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned int x3,unsigned int y3,unsigned short fillColour);

 void Graphics_drawPixel(unsigned short Colour, unsigned int x, unsigned int y);

#endif
```

6.1.2 Graphics initialise

```c
void Graphics_initialise(unsigned volatile int lcd_pio_base,unsigned volatile int lcd_hw_base){
  LT24_initialise(lcd_pio_base,lcd_hw_base);
}
```

6.1.3 Graphics drawBox

```
void Graphics_drawBox(unsigned int x1,unsigned int y1,unsigned int x2,unsigned
    int y2,unsigned short colour,bool noFill,unsigned short fillColour){
  //Signed Values declares
  int sx1 = (int) x1;
  int sx2 = (int) x2;
  int sy1 = (int) y1;
  int sy2 = (int) y2;
  //calculate height and width
  int height = abs(sy2-sy1);
  int width = abs(sx1-sx2);
  //set values for forloops
  int y=0;
  int x=0;
  int oy=0;
  int ox=0;

  //find bottom left value
  int llx = 0;
  int lly = 0;
  if(sx1<sx2){
    llx = sx1;
  }
  else{
    llx = sx2;
  }

  if(sy1<sy2){
    lly = sy1;
  }
  else{
    lly = sy2;
  }

  //cube fill (draws first so it can be overdrawn with outline)
  if(!noFill){
    for(y=0; y <= height; y++){
      for(x=0; x<=width; x++){
        Graphics_drawPixel(fillColour,x+llx,y+lly);
      }
    }
  }

  //cube outline
  //verticle outline
  for(oy = 0; oy <=height; oy++){
    Graphics_drawPixel(colour,sx1,lly+oy);
  }
  for(oy = 0; oy <=height; oy++){
    Graphics_drawPixel(colour,sx2,lly+oy);
  }
  //horizontal outline
  for(ox = 0; ox <=width; ox++){
    Graphics_drawPixel(colour,llx+ox,sy1);
  }
  for(ox = 0; ox <=width; ox++){
        Graphics_drawPixel(colour,llx+ox,sy2);
      }


    }
```

6.1.4 Graphics drawCircle

```
void Graphics_drawCircle(unsigned int x,unsigned int y,unsigned int r,unsigned
    short colour,bool noFill,unsigned short fillColour){
  //Radius as signed int
  int signedr = (int) r;
  //Radius squared
  int rad2 = signedr * signedr;
  //Outline threshold
  int outThres = 230;
  //Go through x's
  int xc = 0;
  int yc = 0;
  //Loop through all X and Y of square the size of radius squared
  for (xc = -signedr-3; xc <= signedr + 3; xc++) {
    for (yc = -signedr-3; yc <= signedr + 3; yc++) {
      //radius squared = yc^2 + xc^2
      int pyr = (yc*yc) + (xc*xc);
      //If no fill then draw outline
      if(noFill && (pyr > rad2-outThres) && (pyr <= rad2)){
        Graphics_drawPixel(colour,xc+x,yc+y);
      }
      //If fill draw fill
      else if(!noFill && pyr <= rad2){
        Graphics_drawPixel(fillColour,xc+x,yc+y);
      }
    }
  }
  //if fill draw outline last over fill
  if(~noFill){
    xc = 0;
    yc = 0;
    for (xc = -signedr-3; xc <= signedr+3; xc++) {
      for (yc = -signedr-3; yc <= signedr+3; yc++) {
        //get r and check if it the same as radius
        int pyr = (yc*yc) + (xc*xc);
        if((pyr > rad2-outThres) && (pyr <= rad2)){
          LT24_drawPixel(colour,xc+x,yc+y);
        }
      }
    }
  }

}
```

6.1.5 Graphics drawLine

```c
void Graphics_drawLine(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned short colour){
  //REFERENCE: drawLine using Bresenhams algorithm. https://rosettacode.org/wiki/Bitmap/Bresenham%27s_line_algorithm

  //calculate deltas
  int dx =  abs (x2 - x1);
  int dy = -abs (y2 - y1);
  //calculate error
  int error = dx + dy;
  int error2;
  //Find quadrant
  int sy;
  int sx;
  if(x1<x2){
    sx = 1;
  }
  else{
    sx = -1;
  }

  if(y1<y2){
    sy = 1;
  }
  else{
    sy = -1;
  }
  //Loop though and calculate line pixels
  while(1){
    Graphics_drawPixel(colour,x1,y1);
    if (x1 == x2 && y1 == y2){ break;}
    error2 = 2 * error;
    //if error2 is larger than delta y then add 1 to x
    if (error2 >= dy) {
      error += dy;
      x1 += sx;
    }
    //if error2 is smaller than delta x then add 1 to y
    if (error2 <= dx) {
      error += dx;
      y1 += sy;
    }
  }
}
```

6.1.6 Graphics drawTriangle

```c
void Graphics_drawTriangle(unsigned int x1,unsigned int y1,unsigned int x2,unsigned int y2,unsigned int x3,unsigned int y3,unsigned short colour,bool noFill,unsigned short fillColour){

  //If fill
  if(!noFill){
    //Run fill traingle on 3 occassions to ensure on small triangles that no pixel is missed. Reset WD.
     Graphics_fillTriangle(x1,y1,x2,y2,x3,y3,fillColour);ResetWDT();
     Graphics_fillTriangle(x3,y3,x1,y1,x2,y2,fillColour);ResetWDT();
     Graphics_fillTriangle(x2,y2,x3,y3,x1,y1,fillColour);ResetWDT();
  }
  //Draw Outline
  Graphics_drawLine(x1,y1,x2,y2,colour);
  Graphics_drawLine(x2,y2,x3,y3,colour);
  Graphics_drawLine(x3,y3,x1,y1,colour);
}
```

6.1.7 Graphics fillTriangle

```c
void Graphics_fillTriangle(unsigned int x1,unsigned int y1,unsigned int x2,
    unsigned int y2,unsigned int x3,unsigned int y3,unsigned short fillColour){
  //A rewrite of the straight line function to allow it to drawlines to fill the
     trangle.
  //calculate deltas
    int dx =  abs (x2 - x1);
    int dy = -abs (y2 - y1);
    //calculate error
    int error = dx + dy;
    int error2;
    int sy;
    int sx;
    if(x1<x2){
      sx = 1;
    }
    else{
      sx = -1;
    }

    if(y1<y2){
      sy = 1;
    }
    else{
      sy = -1;
    }
    while(1){
      Graphics_drawLine(x3,y3,x1,y1,fillColour); //drawLine
        if (x1 == x2 && y1 == y2){ break;}
        error2 = 2 * error;
        if (error2 >= dy) {
          error += dy;
          x1 += sx;
        }
        if (error2 <= dx) {
          error += dx;
          y1 += sy;
        }
    }
}
```

6.1.8 Graphics drawPixel

```c
void Graphics_drawPixel(unsigned short Colour, unsigned int x, unsigned int y){
  int status = LT24_drawPixel(Colour,x,y);
  ResetWDT();
  if(status != 0){
    SDisplay_clearAll();
    SDisplay_set(0, 0x1);
    SDisplay_set(1, 0xE);
  }
}
```

6.2 Timer.c/.h

6.2.1 Timer Header

```c
#ifndef TIMER_H
#define TIMER_H

void timer_Start();

int timer_Stop();


#endif
```

6.2.2 Timer start

```c
void timer_Start(){
//initialise
//start timer
  int timerS = 0;
  *private_timer_load = 100000000;
      // Set the "Prescaler" value to 0, Enable the timer (E = 1), Set Automatic
    reload
      // on overflow (A = 1), and disable ISR (I = 0)
  *private_timer_control = (0 << 8) | (0 << 2) | (1 << 1) | (1 << 0);

  timerS = *private_timer_value;

  timerStartValue = timerS;
}
```

6.2.3 Timer stop

```c
int timer_Stop(){
//stop timer
//print timer end value
//print difference
  int timerEndValue = *private_timer_value;
  int timerDuration = timerStartValue - timerEndValue;
  int freqTimer =  1/225000000 * timerDuration;
  float FPS = 1/(4.44e-9 * timerDuration);
  int FPSint = FPS;
  int freq = freqTimer;

  *private_timer_control   = 0;

  return FPSint;
```

6.3 sevenSeg.c/.h

6.3.1 SDisplay Header

```c
#ifndef SEVENSEG_H
#define SEVENSEG_H

void SDisplay_PNum(int number, int pair);
void SDisplay_clearAll();
void SDisplay_set(int Display, int HexValue);

#endif
```

6.3.2 SDisplay clearAll

```c
void SDisplay_clearAll(){
  //Hex memory base
  volatile int *HEX0 = (int* ) 0XFF200020;
  volatile int *HEX1 = (int* ) 0XFF200030;
  int zero = 0x00;

  *HEX0 &= zero; //clear bits
  *HEX1 &= zero; //clear bits
}
```

6.3.3 SDisplay set

```c
void SDisplay_set(int Display, int HexValue){
  //Hex memory base
  volatile int *HEX0 = (int* ) 0XFF200020;
  volatile int *HEX1 = (int* ) 0XFF200030;

  int invClearBits = 0x7F; //inverted bits to put through and bitwise
  int shiftAmount = 8; //shift multiple amount
  int hex1Adjust = 4; //adjust amount for second memory address

  if(Display < 4){
    *HEX0 &= ~(invClearBits << (Display * shiftAmount)); //clear bits
    *HEX0 |= (HexSDisplay[HexValue] << (Display * shiftAmount)); //set bits
  }
  else{
    *HEX1 &= ~(invClearBits << ((Display - hex1Adjust) * shiftAmount)); //clear bits
    *HEX1 |= (HexSDisplay[HexValue] << ((Display - hex1Adjust) * shiftAmount)); //set bits
  }
}
```

7 Bibliography