LCD

ESP LCD ILI9341 Component Registry

Implementation of the ILI9341 LCD controller with esp_lcd component.

LCD controller Communication interface Component name Link to datasheet ILI9341 SPI esp_lcd_ili9341 Specification Add to project Packages from this repository are uploaded to Espressif's component service. You can add them to your project via idf.py add-dependency, e.g.

Bash

idf.py add-dependency "espressif/esp_lcd_ili9341^2.0.0" Alternatively, you can create idf_component.yml. More is in Espressif's documentation.

Example use C

```
ESP_LOGI(TAG, "Initialize SPI bus");
const spi_bus_config_t bus_config =
ILI9341_PANEL_BUS_SPI_CONFIG(EXAMPLE_PIN_NUM_LCD_PCLK, EXAMPLE_PIN_NUM_LCD_MOSI,
                                                                EXAMPLE_LCD_H_RES
* 80 * sizeof(uint16_t));
ESP_ERROR_CHECK(spi_bus_initialize(EXAMPLE_LCD_HOST, &bus_config,
SPI_DMA_CH_AUTO));

ESP_LOGI(TAG, "Install panel IO");
esp_lcd_panel_io_handle_t io_handle = NULL;
const esp_lcd_panel_io_spi_config_t io_config =
ILI9341_PANEL_IO_SPI_CONFIG(EXAMPLE_PIN_NUM_LCD_CS, EXAMPLE_PIN_NUM_LCD_DC,

example_callback, &example_callback_ctx);
ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t)EXAMPLE_LCD_HOS
T, &io_config, &io_handle));
```

/**

- Uncomment these lines if use custom initialization commands.
- The array should be declared as "static const" and positioned outside the function. */ // static const ili9341_lcd_init_cmd_t lcd_init_cmds[] = { // // {cmd, { data }, data_size, delay_ms} // {0xCF, (uint8_t []) {0x00, 0xAA, 0XE0}, 3, 0}, // {0xED, (uint8_t []){0x67, 0x03, 0X12, 0X81}, 4, 0}, // {0xE8, (uint8_t []){0x8A, 0x01, 0x78}, 3, 0}, // ... // };

  ```
  ESP_LOGI(TAG, "Install ILI9341 panel driver");
  esp_lcd_panel_handle_t panel_handle = NULL;
  // ili9341_vendor_config_t vendor_config = {  // Uncomment these lines if
  use custom initialization commands
  ```

```
//     .init_cmds = lcd_init_cmds,
//     .init_cmds_size = sizeof(lcd_init_cmds) /
sizeof(ili9341_lcd_init_cmd_t),
// };
const esp_lcd_panel_dev_config_t panel_config = {
    .reset_gpio_num = EXAMPLE_PIN_NUM_LCD_RST,   // Set to -1 if not use
    .rgb_ele_order = LCD_RGB_ELEMENT_ORDER_RGB,  // RGB element order: R-G-B
    .bits_per_pixel = 16,                        // Implemented by LCD
command `3Ah` (16/18)
    // .vendor_config = &vendor_config,          // Uncomment this line if
use custom initialization commands
};
ESP_ERROR_CHECK(esp_lcd_new_panel_ili9341(io_handle, &panel_config,
&panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_reset(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_init(panel_handle));
ESP_ERROR_CHECK(esp_lcd_panel_disp_on_off(panel_handle, true));
```

There is an example in ESP-IDF with this LCD controller. Please follow this link.

Predefined init vendor commands This driver contains various predefined vendor initialization commands which in some cases can improve image quality. Usage is in example below.

Include selected commands:

C

#include "esp_lcd_ili9341_init_cmds_1.h" Use vendor config structure:

C

```
...

ili9341_vendor_config_t vendor_config = {
    .init_cmds = ili9341_lcd_init_vendor,
    .init_cmds_size = sizeof(ili9341_lcd_init_vendor) /
sizeof(ili9341_lcd_init_cmd_t),
};

const esp_lcd_panel_dev_config_t panel_config = {
    ...
    .vendor_config = &vendor_config,
};

...
```

LGVL

LVGL ESP Portation Component Registry maintenance-status

This component helps with using LVGL with Espressif's LCD and touch drivers. It can be used with any project with LCD display.

Features Initialization of the LVGL Create task and timer Handle rotating Power saving Add/remove display (using esp_lcd) Add/remove touch input (using esp_lcd_touch) Add/remove navigation buttons input (using button) Add/remove encoder input (using knob) Add/remove USB HID mouse/keyboard input (using usb_host_hid) LVGL Version This component supports LVGL8 and LVGL9. By default, it selects the latest LVGL version. If you want to use a specific version (e.g. latest LVGL8), you can easily define this requirement in idf_component.yml in your project like this:

Plaintext

lvgl/lvgl: version: "^8" public: true LVGL Version Compatibility This component is fully compatible with LVGL version 9. All types and functions are used from LVGL9. Some LVGL9 types are not supported in LVGL8 and there are retyped in esp_lvgl_port_compatibility.h header file. Please, be aware, that some draw and object functions are not compatible between LVGL8 and LVGL9.

Usage Initialization C

```
const lvgl_port_cfg_t lvgl_cfg = ESP_LVGL_PORT_INIT_CONFIG();
esp_err_t err = lvgl_port_init(&lvgl_cfg);
```

Add screen Add an LCD screen to the LVGL. It can be called multiple times for adding multiple LCD screens.

C

```
static lv_disp_t * disp_handle;

/* LCD IO */
esp_lcd_panel_io_handle_t io_handle = NULL;
ESP_ERROR_CHECK(esp_lcd_new_panel_io_spi((esp_lcd_spi_bus_handle_t) 1, &io_config,
&io_handle));

/* LCD driver initialization */
esp_lcd_panel_handle_t lcd_panel_handle;
ESP_ERROR_CHECK(esp_lcd_new_panel_st7789(io_handle, &panel_config,
&lcd_panel_handle));

/* Add LCD screen */
const lvgl_port_display_cfg_t disp_cfg = {
    .io_handle = io_handle,
    .panel_handle = lcd_panel_handle,
    .buffer_size = DISP_WIDTH*DISP_HEIGHT,
    .double_buffer = true,
    .hres = DISP_WIDTH,
    .vres = DISP_HEIGHT,
```

```
    .monochrome = false,
    .mipi_dsi = false,
    .color_format = LV_COLOR_FORMAT_RGB565,
    .rotation = {
        .swap_xy = false,
        .mirror_x = false,
        .mirror_y = false,
    },
    .flags = {
        .buff_dma = true,
        .swap_bytes = false,
    }
};
disp_handle = lvgl_port_add_disp(&disp_cfg);

/* ... the rest of the initialization ... */

/* If deinitializing LVGL port, remember to delete all displays: */
lvgl_port_remove_disp(disp_handle);
```

Note

For adding RGB or MIPI-DSI screen, use functions lvgl_port_add_disp_rgb or lvgl_port_add_disp_dsi. DMA buffer can be used only when you use color format LV_COLOR_FORMAT_RGB565. Add touch input Add touch input to the LVGL. It can be called more times for adding more touch inputs.

C

```
/* Touch driver initialization */
...
esp_lcd_touch_handle_t tp;
esp_err_t err = esp_lcd_touch_new_i2c_gt911(io_handle, &tp_cfg, &tp);

/* Add touch input (for selected screen) */
const lvgl_port_touch_cfg_t touch_cfg = {
    .disp = disp_handle,
    .handle = tp,
};
lv_indev_t* touch_handle = lvgl_port_add_touch(&touch_cfg);

/* ... the rest of the initialization ... */

/* If deinitializing LVGL port, remember to delete all touches: */
lvgl_port_remove_touch(touch_handle);
```

Note

If the screen has another resolution than the touch resolution, you can use scaling by add .scale.x or .scale.y into lvgl_port_touch_cfg_t configuration structure.

Add buttons input Add buttons input to the LVGL. It can be called more times for adding more buttons inputs for different displays. This feature is available only when the component espressif/button was added into the project.

C

```
/* Buttons configuration structure */
const button_gpio_config_t bsp_button_config[] = {
    {
        .gpio_num = GPIO_NUM_37,
        .active_level = 0,
    },
    {
        .gpio_num = GPIO_NUM_38,
        .active_level = 0,
    },
    {
        .gpio_num = GPIO_NUM_39,
        .active_level = 0,
    },
};


const button_config_t btn_cfg = {0};
button_handle_t prev_btn_handle = NULL;
button_handle_t next_btn_handle = NULL;
button_handle_t enter_btn_handle = NULL;
iot_button_new_gpio_device(&btn_cfg, &bsp_button_config[0], &prev_btn_handle);
iot_button_new_gpio_device(&btn_cfg, &bsp_button_config[1], &next_btn_handle);
iot_button_new_gpio_device(&btn_cfg, &bsp_button_config[2], &enter_btn_handle);

const lvgl_port_nav_btns_cfg_t btns = {
    .disp = disp_handle,
    .button_prev = prev_btn_handle,
    .button_next = next_btn_handle,
    .button_enter = enter_btn_handle
};

/* Add buttons input (for selected screen) */
lv_indev_t* buttons_handle = lvgl_port_add_navigation_buttons(&btns);

/* ... the rest of the initialization ... */

/* If deinitializing LVGL port, remember to delete all buttons: */
lvgl_port_remove_navigation_buttons(buttons_handle);
```

Note

When you use navigation buttons for control LVGL objects, these objects must be added to LVGL groups. See LVGL documentation for more info.

Add encoder input Add encoder input to the LVGL. It can be called more times for adding more encoder inputs for different displays. This feature is available only when the component espressif/knob was added into the project.

C

```
static const button_gpio_config_t encoder_btn_config = {
    .gpio_num = GPIO_BTN_PRESS,
    .active_level = 0,
};

const knob_config_t encoder_a_b_config = {
    .default_direction = 0,
    .gpio_encoder_a = GPIO_ENCODER_A,
    .gpio_encoder_b = GPIO_ENCODER_B,
};

const button_config_t btn_cfg = {0};
button_handle_t encoder_btn_handle = NULL;
BSP_ERROR_CHECK_RETURN_NULL(iot_button_new_gpio_device(&btn_cfg,
&encoder_btn_config, &encoder_btn_handle));

/* Encoder configuration structure */
const lvgl_port_encoder_cfg_t encoder = {
    .disp = disp_handle,
    .encoder_a_b = &encoder_a_b_config,
    .encoder_enter = encoder_btn_handle
};

/* Add encoder input (for selected screen) */
lv_indev_t* encoder_handle = lvgl_port_add_encoder(&encoder);

/* ... the rest of the initialization ... */

/* If deinitializing LVGL port, remember to delete all encoders: */
lvgl_port_remove_encoder(encoder_handle);
```

Note

When you use encoder for control LVGL objects, these objects must be added to LVGL groups. See LVGL documentation for more info.

Add USB HID keyboard and mouse input Add mouse and keyboard input to the LVGL. This feature is available only when the component usb_host_hid was added into the project.

C

```
/* USB initialization */
usb_host_config_t host_config = {
    .skip_phy_setup = false,
    .intr_flags = ESP_INTR_FLAG_LEVEL1,
};
ESP_ERROR_CHECK(usb_host_install(&host_config));

...

/* Add mouse input device */
const lvgl_port_hid_mouse_cfg_t mouse_cfg = {
    .disp = display,
    .sensitivity = 1, /* Sensitivity of the mouse moving */
};
lvgl_port_add_usb_hid_mouse_input(&mouse_cfg);

/* Add keyboard input device */
const lvgl_port_hid_keyboard_cfg_t kb_cfg = {
    .disp = display,
};
kb_indev = lvgl_port_add_usb_hid_keyboard_input(&kb_cfg);
```

Keyboard special behavior (when objects are in group):

TAB: Select next object SHIFT + TAB: Select previous object ENTER: Control object (e.g. click to button) ARROWS or HOME or END: Move in text area DEL or Backspace: Remove character in textarea Note

When you use keyboard for control LVGL objects, these objects must be added to LVGL groups. See LVGL documentation for more info.

LVGL API usage Every LVGL calls must be protected with these lock/unlock commands:

C

```
/* Wait for the other task done the screen operation */
lvgl_port_lock(0);
...
lv_obj_t * screen = lv_disp_get_scr_act(disp_handle);
lv_obj_t * obj = lv_label_create(screen);
...
/* Screen operation done -> release for the other task */
lvgl_port_unlock();
```

Rotating screen LVGL port supports rotation of the display. You can select whether you'd like software rotation or hardware rotation. Software rotation requires no additional logic in your flush_cb callback.

Rotation mode can be selected in the lvgl_port_display_cfg_t structure.

C

```
const lvgl_port_display_cfg_t disp_cfg = {
    ...
    .flags = {
        ...
        .sw_rotate = true / false, // true: software; false: hardware
    }
}
```

Display rotation can be changed at runtime.

C

```
lv_disp_set_rotation(disp_handle, LV_DISP_ROT_90);
```

Note

Software rotation consumes more RAM. Software rotation uses PPA if available on the chip (e.g. ESP32P4).

Note

During the hardware rotating, the component call esp_lcd API. When using software rotation, you cannot use neither direct_mode nor full_refresh in the driver. See LVGL documentation for more info.

Detecting gestures LVGL (version 9.4 and higher) includes support for software detection of multi-touch gestures. This detection can be enabled by setting the LV_USE_GESTURE_RECOGNITION config and having ESP_LCD_TOUCH_MAX_POINTS > 1. Example usage of the gesture callback can be found in LVGL documentation.

The LVGL port task is responsible for passing the touch coordinates to the gesture recognizers and calling the registered callback when a gesture is detected.

To correctly distinguish two finger swipe from rotation, we recommend changing the default value (which is 0) for the rotation threshold. From our testing we recommend starting with 0.15 radians.

C

```
lv_indev_t indev = bsp_display_get_input_dev();
lv_indev_set_rotation_rad_threshold(indev, 0.15f);
```

Using PSRAM canvas If the SRAM is insufficient, you can use the PSRAM as a canvas and use a small trans_buffer to carry it, this makes drawing more efficient.

C

```
const lvgl_port_display_cfg_t disp_cfg = {
    ...
    .buffer_size = DISP_WIDTH * DISP_HEIGHT, // in PSRAM, not DMA-capable
    .trans_size = size, // in SRAM, DMA-capable
    .flags = {
        .buff_spiram = true,
        .buff_dma = false,
        ...
    }
}
```

Generating images (C Array) Images can be generated during build by adding these lines to end of the main CMakeLists.txt:

Plaintext

Generate C array for each image

lvgl_port_create_c_image("images/logo.png" "images/" "ARGB8888" "NONE")
lvgl_port_create_c_image("images/image.png" "images/" "ARGB8888" "NONE")

Add generated images to build

lvgl_port_add_images(${COMPONENT_LIB} "images/") Usage of create C image function:

Plaintext

lvgl_port_create_c_image(input_image output_folder color_format compression) Available color formats: L8,I1,I2,I4,I8,A1,A2,A4,A8,ARGB8888,XRGB8888,RGB565,RGB565A8,RGB888,TRUECOLOR,TRUECOLOR_ALPHA,AUTO

Available compression: NONE,RLE,LZ4

Note

Parameters color_format and compression are used only in LVGL 9.

Power Saving The LVGL port can be optimized for power saving mode. There are two main features.

LVGL task sleep For optimization power saving, the LVGL task should sleep, when it does nothing. Set task_max_sleep_ms to big value, the LVGL task will wait for events only.

The LVGL task can sleep till these situations:

LVGL display invalidate LVGL animation in process Touch interrupt Button interrupt Knob interrupt USB mouse/keyboard interrupt Timeout (task_max_sleep_ms in configuration structure) User wake (by function

lvgl_port_task_wake) Warning

This feature is available from LVGL 9.

Note

Don't forget to set the interrupt pin in LCD touch when you set a big time for sleep in task_max_sleep_ms.

Stopping the timer Timers can still work during light-sleep mode. You can stop LVGL timer before use light-sleep by function:

Plaintext

lvgl_port_stop(); and resume LVGL timer after wake by function:

Plaintext

lvgl_port_resume(); Performance Key feature of every graphical application is performance. Recommended settings for improving LCD performance is described in a separate document here.

Performance monitor For show performance monitor in LVGL9, please add these lines to sdkconfig.defaults and rebuild all.

Plaintext

CONFIG_LV_USE_OBSERVER=y CONFIG_LV_USE_SYSMON=y CONFIG_LV_USE_PERF_MONITOR=y

CAMERA

SP32 Camera Driver Build examples Component Registry

General Information This repository hosts ESP32 series Soc compatible driver for image sensors. Additionally it provides a few tools, which allow converting the captured frame data to the more common BMP and JPEG formats.

Supported Soc ESP32 ESP32-S2 ESP32-S3 Supported Sensor model max resolution color type output format Len Size OV2640 1600 x 1200 color YUV(422/420)/YCbCr422 RGB565/555 8-bit compressed data 8/10-bit Raw RGB data 1/4" OV3660 2048 x 1536 color raw RGB data RGB565/555/444 CCIR656 YCbCr422 compression 1/5" OV5640 2592 x 1944 color RAW RGB RGB565/555/444 CCIR656 YUV422/420 YCbCr422 compression 1/4" OV7670 640 x 480 color Raw Bayer RGB Processed Bayer RGB YUV/YCbCr422 GRB422 RGB565/555 1/6" OV7725 640 x 480 color Raw RGB GRB 422 RGB565/555/444 YCbCr 422 1/4" NT99141 1280 x 720 color YCbCr 422 RGB565/555/444 Raw CCIR656 JPEG compression 1/4" GC032A 640 x 480 color YUV/YCbCr422 RAW Bayer RGB565 1/10" GC0308 640 x 480 color YUV/YCbCr422 RAW Bayer RGB565 Grayscale 1/6.5" GC2145 1600 x 1200 color YUV/YCbCr422 RAW Bayer RGB565 1/5" BF3005 640 x 480 color YUV/YCbCr422 RAW Bayer RGB565 1/4" BF20A6 640 x 480 color YUV/YCbCr422 RAW Bayer Only Y 1/10" SC101IOT 1280 x 720 color YUV/YCbCr422 Raw RGB 1/4.2" SC030IOT 640 x 480 color YUV/YCbCr422 RAW Bayer 1/6.5" SC031GS 640 x 480 monochrome RAW MONO Grayscale 1/6" HM0360 656 x 496 monochrome RAW MONO Grayscale 1/6" HM1055 1280 x 720 color 8/10-bit Raw YUV/YCbCr422 RGB565/555/444 1/6" Important to Remember Except when using CIF or lower resolution with JPEG, the driver requires PSRAM to be installed and activated. Using YUV or RGB puts a lot of strain on the chip because writing to PSRAM is not particularly fast. The result is that

image data might be missing. This is particularly true if WiFi is enabled. If you need RGB data, it is recommended that JPEG is captured and then turned into RGB using fmt2rgb888 or fmt2bmp/frame2bmp. When 1 frame buffer is used, the driver will wait for the current frame to finish (VSYNC) and start I2S DMA. After the frame is acquired, I2S will be stopped and the frame buffer returned to the application. This approach gives more control over the system, but results in longer time to get the frame. When 2 or more frame bufers are used, I2S is running in continuous mode and each frame is pushed to a queue that the application can access. This approach puts more strain on the CPU/Memory, but allows for double the frame rate. Please use only with JPEG. The Kconfig option CONFIG_CAMERA_PSRAM_DMA enables PSRAM DMA mode on ESP32-S2 and ESP32-S3 devices. This flag defaults to false. You can switch PSRAM DMA mode at runtime using esp_camera_set_psram_mode(). Installation Instructions Using with ESP-IDF Add a dependency on espressif/esp32-camera component: Bash

idf.py add-dependency "espressif/esp32-camera" (or add it manually in idf_component.yml of your project)

Enable PSRAM in menuconfig (also set Flash and PSRAM frequiencies to 80MHz) Include esp_camera.h in your code These instructions also work for PlatformIO, if you are using framework=espidf.

Using with Arduino Arduino IDE If you are using the arduino-esp32 core in Arduino IDE, no installation is needed! You can use esp32-camera right away.

PlatformIO The easy way -- on the env section of platformio.ini, add the following:

Ini

[env] lib_deps = esp32-camera Now the esp_camera.h is available to be included:

C

#include "esp_camera.h" Enable PSRAM on menuconfig or type it direclty on sdkconfig. Check the official doc for more info.

Plaintext

CONFIG_ESP32_SPIRAM_SUPPORT=y Examples This component comes with a basic example illustrating how to get frames from the camera. You can try out the example using the following command:

Plaintext

idf.py create-project-from-example "espressif/esp32-camera:camera_example" This command will download the example into camera_example directory. It comes already pre-configured with the correct settings in menuconfig.

Initialization C

#include "esp_camera.h"

//WROVER-KIT PIN Map #define CAM_PIN_PWDN -1 //power down is not used #define CAM_PIN_RESET -1 //software reset will be performed #define CAM_PIN_XCLK 21 #define CAM_PIN_SIOD 26 #define CAM_PIN_SIOC 27

#define CAM_PIN_D7 35 #define CAM_PIN_D6 34 #define CAM_PIN_D5 39 #define CAM_PIN_D4 36 #define CAM_PIN_D3 19 #define CAM_PIN_D2 18 #define CAM_PIN_D1 5 #define CAM_PIN_D0 4 #define

CAM_PIN_VSYNC 25 #define CAM_PIN_HREF 23 #define CAM_PIN_PCLK 22

static camera_config_t camera_config = { .pin_pwdn = CAM_PIN_PWDN, .pin_reset = CAM_PIN_RESET, .pin_xclk = CAM_PIN_XCLK, .pin_sccb_sda = CAM_PIN_SIOD, .pin_sccb_scl = CAM_PIN_SIOC,

```
.pin_d7 = CAM_PIN_D7,
.pin_d6 = CAM_PIN_D6,
.pin_d5 = CAM_PIN_D5,
.pin_d4 = CAM_PIN_D4,
.pin_d3 = CAM_PIN_D3,
.pin_d2 = CAM_PIN_D2,
.pin_d1 = CAM_PIN_D1,
.pin_d0 = CAM_PIN_D0,
.pin_vsync = CAM_PIN_VSYNC,
.pin_href = CAM_PIN_HREF,
.pin_pclk = CAM_PIN_PCLK,

.xclk_freq_hz = 20000000,
.ledc_timer = LEDC_TIMER_0,
.ledc_channel = LEDC_CHANNEL_0,

.pixel_format = PIXFORMAT_JPEG,//YUV422,GRAYSCALE,RGB565,JPEG
.frame_size = FRAMESIZE_UXGA,//QQVGA-UXGA, For ESP32, do not use sizes above QVGA
when not JPEG. The performance of the ESP32-S series has improved a lot, but JPEG
mode always gives better frame rates.

.jpeg_quality = 12, //0-63, for OV series camera sensors, lower number means
higher quality
.fb_count = 1, //When jpeg mode is used, if fb_count more than one, the driver
will work in continuous mode.
.grab_mode = CAMERA_GRAB_WHEN_EMPTY//CAMERA_GRAB_LATEST. Sets when buffers should
be filled
```

};

esp_err_t camera_init(){ //power up the camera if PWDN pin is defined if(CAM_PIN_PWDN != -1){ pinMode(CAM_PIN_PWDN, OUTPUT); digitalWrite(CAM_PIN_PWDN, LOW); }

```
//initialize the camera
esp_err_t err = esp_camera_init(&camera_config);
if (err != ESP_OK) {
    ESP_LOGE(TAG, "Camera Init Failed");
    return err;
}

return ESP_OK;
```

}

esp_err_t camera_capture(){ //acquire a frame camera_fb_t * fb = esp_camera_fb_get(); if (!fb) { ESP_LOGE(TAG, "Camera Capture Failed"); return ESP_FAIL; } //replace this with your own function process_image(fb->width, fb->height, fb->format, fb->buf, fb->len);

```
//return the frame buffer back to the driver for reuse
esp_camera_fb_return(fb);
return ESP_OK;
```

} JPEG HTTP Capture C

#include "esp_camera.h" #include "esp_http_server.h" #include "esp_timer.h"

typedef struct { httpd_req_t *req; size_t len; } jpg_chunking_t;

static size_t jpg_encode_stream(void _ arg, size_t index, const void_ data, size_t len){ jpg_chunking_t _j = (jpg_chunking_t _)arg; if(!index){ j->len = 0; } if(httpd_resp_send_chunk(j->req, (const char *)data, len) != ESP_OK){ return 0; } j->len += len; return len; }

esp_err_t jpg_httpd_handler(httpd_req_t _req){ camera_fb_t _ fb = NULL; esp_err_t res = ESP_OK; size_t fb_len = 0; int64_t fr_start = esp_timer_get_time();

```
fb = esp_camera_fb_get();
if (!fb) {
    ESP_LOGE(TAG, "Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
}
res = httpd_resp_set_type(req, "image/jpeg");
if(res == ESP_OK){
    res = httpd_resp_set_hdr(req, "Content-Disposition", "inline;
filename=capture.jpg");
}

if(res == ESP_OK){
    if(fb->format == PIXFORMAT_JPEG){
        fb_len = fb->len;
        res = httpd_resp_send(req, (const char *)fb->buf, fb->len);
    } else {
        jpg_chunking_t jchunk = {req, 0};
        res = frame2jpg_cb(fb, 80, jpg_encode_stream, &jchunk)?ESP_OK:ESP_FAIL;
        httpd_resp_send_chunk(req, NULL, 0);
        fb_len = jchunk.len;
    }
}
esp_camera_fb_return(fb);
int64_t fr_end = esp_timer_get_time();
ESP_LOGI(TAG, "JPG: %uKB %ums", (uint32_t)(fb_len/1024), (uint32_t)((fr_end -
```

```
fr_start)/1000));
return res;
```

} JPEG HTTP Stream C

#include "esp_camera.h" #include "esp_http_server.h" #include "esp_timer.h"

#define PART_BOUNDARY "123456789000000000000987654321" static const char* _STREAM_CONTENT_TYPE = "multipart/x-mixed-replace;boundary=" PART_BOUNDARY; static const char* _STREAM_BOUNDARY = "\r\n--" PART_BOUNDARY "\r\n"; static const char* _STREAM_PART = "Content-Type: image/jpeg\r\nContent-Length: %zu\r\n\r\n";

esp_err_t jpg_stream_httpd_handler(httpd_req_t _req){ camera_fb_t _ fb = NULL; esp_err_t res = ESP_OK; size_t jpg_buf_len = 0; uint8_t * jpg_buf = NULL; char part_buf[64]; static int64_t last_frame = 0; if(!last_frame) { last_frame = esp_timer_get_time(); }

```
res = httpd_resp_set_type(req, _STREAM_CONTENT_TYPE);
if(res != ESP_OK){
    return res;
}

while(true){
    fb = esp_camera_fb_get();
    if (!fb) {
        ESP_LOGE(TAG, "Camera capture failed");
        res = ESP_FAIL;
        break;
    }
    if(fb->format != PIXFORMAT_JPEG){
        bool jpeg_converted = frame2jpg(fb, 80, &jpg_buf, &jpg_buf_len);
        if(!jpeg_converted){
            ESP_LOGE(TAG, "JPEG compression failed");
            esp_camera_fb_return(fb);
            res = ESP_FAIL;
            break;
        }
    } else {
        jpg_buf_len = fb->len;
        jpg_buf = fb->buf;
    }

    if(res == ESP_OK){
        res = httpd_resp_send_chunk(req, _STREAM_BOUNDARY,
strlen(_STREAM_BOUNDARY));
    }
    if(res == ESP_OK){
        int hlen = snprintf(part_buf, sizeof(part_buf), _STREAM_PART,
jpg_buf_len);
        if(hlen < 0 || hlen >= sizeof(part_buf)){
```

```
            ESP_LOGE(TAG, "Header truncated (%d bytes needed >= %zu buffer)",
                     hlen, sizeof(part_buf));
            res = ESP_FAIL;
        } else {
            res = httpd_resp_send_chunk(req, part_buf, (size_t)hlen);
        }
    }
    if(res == ESP_OK){
        res = httpd_resp_send_chunk(req, (const char *)jpg_buf, jpg_buf_len);
    }
    if(fb->format != PIXFORMAT_JPEG){
        free(jpg_buf);
    }
    esp_camera_fb_return(fb);
    if(res != ESP_OK){
        break;
    }
    int64_t fr_end = esp_timer_get_time();
    int64_t frame_time = fr_end - last_frame;
    last_frame = fr_end;
    frame_time /= 1000;
    float fps = frame_time > 0 ? 1000.0f / (float)frame_time : 0.0f;
    ESP_LOGI(TAG, "MJPG: %uKB %ums (%.1ffps)",
        (uint32_t)(jpg_buf_len/1024),
        (uint32_t)frame_time, fps);
}

last_frame = 0;
return res;
```

} BMP HTTP Capture C

#include "esp_camera.h" #include "esp_http_server.h" #include "esp_timer.h"

esp_err_t bmp_httpd_handler(httpd_req_t _req){ camera_fb_t _ fb = NULL; esp_err_t res = ESP_OK; int64_t fr_start = esp_timer_get_time();

```
fb = esp_camera_fb_get();
if (!fb) {
    ESP_LOGE(TAG, "Camera capture failed");
    httpd_resp_send_500(req);
    return ESP_FAIL;
}

uint8_t * buf = NULL;
size_t buf_len = 0;
bool converted = frame2bmp(fb, &buf, &buf_len);
esp_camera_fb_return(fb);
if(!converted){
```

```
        ESP_LOGE(TAG, "BMP conversion failed");
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }

    res = httpd_resp_set_type(req, "image/x-windows-bmp")
       || httpd_resp_set_hdr(req, "Content-Disposition", "inline;
    filename=capture.bmp")
       || httpd_resp_send(req, (const char *)buf, buf_len);
    free(buf);
    int64_t fr_end = esp_timer_get_time();
    ESP_LOGI(TAG, "BMP: %uKB %ums", (uint32_t)(buf_len/1024), (uint32_t)((fr_end -
    fr_start)/1000));
    return res;
```

}

Audio

SP Codec Device Component Registry 中文版本 General Information esp_codec_dev is a driver component for audio codec devices. Following features are supported currently:

Support driver for common audio codec devices Support multiple instance of codec devices (including device of same type) Add unified abstract interface to operate on codec device Support customized codec realization based on provided interface Easy-to-use high-level API for playback and recording Support for volume adjustment in software when it is not supported in hardware Support customized volume curve and customized volume control Easy to port to other platform after replacing codes under platform The currently supported codec devices are listed as below:

Playback Record ES8311 Y Y ES8388 Y Y ES8374 Y Y ZL38063 Y Y TAS6805M Y N AW88298 Y N ES8389 Y Y ES7210 N Y ES7243 N Y ES7243E N Y ES8156 N Y CJC8910 Y Y Architecture overview Hardware connection and software architecture are introduced respectively by taking the codec device (ES8311) as an example. The hardware connection diagram between the codec device (ES8311) and the main IC (ESP32-S3) is as below:

Mermaid

Left Speaker NS4150 ES8311 ESP32-S3 Mic Speaker IN CTRL OUT ADC DAC I2C_Port I2S_Port I2C_Bus I2S_Bus PA_GPIO Microphone

ESP32-S3 sends control data to ES8311 through I2C bus and exchanges audio data through I2S bus. During playing, ES8311 receives digital audio data from I2S bus and performs DAC operation, then the analog signal is amplified by PA chip (NS4150) and finally is output through the speaker. During recording, ES8311 gets the analog signal from the microphone, amplifies it, and performs ADC operation, then digital audio data can be obtained from ESP32-S3.

Communication between ESP32-S3 and ES8311 mainly occurs on two paths:

Control path: Set up the codec chip (using I2C bus) Data path: Exchange audio data (using I2S bus) In software architecture, the above hardware behavior is abstracted as:

Mermaid

audio_codec_ctrl_if_t open() read_reg() write_reg() close() audio_codec_gpio_if_t setup() set() get() es8311_codec_cfg_t audio_codec_ctrl_if_t *ctrl_if audio_codec_gpio_if_t gpio_if int16_t pa_pin esp_codec_dev_hw_gain_t hw_gain audio_codec_if_t audio_codec_ctrl_if_t ctrl_if open() enable() set_fs() set_vol() set_mic_gain() close() audio_codec_data_if_t open() set_fmt() read() write() close() esp_codec_dev audio_codec_data_if_t* data_if audio_codec_if_t* codec_if esp_codec_dev_new() esp_codec_dev_open() esp_codec_dev_read() esp_codec_dev_write() esp_codec_dev_set_out_vol() esp_codec_dev_set_in_gain() esp_codec_dev_set_vol_curve() esp_codec_dev_set_vol_handler() esp_codec_dev_close()

esp_codec_dev abstracts the above communication path into two interfaces:

audio_codec_ctrl_if_t for the control path: The control interface mainly offers read_reg and write_reg APIs to do codec setup Commonly used control channels include I2C, SPI, etc audio_codec_data_if_t for data path: The data interface mainly offers read and write APIs to exchange audio data Commonly used data channels include I2S, SPI, etc esp_codec_dev provides users with convenient high-level API to implement playback and recording functions. It is composed of audio_codec_data_if_t and audio_codec_if_t. audio_codec_if_t abstracts codec control operations and constructed by specified codec configuration (configured by audio_codec_ctrl_if_t and audio_codec_gpio_if_t through es8311_codec_cfg_t). audio_codec_gpio_if_t abstracts the IO control to adapt to the main control IO or the expansion chip IO, and called inside the codec to match the unique set timing.

DAC Volume setting Volume setting is realized by common API: esp_codec_dev_set_out_vol. esp_codec_dev supports the following volume setup methods:

Use codec register to adjust the volume Use built-in software volume audio_codec_new_sw_vol when codec hardware does not support volume adjustment Use customized software volume interface through esp_codec_dev_set_vol_handler The default volume range is 0 - 100. Volume [1:100] is mapped to [-49.5 dB:0 dB] with the scale being 0.5 dB. Volume 0 is mapped to -96 dB. To change this mapping, you can define your own volume curve through API esp_codec_dev_set_vol_curve. The volume curve is an array of esp_codec_dev_vol_map_t which uses linear interpolation to calculate the decibel value at a certain volume point internally (please sort volume maps in advance). To balance the speaker's loudness across different platforms when playing the same content, you need to know some mechanism of the audio gain. In short, audio gain consists of two parts: software gain (adjustable) and hardware gain (fixed). Software gain can be adjusted by changing the input PCM data level or setting the codec volume register. The hardware gain is affected by the peripheral circuit, mainly by the amplification efficiency of the analog signal. The typical impact parameter of hardware gain is extracted into esp_codec_dev_hw_gain_t, which can be configured to codec devices to ensure loudness consistency. For more details, please refer to the comments in esp_codec_dev_vol.h.

Usage The steps below take the ES8311 codec as an example to illustrate how to play and record audio.

Install the driver for codec control and data bus referring to test_board.c c ut_i2c_init(0); ut_i2s_init(0);

Create the control and data interfaces for the codec using the interface provided by default

```
audio_codec_i2c_cfg_t i2c_cfg = {.addr = ES8311_CODEC_DEFAULT_ADDR}; const
audio_codec_ctrl_if_t *out_ctrl_if = audio_codec_new_i2c_ctrl(&i2c_cfg);
```

```
const audio_codec_gpio_if_t *gpio_if = audio_codec_new_gpio(); ```

Create the codec interface based on control interface and codec-specified configuration
c es8311_codec_cfg_t es8311_cfg = { .codec_mode = ESP_CODEC_DEV_WORK_MODE_BOTH, .ctrl_if = out_ctrl_if, .gpio_if = gpio_if, .pa_pin = YOUR_PA_GPIO, .use_mclk = true, }; const audio_codec_if_t *out_codec_if = es8311_codec_new(&es8311_cfg);

Get esp_codec_dev_handle_t through esp_codec_dev_new
Now you can use the handle for further playback and recording as follows:

C


esp_codec_dev_cfg_t dev_cfg = {
    .codec_if = out_codec_if;              // codec interface from es8311_codec_new
    .data_if = data_if;                    // data interface from audio_codec_new_i2s_data
    .dev_type = ESP_CODEC_DEV_TYPE_IN_OUT; // codec support both playback and record
};
esp_codec_dev_handle_t codec_dev = esp_codec_dev_new(&dev_cfg);
// Below code shows how to play
esp_codec_dev_set_out_vol(codec_dev, 60.0);
esp_codec_dev_sample_info_t fs = {
    .sample_rate = 48000,
    .channel = 2,
    .bits_per_sample = 16,
};
esp_codec_dev_open(codec_dev, &fs);
uint8_t data[256];
esp_codec_dev_write(codec_dev, data, sizeof(data));

// Below code shows how to record
esp_codec_dev_set_in_gain(codec_dev, 30.0);
esp_codec_dev_read(codec_dev, data, sizeof(data));
esp_codec_dev_close(codec_dev);
How to customize for new codec device
Implement audio_codec_ctrl_if_t and audio_codec_data_if_t
If you are using I2C bus for control and I2S bus for data, you can use the implementation provided by default:
audio_codec_new_i2c_ctrl and audio_codec_new_i2s_data

Implement audio_codec_if_t based on the interface built in step 1

C


typedef struct {
    const audio_codec_ctrl_if_t *ctrl_if;     /*!< Codec Control interface */
    const audio_codec_gpio_if_t *gpio_if;     /*!< If you want to operate GPIO */
    //.................................     Other settings
```

```
} my_codec_cfg_t;
const audio_codec_if_t *my_codec_new(my_codec_cfg_t *codec_cfg);
For details, refer to the sample code my_codec.c.
```