ELEC5620M
EMBEDDED MICROPROCESSOR SYSTEM DESIGN
MINI-PROJECT REPORT

SIMPLE LT24 LCD GRAPHICS ENGINE

Phaneeshwar Mallakkagari
Student ID: 201387234

Contents

Phaneeshwar Mallakkagari
Student ID: 201387234

1.Abstract

This report covers the design, implementation, and verification of different shapes, programmed using embedded C. The model is well structured using various concepts of embedded systems. This report briefs about the implementation design and the code of the task to create a Lt24 graphics driver for DE1-SoC.

2.Introduction

The design aims to create different shapes of various sizes using embedded C programming. The Shape of the sizes can be varied according to the user needs. In this project, different drivers were built with source and header files. This task was demonstrated using Terasic LT24 LCD controller, that is used to display 240(H) x 320(W) image. The LCD module should be initialized before sending image data to LT24 image display. Throughout this project, firstly build a simple driver with source and header files to display different sizes of various shapes such as draw line, circle, rectangle box and triangles. These shapes have an option to be filled with different colours or left unfilled. Furthermore, this project also uses LT24 display driver for utilizing LT24drawpixel and initialization functions.

3. Code Design

3.1 Graphics_drawBox

This function is created with number of steps to design a box. Firstly, it determines the lowest bottom left coordinates of the box. The height and width of the Box is calculated. Furthermore, every pixel that connects all four coordinates are set with the colour defined by the user and the respective border is drawn. If the box needs to be filled with a unique colour then the interested fillcolour is passed as a parameter to the function “Graphics_drawbox” and the parameter “nofill” is set false. The algorithm fills the respective colour in the box by traversing through all the pixels within the region of interest.

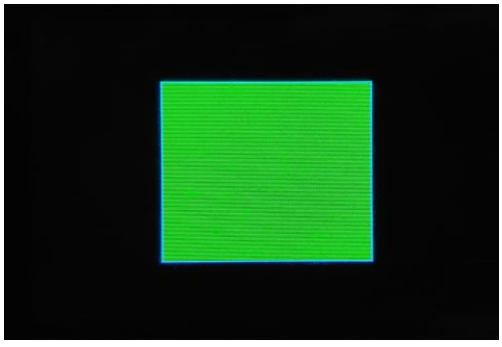

Figure 1: Graphics Box from LT24 display

3.2 Graphics_drawCircle

The Circle algorithm is designed using the Pythagoras theorem. As Pythagoras theorem states that Sum of squares of the shorter sides is to be equal to the square of the hypotenuse. A circle can be defined as a locus of all points that satisfy the equation (1). This means that any point on the circle ,this equation is true.

$$(x\text{-}x_1)^2 + (y\text{-}x_2)^2 = r^2 \quad (1)$$

In the above equation x_1 & x_2 are the centers of the circle. In this algorithm I have considered hypotenuse as the radius of the circle and drag around through all x and y coordinates, this will create a circle. This code will set all the pixels that could create a circle of defined dimensions. Moreover, the circle can be filled with different colours by traversing through all the pixels in the circle and set the respective pixels to user defined colours. A threshold variable has been utilized; this would ensure that the circle has full outline rather than just dotted points.

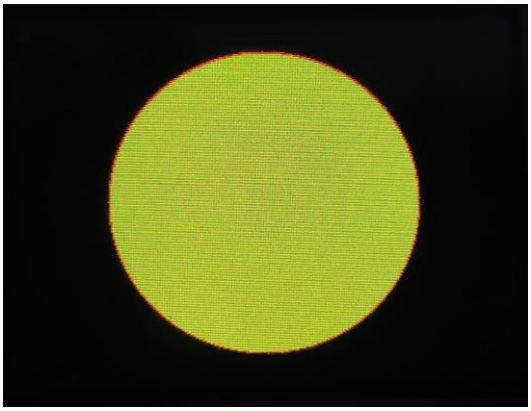

Figure 2: Graphics Circle from LT24 display

3.4 Graphics_drawLine

This Function was designed successfully with help of the algorithm called Bresenhams's line algorithm. Most widely used algorithm in field of computer graphics, which sets up the required pixel by deciding the degree of closeness. Basic idea of this algorithm is to avoid any floating-point multiplication and additions. In this code we increase x by factor of one and choose about next y, to decide whether to remain in y coordinate or increment to y + 1 coordinate. Moreover, we would pick up y-value that is closer to the original line. Hence decision parameter is used to decide which pixel to set high , and therefore to achieve this, the slope error is kept in track from previous increment to y. This algorithm is hard coded in embedded C and required function line is determined for any pair of coordinate points.

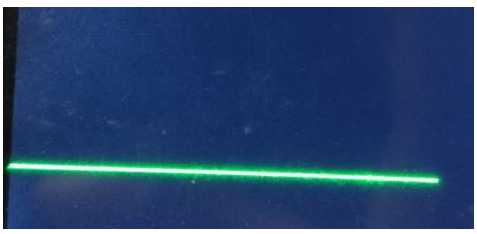

Figure 3: Graphics drawline from LT24 display

3.5 Graphics_drawTriangle

This function is used for designing different types of triangles on LT24 display. I have designed this function with help of two other functions. One of the functions is constructed to create outline of the triangle while another function is used for filling up the region of interest. To draw the outline of the shape we use “Graphics_drawLine” function. This function achieves the task by connecting a pair of points of the triangle, in this function the coordinates are passed as a function parameter.

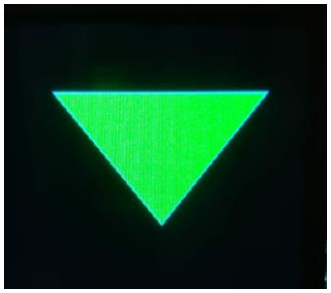

Figure 4: Graphics Triangle from LT24 display

3.6 Timer

In this module the timer headers are designed using advance concepts of pointers and memory accessing. In this project timer module is been used to calculate the latency of the code. To determine how long it took for the board to run the given test code.

Load = (System frequency * Required frequency) / (4*(Prescaler + 1))

The above equation is used to determine Required frequency and can set the timer with any frequency.

4 Testing and Debugging

Various testing techniques were used in both software and hardware using both graphics and timer driver.

4.1 Software

While Eclipse IDE has been used for debugging and testing the code through various steps, to check if the results obtained are as expected. Several bugs were solved while creating header and source files for the timer operation and lt24graphics module. Number of breakpoints were used to find the time of execution of the test code using DE1-SoC. Hence timer operation was specifically designed for this application. This could help us to debug and grab variables to find the duration of execution of the main code. Furthermore, the all the variables were verified using the variable explorer in the debugger. Hardware optimizations were tested using the timer operation. It was observed that the hardware optimization was achieved with decreasing in frame rates.

4.2 Hardware

When testing the hardware, part of a code was checked every time to see if the respective function plots were obtained on the display LT24 display. After testing the test code , different shapes and sizes were tested with the help of rand() function as an input parameter and achieved successful results. Furthermore, the assigning of the pixel can analyze with help of Oscilloscope by using GPIO pins on the board. By analyzing variations in output digital waveforms.

5. Conclusion

In Conclusion, creating the graphic driver and the timer operations was a stimulating task in which numerous algorithms and testing techniques had been used. Eclipse IDE was found to be very advancing tool for debugging and the testing the code through step by step with help of breakpoints and checking memory variables. This design was thoroughly tested and verified.

6. Appendix

6.1 Code

6.1.1 Main.c

```
#include "DE1SoC_LT24/DE1SoC_LT24.h"
#include "HPS_Watchdog/HPS_Watchdog.h"
#include "Timer11/Timer11.h"
#include "lt24graphics.h"
#include <math.h>
#include <stdlib.h>


void exitOnFail(signed int status, signed int successStatus)
{
    if (status != successStatus)
    {
        exit((int)status);
    }
}

int main(void) {

            volatile unsigned int base_add = 0xFFFEC600;
            volatile unsigned int load = 0x00;
            volatile unsigned int control = 0x0003;
            volatile unsigned int prescaler = 0x00;
            unsigned int timer_interrupt = 0;
            unsigned int Starttimer = 0;
            unsigned int Endtimer = 0;
            unsigned int duration = 0;
            int FPS = 0;


           load = 0xFFFFFFFF;
              exitOnFail (Timer_intialise(base_add),TIMER_SUCCESS);
                      exitOnFail (Timer_load(load),TIMER_SUCCESS);
                      exitOnFail (Timer_Prescaler(prescaler),TIMER_SUCCESS);
```

```
                    exitOnFail (Timer_Control(control),TIMER_SUCCESS);
                    ResetWDT();


      while(1)
      {

       Starttimer = Timer_counterread();

             exitOnFail
(Graphics_initialise(0xFF200060,0xFF200080),GRAPHICS_SUCCESS); ResetWDT();
             //Rectangle. Red Border. Grey Fill.

             exitOnFail
(Graphics_drawBox(10,10,230,310,LT24_RED,false,0x39E7),GRAPHICS_SUCCESS);
ResetWDT();
             //Circle. Blue Border, White Fill. Centre of screen. 100px radius
             exitOnFail
(Graphics_drawCircle(120,160,100,LT24_RED,false,LT24_YELLOW),GRAPHICS_SUCCESS);
ResetWDT();
             //Circle. Yellow Border, No Fill. Centre of screen. 102px radius
             exitOnFail
(Graphics_drawCircle(120,160,102,LT24_BLUE,false,0),GRAPHICS_SUCCESS); ResetWDT();
             //Rectangle. Cyan Border, No Fill.
             exitOnFail
(Graphics_drawBox(49,89,191,231,LT24_CYAN,false,LT24_GREEN),GRAPHICS_SUCCESS);
ResetWDT();
             //Line. Green. 45 degree Radius of circle.
             exitOnFail
(Graphics_drawLine(191,89,120,160,LT24_GREEN),GRAPHICS_SUCCESS); ResetWDT();
             //Line. Magenta. 270 degree Radius of circle.
             exitOnFail
(Graphics_drawLine(120,160,20,160,LT24_MAGENTA),GRAPHICS_SUCCESS); ResetWDT();
             //Triangle. Blue Border, No Fill. Bottom left corner. Right-angle
triangle.
             exitOnFail
(Graphics_drawTriangle(18,283,18,302,37,302,LT24_BLUE,false,0),GRAPHICS_SUCCESS);
ResetWDT();
             //Triangle. Yellow Border, Green Fill. Bottom left corner
Equilateral triangle.
             exitOnFail
(Graphics_drawTriangle(213,283,204,302,222,302,LT24_YELLOW,false,LT24_GREEN),GRAPH
ICS_SUCCESS);ResetWDT();
             //Done.

          Endtimer = Timer_counterread();
          duration = Starttimer - Endtimer;
          FPS = 1/(4.44e-9 * duration);

       while (1) { HPS_ResetWatchdog(); } //Watchdog reset.
       }
}
```

6.1.2 Timer11.c

```
#include "Timer11.h"

volatile unsigned int *base_address_ptr = 0x00;
```

```
bool timer_intialized = false;
bool load_intialized = false;

#define Timer_Load      (0x00/sizeof(unsigned int))
#define Timer_counter   (0x04/sizeof(unsigned int))
#define Timer_control   (0x08/sizeof(unsigned int))
#define Timer_interrupt (0x0C/sizeof(unsigned int))

signed int Timer_intialise (unsigned int base_address)
{
       base_address_ptr = (unsigned int *) base_address;
       base_address_ptr[Timer_control] = 0x00;

       timer_intialized = true;

       return TIMER_SUCCESS;
}

bool Timer_isIntialised(void)
{
       return timer_intialized;
}

signed int Timer_load (unsigned int load)
{
       if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
       load_intialized = true;
    base_address_ptr[Timer_Load] = load;

    return TIMER_SUCCESS;

}
bool Load_isIntialised(void)
{
       return load_intialized;
}


signed int Timer_Prescaler (unsigned int prescaler)
{
       unsigned int variable;
    unsigned int *timer_ptr;  // declare a seprate pointer
       if (!Timer_isIntialised()) return TIMER_ERRORNOINT;

       variable =  *base_address_ptr;
       timer_ptr = &variable;
       timer_ptr[Timer_control] &= 0x00FF;
       timer_ptr[Timer_control] |= (prescaler << 8 );
       base_address_ptr[Timer_control] |= timer_ptr[Timer_control];
       return TIMER_SUCCESS;
}

signed int Timer_Control (unsigned int control)
{
       unsigned int *timer_ptr1;
       unsigned int variable1;
       if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
       if (!Load_isIntialised())  return TIMER_ERRORNOINT;
```

```
      variable1 = *base_address_ptr;
    timer_ptr1 = &variable1;
    timer_ptr1[Timer_control] &= ~0x7;
    timer_ptr1[Timer_control] = 0x7 & control;
    base_address_ptr[Timer_control] |= timer_ptr1[Timer_control];
    return TIMER_SUCCESS;
}

signed int Timer_counterread (void)
{

      if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
      if (!Load_isIntialised())  return TIMER_ERRORNOINT;
    return base_address_ptr[Timer_counter];
}

signed int Timer_interuppt(void)
{
      unsigned int value = 0;
      unsigned int *timer_ptr;
      if (!Timer_isIntialised()) return TIMER_ERRORNOINT;
      if (!Load_isIntialised())  return TIMER_ERRORNOINT;
      value = base_address_ptr[Timer_interrupt];

      if (value == 1)
      {
            base_address_ptr[Timer_interrupt] = 0x01;
      }

}
```

6.1.3 Timer11.h

```
#ifndef TIMER11_TIMER11_H_
#define TIMER11_TIMER11_H_

#include <stdbool.h>

#define TIMER_SUCCESS 0
#define TIMER_ERRORNOINT -1
#define INTERUPPT_HIGH 1
#define INTERUPPT_LOW 2

signed int Timer_intialise (unsigned int base_address);
bool Timer_isIntialised(void);

signed int Timer_load (unsigned int load);
signed int Timer_Prescaler (unsigned int prescaler);
signed int Timer_Control (unsigned int control);
signed int Timer_counterread (void);
signed int Timer_interuppt(void );


#endif /* TIMER11_TIMER11_H_ */
```

6.1.4 lt24graphics.h

```
#ifndef LT24GRAPHICS_H_
```

```
#define LT24GRAPHICS_H_
#include <math.h>
#include "DE1SoC_LT24/DE1SoC_LT24.h"
#include "HPS_Watchdog/HPS_Watchdog.h"
#include <stdbool.h>

#define GRAPHICS_SUCCESS 0


signed int Graphics_initialise(unsigned int lcd_pio_base,unsigned int
lcd_hw_base);
signed int  Graphics_drawBox(unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned short colour, bool noFill, unsigned short fillColour);
signed int  Graphics_drawCircle(unsigned int x, unsigned int y, unsigned int r,
unsigned short colour, bool noFill, unsigned short fillColour);
signed int  Graphics_drawLine(unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned short colour);
signed int  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);
signed int Graphics_fill_Tri (unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned int x3, unsigned int y3, unsigned short fillColour);
signed int  Graphics_SetPixel (unsigned short colour,unsigned int x, unsigned int
y);


#endif /* LT24GRAPHICS_H_ */
```

6.1.5 lt24graphics.c

```
#include "lt24graphics.h"
#include <math.h>

signed int  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 (!noFill)   // to check if the triangle needed to be filled or not
	{
		Graphics_fill_Tri (x1,y1,x2,y2,x3,y3,fillColour); ResetWDT(); //
this fnction is used for filling up all the pixels in the region of interest
		Graphics_fill_Tri (x3,y3,x1,y1,x2,y2,fillColour); ResetWDT();
		Graphics_fill_Tri (x2,y2,x3,y3,x1,y1,fillColour); ResetWDT();
	}

	Graphics_drawLine (x1,y1,x2,y2,colour);  // drawline function is used for
joining two coordinate points, Hence if all the three coordinates
	Graphics_drawLine (x2,y2,x3,y3,colour); //.... are joined then we obtain a
triangle
	Graphics_drawLine (x3,y3,x1,y1,colour);

	return GRAPHICS_SUCCESS;		// to return true if the function is
executed without any errors

}
```

```
signed int  Graphics_fill_Tri (unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned int x3, unsigned int y3, unsigned short fillColour)
{
	int diffx = abs(1.0*(x2 - x1));		// obtain the difference between
the two x-coordinate values. here absolute function is used. and respective deltas
are calculated
	int diffy = -abs(1.0*(y2-y1));

	int err1 = diffx + diffy;  // to calculate the error function
	int err2;
    int kx = x1<x2 ? 1 : -1; // assign the kx and ky to either -1 or +1 depending
on the position of the coordinates x1,x2 and y1,y2 respectively.
	int ky = y1<y2 ? 1 : -1;

	while (1)
	{
		Graphics_drawLine (x3,y3,x1,y1,fillColour); // to draw the line with
the fill colour given byt the user

		if (x1 == x2 && y1 == y2)
		{
			break;
		}
		err2 = 2 * err1;		// to avoid floating point values hence
error is determined.
		if (err2 >= diffy)		//.. it acts like a decision parameter,
and to keep track of the pixel value.
		{
			err1 = err1 + diffy;
			x1 = x1 + kx;
		}
		if (err2 <= diffx)
		{
			err1 = err1 + diffx;
			y1 = y1 + ky;
		}
	}

	return GRAPHICS_SUCCESS;   // to return true if the function is executed
without any errors

}

signed int  Graphics_drawLine (unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned short colour) {

	int diffx = abs(1.0 * (x2 - x1));  // obtain the difference between the two
x-coordinate values. here absolute function is used. and respective deltas are
calculated
	int diffy = -abs(1.0* (y2 - y1));

	int err1 = diffx + diffy;
	int err2;

	int kx = x1<x2 ? 1:-1;
	int ky = y1<y2 ? 1:-1;

	while(1)
	{
```

```
		Graphics_SetPixel(colour,x1,y1);

		if (x1 == x2 && y1 == y2)
		{
			break;
		}
		err2 = 2 * err1;   // this equation is used to avoid floating point
values
		//... Also to determine which pixel to set high based on the degree
of closeness from the original line to the midpoint of the pixel
		// ... Bresenhams line algorithm has been utilized. Slope error is
used to determine the required pixel.

		if (err2 >= diffy)
		{
			err1 = err1 + diffy;
			x1 = x1 + kx;
		}

		if (err2 <= diffx)
		{
			err1 = err1 + diffx;
			y1 = y1 + ky;
		}
	}
	return GRAPHICS_SUCCESS; // to return true if the function is executed
without any errors
}

signed int  Graphics_drawCircle (unsigned int x, unsigned int y, unsigned int r,
unsigned short colour, bool noFill, unsigned short fillColour)
{
	// Circle determined using concept of pythagorous theorem
	int rad1 = (int) r;
	int rad2 = rad1*rad1;
	int threshold = 230;  // to outline the threhold value, used to set pixels
within the scope tof the defined region
	int pythagorous = 0;  // using pythagorous theorem , this variable is used
for determining the hypotaneous
	int xc = 0;
	int yc = 0;


	for (xc = -rad1-3; xc <= rad1 + 3; xc++)
	{
		for(yc = -rad1-3; yc <= rad1 + 3; yc++)
		{
			pythagorous = (xc*xc) + (yc*yc);
			// if nofill is true then fill outline of circle with colour
parameter
			if (noFill && (pythagorous > rad2-threshold) && (pythagorous
<= rad2))
			{
				Graphics_SetPixel(colour, xc+x,yc+y);
			}
			// if nofill is false then fill outline of circle with
fillcolour parameter
			else if(!noFill && pythagorous <= rad2)
			{
```

```
                        Graphics_SetPixel (fillColour, xc+x,yc+y);
                }
        }
}

        // This function is used to set user defined colour within the defined
circle region
        //..... all the pixels within the region of interest are assigned high.
        if(~noFill)
        {
                xc = 0;
                yc = 0;

                for (xc = -rad1-3;xc <= rad1+3; xc++)
                {
                        for (yc = -rad1-3;yc <= rad1+3; yc++)
                        {
                                pythagorous = (xc*xc) + (yc*yc);
                                if ((pythagorous > rad2-threshold)&&(pythagorous <=
rad2))
                                {
                                        Graphics_SetPixel(colour,xc+x,yc+y);
                                }
                        }
                }
        }
        return GRAPHICS_SUCCESS; // to return true if the function is executed
without any errors
}

signed int  Graphics_drawBox (unsigned int x1, unsigned int y1, unsigned int x2,
unsigned int y2, unsigned short colour, bool noFill, unsigned short fillColour)
{
        int X1 = (int) x1;
        int X2 = (int) x2;
        int Y1 = (int) y1;
        int Y2 = (int) y2;

        int H = abs(1.0*(Y2 - Y1));  // Height and Width of the box is determined
        int W = abs(1.0*(X1 - X2)); //... Used for determing the coordinates for
remaining points

        int i=0;
        int j=0;
        int z1=0;
        int z2=0;

        int kx = X1<X2 ? X1:X2;
        int ky = Y1<Y2 ? Y1:Y2;

        // This conditional statement below is used for setting up all the pixels
in the given dimensions of the box
        if(!noFill)
        {
                for (i = 0; i<= H; i++)
                {
                        for (j=0; j<= W; j++)
                        {
                                Graphics_SetPixel(fillColour, j+kx, i+ky);
```

```
                }
        }
}

// if nofill is true then only outline of the box is drawn
//... two vertical lines are drawn using two for loops
for (z1 = 0; z1 <= H; z1++)
{
        Graphics_SetPixel(colour, X1, ky+z1);
}

for (z1 = 0; z1 <= H; z1++)
        {
                Graphics_SetPixel(colour, X2, ky+z1);
        }
//..//... two horizontal lines are drawn using two for loops

for (z2 = 0; z2 <= W; z2++)
                {
                        Graphics_SetPixel(colour, kx+z2, Y1);
                }

for (z2 = 0; z2 <= W; z2++)
                {
                        Graphics_SetPixel(colour, kx+z2, Y2);
                }

        return GRAPHICS_SUCCESS; // to return true if the function is executed
without any errors
}

signed int  Graphics_initialise(unsigned volatile int lcd_pio_base, unsigned
volatile int lcd_hw_base)
{
        LT24_initialise(lcd_pio_base,lcd_hw_base); // used for initialising LT24
display
        return GRAPHICS_SUCCESS;
}

signed int  Graphics_SetPixel(unsigned short colour,unsigned int x, unsigned int
y)
{
        LT24_drawPixel(colour,x,y);  // to set pixel colour for the selected pixel
on the display.
        ResetWDT();
        return GRAPHICS_SUCCESS;
}
```

6.1.6 DE1SoC_LT24.c (Given Code From University)

```
#include "DE1SoC_LT24.h"

#include "../HPS_Watchdog/HPS_Watchdog.h"

#include "../HPS_usleep/HPS_usleep.h" //some useful delay routines
```

```
//
// Driver global static variables (visible only to this .c file)
//

//Driver Base Addresses
volatile unsigned int   *lt24_pio_ptr    = 0x0;  //0xFF200060
volatile unsigned short *lt24_hwbase_ptr = 0x0;  //0xFF200080
//Driver Initialised
bool lt24_initialised = false;

//
// Useful Defines
//

//Uncomment this #define to enable the Hardware Optimised mode.
//#define HARDWARE_OPTIMISED

//PIO Bit Map
#define LT24_WRn       (1 << 16)
#define LT24_RS        (1 << 17)
#define LT24_RDn       (1 << 18)
#define LT24_CSn       (1 << 19)
#define LT24_RESETn    (1 << 20)
#define LT24_LCD_ON    (1 << 21)
#define LT24_HW_OPT    (1 << 23)
#define LT24_CMDDATMASK (LT24_CSn | LT24_RDn | LT24_RS | LT24_WRn | 0x0000FFFF) //CMD and Data bits in PIO

//LT24 Dedicated Address Offsets
#define LT24_DEDCMD  (0x00/sizeof(unsigned short))
```

```
#define LT24_DEDDATA (0x02/sizeof(unsigned short))


//LT24 PIO Address Offsets
#define LT24_PIO_DATA (0x00/sizeof(unsigned int))
#define LT24_PIO_DIR  (0x04/sizeof(unsigned int))


//Display Initialisation Data
//You don't need to worry about what all these registers are.
//The LT24 LCDs are complicated things with many settings that need
//to be configured - Contrast/Brightness/Data Format/etc.
#define LT24_INIT_DATA_LEN (sizeof(LT24_initData)/sizeof(LT24_initData[0]))
unsigned short LT24_initData [][2] = {
  //isDat, value
   {false, 0x00EF},
   {true , 0x0003},
   {true , 0x0080},
   {true , 0X0002},
   {false, 0x00CF},
   {true , 0x0000},
   {true , 0x0081},
   {true , 0x00c0},
   {false, 0x00ED},
   {true , 0x0064},
   {true , 0x0003},
   {true , 0X0012},
   {true , 0X0081},
   {false, 0x00E8},
   {true , 0x0085},
   {true , 0x0001},
   {true , 0x0078},
   {false, 0x00CB},
```

```
  {true , 0x0039},
  {true , 0x002C},
  {true , 0x0000},
  {true , 0x0034},
  {true , 0x0002},
  {false, 0x00F7},
  {true , 0x0020},
  {false, 0x00EA},
  {true , 0x0000},
  {true , 0x0000},
  //Power control
  {false, 0x00C0},
  {true , 0x0023}, //VRH[5:0]
  {false, 0x00C1},
  {true , 0x0010}, //SAP[2:0];BT[3:0]
  //VCM control
  {false, 0x00C5},
  {true , 0x003E},
  {true , 0x0028},
  {false, 0x00C7},
   {true, 0X0086},
  // Memory Access Control (MADCTL)
  {false, 0x0036},
  {true , 0x0048},
  // More settings...
  {false, 0x003A},
  {true , 0x0055},
  {false, 0x00B1},
  {true , 0x0000},
  {true , 0x001b},
  {false, 0x00B6},
```

```
{true , 0x0008}, //Non-Display Area Inaccessible

{true , 0x0082}, //Normally White, Normal Scan Direction (A2 = Reverse Scan Direction)

{true , 0x0027}, //320 Lines

//3-Gamma Function Disable

{false, 0x00F2},

{true , 0x0000},

//Gamma curve selected

{false, 0x0026},

{true , 0x0001},

//Set Gamma

{false, 0x00E0},

{true , 0x000F},

{true , 0x0031},

{true , 0x002B},

{true , 0x000C},

{true , 0x000E},

{true , 0x0008},

{true , 0x004E},

{true , 0X00F1},

{true , 0x0037},

{true , 0x0007},

{true , 0x0010},

{true , 0x0003},

{true , 0x000E},

{true , 0x0009},

{true , 0x0000},

{false, 0X00E1},

{true , 0x0000},

{true , 0x000E},

{true , 0x0014},

{true , 0x0003},
```

```
    {true , 0x0011},
    {true , 0x0007},
    {true , 0x0031},
    {true , 0x00C1},
    {true , 0x0048},
    {true , 0x0008},
    {true , 0x000F},
    {true , 0x000C},
    {true , 0x0031},
    {true , 0x0036},
    {true , 0x000f},
    //Frame Rate
    {false, 0x00B1},
    {true , 0x0000},
    {true , 0x0001},
    //Interface Control
    {false, 0x00f6},
    {true , 0x0001},
    {true , 0x0010},
    {true , 0x0000},
    //Disable Internal Sleep
    {false, 0x0011},
};

signed int LT24_initialise( unsigned int pio_base_address, unsigned int pio_hw_base_address )
{
    unsigned int regVal;
    unsigned int idx;

    //Set the local base address pointers
    lt24_pio_ptr = (unsigned int *) pio_base_address;
```

```
   lt24_hwbase_ptr = (unsigned short *) pio_hw_base_address;


   //Initialise LCD PIO direction

   //Read-Modify-Write

   regVal = lt24_pio_ptr[LT24_PIO_DIR]; //Read

   regVal = regVal | (LT24_CMDDATMASK | LT24_LCD_ON | LT24_RESETn | LT24_HW_OPT); //All data/cmd bits are outputs

   lt24_pio_ptr[LT24_PIO_DIR] = regVal; //Write


   //Initialise LCD data/control register.

   //Read-Modify-Write

   regVal = lt24_pio_ptr[LT24_PIO_DATA]; //Read

   regVal = regVal & ~(LT24_CMDDATMASK | LT24_LCD_ON | LT24_RESETn | LT24_HW_OPT); //Mask all data/cmd bits

   regVal = regVal | (LT24_CSn | LT24_WRn | LT24_RDn);  //Deselect Chip and set write and read signals to idle.

#ifdef HARDWARE_OPTIMISED

   regVal = regVal | LT24_HW_OPT; //Enable HW opt bit.

#endif

   lt24_pio_ptr[LT24_PIO_DATA] = regVal; //Write


   //LCD requires specific reset sequence:

   LT24_powerConfig(true);  //turn on for 1ms

   usleep(1000);

   LT24_powerConfig(false); //then off for 10ms

   usleep(10000);

   LT24_powerConfig(true);  //finally back on and wait 120ms for LCD to power on

   usleep(120000);


   //Upload Initialisation Data

   for (idx = 0; idx < LT24_INIT_DATA_LEN; idx++) {

      LT24_write(LT24_initData[idx][0], LT24_initData[idx][1]);
```

```
    }

    //Allow 120ms time for LCD to wake up
    usleep(120000);

    //Turn on display drivers
    LT24_write(false, 0x0029);

    //Mark as initialised so later functions know we are ready
    lt24_initialised = true;

    //And clear the display
    return LT24_clearDisplay(LT24_BLACK);
}

//Check if driver initialised
bool LT24_isInitialised() {
    return lt24_initialised;
}

#ifdef HARDWARE_OPTIMISED
//If HARDWARE_OPTIMISED is defined, use this optimised function. It will be discussed in the lab on LCDs.

//Function for writing to LT24 Registers (using dedicated HW)
//You must check LT24_isInitialised() before calling this function
void LT24_write( bool isData, unsigned short value )
{
    if (isData) {
        lt24_hwbase_ptr[LT24_DEDDATA] = value;
    } else {
```

```
        lt24_hwbase_ptr[LT24_DEDCMD ] = value;
    }
}

#else
//Otherwise use the non-optimised function.

//Function for writing to LT24 Registers (using PIO)
//You must check LT24_isInitialised() before calling this function
void LT24_write( bool isData, unsigned short value )
{
    //PIO controls more than just LT24, so need to Read-Modify-Write
    //First we have to output the value with the LT24_WRn bit low (first cycle of write)
    //Read
    unsigned int regVal = lt24_pio_ptr[LT24_PIO_DATA];
    //Modify
    regVal = regVal & ~LT24_CMDDATMASK;      //Mask all bits for command and data (sets them all to
0)
    regVal = regVal | ((unsigned int)value); //Set the data bits (unsigned value, so cast pads MSBs with
0's)
    if (isData) {
        //For data we set the RS bit high.
        regVal = regVal | (LT24_RS | LT24_RDn);
    } else {
        //For command we don't set the RS bit
        regVal = regVal | (LT24_RDn);
    }
    //Write
    lt24_pio_ptr[LT24_PIO_DATA] = regVal;
    //Then we need to output the value again with the LT24_WRn bit high (second cycle of write)
    regVal = regVal | (LT24_WRn); //Rest of regVal is unchanged, so we just or on the LT24_WRn bit
```

```
   //Write
   lt24_pio_ptr[LT24_PIO_DATA] = regVal;
}

#endif



//Function for configuring LCD reset/power (using PIO)
//You must check LT24_isInitialised() before calling this function
void LT24_powerConfig( bool isOn )
{
   //Read
   unsigned int regVal = lt24_pio_ptr[LT24_PIO_DATA];
   //Modify
   if (isOn) {
      //To turn on we must set the RESETn and LCD_ON bits high
      regVal = regVal |  (LT24_RESETn | LT24_LCD_ON);
   } else {
      //To turn off we must set the RESETn and LCD_ON bits low
      regVal = regVal & ~(LT24_RESETn | LT24_LCD_ON);
   }
   //Write
   lt24_pio_ptr[LT24_PIO_DATA] = regVal;
}

//Function to clear display to a set colour
// - Returns true if successful
signed int LT24_clearDisplay(unsigned short colour)
{
   signed int status;
   unsigned int idx;
```

```
  //Reset watchdog.
  ResetWDT();
  //Define window as entire display (LT24_setWindow will check if we are initialised).
  status = LT24_setWindow(0, 0, LT24_WIDTH, LT24_HEIGHT);
  if (status != LT24_SUCCESS) return status;
  //Loop through each pixel in the window writing the required colour
  for(idx=0;idx<(LT24_WIDTH*LT24_HEIGHT);idx++)
  {
    LT24_write(true, colour);
  }
  //And done.
  return LT24_SUCCESS;
}

//Function to convert Red/Green/Blue to RGB565 encoded colour value
unsigned short LT24_makeColour( unsigned int R, unsigned int G, unsigned int B ) {
  unsigned short colour;
  //Limit the colours to the maximum range
  if (R > 0x1F) R = 0x1F;
  if (G > 0x3F) G = 0x3F;
  if (B > 0x1F) B = 0x1F;
  //Move the RGB values to the correct place in the encoded colour
  colour = (R << 11) + (G << 5) + (B << 0);
  return colour;
}

//Function to set the drawing window on the display
//  Returns 0 if successful
signed int LT24_setWindow( unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int
height) {
  unsigned int xright, ybottom;
```

```
  if (!LT24_isInitialised()) return LT24_ERRORNOINIT; //Don't run if not yet initialised
  //Calculate bottom right corner location
  xright = xleft + width - 1;
  ybottom = ytop + height - 1;
  //Ensure end coordinates are in range
  if (xright >= LT24_WIDTH)   return LT24_INVALIDSIZE; //Invalid size
  if (ybottom >= LT24_HEIGHT) return LT24_INVALIDSIZE; //Invalid size
  //Ensure start coordinates are in range (top left must be <= bottom right)
  if (xleft > xright) return LT24_INVALIDSHAPE; //Invalid shape
  if (ytop > ybottom) return LT24_INVALIDSHAPE; //Invalid shape
  //Define the left and right of the display
  LT24_write(false, 0x002A);
  LT24_write(true , (xleft >> 8) & 0xFF);
  LT24_write(true , xleft & 0xFF);
  LT24_write(true , (xright >> 8) & 0xFF);
  LT24_write(true , xright & 0xFF);
  //Define the top and bottom of the display
  LT24_write(false, 0x002B);
  LT24_write(true , (ytop >> 8) & 0xFF);
  LT24_write(true , ytop & 0xFF);
  LT24_write(true , (ybottom >> 8) & 0xFF);
  LT24_write(true , ybottom & 0xFF);
  //Create window and prepare for data
  LT24_write(false, 0x002c);
  //Done
  return LT24_SUCCESS;
}


//Internal function to generate Red/Green corner of test pattern
signed int LT24_redGreen(unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int height){
```

```
   signed int status;
            unsigned int i, j;
   unsigned short colour;
   //Reset watchdog
   ResetWDT();
   //Define Window
   status = LT24_setWindow(xleft,ytop,width,height);
   if (status != LT24_SUCCESS) return status;
   //Create test pattern
   for (j = 0;j < height;j++){
      for (i = 0;i < width;i++){
         colour = LT24_makeColour((i * 0x20)/width, (j * 0x20)/height, 0);
         LT24_write(true, colour);
      }
   }
   //Done
   return LT24_SUCCESS;
}


//Internal function to generate Green/Blue corner of test pattern
signed int LT24_greenBlue(unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int height){
   signed int status;
            unsigned int i, j;
   unsigned short colour;
   //Reset watchdog
   ResetWDT();
   //Define Window
   status = LT24_setWindow(xleft,ytop,width,height);
   if (status != LT24_SUCCESS) return status;
   //Create test pattern
```

```
  for (j = 0;j < height;j++){
    for (i = 0;i < width;i++){
      colour = LT24_makeColour(0, (i * 0x20)/width, (j * 0x20)/height);
      LT24_write(true, colour);
    }
  }
  //Done
  return LT24_SUCCESS;
}


//Internal function to generate Blue/Red of test pattern
signed int LT24_blueRed(unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int height){
  signed int status;
       unsigned int i, j;
  unsigned short colour;
  //Reset watchdog
  ResetWDT();
  //Define Window
  status = LT24_setWindow(xleft,ytop,width,height);
  if (status != LT24_SUCCESS) return status;
  //Create test pattern
  for (j = 0;j < height;j++){
    for (i = 0;i < width;i++){
      colour = LT24_makeColour((j * 0x20)/height, 0, (i * 0x20)/width);
      LT24_write(true, colour);
    }
  }
  //Done
  return LT24_SUCCESS;
}
```

```
//Internal function to generate colour bars of test pattern
signed int LT24_colourBars(unsigned int xleft, unsigned int ytop, unsigned int width, unsigned int height){
  signed int status;
        unsigned int i, j;
  unsigned int colourbars[6] = {LT24_RED,LT24_YELLOW,LT24_GREEN,LT24_CYAN,LT24_BLUE,LT24_MAGENTA};
  unsigned short colour;
  //Reset watchdog
  ResetWDT();
  //Define Window
  status = LT24_setWindow(xleft,ytop,width,height);
  if (status != LT24_SUCCESS) return status;
  //Generate Colour Bars
  for (j = 0;j < height/2;j++){
    for (i = 0;i < width;i++){
      colour = LT24_makeColour((i * 0x20)/width, (i * 0x20)/width, (i * 0x20)/width);
      LT24_write(true, colour);
    }
  }
  //Generate tone shades
  for (j = height/2;j < height;j++){
    for (i = 0;i < width;i++){
      LT24_write(true, colourbars[(i*6)/width]);
    }
  }
  //Done
  return LT24_SUCCESS;
}
```

```
//Generates test pattern on display
// - returns 0 if successful
signed int LT24_testPattern(){
   signed int status;
   status = LT24_redGreen  (         0,          0,LT24_WIDTH/2,LT24_HEIGHT/2);
   if (status != LT24_SUCCESS) return status;
   status = LT24_greenBlue (         0,LT24_HEIGHT/2,LT24_WIDTH/2,LT24_HEIGHT/2);
   if (status != LT24_SUCCESS) return status;
   status = LT24_blueRed   (LT24_WIDTH/2,          0,LT24_WIDTH/2,LT24_HEIGHT/2);
   if (status != LT24_SUCCESS) return status;
   status = LT24_colourBars(LT24_WIDTH/2,LT24_HEIGHT/2,LT24_WIDTH/2,LT24_HEIGHT/2);
   return status;
}

//Copy frame buffer to display
// - returns 0 if successful
signed int LT24_copyFrameBuffer(const unsigned short* framebuffer, unsigned int xleft, unsigned int
ytop, unsigned int width, unsigned int height)
{   unsigned int cnt;
   //Define Window
   signed int status = LT24_setWindow(xleft,ytop,width,height);
   if (status != LT24_SUCCESS) return status;
   //And Copy
   cnt = (height * width); //How many pixels.
   while (--cnt) {
      LT24_write(true, *framebuffer++);
   }
   //Done    return LT24_SUCCESS;
}
//Plot a single pixel on the LT24 display
// - returns 0 if successful
```

```
signed int LT24_drawPixel(unsigned short colour,unsigned int x,unsigned int y)

{

   signed int status = LT24_setWindow(x,y,1,1); //Define single pixel window

   if (status != LT24_SUCCESS) return status;   //Check for any errors

   LT24_write(true, colour);              //Write one pixel of colour data

   return LT24_SUCCESS;                   //And Done

}
```

6.1.7 DE1SoC_LT24.h

```
#ifndef DE1SoC_LT24_H_
#define DE1SoC_LT24_H_

//Include required header files
#include <stdbool.h> //Boolean variable type "bool" and "true"/"false" constants.

//Error Codes
#define LT24_SUCCESS       0
#define LT24_ERRORNOINIT  -1
#define LT24_INVALIDSIZE  -4
#define LT24_INVALIDSHAPE -6

//Size of the LCD
#define LT24_WIDTH  240
#define LT24_HEIGHT 320

//Some basic colours
#define LT24_BLACK   (0x0000)
#define LT24_WHITE   (0xFFFF)
#define LT24_RED     (0x1F << 11)
#define LT24_GREEN   (0x1F << 6)
#define LT24_BLUE    (0x1F << 0)
#define LT24_YELLOW  (LT24_RED | LT24_GREEN)
#define LT24_CYAN    (LT24_GREEN | LT24_BLUE)
#define LT24_MAGENTA (LT24_BLUE | LT24_RED)


//Function to initialise the LCD
// - Returns 0 if successful
signed int LT24_initialise( unsigned int pio_base_address, unsigned int
pio_hw_base_address );

//Check if driver initialised
// - returns true if initialised
bool LT24_isInitialised( void );

//Function for writing to LT24 Registers (using dedicated HW)
//You must check LT24_isInitialised() before calling this function
void LT24_write( bool isData, unsigned short value );

//Function for configuring LCD reset/power (using PIO)
//You must check LT24_isInitialised() before calling this function
void LT24_powerConfig( bool isOn );
```

```
//Function to clear display to a set colour
// - Returns 0 if successful
signed int LT24_clearDisplay(unsigned short colour);

//Function to convert Red/Green/Blue to RGB565 encoded colour value
unsigned short LT24_makeColour( unsigned int R, unsigned int G, unsigned int B );

//Function to set the drawing window on the display
//  Returns 0 if successful
signed int LT24_setWindow( unsigned int xleft, unsigned int ytop, unsigned int
width, unsigned int height);

//Generates test pattern on display
// - returns 0 if successful
signed int LT24_testPattern( void );

//Copy frame buffer to display
// - returns 0 if successful
signed int LT24_copyFrameBuffer(const unsigned short* framebuffer, unsigned int
xleft, unsigned int ytop, unsigned int width, unsigned int height);

//Plot a single pixel on the LT24 display
// - returns 0 if successful
signed int LT24_drawPixel(unsigned short colour,unsigned int x,unsigned int y);


#endif /*DE1SoC_LT24_H_*/
```

6.1.8 HPS_Watchdog.h

```
#ifndef HPS_WATCHDOG_H_
#define HPS_WATCHDOG_H_

//#define for backwards compatibility
#define ResetWDT() HPS_ResetWatchdog()

// Function to reset the watchdog timer.
__forceinline void HPS_ResetWatchdog() {
    *((volatile unsigned int *) 0xFFD0200C) = 0x76;
}

// Function to get value of the watchdog timer
__forceinline unsigned int HPS_WatchdogValue() {
    return *((volatile unsigned int *) 0xFFD02008);
}

#endif /* HPS_WATCHDOG_H_ */
```

6.1.9 HPS_usleep.c

```
#include "HPS_usleep.h"
#include "../HPS_Watchdog/HPS_Watchdog.h"

//Microsecond sleep function based on Cyclone V HPS SP Timer 1
void usleep(int x) //Max delay ~2.09 seconds
```

```
{
    volatile unsigned int *sptimer1_load = (unsigned int *)0xFFC09000;
    volatile unsigned int *sptimer1_ctrl = (unsigned int *)0xFFC09008;
    volatile unsigned int *sptimer1_irqs = (unsigned int *)0xFFC090A8; //Raw
interrupt status (unmasked)

    if (x <= 0) return; //For delays of 0 we just assume that we are done.
    if (x > 0x200000) x = 0x200000; //For delays longer than the max, set to max.

    //Reset the watchdog before we sleep
    ResetWDT();

    //Perform sleep operation
    *sptimer1_load = (x * 100) - 1; //SPT1 doesn't have prescaler. Clock rate is
100MHz, so times us by 100 to get ticks)
    *sptimer1_ctrl = 0x7;           //IRQ Masked, User Count Value, Enabled
    while(!(*sptimer1_irqs));       //Wait until timer overflows
    *sptimer1_ctrl = 0x4;           //IRQ Masked, Disable timer

    //Reset the watchdog after we sleep
    ResetWDT();
}
```

6.1.10 HPS_usleep.h

```
#ifndef HPS_USLEEP_H_
#define HPS_USLEEP_H_

//Delay for x microseconds
void usleep(int x);

#endif //DELAY_H_
```

7.References

[1] Bresenham's Line Generation Algorithm. (2019, October 15). Retrieved May 16, 2020, from https://www.geeksforgeeks.org/bresenhams-line-generation-algorithm/

[2] Features of a circle from its standard equation | Analytic geometry (video). (n.d.). Retrieved May 16, 2020, from https://www.khanacademy.org/math/geometry/hs-geo-circles/hs-geo-circle-standard-equation/v/radius-and-center-for-a-circle-equation-in-standard-form