Lab-1 : Running ESP-S3-EYE Examples

Compilation and modification of color detection sample

2023/2024

PREVIEW

In this lab you will compile the color detection sample, play a bit with it and then modify it to take an image and show a modified version of the image.

GOAL

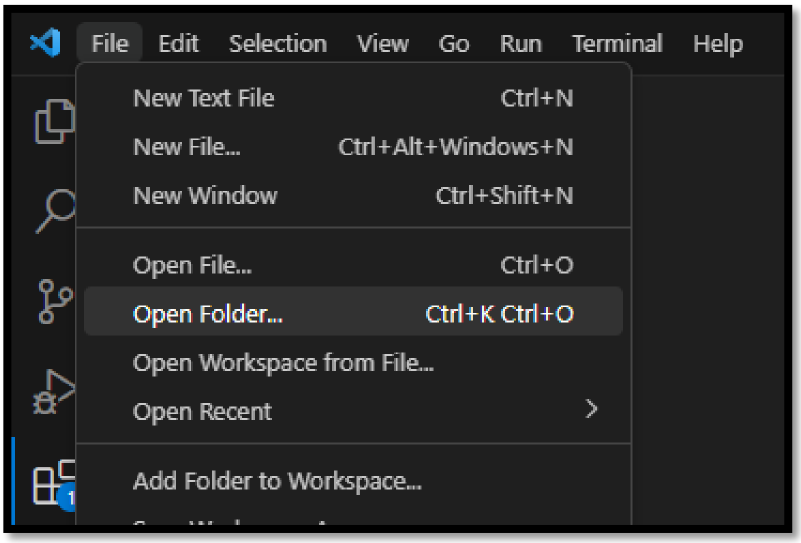
The goal of this labs is to get familiar with the ESP32-S3-EYE boards compilation and flashing process as well as the built in image format.

Reminder

Installing ESP-IDF and ESP-WHO

Opening the project

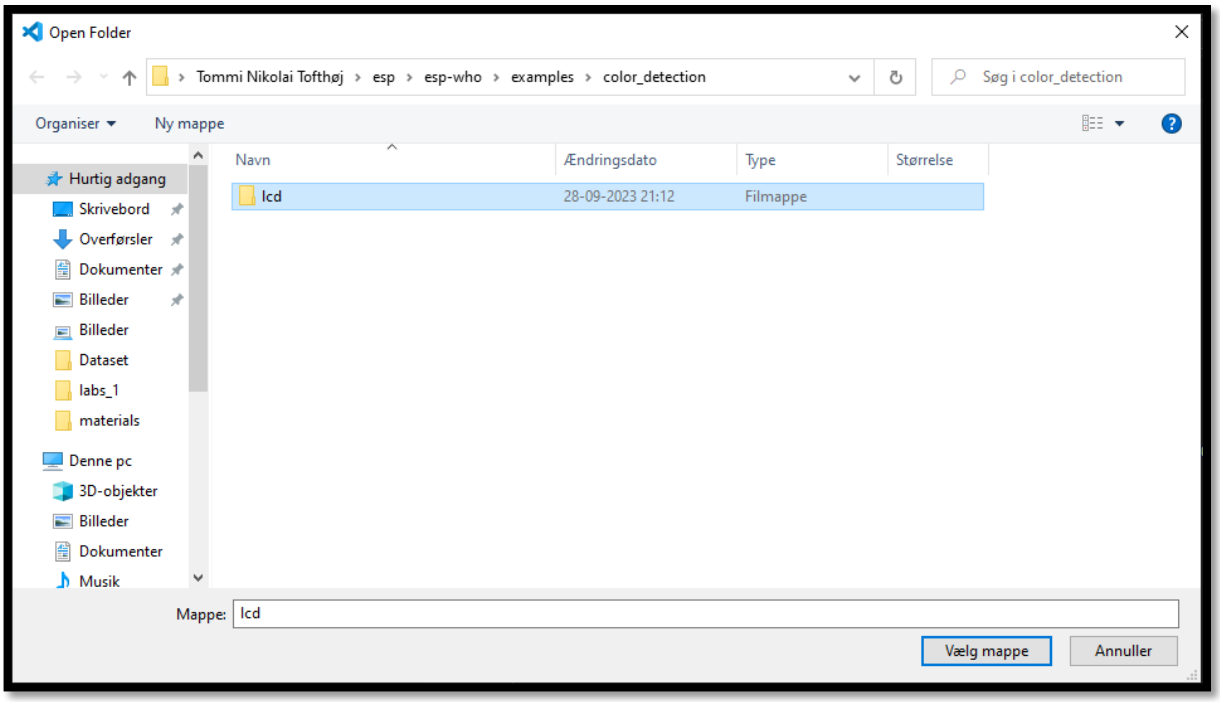
First open a new window of ***vscode***. Now select file and open folder.


File
Edit
Selection
View
Go
Run
Terminal
Help
New Text File Ctrl+N
New File... Ctrl+Alt+Windows+N
New Window Ctrl+Shift+N
Open File... Ctrl+O
Open Folder... Ctrl+K Ctrl+O
Open Workspace from File...
Open Recent
Add Folder to Workspace...

Navigate to the color detection sample usually found in:

C:\Users\[your username]\esp\esp-who\examples\color_detection

Open the ***LCD folder*** as the project folder.

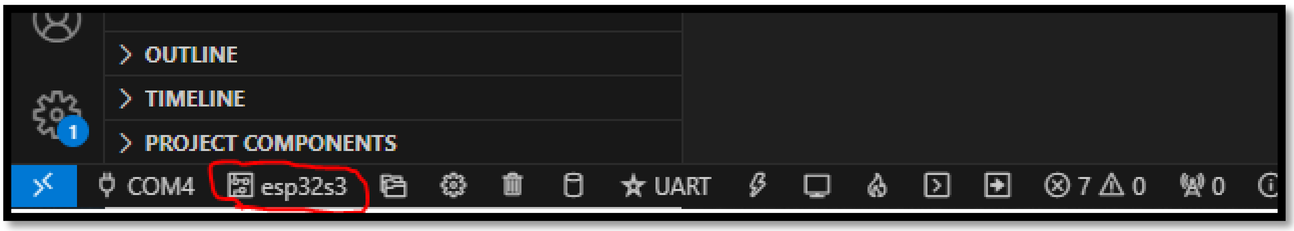


Navn	Ændringsdato	Type	Størrelse
lcd	28-09-2023 21:12	Filmappe	

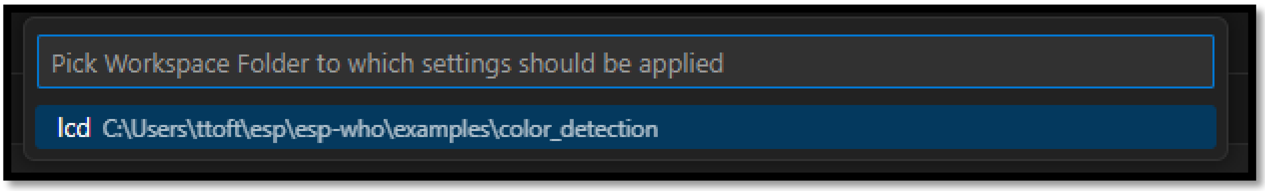
Mappe: lcd

Setting up and compiling project

First you need to set the board used. Press this icon on the bottom bar.



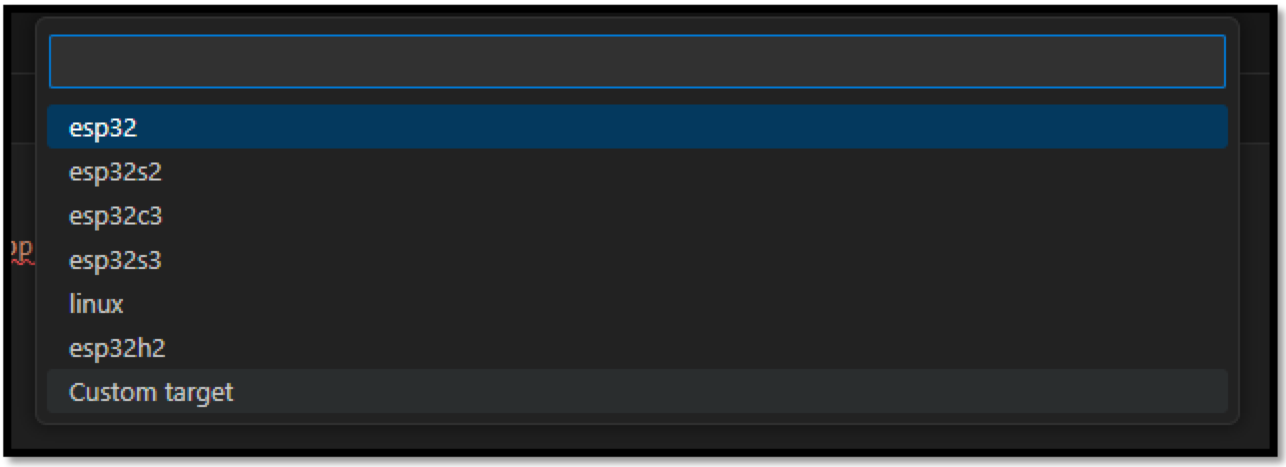
Select ***lcd***

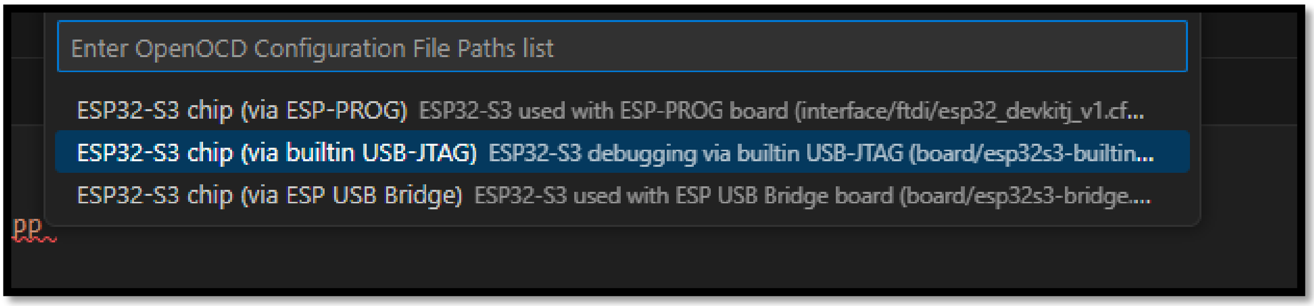


Pick Workspace Folder to which settings should be applied

lcd C:\Users\ttoft\esp\esp-who\examples\color_detection

Then select **esp32s3 → ESP32-S3** chip (via builtin USB-JTAG)





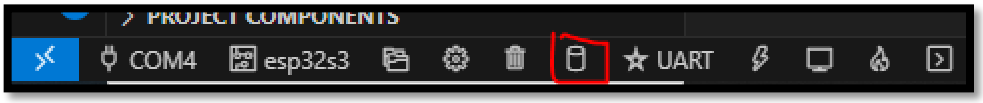
Enter OpenOCD Configuration File Paths list

ESP32-S3 chip (via ESP-PROG) ESP32-S3 used with ESP-PROG board (interface/ftdi/esp32_devkitj_v1.cf...

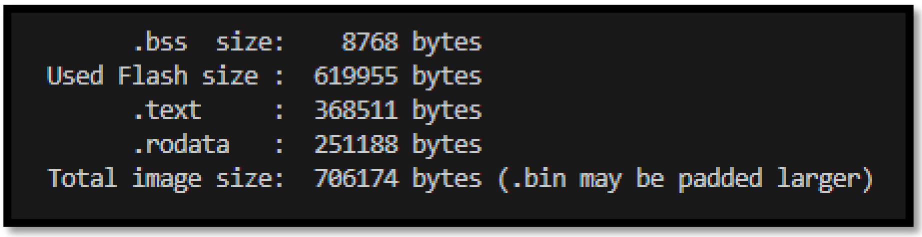
ESP32-S3 chip (via builtin USB-JTAG) ESP32-S3 debugging via builtin USB-JTAG (board/esp32s3-builtin...

ESP32-S3 chip (via ESP USB Bridge) ESP32-S3 used with ESP USB Bridge board (board/esp32s3-bridge....

Now you can compile the code using the **bin** shaped icon



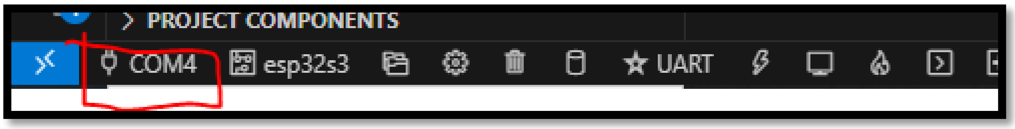
When the process is successful you should see this text



```
        .bss  size:    8768 bytes
 Used Flash size :  619955 bytes
        .text     :  368511 bytes
        .rodata   :  251188 bytes
 Total image size:  706174 bytes (.bin may be padded larger)
```

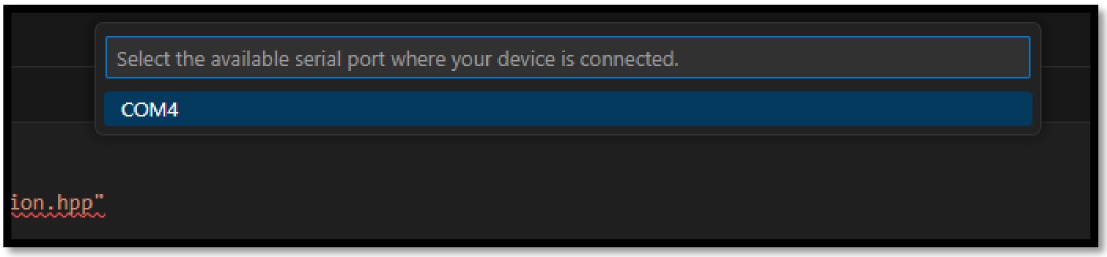
Setting com port and flashing board.

First you need to set the **COM** port. Press the first icon saying **COM** on the bar.



If you have connected the **ESP32** board a list of **COM** ports should show up (if you have multiple COM ports showing unplug the board and plug it in again to identify the correct port).

Select the correct comport (in my case COM 4).

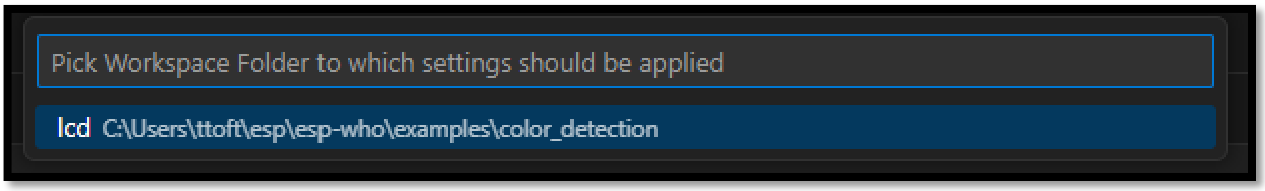


Select the available serial port where your device is connected.

COM4

ion.hpp"

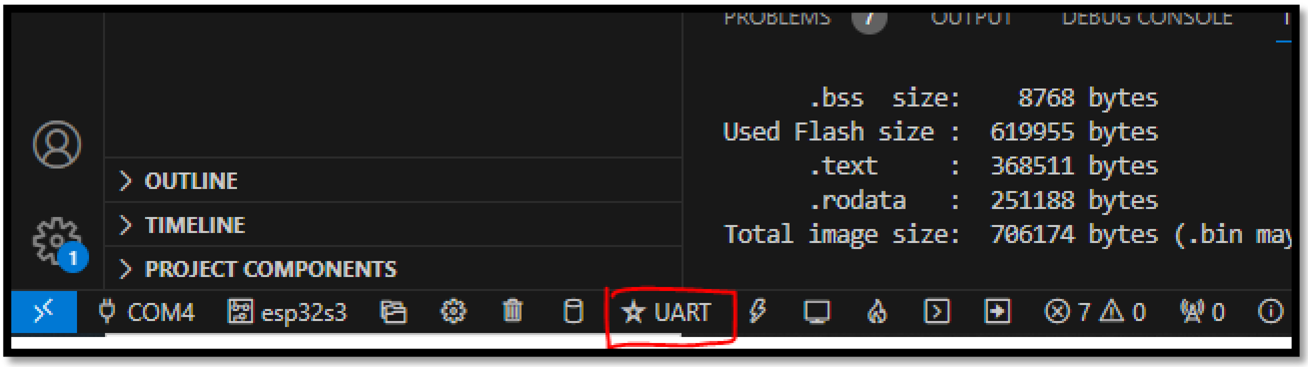
Then select the lcd workspace like earlier.



Pick Workspace Folder to which settings should be applied

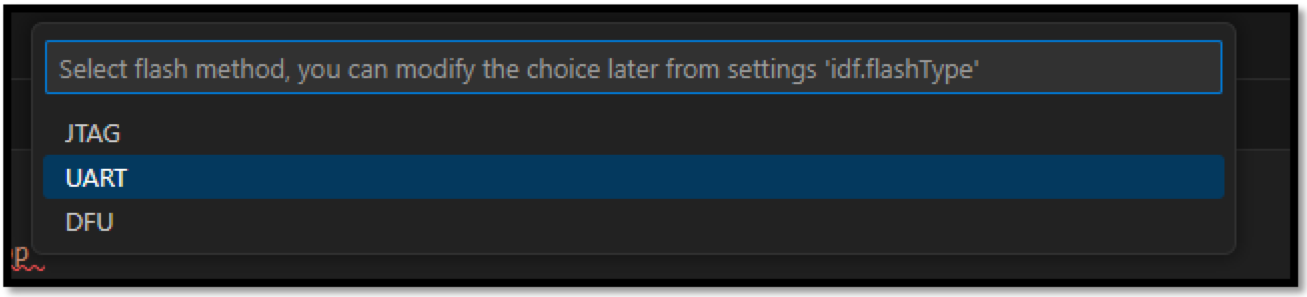
lcd C:\Users\ttoft\esp\esp-who\examples\color_detection

Now click the **star** to select the flashing method.



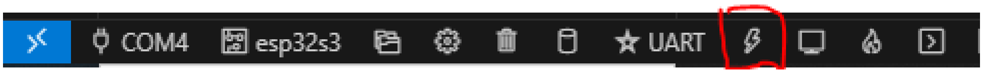
```
      .bss  size:    8768 bytes
Used Flash size :  619955 bytes
      .text     :  368511 bytes
      .rodata   :  251188 bytes
Total image size:  706174 bytes (.bin may
```

Select **UART**

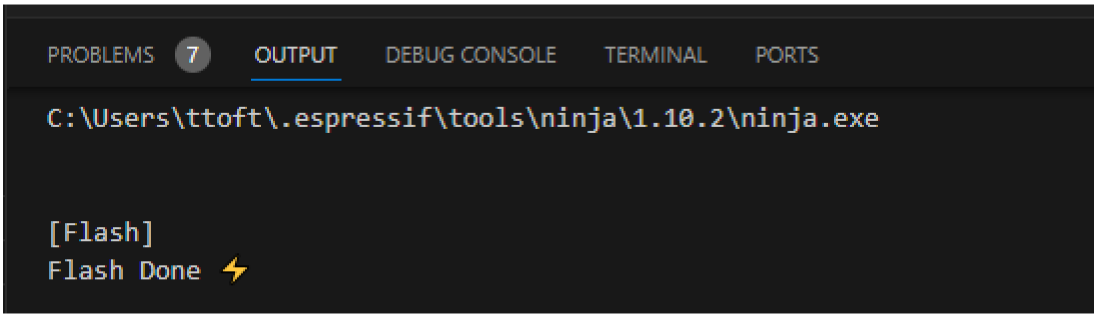


Select flash method, you can modify the choice later from settings 'idf.flashType'

Now click the **lightning icon** to flash the compiled code onto the board.



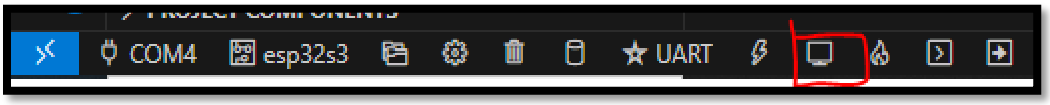
It should show the process of erasing and writing to the flash storage on the board and then show this when it's finished.



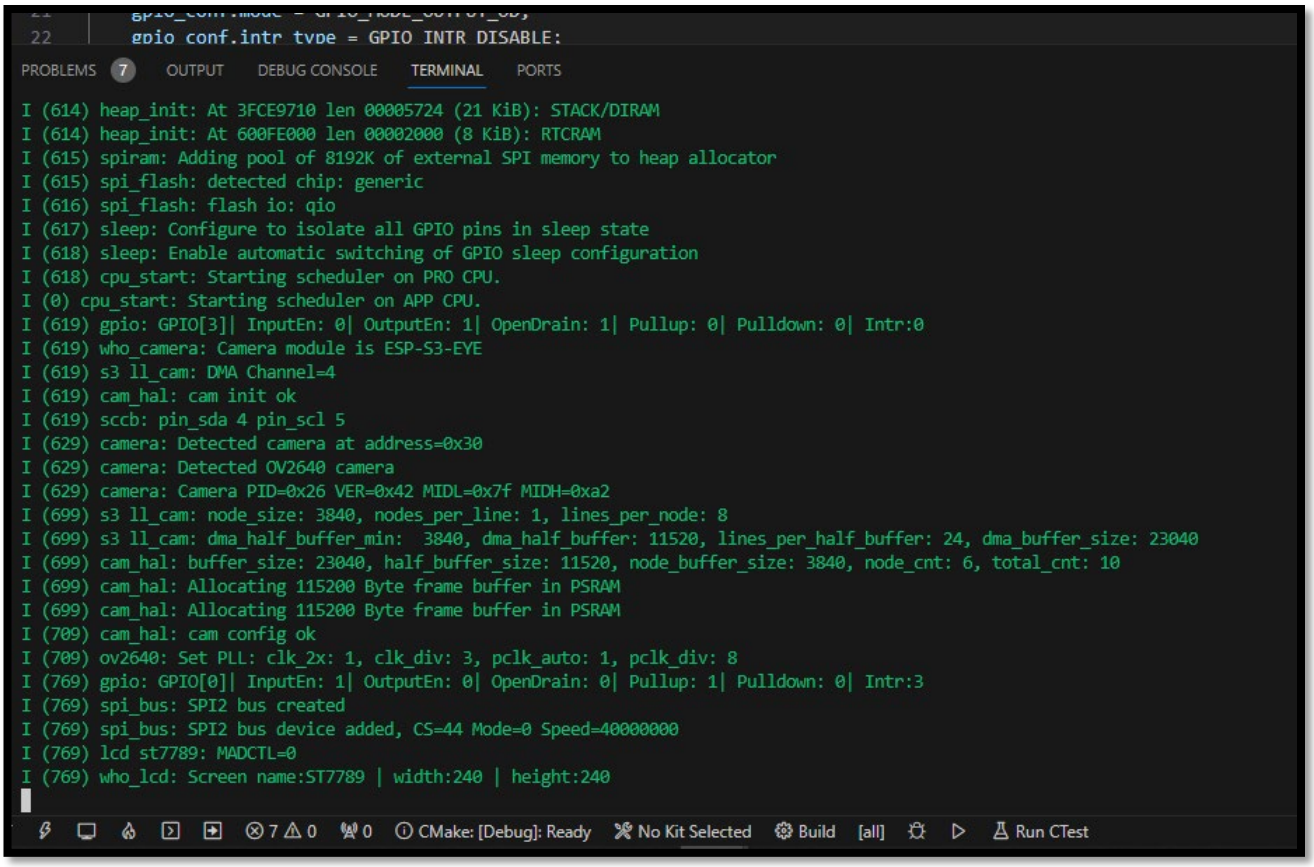
```
C:\Users\ttoft\.espressif\tools\ninja\1.10.2\ninja.exe


[Flash]
Flash Done ⚡
```

Monitoring the application

The ESP features is a very good logging system. This is basically all sent over serial communication and can be read with different serial communication programs. However the IDF has a built in monitoring function for this. Click the monitor icon on the toolbar.



Now the board reboots and you can see the logs in the **Terminal**



```
gpio conf.intr type = GPIO INTR DISABLE:
```

```
I (614) heap_init: At 3FCE9710 len 00005724 (21 KiB): STACK/DIRAM
I (614) heap_init: At 600FE000 len 00002000 (8 KiB): RTCRAM
I (615) spiram: Adding pool of 8192K of external SPI memory to heap allocator
I (615) spi_flash: detected chip: generic
I (616) spi_flash: flash io: qio
I (617) sleep: Configure to isolate all GPIO pins in sleep state
I (618) sleep: Enable automatic switching of GPIO sleep configuration
I (618) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
I (619) gpio: GPIO[3]| InputEn: 0| OutputEn: 1| OpenDrain: 1| Pullup: 0| Pulldown: 0| Intr:0
I (619) who_camera: Camera module is ESP-S3-EYE
I (619) s3 ll_cam: DMA Channel=4
I (619) cam_hal: cam init ok
I (619) sccb: pin_sda 4 pin_scl 5
I (629) camera: Detected camera at address=0x30
I (629) camera: Detected OV2640 camera
I (629) camera: Camera PID=0x26 VER=0x42 MIDL=0x7f MIDH=0xa2
I (699) s3 ll_cam: node_size: 3840, nodes_per_line: 1, lines_per_node: 8
I (699) s3 ll_cam: dma_half_buffer_min:  3840, dma_half_buffer: 11520, lines_per_half_buffer: 24, dma_buffer_size: 23040
I (699) cam_hal: buffer_size: 23040, half_buffer_size: 11520, node_buffer_size: 3840, node_cnt: 6, total_cnt: 10
I (699) cam_hal: Allocating 115200 Byte frame buffer in PSRAM
I (699) cam_hal: Allocating 115200 Byte frame buffer in PSRAM
I (709) cam_hal: cam config ok
I (709) ov2640: Set PLL: clk_2x: 1, clk_div: 3, pclk_auto: 1, pclk_div: 8
I (769) gpio: GPIO[0]| InputEn: 1| OutputEn: 0| OpenDrain: 0| Pullup: 1| Pulldown: 0| Intr:3
I (769) spi_bus: SPI2 bus created
I (769) spi_bus: SPI2 bus device added, CS=44 Mode=0 Speed=40000000
I (769) lcd st7789: MADCTL=0
I (769) who_lcd: Screen name:ST7789 | width:240 | height:240
```

If you click the flame icon to the right of the monitor icon it will automatically build, flash and then monitor the application combining all three steps.

Using the color detection sample

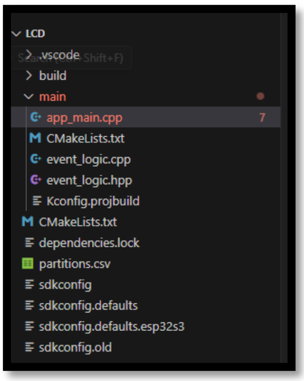
This sample tries to find large blobs of the same color and you can modify the size of the blobs. The full documentation can be found here.

https://github.com/espressif/esp-who/tree/master/examples/color_detection

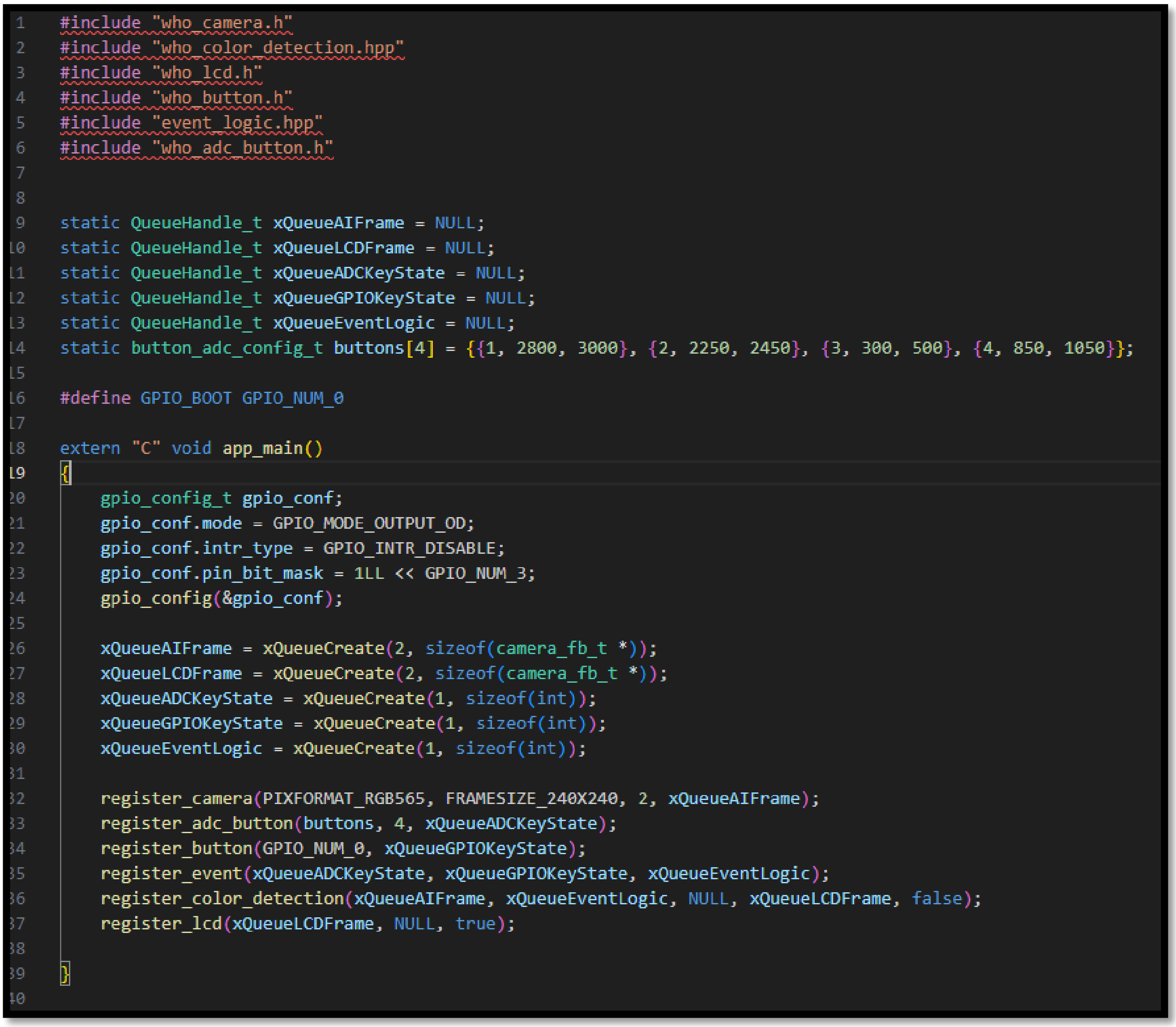
Now you can explore the functionality for a while.

Modifying the color detection sample

Open the ***app_main.cpp*** file in the main folder



```
LCD
  .vscode
  build
  main
    app_main.cpp        7
    CMakeLists.txt
    event_logic.cpp
    event_logic.hpp
    Kconfig.projbuild
  CMakeLists.txt
  dependencies.lock
  partitions.csv
  sdkconfig
  sdkconfig.defaults
  sdkconfig.defaults.esp32s3
  sdkconfig.old
```



```cpp
#include "who_camera.h"
#include "who_color_detection.hpp"
#include "who_lcd.h"
#include "who_button.h"
#include "event_logic.hpp"
#include "who_adc_button.h"


static QueueHandle_t xQueueAIFrame = NULL;
static QueueHandle_t xQueueLCDFrame = NULL;
static QueueHandle_t xQueueADCKeyState = NULL;
static QueueHandle_t xQueueGPIOKeyState = NULL;
static QueueHandle_t xQueueEventLogic = NULL;
static button_adc_config_t buttons[4] = {{1, 2800, 3000}, {2, 2250, 2450}, {3, 300, 500}, {4, 850, 1050}};

#define GPIO_BOOT GPIO_NUM_0

extern "C" void app_main()
{
    gpio_config_t gpio_conf;
    gpio_conf.mode = GPIO_MODE_OUTPUT_OD;
    gpio_conf.intr_type = GPIO_INTR_DISABLE;
    gpio_conf.pin_bit_mask = 1LL << GPIO_NUM_3;
    gpio_config(&gpio_conf);

    xQueueAIFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueLCDFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueADCKeyState = xQueueCreate(1, sizeof(int));
    xQueueGPIOKeyState = xQueueCreate(1, sizeof(int));
    xQueueEventLogic = xQueueCreate(1, sizeof(int));

    register_camera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2, xQueueAIFrame);
    register_adc_button(buttons, 4, xQueueADCKeyState);
    register_button(GPIO_NUM_0, xQueueGPIOKeyState);
    register_event(xQueueADCKeyState, xQueueGPIOKeyState, xQueueEventLogic);
    register_color_detection(xQueueAIFrame, xQueueEventLogic, NULL, xQueueLCDFrame, false);
    register_lcd(xQueueLCDFrame, NULL, true);

}

```

This code sets up some things and then it uses some functions to start 5 main processes.

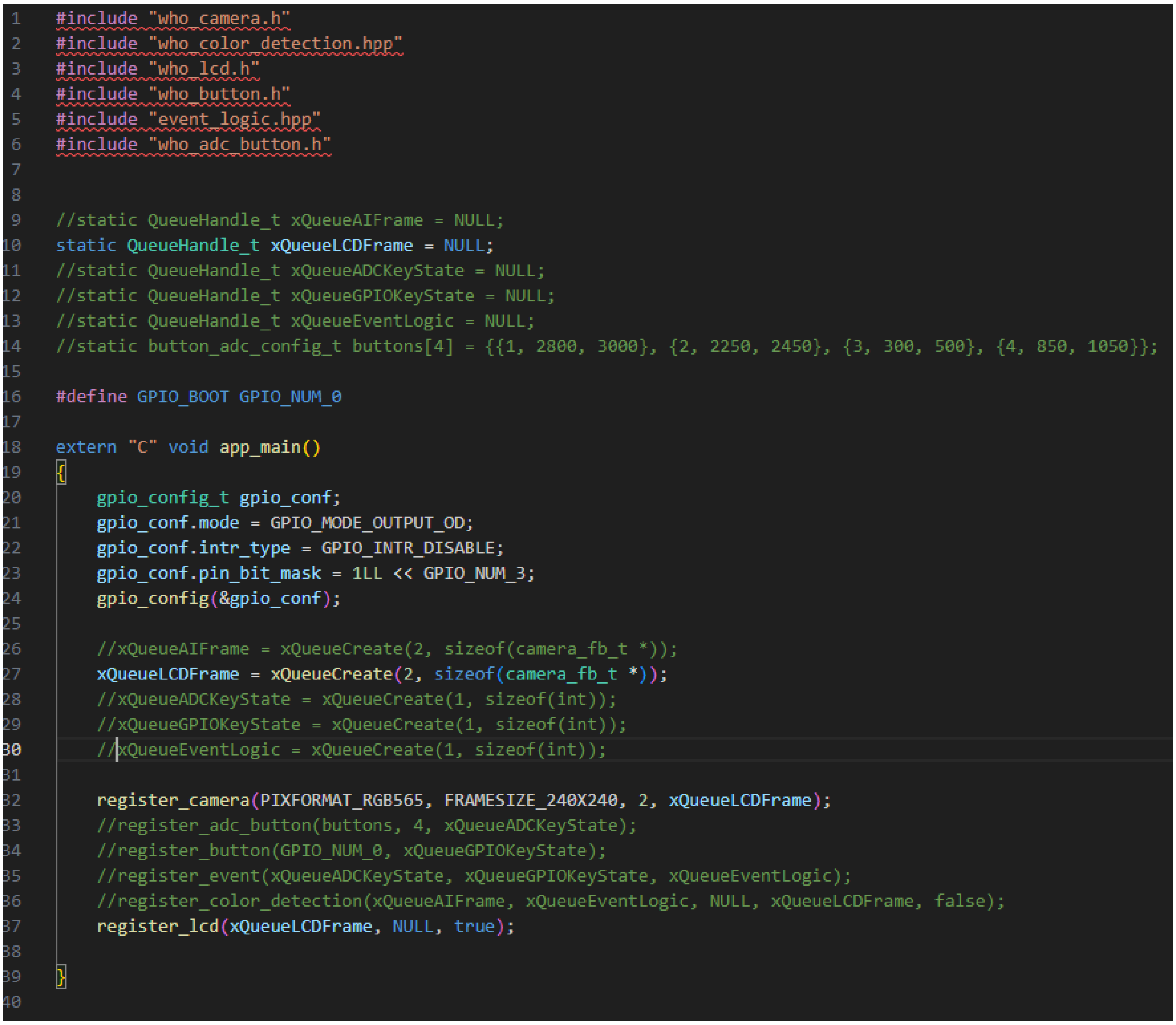
register_camera: starts the camera process and gives it a **Freertos** queue to output the framebuffer pointer to the next three functions register the buttons and setup event logic for the buttons.

register_color_detection: starts the color detection process which also draws the boxes on the frame which is then fed to another **Freertos** queue.

register_lcd: sets up the lcd and shows the frame.

For our modified sample we will mainly focus on the camera and lcd part.

The first step is the feed the framebuffer pointer directly to the lcd from the camera. We do this by setting **xQueueAIFrame** to be **xQueueLCDFrame** instead. We can also comment out the **register_color_detection** and button functions now.



```
#include "who_camera.h"
#include "who_color_detection.hpp"
#include "who_lcd.h"
#include "who_button.h"
#include "event_logic.hpp"
#include "who_adc_button.h"


//static QueueHandle_t xQueueAIFrame = NULL;
static QueueHandle_t xQueueLCDFrame = NULL;
//static QueueHandle_t xQueueADCKeyState = NULL;
//static QueueHandle_t xQueueGPIOKeyState = NULL;
//static QueueHandle_t xQueueEventLogic = NULL;
//static button_adc_config_t buttons[4] = {{1, 2800, 3000}, {2, 2250, 2450}, {3, 300, 500}, {4, 850, 1050}};

#define GPIO_BOOT GPIO_NUM_0

extern "C" void app_main()
{
    gpio_config_t gpio_conf;
    gpio_conf.mode = GPIO_MODE_OUTPUT_OD;
    gpio_conf.intr_type = GPIO_INTR_DISABLE;
    gpio_conf.pin_bit_mask = 1LL << GPIO_NUM_3;
    gpio_config(&gpio_conf);

    //xQueueAIFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    xQueueLCDFrame = xQueueCreate(2, sizeof(camera_fb_t *));
    //xQueueADCKeyState = xQueueCreate(1, sizeof(int));
    //xQueueGPIOKeyState = xQueueCreate(1, sizeof(int));
    //xQueueEventLogic = xQueueCreate(1, sizeof(int));

    register_camera(PIXFORMAT_RGB565, FRAMESIZE_240X240, 2, xQueueLCDFrame);
    //register_adc_button(buttons, 4, xQueueADCKeyState);
    //register_button(GPIO_NUM_0, xQueueGPIOKeyState);
    //register_event(xQueueADCKeyState, xQueueGPIOKeyState, xQueueEventLogic);
    //register_color_detection(xQueueAIFrame, xQueueEventLogic, NULL, xQueueLCDFrame, false);
    register_lcd(xQueueLCDFrame, NULL, true);

}
```

Now you can build flash and monitor and you should see that the image is just shown directly on the screen.

For the next step we need to create a new module. Go into:

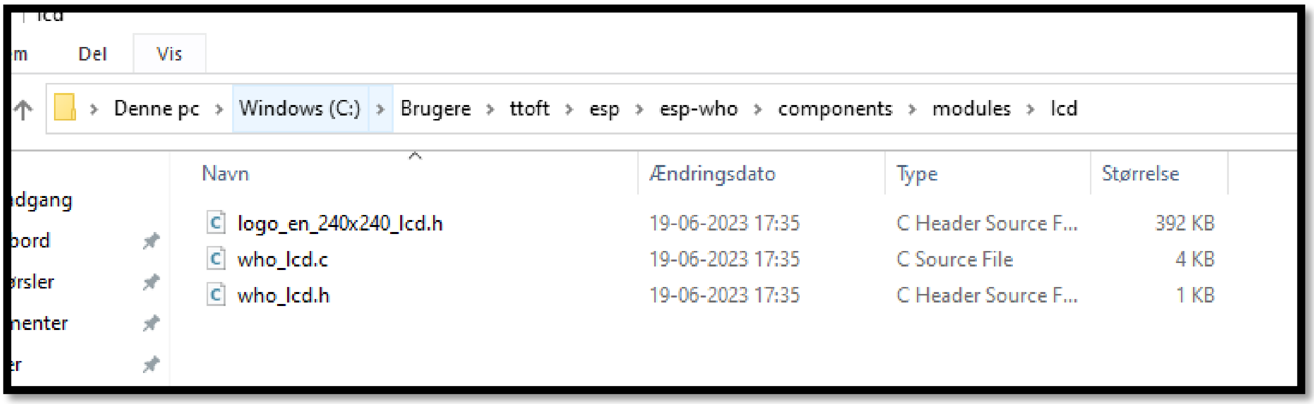
C:\Users\[your username]\esp\esp-who\components\modules

Make a new folder called **custom_lcd**

Copy **who_lcd.h** and **who_lcd.c** into our the new folder and rename them to **custom_lcd.h** and **custom_lcd.c**

The files are usually found at:

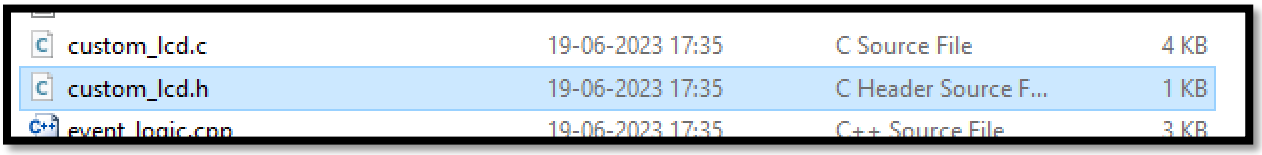
C:\Users\[your username]\esp\esp-who\components\modules\lcd



Denne pc > Windows (C:) > Brugere > ttoft > esp > esp-who > components > modules > lcd

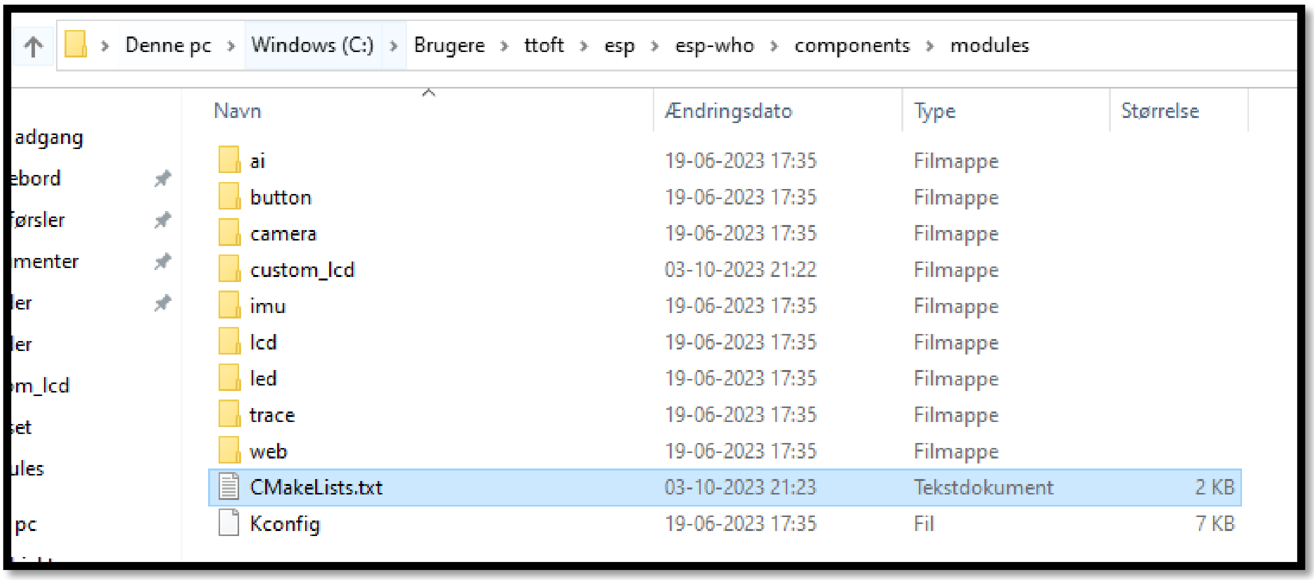
Navn	Ændringsdato	Type	Størrelse
logo_en_240x240_lcd.h	19-06-2023 17:35	C Header Source F...	392 KB
who_lcd.c	19-06-2023 17:35	C Source File	4 KB
who_lcd.h	19-06-2023 17:35	C Header Source F...	1 KB

Rename them to custom_lcd.c and custom_lcd.h



custom_lcd.c	19-06-2023 17:35	C Source File	4 KB
custom_lcd.h	19-06-2023 17:35	C Header Source F...	1 KB
event_logic.cpp	19-06-2023 17:35	C++ Source File	3 KB

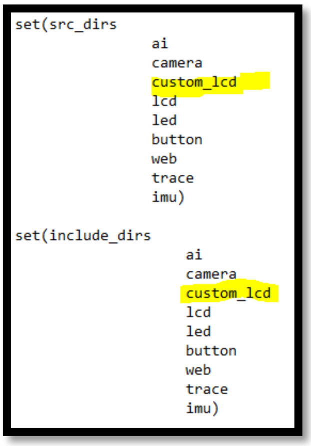
Now in the modules folder we need to edit the **CMakeList.txt**



Denne pc > Windows (C:) > Brugere > ttoft > esp > esp-who > components > modules
adgang
ebord
førsler
menter
er
er
m_lcd
et
ules
pc

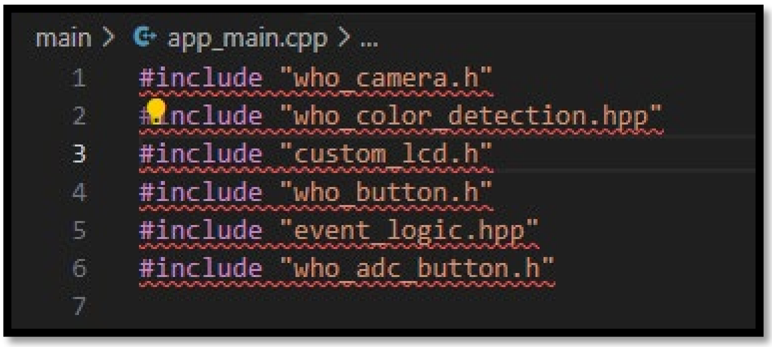
Navn	Ændringsdato	Type	Størrelse
ai	19-06-2023 17:35	Filmappe	
button	19-06-2023 17:35	Filmappe	
camera	19-06-2023 17:35	Filmappe	
custom_lcd	03-10-2023 21:22	Filmappe	
imu	19-06-2023 17:35	Filmappe	
lcd	19-06-2023 17:35	Filmappe	
led	19-06-2023 17:35	Filmappe	
trace	19-06-2023 17:35	Filmappe	
web	19-06-2023 17:35	Filmappe	
CMakeLists.txt	03-10-2023 21:23	Tekstdokument	2 KB
Kconfig	19-06-2023 17:35	Fil	7 KB

We need to add the two lines



```
set(src_dirs
                ai
                camera
                custom_lcd
                lcd
                led
                button
                web
                trace
                imu)

set(include_dirs
                    ai
                    camera
                    custom_lcd
                    lcd
                    led
                    button
                    web
                    trace
                    imu)
```

Now replace the #include “who_lcd.h” line with #include “custom_lcd.h” in app_main.cpp



```
#include "who_camera.h"
#include "who_color_detection.hpp"
#include "custom_lcd.h"
#include "who_button.h"
#include "event_logic.hpp"
#include "who_adc_button.h"
```

Now you should still be able to build, flash and monitor without any issues.

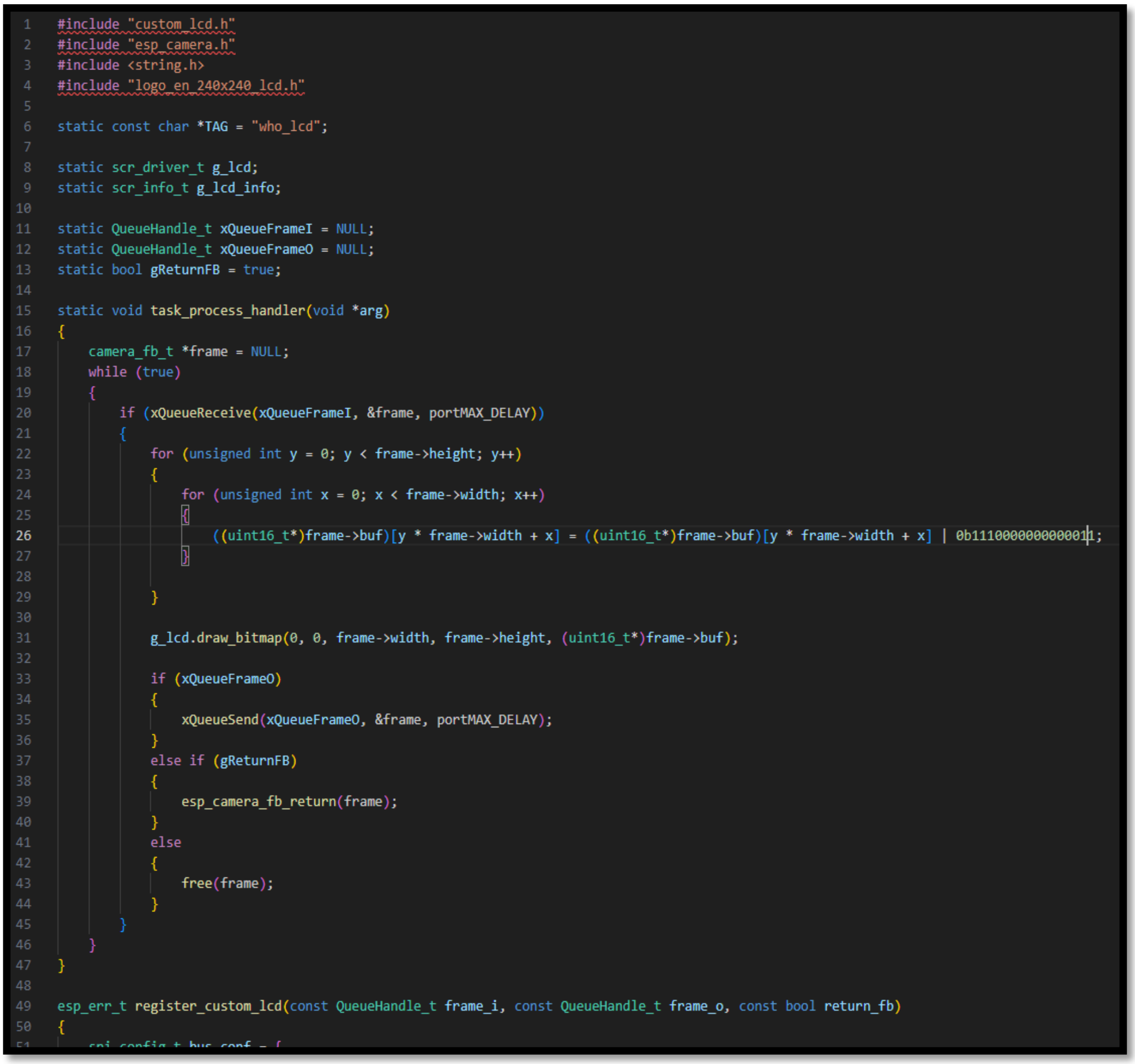
For the next part you can have a look around in custom_lcd.c, try to understand the different functions.

Now your task is to modify the image in some way before showing it on the LCD. The image format is RGB565 this means that the images have the following bit order from most significant to least significant:

G2 G1 G0 R4 R3 R2 R1 R0 B4 B3 B2 B1 B0 G5 G4 G3

It is important to rename the function **register_lcd** to **register_custom_lcd** both in the **.c** and the **.h** and the **app_main.cpp** file.

On the next page you will see sample code to modify all of the pixels to have at least 5/6 of the max value for green.



```c
#include "custom_lcd.h"
#include "esp_camera.h"
#include <string.h>
#include "logo_en_240x240_lcd.h"

static const char *TAG = "who_lcd";

static scr_driver_t g_lcd;
static scr_info_t g_lcd_info;

static QueueHandle_t xQueueFrameI = NULL;
static QueueHandle_t xQueueFrameO = NULL;
static bool gReturnFB = true;

static void task_process_handler(void *arg)
{
    camera_fb_t *frame = NULL;
    while (true)
    {
        if (xQueueReceive(xQueueFrameI, &frame, portMAX_DELAY))
        {
            for (unsigned int y = 0; y < frame->height; y++)
            {
                for (unsigned int x = 0; x < frame->width; x++)
                {
                    ((uint16_t*)frame->buf)[y * frame->width + x] = ((uint16_t*)frame->buf)[y * frame->width + x] | 0b1110000000000011;
                }

            }

            g_lcd.draw_bitmap(0, 0, frame->width, frame->height, (uint16_t*)frame->buf);

            if (xQueueFrameO)
            {
                xQueueSend(xQueueFrameO, &frame, portMAX_DELAY);
            }
            else if (gReturnFB)
            {
                esp_camera_fb_return(frame);
            }
            else
            {
                free(frame);
            }
        }
    }
}

esp_err_t register_custom_lcd(const QueueHandle_t frame_i, const QueueHandle_t frame_o, const bool return_fb)
{
    spi_config_t bus_conf = {
```