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 driver for the DE1-SoC using an LT24 LCD screen.[1] In this report the code and testing to create various shapes and lines will be discussed and design choices explained.

Contents

2 Introduction

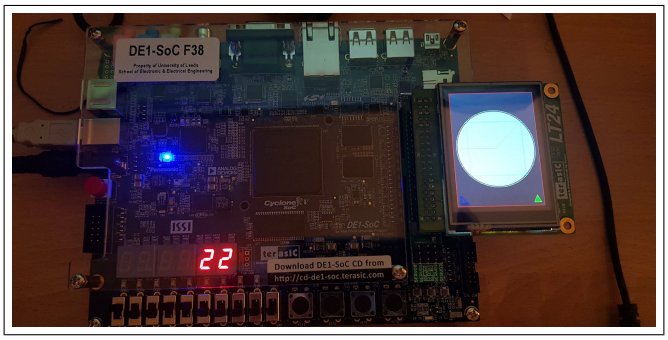

Figure 1: The final working driver displayed on the DE1-SoC

In this assignment there was 3 different prerequisite's, firstly build a simple driver with a source file and a header file, secondly it must be able to draw lines, circles, rectangles, and triangles, finally the shapes must also allow the option to be filled and unfilled. [1] A piece of test code was provided to which produces a test image. This can be seen in appendix 6.1.

The graphics driver utilised the LT24 display driver. It used the function of LT24 drawPixel and initialisation. The graphics driver comprises of algorithms to create the shapes utilising loops. Every pixel drawn is checked to see if it is still in the window of the LCD screen. A window is a defined area which is within the LCD screen. A wrapper was also created to control the 7 segment display to allow the FPS speed to be outputted onto the board.

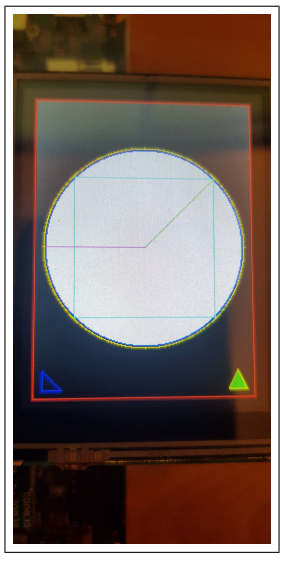

Figure 2: A close up of the final working driver on the LT24 screen

3 Code Design and checking

3.1 Draw Pixel

```
1 void Graphics_drawPixel(unsigned short Colour, unsigned int x, unsigned int y){
2   int status = LT24_drawPixel(Colour,x,y);
3   ResetWDT();
4   if(status != 0){
5     SDisplay_clearAll();
6     SDisplay_set(0, 0x1);
7     SDisplay_set(1, 0xE);
8   }
9 }
```

Figure 3: The function to draw pixel

This function was created to check each pixel is within the LCD window. If a pixel is set outside of the window then the status outputted is not equal to 0. If this happens then the 7 segment display is set to display "E1" for error 1. When error checking is enabled as such in this function it significantly slows down the rendering process for all shapes. An example of this was with hardware optimisation disabled the LCD outputs at 14 FPS. However when enabled it the LCD outputs at 6 FPS. This is a significant reduction in speed. To compensate for this reduction of speed the watchdog would have to be reset as it was causing time outs. Incorporated this to be reset every pixel to prevent a time out.

3.2 Draw Box

The draw box function uses a number of steps to create a box (code in Appendix 6.2.3). Firstly it finds the lowest bottom left corner value of the box by finding the smallest x and smallest y value. It then uses a calculated value for height and width and creates the box using 2 for loops which renders the box one pixel at a time in a typewriter style. Finally, the outline is added to the box.

3.3 Draw Circle

To draw the circle an algorithm was designed utilising Pythagoras theorem (code in Appendix 6.2.4). Pythagoras theorem states that the square of the hypotenuse is equal to the sum of the squares of the other two sides.

When used on a grid such as in Figure 5 where you keep R (the hypotenuse) as the radius and cycle through all Y and X co-ordinates it will create a circle. When doing this programmatically you can create an outline by checking if R is equal to the radius and the fill of the shape by checking if it is smaller than R. A threshold had to be used to ensure that the circle has a full outline and not just 12 dots which will appear if only set the equal R. This specific code can be seen on Appendix 6.2.4 lines 12 to 25.

$$x^2 + y^2 = r^2$$

Figure 4: Pythagoras Theorem

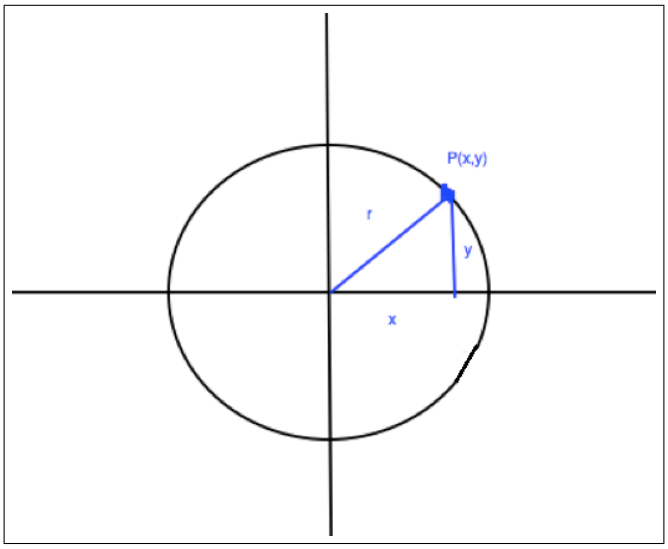

Figure 5: Pythagoras theorem to create circle diagram [n]

3.4 Draw Line

To draw the line (code in appendix 6.2.5) it was more difficult than expected. An algorithm called Bresenham's line algorithm. The code was based off an example found on Rosetta code. [3] The algorithm essentially draws a line across the pixels. If error 2 is larger than dy then x1 is equal to x1 plus a value dependent on the quadrant which the line's angle lies in. If error 2 is smaller than dx then error is increased by dx and y1 is equal to y1 plus a value dependent on the quadrant. This can be better shown in figure 6 where sx and sy are set depending on the quadrant. This turns on the appropriate pixels of where the line lands.

```
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;
  }
}
```

Figure 6: The function to draw pixel

3.5 Draw Triangle

To draw a triangle two functions where created. One function created the outline using the line function and another function was created which was a modification of the line function to fill the triangle.

This modified version of the line function created worked by creating numerous straight lines from with one point set to x3,y3, and another set which moves between x1,y1 and x1,y2. This code was run 3 times to ensure it is filled as in smaller triangles the lines are more distorted. This modification was a single line of code on line 2 of Figure 8.

```
1  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){
2
3    //If fill
4    if(!noFill){
5      //Run fill traingle on 3 occassions to ensure on small triangles that no
       pixel is missed. Reset WD.
6      Graphics_fillTriangle(x1,y1,x2,y2,x3,y3,fillColour);ResetWDT();
7      Graphics_fillTriangle(x3,y3,x1,y1,x2,y2,fillColour);ResetWDT();
8      Graphics_fillTriangle(x2,y2,x3,y3,x1,y1,fillColour);ResetWDT();
9    }
10   //Draw Outline
11   Graphics_drawLine(x1,y1,x2,y2,colour);
12   Graphics_drawLine(x2,y2,x3,y3,colour);
13   Graphics_drawLine(x3,y3,x1,y1,colour);
14 }
```

Figure 7: The main triangle function

```
1      while(1){
2        Graphics_drawLine(x3,y3,x1,y1,fillColour); //drawLine
3          if (x1 == x2 && y1 == y2){ break;}
4          error2 = 2 * error;
5          if (error2 >= dy) {
6            error += dy;
7            x1 += sx;
8          }
9          if (error2 <= dx) {
10           error += dx;
11           y1 += sy;
12         }
13     }
```

Figure 8: The fill triangle function sample showing the single line modification

.

4 Testing and Debugging

Various testing techniques were used to test both software and hardware using the graphics driver.

4.1 Software

In software testing and debugging the Eclipse IDE was used to step through code and test variables to see if they are to be as what is expected. A number of bugs were solved by doing this, an example was such as when a boolean NOT symbol was used instead of the logical NOT symbol (!) which stopped fills working correctly for the shapes as this made the if statements act erratically.

Another example of using this method was testing the timer where it was possible to grab the variables to find how long it took the DE1-SoC to run the test code. A breakpoint would be placed on line 6 of Figure 9 allowing the variables to be seen in the variable explorer in the debugger. Using this for the timer the hardware optimisation code was tested which was enabled from the LT24 driver. It was found that when the hardware optimisation was enabled that the frame rate decreased from 6 FPS to 3 FPS.

```
1  int timerEndValue = *private_timer_value;
2  int timerDuration = timerStartValue - timerEndValue;
3  int freqTimer =  1/225000000 * timerDuration;
4  float FPS = 1/(4.44e-9 * timerDuration);
5  int FPSint = FPS;
6  int freq = freqTimer;
```

Figure 9: The timer stop function snippet

4.2 Hardware

When testing on hardware the example code would be run and checked to see if it was as displayed as expected. The frame rate was outputted from the timer to the 7 segment display using a library which was created. (Appendix 6.4) The error function was also tested by trying to draw a pixel which was not on the physical screen which displayed error E1 on the 7 segment display.

5 Conclusion

In conclusion, creating the Graphics driver was an intellectually stimulating task in which numerous algorithms and testing techniques had to be used. The most difficult challenges were the create a line algorithm and triangle fill algorithm. The Eclipse IDE was found to be a very powerful tool for testing and debugging as it allows us to step through code whilst checking registers and memory variables. The final graphics driver was a success however did not run as fast as wanted. More optimisation would've been possible with more time which could have increased the frames per second.

6 Appendix

6.1 Test code

```
//Initialise the LCD Display.
int main(void) {
 //Initialise the LCD Display.
 Graphics_initialise(0xFF200060,0xFF200080); ResetWDT();
 //Rectangle. Red Border. Grey Fill.
 Graphics_drawBox(10,10,230,310,LT24_RED,false,0x39E7); ResetWDT();
 //Circle. Blue Border, White Fill. Centre of screen. 100px radius
 Graphics_drawCircle(120,160,100,LT24_BLUE,false,LT24_WHITE); ResetWDT();
 //Circle. Yellow Border, No Fill. Centre of screen. 102px radius
 Graphics_drawCircle(120,160,102,LT24_YELLOW,true,0); ResetWDT();
 //Rectangle. Cyan Border, No Fill.
 Graphics_drawBox(49,89,191,231,LT24_CYAN,true,0); ResetWDT();
 //Line. Green. 45 degree Radius of circle.
 Graphics_drawLine(191,89,120,160,LT24_GREEN); ResetWDT();
 //Line. Magenta. 270 degree Radius of circle.
 Graphics_drawLine(120,160,20,160,LT24_MAGENTA); ResetWDT();
 //Triangle. Blue Border, No Fill. Bottom left corner. Right-angle triangle.
 Graphics_drawTriangle(18,283,18,302,37,302,LT24_BLUE,true,0); ResetWDT();
 //Triangle. Yellow Border, Green Fill. Bottom left corner Equilateral triangle.
 Graphics_drawTriangle(213,283,204,302,222,302,LT24_YELLOW,false,LT24_GREEN);
 //Done.
 while (1) { HPS_ResetWatchdog(); } //Watchdog reset.
}
```

6.2 Graphics.c/.h

6.2.1 Graphics Header

```
#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);
```

```
19
20  void Graphics_drawPixel(unsigned short Colour, unsigned int x, unsigned int y);
21
22 #endif
```

6.2.2 Graphics initialise

```
1 void Graphics_initialise(unsigned volatile int lcd_pio_base,unsigned volatile
      int lcd_hw_base){
2   LT24_initialise(lcd_pio_base,lcd_hw_base);
3 }
```

6.2.3 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);
  }


}
```

6.2.4 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);
        }
      }
    }
  }

}
```

6.2.5 Graphics drawLine

```
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;
    }
  }
}
```

6.2.6 Graphics drawTriangle

```
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);
}
```

6.2.7 Graphics fillTriangle

```
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;
        }
    }
}
```

6.2.8 Graphics drawPixel

```
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);
  }
}
```

6.3 Timer.c/.h

6.3.1 Timer Header

```
#ifndef TIMER_H
#define TIMER_H

void timer_Start();

int timer_Stop();


#endif
```

6.3.2 Timer start

```
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;
}
```

6.3.3 Timer stop

```
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;
```

6.4 sevenSeg.c/.h

6.4.1 SDisplay Header

```
#ifndef SEVENSEG_H
#define SEVENSEG_H

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

#endif
```

6.4.2 SDisplay clearAll

```
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
}
```

6.4.3 SDisplay set

```
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

[1] D. Cowell, S. Freear and T. Carpenter, "ELEC5620M: Embedded Microprocessor System Design - Assignment 1", 2019.
[2] Veritas Prep, Pythagoras Theorem to create circle. 2016 [Online]. Available: https://www.veritasprep.com/blog/wp-content/uploads/2016/10/DG-Blog-4.png. [Accessed: 19-Mar-2019].
[3]"Bitmap/Bresenham's line algorithm - Rosetta Code", Rosettacode.org. [Online]. Available: https://rosettacode.org/wiki/Bitmap/Bresenham%27s_line_algorithm. [Accessed: 19-Mar-2019]