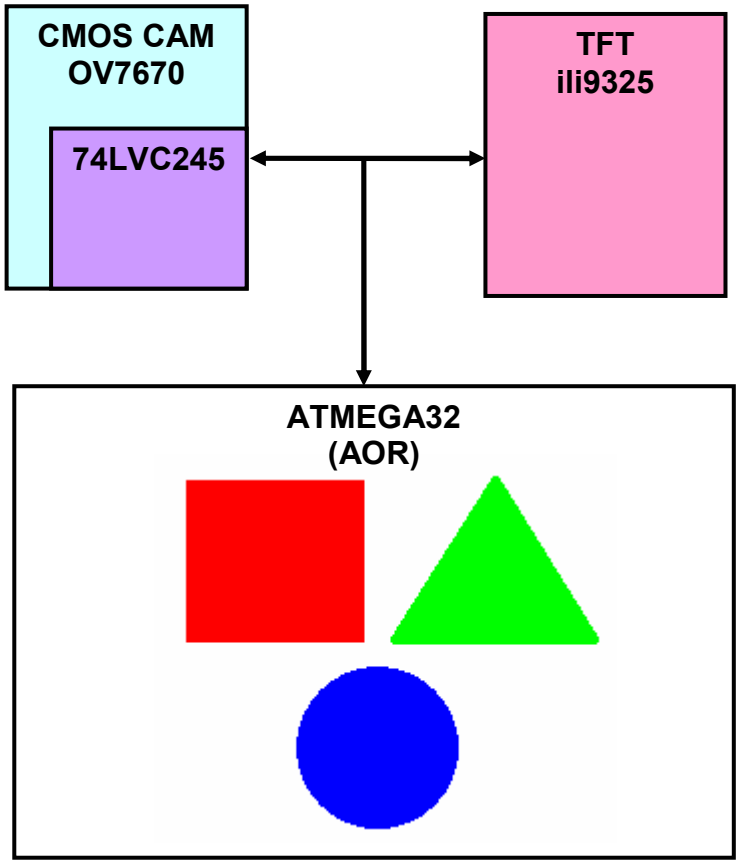
# Documentation AOR

## Content



## 1. System overview

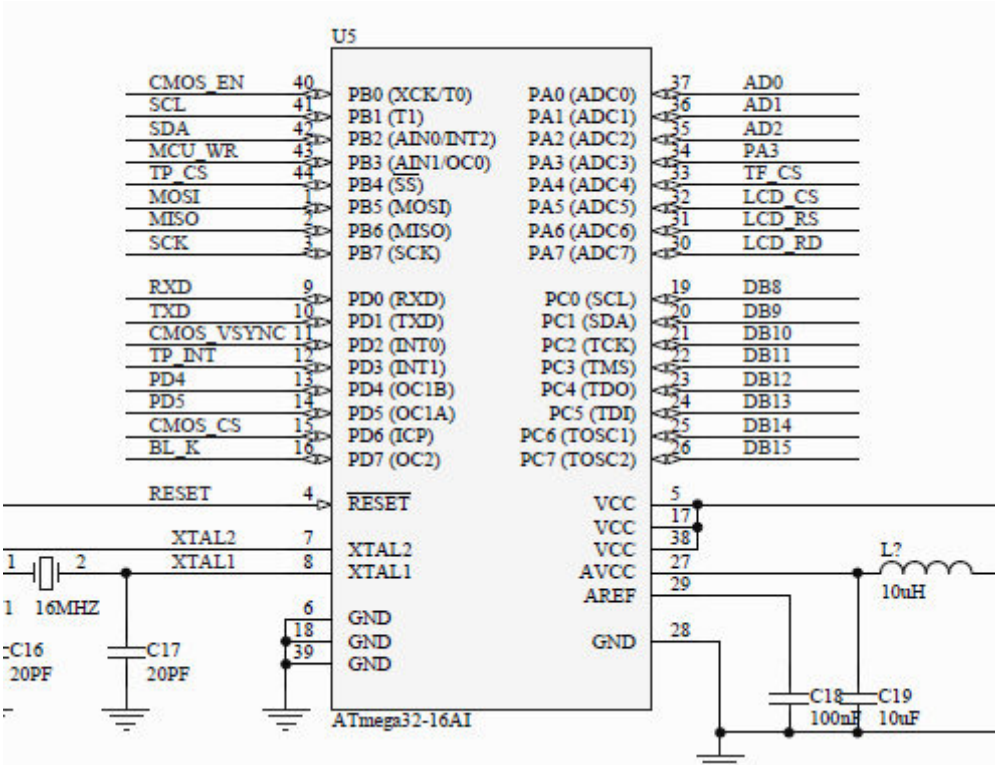


# 2. HW components AOR

## *2.1. ATmega32*

The RISC is used for communication with the ili9325 TFT display and the CMOS camera controller OV7670.

**Port mapping µC:**



## 2.2. Camera CMOS OV7670

The camera is part of a complete developed and already populated PCB which is available for example at www.elechouse.com. The camera chip is from OmniVision Technoligies, specification is available at www.ovt.com.

## 2.3. TFT ili9325

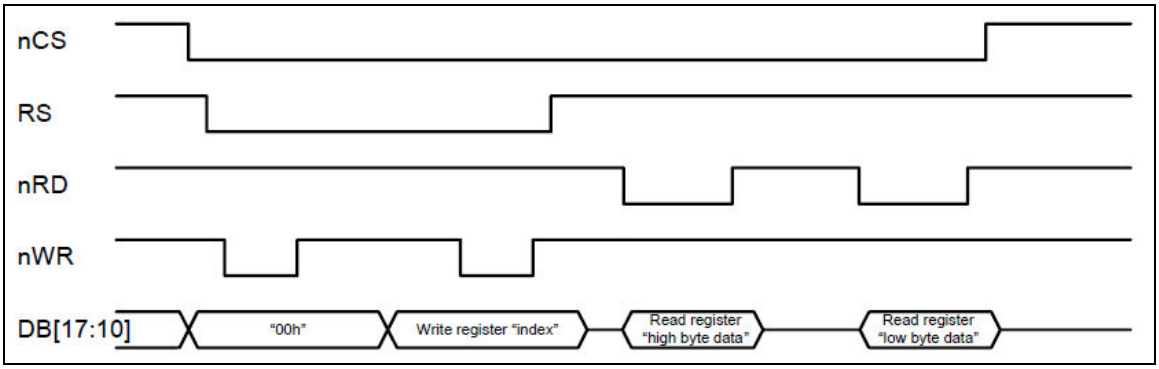
The TFT display is also part of the populated PCB. The display comes from Ilitek and the specification is available at www.ilitek.com.

# 3. SW functions

The SW package which was delivered with the populated PCB is available at www.elechouse.com. Following code examples are programmed in addition.

## *3.1. Get RGB Pixel*

First challenge was to read in the RGB Pixel out of the GRAM. The sequence for the control lines are given in the ili9325 datasheet for reading out the GRAM data.




Before the data will be set on to the data bus and the µC pins at PORTC, the GRAM register has to be addressed as followed.

```
LCD_WR_REG8(0);
LCD_WR_REG8(LCD_GRAM_DATA_REG);
```

The two bytes will be read in over the µC at PORTC, all pins must be set and defined as inputs. The valid RGB data can be taken out of the port register after two dummy reads.

```
void LCD_RD_DATA(unsigned char *data)
{
        DATA_INPUT();      // make µC port C to input
        LCD_RD_H();
        LCD_RS_H();
        LCD_CS_L();

        LCD_RD_L();
        _delay_us(1);
        data[0] = DATA_LCD_PIN;   // dummy read
        _delay_us(1);
        LCD_RD_H();

        LCD_RD_L();
        _delay_us(1);
        data[1] = DATA_LCD_PIN;   // dummy read
        _delay_us(1);
        LCD_RD_H();

        LCD_RD_L();
        _delay_us(1);
        data[1] = DATA_LCD_PIN;   // high byte
        _delay_us(1);
```

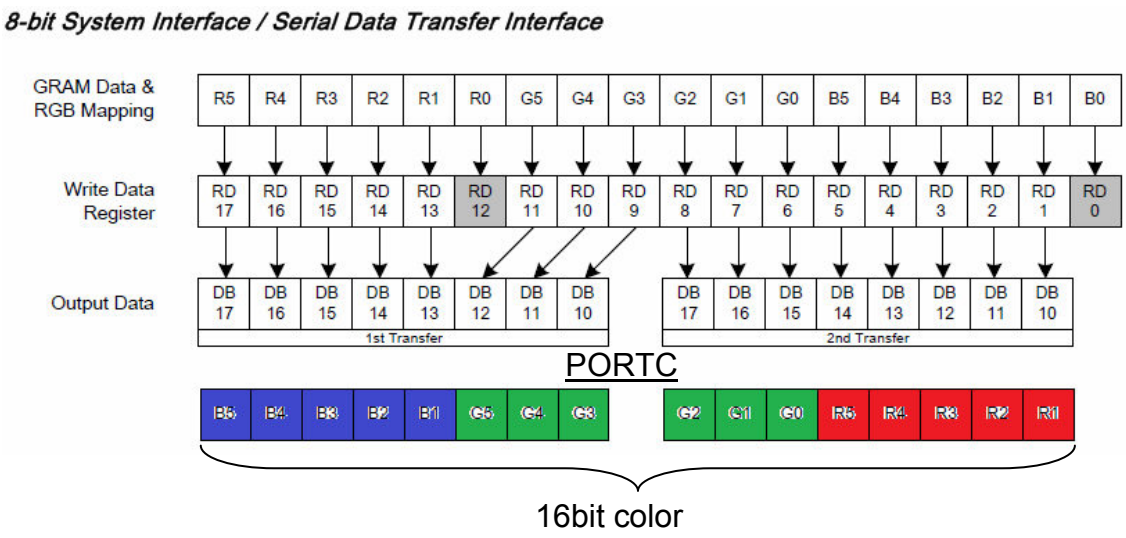
```
        LCD_RD_H();

        LCD_RD_L();
        _delay_us(1);
        data[0] = DATA_LCD_PIN;   // low byte
        _delay_us(1);
        LCD_RD_H();

        LCD_CS_H();
        DATA_OUTPUT();     // make µC Port C to output
}
```

For a better handling the two bytes were transformed to one 16bit variable.

```
color = data[1] << 8;
color |= data[0];
```

The sorting of RGB colors in `color` are different than described in the datasheet of ili9325 due to unknown reasons.

*8-bit System Interface / Serial Data Transfer Interface*




## 3.2. Convert RGB color pixel to gray

To check the ability of manipulating the pixels and the displayed image on the TFT screen one half of the RGB colored screen was transferred into grey.

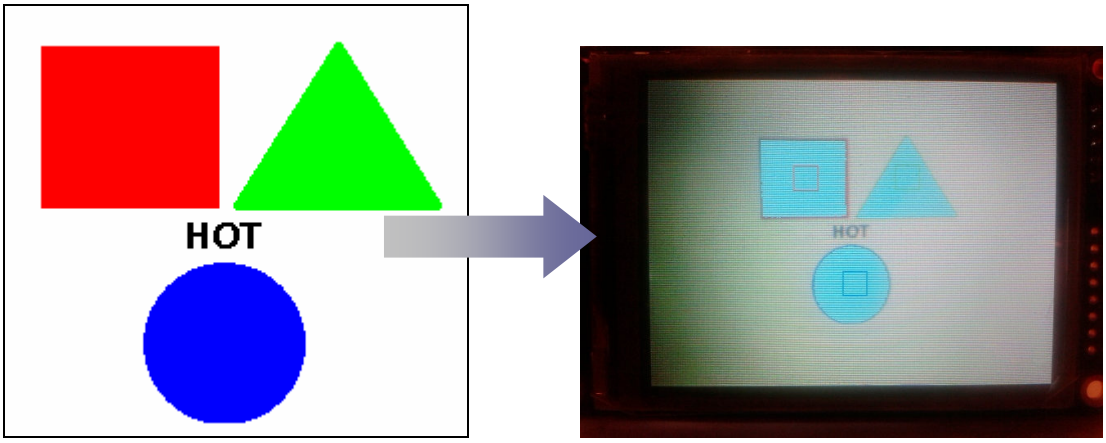
```
void image_processing(void)
{
        uint16 RGBpixel, color, colorR, colorG, colorB;
        unsigned short r, g, b;
        unsigned int RMove = 12, GMove = 7, BMove = 1;
        unsigned short GMask = 0x07E0;
        unsigned short BMask = 0x001F;
        unsigned int gray;

        RGBpixel = LCD_get_pixel(x,y);

        colorR = (RGBpixel & 0x1F) << 11;
        colorG = (((((RGBpixel & 0xFF00) >> 8) & 0x07) << 8)) | (RGBpixel & 0xE0);
```

```
        colorB = ((((RGBpixel & 0xFF00) >> 8) & 0xF8) >> 3);
        color = (colorR) | (colorG) | (colorB);

        // make grayscale
        r = color >> RMove;
        g = (GMask & color) >> GMove;
        b = (BMask & color) >> BMove;
        gray = (r * 3 + g * 6 + b) / 10;
        color = gray << 11 | gray << 6 | gray;

        LCD_WR_REG16(0x0022);
        LCD_WR_DATA16(color);
}
```

## 3.3. Color recognition

For a simple color recogniton every RGB pixel of an image was seperated into the three colors red, green and blue. With defined values and fixed thresholds the different colors can be filtered.

In a test application every pixel which was recognized within the range of the filter was colored in cyan. In addition the middle of the object was roughly calculated and marked with a box.



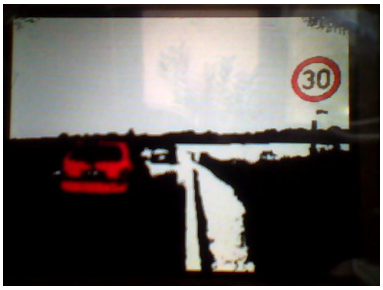

```
void image_processing(void)
{
    uint32 x = 0, y = 0;
    uint16 RGBpixel, color, colorR, colorG, colorB, reCnt, grCnt, blCnt, rCnt, gCnt, rLine,
    gLine, rStartx, rStarty, rEndx, rEndy, gStartx, gStarty, gEndx, gEndy, bStartx, bStarty,
    bEndx, bEndy;
    uint8 R,G,B, red, green, blue;
    uint8 value[2] = {1,1};
    unsigned short t, r, g, b;
    unsigned int RMove = 12, GMove = 7, BMove = 1;
    unsigned short GMask = 0x07E0;
    unsigned short BMask = 0x001F;
    unsigned int gray;

    red = green = blue = rCnt = gCnt = reCnt = grCnt = blCnt = rLine = gLine = rStartx =
    rStarty = rEndx = rEndy = rEndySum = rEndxSum = gStartx = gStarty = gEndx = gEndy =
    bStartx = bStarty = bEndx = bEndy = 0;

        // pixel manipulation

    for(x=0; x<240; x++)
    {
            for(y=0; y<320; y++)
            {
                    RGBpixel = LCD_get_pixel(x,y);

                    R = (((RGBpixel & 0x1F) & 0x1F) << 3);
```

```
                        G = (((((RGBpixel & 0xFF00) >> 8) & 0x07) << 5)) | ((((RGBpixel & 0x1F)
                        & 0xE0) >> 3));
                        B = ((RGBpixel & 0xFF00) >> 8) & 0xF8;

                        // Red
                        if(((R >= 0xA0) && (R <= 0xE0)) && ((G >= 0x30) && (G <= 0x70)) && ((B
                        >= 0x30) && (B <= 0x50)))
                        {
                                if(!reCnt)
                                {
                                        rStartx = x;
                                        rStarty = y;
                                }

                                reCnt++;
                                rEndx = x;
                                rEndy = y;
                                LCD_set_pixel(CYAN);
                        }

                        // Green
                        if(((R >= 0x55) && (R <= 0x80)) && ((G >= 0xBE) && (G <= 0xE7)) && ((B
                        >= 0x55) && (B <= 0x80)))
                        {
                                if(!grCnt)
                                {
                                        gStartx = x;
                                        gStarty = y;
                                }

                                grCnt++;
                                gEndx = x;
                                gEndy = y;
                                LCD_set_pixel(CYAN);
                        }

                        //Blue
                        if(((R >= 0x30) && (R <= 0x50)) && ((G >= 0x70) && (G <= 0x90)) && ((B
                        >= 0xA0) && (B <= 0xC0)))
                        {
                                if(!blCnt)
                                        bStartx = x;
                                if(y > bEndy)
                                        bEndy = y;

                                blCnt++;
                                bEndx = x;
                                //bEndy = y;
                                LCD_set_pixel(CYAN);
                        }
                }
        }

        if(reCnt >= 20)
                LCD_write_Box(rStarty+(rEndy-rStarty)/2-10,rStartx+(rEndx-rStartx)/2-10,20,20,
                RED);

        if(grCnt >= 20)
                LCD_write_Box(gStarty-10,gStartx+(gEndx-gStartx)/2-10,20,20, GREEN);

        if(blCnt >= 20)
                LCD_write_Box(bEndy-((bEndx-bStartx)/2)+10,bStartx+(bEndx-bStartx)/2-10,20,20,
                BLUE);
}
```

## 3.4. Speed limit sign recognition

To recognise a red sign in the first step the color was separated from the rest of the image.

Original picture



Red colors intensified, other colors black or white

Edge filter, only in relevant areas