



**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 design solutions and testing will be discussed in depth.

# Contents

<b>1</b>	<b>Abstract</b>	<b>1</b>
<b>2</b>	<b>Introduction</b>	<b>3</b>
<b>3</b>	<b>Code Design and checking</b>	<b>4</b>
3.1	Draw Pixel . . . . .	4
3.2	Draw Box . . . . .	4
3.3	Draw Circle . . . . .	4
3.4	Draw Line . . . . .	5
3.5	Draw Triangle . . . . .	6
<b>4</b>	<b>Testing</b>	<b>6</b>
4.1	optimisation . . . . .	6
<b>5</b>	<b>Conclusion</b>	<b>6</b>
<b>6</b>	<b>Appendix</b>	<b>7</b>
6.1	Test code . . . . .	7
6.2	Graphics.c/.h . . . . .	7
6.2.1	Graphics Header . . . . .	7
6.2.2	Graphics initialise . . . . .	8
6.2.3	Graphics drawBox . . . . .	9
6.2.4	Graphics drawCircle . . . . .	10
6.2.5	Graphics drawLine . . . . .	11
6.2.6	Graphics drawTriangle . . . . .	12
6.2.7	Graphics fillTriangle . . . . .	13
6.2.8	Graphics drawPixel . . . . .	13
6.3	Timer.c/.h . . . . .	14
6.3.1	Timer Header . . . . .	14
6.3.2	Timer start . . . . .	14
6.3.3	Timer stop . . . . .	14
6.4	sevenSeg.c/.h . . . . .	15
6.4.1	SDisplay Header . . . . .	15
6.4.2	SDisplay clearAll . . . . .	15
6.4.3	SDisplay set . . . . .	15
<b>7</b>	<b>Bibliography</b>	<b>16</b>

## 2 Introduction

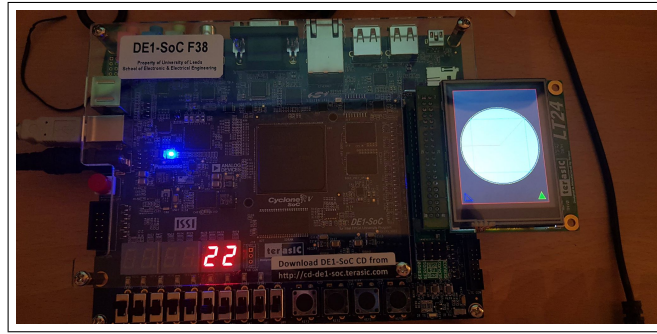


Figure 1: The final working driver displayed on the DE1-SoC

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 LT24 drawPixel and initialisation. The graphics driver comprises of algorithms to create the shapes utilising loops. Every pixel drawn is checked to see if 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.

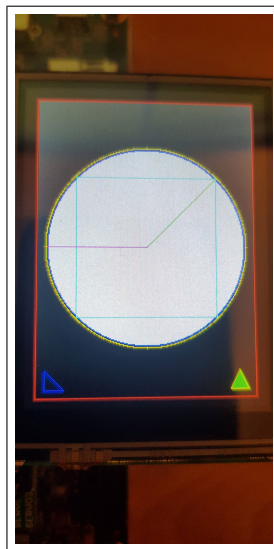


Figure 2: A close up of the final working driver on the LT24 screen

## 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

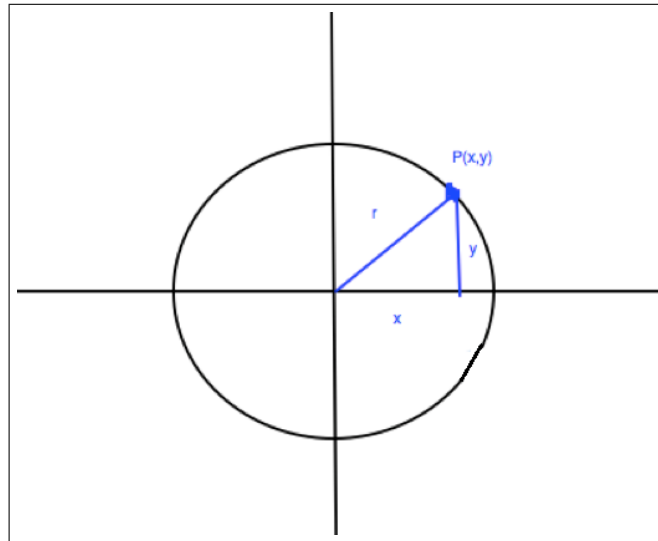


Figure 5: Pythagoras theorem to create circle diagram [n]

### 3.4 Draw Line

To draw the line (code in appendix 6.2.5) is was more difficult than expected. An algorithm called Bresenham's line algorithm. The code was based off an examples 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 dependent on the quadrant. This turns on the appropriate pixels of where the line lands.

```

1  while(1){
2      Graphics_drawPixel(colour ,x1 ,y1);
3      if (x1 == x2 && y1 == y2){ break;}
4      error2 = 2 * error;
5      //if error2 is larger than delta y then add 1 to x
6      if (error2 >= dy) {
7          error += dy;
8          x1 += sx;
9      }
10     //if error2 is smaller than delta x then add 1 to y
11     if (error2 <= dx) {
12         error += dx;
13         y1 += sy;
14     }
15 }

```

Figure 6: The function to draw pixel

### 3.5 Draw Triangle

To draw a triangle two functions were 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.

```
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 triangle on 3 occasions 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

## 4 Testing

### 4.1 optimisation

## 5 Conclusion

## 6 Appendix

### 6.1 Test code

```
1 // Initialise the LCD Display.
2 int main(void) {
3     // Initialise the LCD Display.
4     Graphics_initialise(0xFF200060, 0xFF200080); ResetWDT();
5     // Rectangle. Red Border. Grey Fill.
6     Graphics_drawBox(10, 10, 230, 310, LT24_RED, false, 0x39E7); ResetWDT();
7     // Circle. Blue Border, White Fill. Centre of screen. 100px radius
8     Graphics_drawCircle(120, 160, 100, LT24_BLUE, false, LT24_WHITE); ResetWDT();
9     // Circle. Yellow Border, No Fill. Centre of screen. 102px radius
10    Graphics_drawCircle(120, 160, 102, LT24_YELLOW, true, 0); ResetWDT();
11    // Rectangle. Cyan Border, No Fill.
12    Graphics_drawBox(49, 89, 191, 231, LT24_CYAN, true, 0); ResetWDT();
13    // Line. Green. 45 degree Radius of circle.
14    Graphics_drawLine(191, 89, 120, 160, LT24_GREEN); ResetWDT();
15    // Line. Magenta. 270 degree Radius of circle.
16    Graphics_drawLine(120, 160, 20, 160, LT24_MAGENTA); ResetWDT();
17    // Triangle. Blue Border, No Fill. Bottom left corner. Right-angle triangle.
18    Graphics_drawTriangle(18, 283, 18, 302, 37, 302, LT24_BLUE, true, 0); ResetWDT();
19    // Triangle. Yellow Border, Green Fill. Bottom left corner Equilateral triangle.
20    Graphics_drawTriangle(213, 283, 204, 302, 222, 302, LT24_YELLOW, false, LT24_GREEN);
21    // Done.
22    while (1) { HPS_ResetWatchdog(); } // Watchdog reset.
23 }
```

### 6.2 Graphics.c/.h

#### 6.2.1 Graphics Header

```
1 #ifndef GRAPHICS_H
2 #define GRAPHICS_H
3
4 #include "../DE1SoC_LT24/DE1SoC_LT24.h"
5 #include "../HPS_Watchdog/HPS_Watchdog.h"
6 #include "../sevenSeg/sevenSeg.h"
7
8 void Graphics_initialise(unsigned volatile int lcd_pio_base, unsigned volatile
    int lcd_hw_base);
9
10 void Graphics_drawBox(unsigned int x1, unsigned int y1, unsigned int x2, unsigned
    int y2, unsigned short colour, bool noFill, unsigned short fillColour);
11
12 void Graphics_drawCircle(unsigned int x, unsigned int y, unsigned int r, unsigned
    short colour, bool noFill, unsigned short fillColour);
13
14 void Graphics_drawLine(unsigned int x1, unsigned int y1, unsigned int x2, unsigned
    int y2, unsigned short colour);
15
16 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);
17
18 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

```
1 void Graphics_drawBox(unsigned int x1,unsigned int y1,unsigned int x2,unsigned
  int y2,unsigned short colour,bool noFill,unsigned short fillColour){
2 //Signed Values declares
3 int sx1 = (int) x1;
4 int sx2 = (int) x2;
5 int sy1 = (int) y1;
6 int sy2 = (int) y2;
7 //calculate height and width
8 int height = abs(sy2-sy1);
9 int width = abs(sx1-sx2);
10 //set values for forloops
11 int y=0;
12 int x=0;
13 int oy=0;
14 int ox=0;
15
16 //find bottom left value
17 int llx = 0;
18 int lly = 0;
19 if(sx1<sx2){
20     llx = sx1;
21 }
22 else{
23     llx = sx2;
24 }
25
26 if(sy1<sy2){
27     lly = sy1;
28 }
29 else{
30     lly = sy2;
31 }
32
33 //cube fill (draws first so it can be overdrawn with outline)
34 if(!noFill){
35     for(y=0; y <= height; y++){
36         for(x=0; x<=width; x++){
37             Graphics_drawPixel(fillColour ,x+llx ,y+lly );
38         }
39     }
40 }
41
42 //cube outline
43 //verticle outline
44 for(oy = 0; oy <=height; oy++){
45     Graphics_drawPixel(colour ,sx1 ,lly+oy);
46 }
47 for(oy = 0; oy <=height; oy++){
48     Graphics_drawPixel(colour ,sx2 ,lly+oy);
49 }
50 //horizontal outline
51 for(ox = 0; ox <=width; ox++){
52     Graphics_drawPixel(colour ,llx+ox ,sy1 );
53 }
54 for(ox = 0; ox <=width; ox++){
```

```

55 Graphics_drawPixel(colour , llx+ox , sy2);
56 }
57
58
59 }

```

## 6.2.4 Graphics drawCircle

```

1 void Graphics_drawCircle(unsigned int x, unsigned int y, unsigned int r, unsigned
  short colour, bool noFill, unsigned short fillColour){
2 //Radius as signed int
3 int signedr = (int) r;
4 //Radius squared
5 int rad2 = signedr * signedr;
6 //Outline threshold
7 int outThres = 230;
8 //Go through x's
9 int xc = 0;
10 int yc = 0;
11 //Loop through all X and Y of square the size of radius squared
12 for (xc = -signedr-3; xc <= signedr + 3; xc++) {
13     for (yc = -signedr-3; yc <= signedr + 3; yc++) {
14         //radius squared = yc^2 + xc^2
15         int pyr = (yc*yc) + (xc*xc);
16         //If no fill then draw outline
17         if (noFill && (pyr > rad2-outThres) && (pyr <= rad2)){
18             Graphics_drawPixel(colour ,xc+x,yc+y);
19         }
20         //If fill draw fill
21         else if (!noFill && pyr <= rad2){
22             Graphics_drawPixel(fillColour ,xc+x,yc+y);
23         }
24     }
25 }
26 //if fill draw outline last over fill
27 if (~noFill){
28     xc = 0;
29     yc = 0;
30     for (xc = -signedr-3; xc <= signedr+3; xc++) {
31         for (yc = -signedr-3; yc <= signedr+3; yc++) {
32             //get r and check if it the same as radius
33             int pyr = (yc*yc) + (xc*xc);
34             if ((pyr > rad2-outThres) && (pyr <= rad2)){
35                 LT24_drawPixel(colour ,xc+x,yc+y);
36             }
37         }
38     }
39 }
40
41 }

```

## 6.2.5 Graphics drawLine

```
1 void Graphics_drawLine(unsigned int x1, unsigned int y1, unsigned int x2, unsigned
  int y2, unsigned short colour){
2 //REFERENCE: drawLine using Bresenham's algorithm. https://rosettacode.org/wiki/Bitmap/Bresenham%27s\_line\_algorithm
3
4 //calculate deltas
5 int dx = abs (x2 - x1);
6 int dy = -abs (y2 - y1);
7 //calculate error
8 int error = dx + dy;
9 int error2;
10 //Find quadrant
11 int sy;
12 int sx;
13 if (x1 < x2) {
14     sx = 1;
15 }
16 else {
17     sx = -1;
18 }
19
20 if (y1 < y2) {
21     sy = 1;
22 }
23 else {
24     sy = -1;
25 }
26 //Loop though and calculate line pixels
27 while(1){
28     Graphics_drawPixel(colour, x1, y1);
29     if (x1 == x2 && y1 == y2) { break; }
30     error2 = 2 * error;
31     //if error2 is larger than delta y then add 1 to x
32     if (error2 >= dy) {
33         error += dy;
34         x1 += sx;
35     }
36     //if error2 is smaller than delta x then add 1 to y
37     if (error2 <= dx) {
38         error += dx;
39         y1 += sy;
40     }
41 }
42 }
```

## 6.2.6 Graphics drawTriangle

```
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 triangle on 3 occasions 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 }
```

### 6.2.7 Graphics fillTriangle

```
1 void Graphics_fillTriangle(unsigned int x1,unsigned int y1,unsigned int x2,
2   unsigned int y2,unsigned int x3,unsigned int y3,unsigned short fillColour){
3   //A rewrite of the straight line function to allow it to drawlines to fill the
4   triangle.
5   //calculate deltas
6   int dx = abs (x2 - x1);
7   int dy = -abs (y2 - y1);
8   //calculate error
9   int error = dx + dy;
10  int error2;
11  int sy;
12  int sx;
13  if (x1<x2){
14    sx = 1;
15  }
16  else{
17    sx = -1;
18  }
19  if (y1<y2){
20    sy = 1;
21  }
22  else{
23    sy = -1;
24  }
25  while(1){
26    Graphics_drawLine(x3,y3,x1,y1,fillColour); //drawLine
27    if (x1 == x2 && y1 == y2){ break;}
28    error2 = 2 * error;
29    if (error2 >= dy) {
30      error += dy;
31      x1 += sx;
32    }
33    if (error2 <= dx) {
34      error += dx;
35      y1 += sy;
36    }
37  }
```

### 6.2.8 Graphics drawPixel

```
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 }
```

## 6.3 Timer.c/.h

### 6.3.1 Timer Header

```
1 #ifndef TIMER_H
2 #define TIMER_H
3
4 void timer_Start();
5
6 int timer_Stop();
7
8
9 #endif
```

### 6.3.2 Timer start

```
1 void timer_Start() {
2 //initialise
3 //start timer
4 int timerS = 0;
5 *private_timer_load = 100000000;
6 // Set the "Prescaler" value to 0, Enable the timer (E = 1), Set Automatic
  reload
7 // on overflow (A = 1), and disable ISR (I = 0)
8 *private_timer_control = (0 << 8) | (0 << 2) | (1 << 1) | (1 << 0);
9
10 timerS = *private_timer_value;
11
12 timerStartValue = timerS;
13 }
```

### 6.3.3 Timer stop

```
1 int timer_Stop() {
2 //stop timer
3 //print timer end value
4 //print difference
5 int timerEndValue = *private_timer_value;
6 int timerDuration = timerStartValue - timerEndValue;
7 int freqTimer = 1/225000000 * timerDuration;
8 float FPS = 1/(4.44e-9 * timerDuration);
9 int FPSint = FPS;
10 int freq = freqTimer;
11
12 *private_timer_control = 0;
13
14 return FPSint;
```

## 6.4 sevenSeg.c/.h

### 6.4.1 SDisplay Header

```
1 #ifndef SEVENSEG_H
2 #define SEVENSEG_H
3
4 void SDisplay_PNum(int number, int pair);
5 void SDisplay_clearAll();
6 void SDisplay_set(int Display, int HexValue);
7
8 #endif
```

### 6.4.2 SDisplay clearAll

```
1 void SDisplay_clearAll(){
2     //Hex memory base
3     volatile int *HEX0 = (int*) 0XFF200020;
4     volatile int *HEX1 = (int*) 0XFF200030;
5     int zero = 0x00;
6
7     *HEX0 &= zero; //clear bits
8     *HEX1 &= zero; //clear bits
9 }
```

### 6.4.3 SDisplay set

```
1 void SDisplay_set(int Display, int HexValue){
2     //Hex memory base
3     volatile int *HEX0 = (int*) 0XFF200020;
4     volatile int *HEX1 = (int*) 0XFF200030;
5
6     int invClearBits = 0x7F; //inverted bits to put through and bitwise
7     int shiftAmount = 8; //shift multiple amount
8     int hex1Adjust = 4; //adjust amount for second memory address
9
10    if(Display < 4){
11        *HEX0 &= ~(invClearBits << (Display * shiftAmount)); //clear bits
12        *HEX0 |= (HexSDisplay[HexValue] << (Display * shiftAmount)); //set bits
13    }
14    else{
15        *HEX1 &= ~(invClearBits << ((Display - hex1Adjust) * shiftAmount)); //clear
16        *HEX1 |= (HexSDisplay[HexValue] << ((Display - hex1Adjust) * shiftAmount)); //set bits
17    }
18 }
```

## 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](https://rosettacode.org/wiki/Bitmap/Bresenham%27s_line_algorithm). [Accessed: 19- Mar- 2019]