Quick Start: ESP32-S3

Supported Development Boards

Introduction to ESP32-S3 development boards:

Model Name	Description	Datasheet Link
ZX3D50CE02S-USRC-4832	SoC: WT32-S3-WROVER-N16R8 Screen Size: 3.5" LCD (480*320) Flash: 8MB NOR FLASH PSRAM: 2MB Peripherals: CODEC, RS485	Datasheet
ZX3D95CE01S-UR-4848	SoC: WT32-S3-WROVER-N16R8 Screen Size: 3.95" RGB LCD (480*480) Flash: 16MB NOR FLASH PSRAM: 8MB Peripherals: USB, RS485	Datasheet
ZX3D95CE01S-AR-4848	SoC: WT32-S3-WROVER-N16R8 Screen Size: 3.95" RGB LCD (480*480) Flash: 16MB NOR FLASH PSRAM: 8MB Peripherals: CODEC, RS485	Datasheet
ZX2D10GE01R-V4848	SoC: WT32-S3-WROVER-N16R8 Screen Size: 2.1" RGB LCD (480*480) Flash: 16MB NOR FLASH PSRAM: 8MB	Datasheet

Model Name	Description	Datasheet Link
ZX4D30NE01S-UR-4827	SoC: WT32-S3-WROVER-N16R8 Screen Size: 4.3" RGB LCD (480*272) Flash: 16MB NOR FLASH PSRAM: 8MB	Datasheet
ZX7D00CE01S	SoC: WT32-S3-WROVER-N16R8 Screen Size: 7" RGB LCD (800*480) Flash: 16MB NOR FLASH PSRAM: 8MB	Datasheet

Environment Setup

Online Setup

Linux

Ubuntu

shell

```shell
sudo apt update
sudo apt install git
sudo apt install python3
sudo apt install cmake
sudo apt install libusb-1.0-0-dev
git clone -b release/v4.4 --recursive https://github.com/espressif/esp-
cd esp-idf
export IDF_GITHUB_ASSETS="dl.espressif.com/github_assets"
bash install.sh
source export.sh
```

Windows

Refer to the following link for setup instructions. Offline installation is recommended: https://docs.espressif.com/projects/esp-idf/zh_CN/release-v4.4/esp32s3/get-started/windows-setup.html#get-started-windows-tools-installer

Setup using Docker

```shell
docker pull shukewt/qmsd_idf_4_4
docker container run -it shukewt/qmsd_idf_4_4 /bin/bash
source ~/esp/esp-idf/export.sh
```

References

- Espressif Development Documentation

Obtaining the SDK

Repository Links

ESP-IDF for compatible SDK

https://github.com/espressif/esp-idf.git

8ms SDK

For international users: https://github.com/wireless-tag-com/8ms-esp32/tree/release/2.2

For users in China: https://gitee.com/qiming-zhixian/sdk-8ms-esp32/tree/release%2F2.2/

8MS Operation Manual

https://docs.8ms.xyz/

Compiling the SDK

Load the corresponding development board model

Linux Environment

```shell
bash load.sh
1): WT32_SC01
2): WT154_C3SI1
```

```
3): WT154_S2MI1
4): WT_86_32_3ZW1
5): WT280_S2MX1
6): WT240_C3SI1
7): WT_0_S2_240MW1
8): ZX3D50CE02S_USRC_4832
9): ZX3D95CE01S_AR_4848
10): ZX3D95CE01S_UR_4848
11): ZX4D30NE01S_UR_4827
12): ZX4D60_AR_4896
13): ZX2D10ECS_cESCP01
1
Set to 1-WT32_SC01
--- sdkconfig done

Done
```

Windows Environment

Double-click load_windows.bat and select the corresponding development board model.

Local Compilation and Flashing

shell

```
idf.py build
idf.py flash
```

- \After setting the development board model in the previous section, compile and flash directly.
- \Choosing the development board and using the idf.py set-target command may reset the preset sdkconfig.

Firmware Flashing

Enter Flashing Mode

When burning ESP32S3 via UART, connect VCC, GND, EN, IO0 (IO9 for ESP32-C3),

TXD, and RXD, a total of six pins. When using most commercially available ESP32 burning devices, the hardware will automatically operate IO0 and EN to enter the burning mode. When using an unadapted burning device, you can ground IO0 and control EN to restart the chip and enter the burning mode.

Flashing Local Development Project

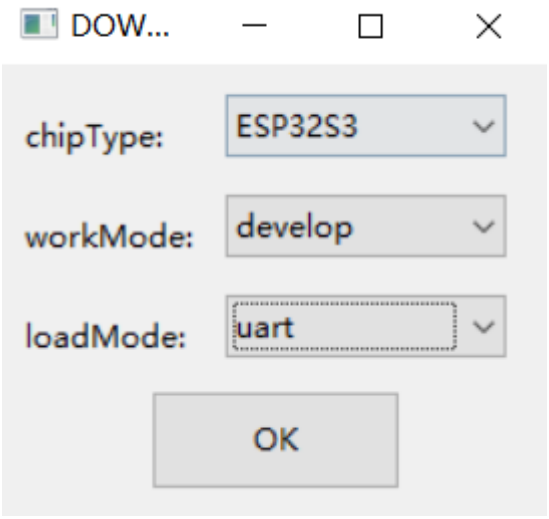
After correctly connecting the module and development board, use idf to flash:

```shell
idf.py flash
```

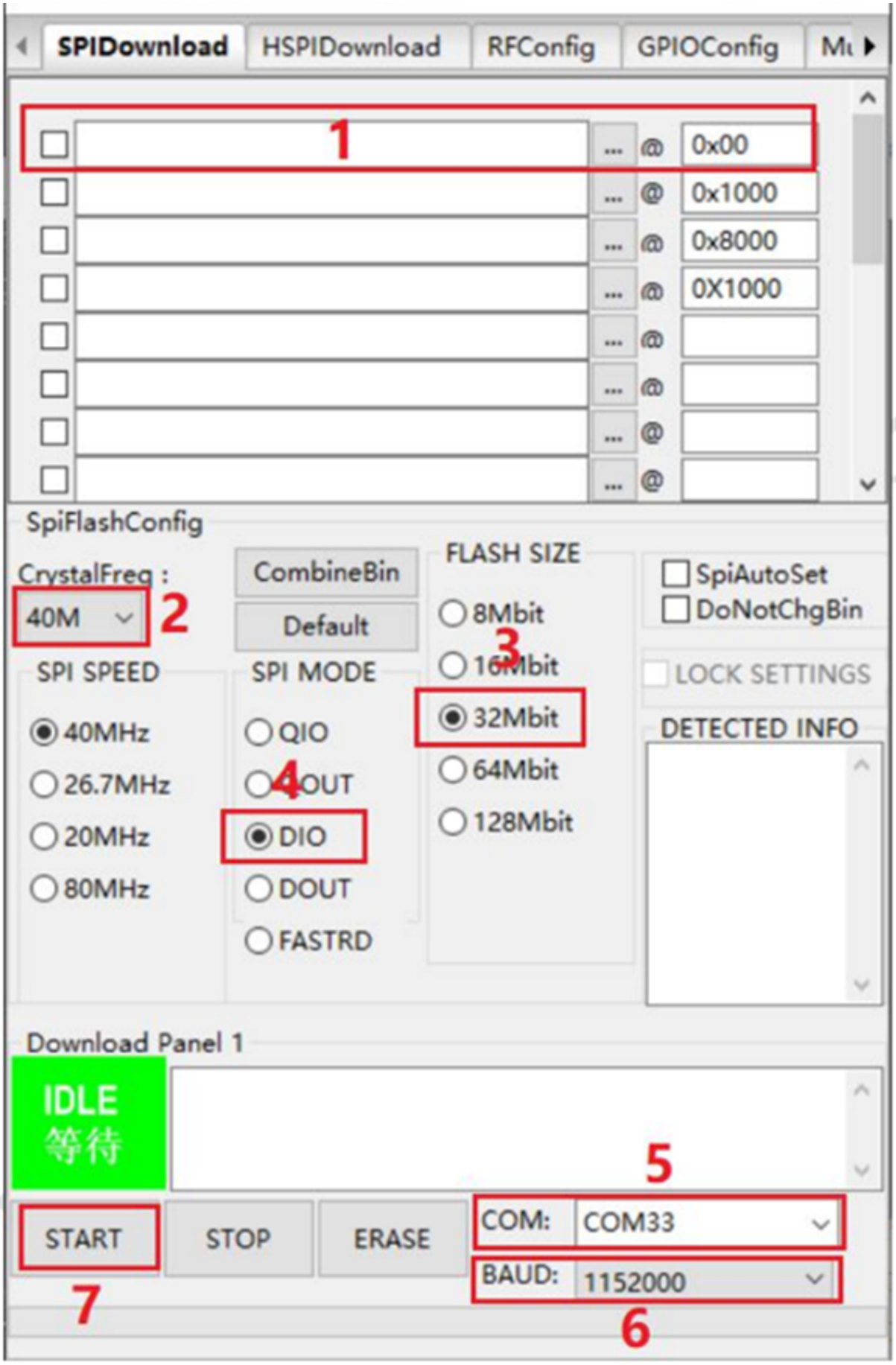
In some environments, you may need to use the -p parameter to manually specify the serial port or the -b parameter to specify the baud rate.

Flashing Local Firmware

1. Obtain the Espressif flash_download_tool.
2. Select the corresponding chip model and interface, choose ESP32S3.



3. Select the corresponding firmware image and enter the corresponding flashing address, as shown in the following figure: Select the firmware path to be burned at location 1, the address is usually 0X00, remember to check the box in front; select the system clock as 40MHz at location 2; select the Flash size as 32Mbit at location 3; select SPI MODE as DIO mode at location 4; select the port number recognized by the computer for the current board at location 5; select the baud rate at location 6 (the larger the value, the faster the firmware download speed, with a maximum support of 1152000bps).



4. After completing the previous configuration, click 7 to start flashing the firmware.
5. Manually reset the development board after flashing is complete.

Reference Links:

Espressif Development Documentation:
https://docs.espressif.com/projects/esp-idf/zh_CN/v4.4.1/esp32/get-started/index.html#get-started-flash

---

Code Example

ZX2D10GE01R-V4848

Load the corresponding development board model

In Linux environment:

```shell
bash load.sh
1): WT32_SC01
2): WT154_C3SI1
3): WT154_S2MI1
4): WT_86_32_3ZW1
5): WT280_S2MX1
6): WT240_C3SI1
7): WT_0_S2_240MW1
8): ZX3D50CE02S_USRC_4832
9): ZX3D95CE01S_AR_4848
10): ZX3D95CE01S_UR_4848
11): ZX4D30NE01S_UR_4827
12): ZX4D60_AR_4896
13): ZX2D10GE01R_V_4848
14): ZX7D00CE01S_UR_8048
13
Set to 13-ZX2D10GE01R_V_4848
--- sdkconfig done

Done
```

In Windows environment, double-click `load_windows.bat` and select the corresponding development board model.

Compile and test the demo

shell

```
cd examples/demo_knob
cp ../../sdkconfig sdkconfig
idf.py build
idf.py flash
```

Description

The following code detects events from a button and a knob, and sends related events through `qmsd_notifier_call` , ultimately calling the `__qmsd_got_encoder_func` function to handle these events.

shell

```
mt8901_init(5,6);
    qmsd_button_config_t config = {
        .ticks_interval_ms = 10,
        .debounce_ticks = 2,
        .short_ticks = 200 / 10,
        .long_ticks = 1000 / 10,
        .update_task = {
            .en = 1,
            .core = 1,
            .priority = 1,
        }};
    qmsd_button_init(&config);

    btn_handle_t *btn0 = qmsd_button_create_gpio(3, 0, NULL);
    qmsd_button_start(btn0);
    for (;;)
    {
        static int16_t cont_last = 0;
        int16_t cont_now = mt8901_get_count();
        int16_t enc_diff = ECO_STEP(cont_now - cont_last);
        cont_last = cont_now;
        if (enc_diff == 1)
        {
            // Right rotation
            qmsd_notifier_call(ENCODER_RIGHT,NULL);
        }
        else if(enc_diff == -1)
```

```
28          {
29              // Left rotation
30              qmsd_notifier_call(ENCODER_LEFT,NULL);
31          }
32          else
33          {
34              qmsd_notifier_call(ENCODER_NONE,NULL);
35          }
36
37          if (qmsd_button_wait_event(btn0, BUTTON_SINGLE_CLICK, pdMS_TO_T
38          {
39              // Short press
40              qmsd_notifier_call(ENCODER_SHORT_CLICK,NULL);
41          }
42          if (qmsd_button_wait_event(btn0, BUTTON_LONG_PRESS_START, pdMS_
43          {
44              // Long press
45              qmsd_notifier_call(ENCODER_LONG_CLICK,NULL);
46          }
47          vTaskDelay(pdMS_TO_TICKS(20));
48      }
```

Development Instructions

API Guide

8ms Related

- Initialize the screen list void qmsd_screen_list_init(int max);
- Register a screen void qmsd_screen_register(lv_obj_t* obj, const char* id);
- Remove a registered screen void qmsd_screen_remove(const char* id);
- Set an identification ID for a specified object void qmsd_obj_set_id(lv_obj_t* obj, const char* id);
- Print all currently registered screens void qmsd_screen_print();
- Search for a screen based on the registered ID lv_obj_t*

qmsd_search_screen(const char* id);

- Search for a widget based on the registered ID lv_obj_t* qmsd_search_widget(const char* id);
- Get the image resource with the corresponding name lv_img_src_t* qmsd_get_img(const char* w_name);
- Get the font resource with the corresponding name lv_font_t* qmsd_get_font(const char* w_name);
- Send events to all child widgets of a screen void qmsd_send_event_to_chill(lv_obj_t* obj, lv_obj_t* prev, qmsd_event event)

Control Protocol Related

- Send the control protocol as a text message to the UI thread internally int qmsd_ctrl_str(const char* json_str);
- Wait for the UI thread to process and execute the control protocol with interval, using mutex lock char* qmsd_ctrl_str_sync(const char* json_str);
- Parse and process the control protocol within the GUI thread, limited to internal use within the GUI thread char* qmsd_ctrl_str_gui(const char* json_str);

Initialization

- GUI initialization, custom_fb_size for custom LVGL buffer size (0 for default), dir for screen orientation (refer to qmsd_screen_rotation_t definition) esp_err_t qmsd_gui_init(uint32_t custom_fb_size, uint8_t dir);
- Storage initialization, initialize the NVS partition of qmsd_storage void qmsd_storage_init(void);
- Initialize the message queue, generally used for widget callbacks to send messages to the main thread int qmsd_main_msgque_init(int msgmax);
- Notification chain initialization, qmsd_api uses many notification chains to notify events int qmsd_notifier_register(struct qmsd_notifier_block* n);

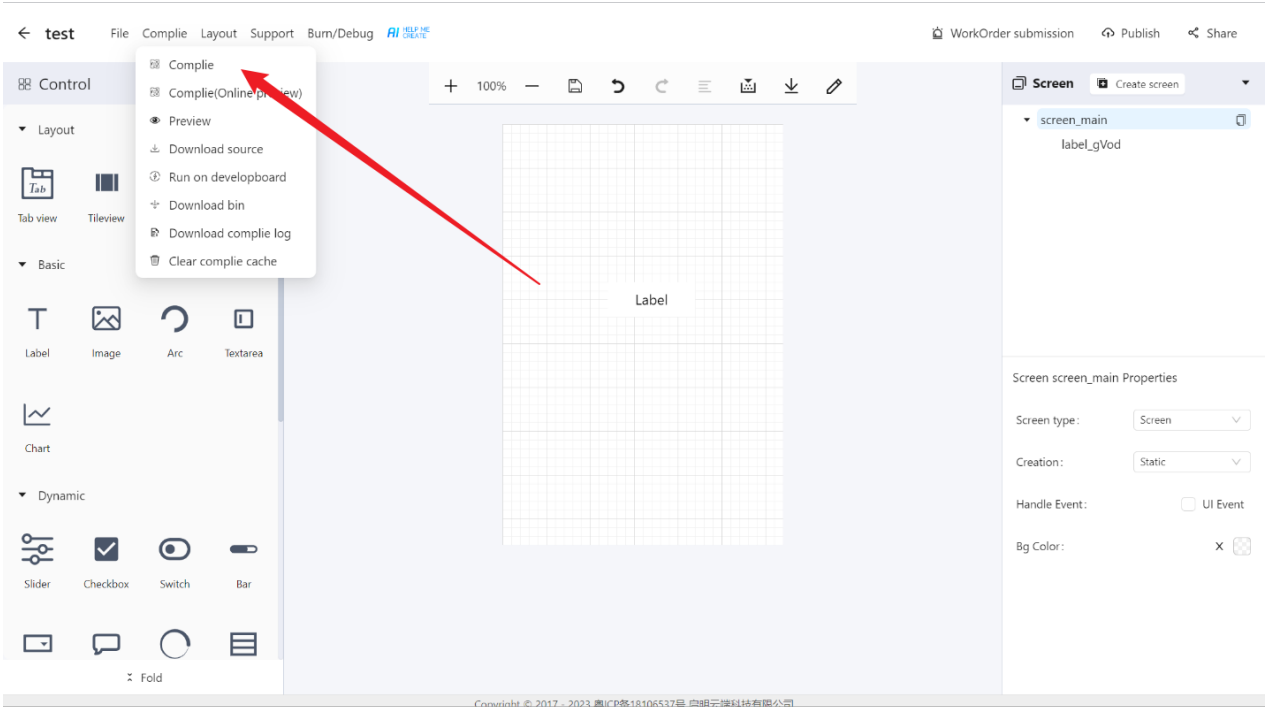
Development Process

Third-Party Development Board Adaptation

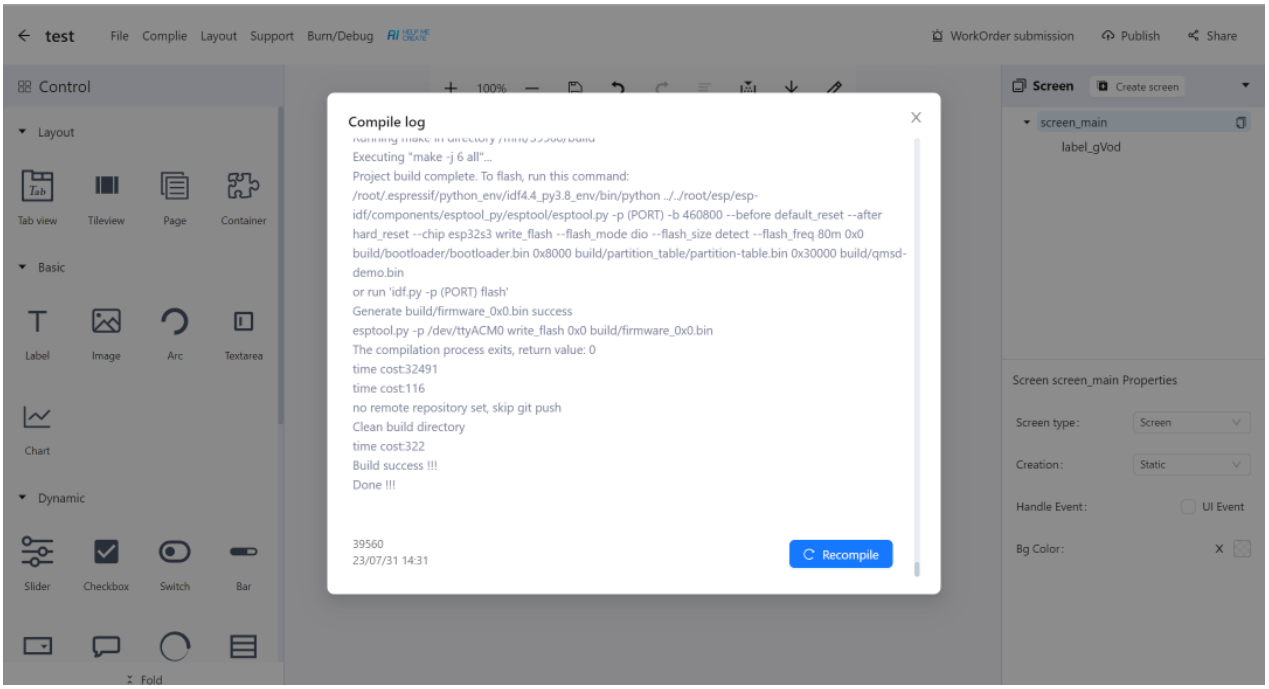
If using the development board provided by Espressif, the initialization of various peripherals of the development board can be completed by loading the configuration file directly. When using a third-party development board, you need to implement the relevant initialization and integration with the UI library according to the function pointer definition typedef void (*qmsd_board_init_cus)(qmsd_screen_rotation_t dir). Then use void qmsd_set_board_init_cus(qmsd_board_init_cus init_cus); to specify the use of the third-party initialization method.

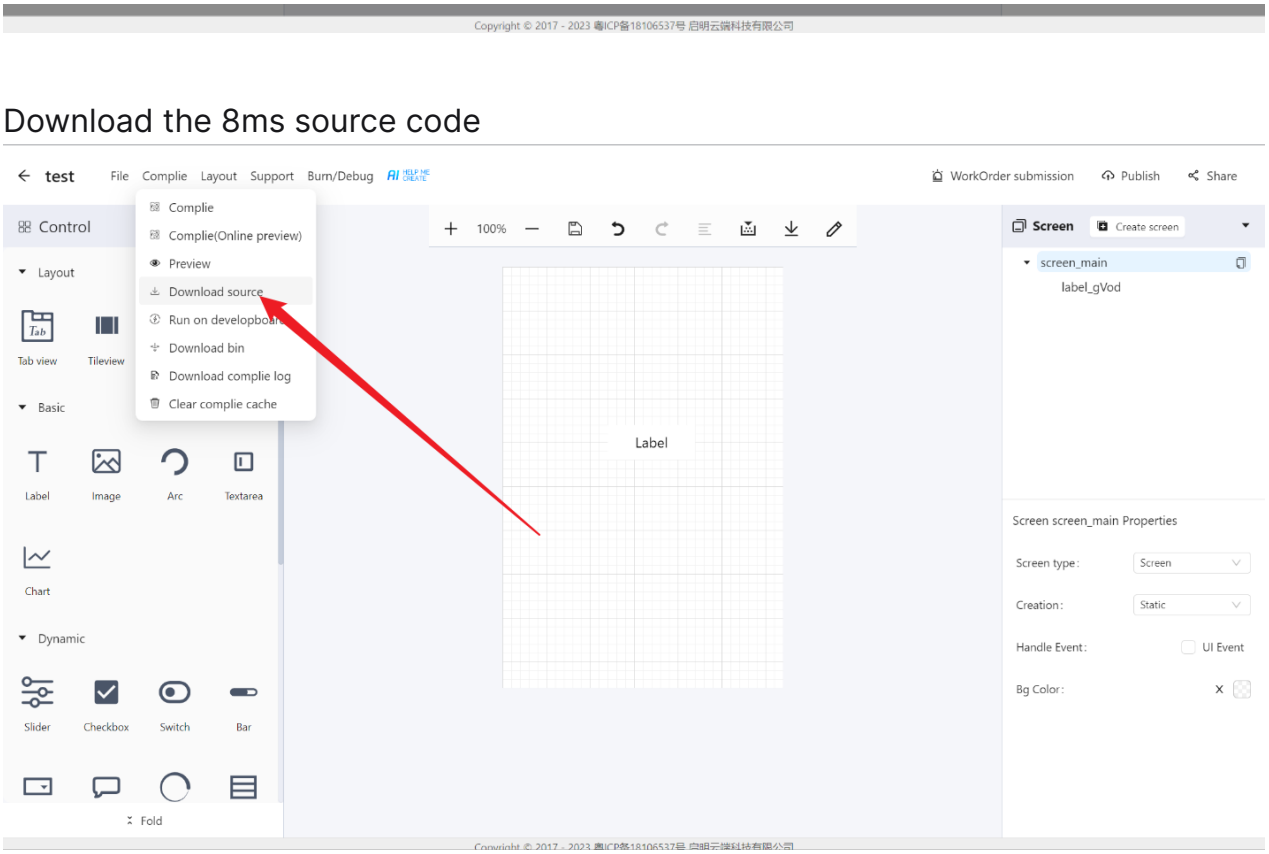
Adding UI Code Generated by 8ms

1. Compile the UI project on the 8ms platform



2. Confirm the successful compilation of the project





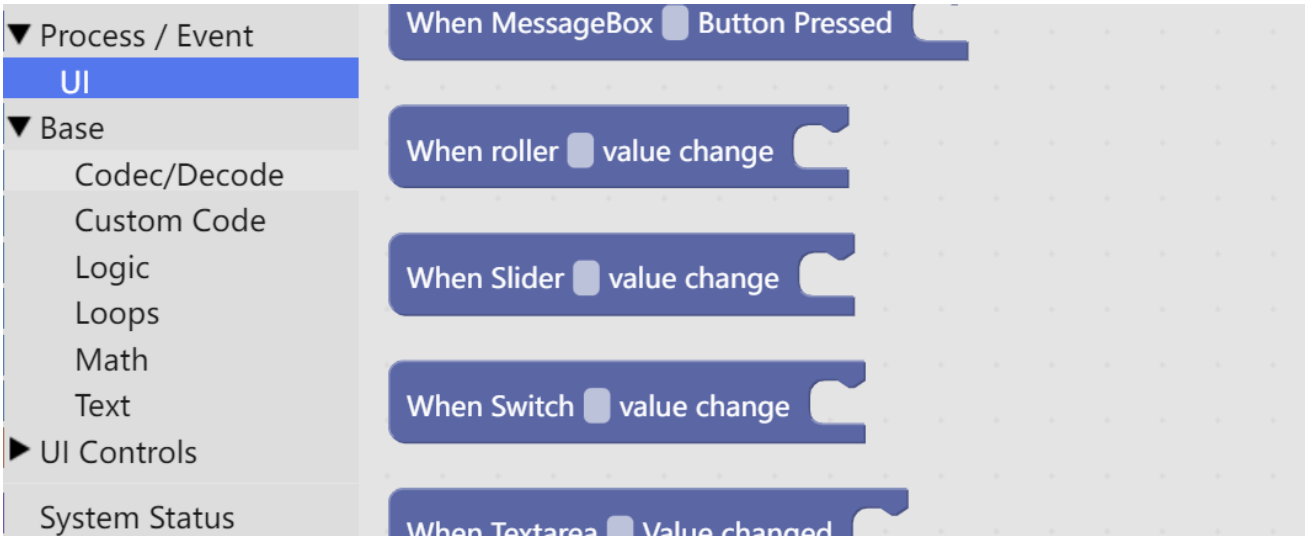
3. Download the 8ms source code

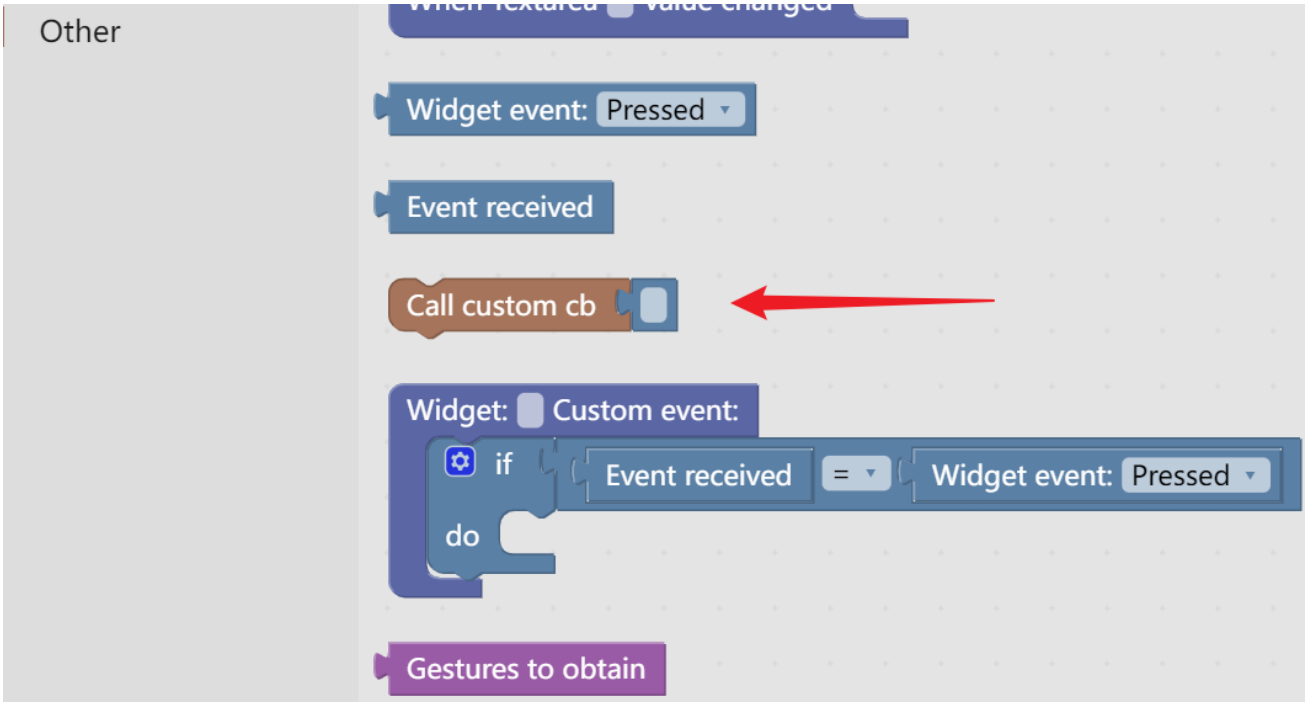
4. Delete the {sdk_dir}/components/qmsd_ui directory and extract the downloaded source code to the corresponding directory

5. Compile and flash according to the above sections

Adding Custom Logic Code

Code related to the UI can be added to the callback functions of various widgets in {sdk_dir}/components/qmsd_ui/ui/qmsd_internal_ui_cb.c. Alternatively, you can pass the time parameters to qmsd_ui_cb in {sdk_dir}/main/control/qmsd_ui_cb.c through custom callbacks in 8ms for further processing. It is worth noting that these functions normally run in the GUI thread, and adding time-consuming or blocking tasks may lead to UI performance degradation or freezing.





Cross-Thread UI Scheduling

When modifying resources within the UI thread across threads, it is necessary to use mutex locks to ensure thread safety. You can use the following APIs to acquire and release mutex locks.

shell

```
int qmsd_gui_lock(uint32_t timeout)
void qmsd_gui_unlock(void)
```

Alternatively, you can use the built-in text protocol of the UI library, see https://docs.8ms.xyz/develop/sdk/api.html for details.

Previous page
Supported Chipes

Next page
ESP32-S2