This is the documentation for esp-dev-kits.
### SUPPORTED DEVELOPMENT BOARDS

<table>
<thead>
<tr>
<th><strong>ESP32-S3 Development Boards</strong></th>
<th><img src="image1.png" alt="ESP32-S3-USB-OTG" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESP32-S3-USB-OTG</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>ESP32-S2 Development Boards</strong></th>
<th><img src="image2.png" alt="ESP32-S2-Kaluga-1" /></th>
<th><img src="image3.png" alt="ESP32-S2-Touch-DevKit-1" /></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESP32-S2-Kaluga-1</strong></td>
<td></td>
<td><strong>ESP32-S2-Touch-DevKit-1</strong></td>
</tr>
</tbody>
</table>
## ESP32 Development Boards

| ESP32-LCDKit |

## Other Boards

| ESP-Prog | ESP32-MeshKit-Sense |

## EOL (End of Life) Boards

| ESP32-S2-HMI-DevKit-1 | ESP32-Sense-Kit |
1.1 Related

- ESP-IDF Get Started
- ESP-IDF Programming Guide
- ESP Product Selector

1.1.1 ESP32-S3-USB-OTG

ESP32-S3-USB-OTG is a development board that focuses on USB-OTG function verification and application development.

ESP32-S3-USB-OTG

ESP32-S3-USB-OTG is a development board that focuses on USB-OTG function verification and application development. It is based on ESP32-S3 SoC, supports Wi-Fi and BLE 5.0 wireless functions, and supports USB host and USB device functions. It can be used to develop applications such as wireless storage devices, Wi-Fi network cards, LTE MiFi, multimedia devices, virtual keyboards and mice. The development board has the following features:

- Onboard ESP32-S3-MINI-1-N8 module, with built-in 8 MB flash
- Onboard USB Type-A host and device interface, with built-in USB interface switching circuit
- Onboard USB to serial debugging chip (Micro USB interface)
- Onboard 1.3-inch LCD color screen, supports GUI
- Onboard SD card interface, compatible with SDIO and SPI interfaces
- Onboard charging IC, can be connected to lithium battery

The document consists of the following major sections:

- **Getting Started**: Provides a brief overview of ESP32-S3-USB-OTG and necessary hardware and software information.
- **Hardware Reference**: Provides detailed hardware information of ESP32-S3-USB-OTG.
- **Related Documents**: Provides links to related documents.

Getting Started

This section describes how to start using ESP32-S3-USB-OTG. It includes introduction to basic information about ESP32-S3-USB-OTG first, and then on how to start using the development board for application development, as well as board packaging and retail information.
Fig. 1: ESP32-S3-USB-OTG (click to enlarge)
Description of Components

The ESP32-S3-USB-OTG development board includes the following parts:

- **Motherboard**: ESP32-S3-USB-OTG motherboard is the core of the kit. The motherboard integrates the ESP32-S3-MINI-1 module and provides an interface of the 1.3-inch LCD screen.

![ESP32-S3-USB-OTG Top View](image)

The following table starts with the USB_HOST Interface on the left, and introduces the main components in the above figure in an anticlockwise order.

1.1. Related
**Main components** | **Description**
--- | ---
USB_HOST Interface | USB Type-A female port, used to connect other USB devices.
ESP32-S3-MINI-1 Module | ESP32-S3-MINI-1 is a powerful, generic Wi-Fi + Bluetooth LE MCU module that has a rich set of peripherals. It has strong ability for neural network computing and signal processing. ESP32-S3-MINI-1 comes with a PCB antenna and is pin-to-pin compatible with ESP32-S2-MINI-1.
MENU Button | Menu button.
Micro SD Card Slot | Micro SD card can be inserted. Both four-line SDIO and SPI mode are supported.
USB Switch IC | By setting the level of USB_SEL, you can switch USB peripherals to make them either connected to the USB_DEV interface or the USB_HOST interface. USB_DEV will be connected by default.
Reset Button | Press this button to restart the system.
USB_DEV Interface | USB Type-A male port, can be connected to the USB host, and also used as a lithium battery charge power source.
Power Switch | Switch to ON to use battery power. Switch to OFF to power off battery.
Boot Button | Download button. Holding down Boot and then pressing Reset initiates Firmware Download mode for downloading firmware through the serial port.
DW- Button | Down button.
LCD FPC Connector | Used to connect the 1.3-inch LCD screen.
UP+ Button | Up button.
USB-to-UART Interface | A Micro-USB port used for power supply to the board, for flashing applications to the chip, as well as for communication with the chip via the on-board USB-to-UART bridge.

The following table starts with the Yellow LED on the left, and introduces the main components in the above figure in an anticlockwise order.

<table>
<thead>
<tr>
<th>Main components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yellow LED</td>
<td>Driven by GPIO16, set high level to turn on.</td>
</tr>
<tr>
<td>Green LED</td>
<td>Driven by GPIO15, set high level to turn on.</td>
</tr>
<tr>
<td>Charging LED</td>
<td>During charging, the red light is on, which will be turned off when charged.</td>
</tr>
<tr>
<td>Battery Solder Joints</td>
<td>3.6 V lithium battery can be welded to power the motherboard.</td>
</tr>
<tr>
<td>Charging Circuit</td>
<td>Used to charge lithium battery.</td>
</tr>
<tr>
<td>Free Pins</td>
<td>Idle pins that can be customized.</td>
</tr>
<tr>
<td>USB-to-UART Bridge</td>
<td>Single USB-to-UART bridge chip provides transfer rates up to 3 Mbps.</td>
</tr>
</tbody>
</table>

- **Subboard**: ESP32-S3-USB-OTG-SUB mount the 1.3-inch LCD screen
1.1. Related

Fig. 3: ESP32-S3-USB-OTG Bottom View (click to enlarge)

Fig. 4: ESP32-S3-USB-OTG Subboard (click to enlarge)
Start Application Development

Before powering on the ESP32-S3-USB-OTG, please make sure that the development board is intact.

Required Hardware

- ESP32-S3-USB-OTG
- A USB 2.0 data cable (standard A to Micro-B)
- Computer (Windows, Linux or macOS)

Software Setup

Please proceed to Get Started, where Section Installation Step by Step will quickly help you set up the development environment and then flash an application example onto your board.

Contents and Packaging

Retail Orders

If you order a few samples, each board comes in an individual package in either antistatic bag or any packaging depending on your retailer.

Which contains the following parts:

- Motherboard:
  - ESP32-S3-USB-OTG
- Subboard:
  - ESP32-S3-USB-OTG_SUB
- Fastener
  - Mounting bolt (x4)
  - Screw (x4)
  - Nut (x4)

For retail orders, please go to https://www.espressif.com/zh-hans/company/contact/buy-a-sample.

Wholesale Order

If purchased in bulk, the development board will be packaged in a large cardboard box.

For wholesale orders, please go to https://www.espressif.com/en/contact-us/sales-questions.
Fig. 5: ESP32-S3-USB-OTG Package (click to enlarge)
Hardware Reference

Block Diagram

The block diagram below shows the components of ESP32-S3-USB-OTG and their interconnections.

Fig. 6: ESP32-S3-USB-OTG Block Diagram (click to enlarge)

Please note that the external interface corresponding to the USB_HOST D+ D− signal in the functional block diagram is USB_DEV, which means that ESP32-S3 is used as a device to receive signals from other USB hosts. The external interface corresponding to the USB_DEV D+ D− signal is USB_HOST, which means that ESP32-S3 acts as a host to control other devices.

Power Supply Options

There are three power supply methods for the development board:

1. Power supply through the Micro_USB interface
   - Use the USB cable (standard A to Micro-B) to connect the motherboard to a power supply device, and set battery switch to OFF. Please note that in this power supply mode, only the motherboard and display are powered.

2. Power supply through the USB_DEV interface
   - Set DEV_VBUS_EN to high level, and set battery switch to OFF. This mode can supply power to the USB_HOST interface. The lithium battery will be charged at the same time (if the lithium battery is installed)
3. Power supply through battery

- Set BOOST_EN to high level, and set battery switch to ON. You should solder a 1-Serial lithium battery (3.7 V ~ 4.2 V) to the power solder joint reserved on the back of the motherboard first. This mode can supply power to the USB HOST interface at the same time. The battery interface description is as follows:

![Battery Connection](image)

**USB HOST Interface Power Options**

The USB HOST interface (Type-A female port) can supply power to the connected USB device. The power supply voltage is 5 V and the maximum current is 500 mA.

- There are two power supply methods for the USB HOST interface:
  1. Power is supplied through the USB_DEV interface, and the 5 V power is directly from the power source connected to the interface.
  2. Power is supplied through the lithium battery, and the 3.6 V ~ 4.2 V voltage of the lithium battery is boosted to 5 V through the Boost circuit. The working status of Boost IC can be controlled by BOOST_EN/GPIO13, set high to enable Boost.

- **USB HOST interface power supply selection:**
500 mA current limiting circuit:

1. The current limiting IC MIC2005A can limit the maximum output current of the USB HOST interface to 500 mA. Please set the IDEV_LIMIT_EN (GPIO17) to high level to enable the current-limiting IC to output...
voltage.

![Fig. 10: 500 mA Current Limiting Circuit (click to enlarge)](image_url)

**USB Interface Switch Circuit**

- When **USB_SEL** (GPIO18) is set to high level, the USB D+/D- Pin (GPIO19, 20) will be connected to **USB_DEV D+ D-**. Then you can use the **USB HOST** interface (Type-A female Port) to connect other USB devices.

- When **USB_SEL** (GPIO18) is set to low level, the USB D+/D- Pin (GPIO19, 20) will be connected to **USB_HOST D+ D-**. Then you can use the **USB DEV** interface (Type-A male port) to connect to host like PC.

  - **USB_SEL** is pulled low level by default.

**LCD Interface**

Please note that this interface supports connecting SPI interface screens. The screen controller used by this development board is ```dev-kits: ST7789 <esp32-s3-usb-otg/datasheet/ST7789VW_datasheet.pdf>```, and **LCD_BL** (GPIO9) can be used to control the screen backlight.

**SD Card Interface**

Please note that the SD card interface is compatible with 1-wire, 4-wire SDIO mode and SPI mode. After powered on, the card will be in 3.3 V signaling mode. Please send the first CMD0 command to select the bus mode: SD mode or SPI mode.
Fig. 11: USB Interface Switch Circuit (click to enlarge)

Fig. 12: LCD Interface Circuit (click to enlarge)
Charging Circuit

Please note that the Type-A male port can be connected to a power adapter that outputs 5 V. When charging the battery, the red indicator LED is on, after fully charged, the red indicator LED is off. When using the charging circuit, please set the battery switch to OFF. The charging current is 212.7 mA.

Pin Layout

Function pin:
### Chapter 1. Supported Development Boards

**Fig. 14: Charging Circuit (click to enlarge)**

<table>
<thead>
<tr>
<th>No.</th>
<th>ESP32-S3-MINI-1 Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO18</td>
<td>USB_SEL: Used to switch the USB interface. When high level, the USB_HOST interface is enabled. When low level, the USB_DEV interface is enabled.</td>
</tr>
<tr>
<td>2</td>
<td>GPIO19</td>
<td>Connect with USB D-.</td>
</tr>
<tr>
<td>3</td>
<td>GPIO20</td>
<td>Connect with USB D+.</td>
</tr>
<tr>
<td>4</td>
<td>GPIO15</td>
<td>LED_GREEN: the light is lit when set high level.</td>
</tr>
<tr>
<td>5</td>
<td>GPIO16</td>
<td>LED_YELLOW: the light is lit when set high level.</td>
</tr>
<tr>
<td>6</td>
<td>GPIO0</td>
<td>BUTTON_OK: OK button, low level when pressed.</td>
</tr>
<tr>
<td>7</td>
<td>GPIO11</td>
<td>BUTTON_DW: Down button, low level when pressed.</td>
</tr>
<tr>
<td>8</td>
<td>GPIO10</td>
<td>BUTTON_UP: UP button, low level when pressed.</td>
</tr>
<tr>
<td>9</td>
<td>GPIO14</td>
<td>BUTTON_MENU: Menu button, low level when pressed.</td>
</tr>
<tr>
<td>10</td>
<td>GPIO8</td>
<td>LCD_RET: used to reset LCD, low level to reset.</td>
</tr>
<tr>
<td>11</td>
<td>GPIO5</td>
<td>LCD_EN: used to enable LCD, low level to enable.</td>
</tr>
<tr>
<td>12</td>
<td>GPIO4</td>
<td>LCD_DC: Used to switch data and command status.</td>
</tr>
<tr>
<td>13</td>
<td>GPIO6</td>
<td>LCD_SCLK: LCD SPI Clock.</td>
</tr>
<tr>
<td>14</td>
<td>GPIO7</td>
<td>LCD_SDA: LCD SPI MOSI.</td>
</tr>
<tr>
<td>15</td>
<td>GPIO9</td>
<td>LCD_BL: LCD backlight control.</td>
</tr>
<tr>
<td>16</td>
<td>GPIO36</td>
<td>SD_SCK: SD SPI CLK / SDIO CLK.</td>
</tr>
<tr>
<td>17</td>
<td>GPIO37</td>
<td>SD_DO: SD SPI MISO / SDIO Data0.</td>
</tr>
<tr>
<td>18</td>
<td>GPIO38</td>
<td>SD_D1: SDIO Data1.</td>
</tr>
<tr>
<td>19</td>
<td>GPIO33</td>
<td>SD_D2: SDIO Data2.</td>
</tr>
<tr>
<td>20</td>
<td>GPIO34</td>
<td>SD_D3: SD SPI CS / SDIO Data3.</td>
</tr>
<tr>
<td>21</td>
<td>GPIO1</td>
<td>HOST_VOL: USB_DEV voltage monitoring, ADC1 channel 0.</td>
</tr>
<tr>
<td>22</td>
<td>GPIO2</td>
<td>BAT_VOL: Battery voltage monitoring, ADC1 channel 1.</td>
</tr>
<tr>
<td>23</td>
<td>GPIO17</td>
<td>LIMIT_EN: Enable current limiting IC, high level enable.</td>
</tr>
<tr>
<td>24</td>
<td>GPIO21</td>
<td>0VER_CURRENT: Current overrun signal, high level means overrun.</td>
</tr>
<tr>
<td>25</td>
<td>GPIO12</td>
<td>DEV_VBUS_EN: High level to enable DEV_VBUS power supply.</td>
</tr>
<tr>
<td>26</td>
<td>GPIO13</td>
<td>BOOST_EN: High level to enable Boost boost circuit.</td>
</tr>
</tbody>
</table>

Trun off for long battery life
Charge current: 1000/4.7k=212.7mA
Extended pin:

<table>
<thead>
<tr>
<th>No.</th>
<th>ESP32-S3-MINI-1 Pin</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>GPIO45</td>
<td>FREE_1: Idle, can be customized.</td>
</tr>
<tr>
<td>2</td>
<td>GPIO46</td>
<td>FREE_2: Idle, can be customized.</td>
</tr>
<tr>
<td>3</td>
<td>GPIO48</td>
<td>FREE_3: Idle, can be customized.</td>
</tr>
<tr>
<td>4</td>
<td>GPIO26</td>
<td>FREE_4: Idle, can be customized.</td>
</tr>
<tr>
<td>5</td>
<td>GPIO47</td>
<td>FREE_5: Idle, can be customized.</td>
</tr>
<tr>
<td>6</td>
<td>GPIO3</td>
<td>FREE_6: Idle, can be customized.</td>
</tr>
</tbody>
</table>

Related Documents

- [ESP32-S3 Datasheet](#) (PDF)
- [ESP32-S3-MINI-1/1U Datasheet](#) (PDF)
- [Espressif Product Selection Tool](#)
- [ESP32-S3-USB-OTG Schematic Diagram](#) (PDF)
- [ESP32-S3-USB-OTG PCB Layout Drawing](#) (PDF)
- [ST7789VW Datasheet](#) (PDF)

1.1.2 ESP32-S2-Kaluga-1

The ESP32-S2-Kaluga-1 kit v1.3 is a development kit by Espressif.

**ESP32-S2-Kaluga-1**

For the latest version, please go to [ESP32-S2-Kaluga-1 Kit](#).

Reference Documentation

Schematic

- [ESP32-S2-Kaluga-1 v1.2 Mainboard](#) (PDF)
- [ESP-LyraT-8311A v1.2 Audio Board](#) (PDF)
- [ESP-LyraP-LCD32 v1.2 Screen](#) (PDF)
- [ESP-LyraP-TOUCHA v1.2 Touchpad](#) (PDF)
- [ESP-LyraP-CAM v1.2 Camera](#) (PDF)
1.1.3 ESP32-S2-Touch-Devkit-1

ESP32-S2-Touch-Devkit-1 is a development kit that is aimed at helping evaluate and develop capacitive touch sensor applications on ESP32-S2. It is made up of Motherboard-Subboard structure.

Fig. 15: ESP32-S2-Touch-Devkit-1 Board-set
Overview

ESP32-S2-Touch-Devkit-1 is a development kit that is aimed at helping evaluate and develop capacitive touch sensor applications on ESP32-S2. It is made up of Motherboard-Subboard structure. The motherboard of ESP32-S2-Touch-Devkit-1 integrates ESP32-S2-MINI-1 controller module and several useful little components such as buzzer, digital tube, RGB light, and so on. There are several kinds of subboards in ESP32-S2-Touch-Devkit-1 with different kinds of capacitive touch sensor pads, developers can choose one of them and connect it with motherboard so that they could develop different kinds of capacitive touch sensor applications. The motherboard and subboard use the pin header/female pin header as the socket connector which makes it plug in and plug out smoothly.

Motherboard

![Motherboard Image]

Fig. 16: Motherboard

Subboards

- Button-board: Three capacitive touch button with waterproof sensor.
Button-board Water-proof sensor

- Slider-board: Capacitive touch linear slider, the relative distance of slider has up to 8 bit precision.

Slider-board Slider

- Matrix-board: 3 x 4 capacitive touch matrix button, 7 channels of touch sensor make up of 12 capacitive touch buttons.
Matrix-board Matrix button

- Touchpad-board: 7 x 6 two-dimension capacitive touchpad.

Touchpad-board Touchpad

- Proximity-board: Three capacitive touch proximity sensor.
Get Started

Install SDK and it's dependencies:

1. Install ESP-IDF, please refer to ESP-IDF Programming Guide.
2. Use the release-4.3 version of ESP-IDF.

```
cd esp-idf  #Enter esp-idf folder
git checkout release/v4.3  #Checkout release-v4.3 version of ESP-IDF

git submodule update  #Maybe is needed
```

Get ESP32-S2-Touch-Devkit-1 project:

1. Clone esp-dev-kits repo.

```
git clone https://github.com/espressif/esp-dev-kits.git
```

2. Enter ESP32-S2-Touch-Devkit-1 project folder.

```
cd esp-dev-kits/esp32-s2-touch-devkit-1
```

The project's structure is show as followed:

```
├── CMakeLists.txt
│   └── components  #components' driver
│       └── board_detect  #subboard detector
│       └── buzzer  #buzzer driver
│       └── digital_tube  #digital tube controller driver
│       └── rgb_light  #RGB light driver(ws2812)
│       └── subboards  #Subboards' application source file
│       └── touch_element  #Touch element library
│       └── main  #Main application demo logic
```

(continues on next page)
Build & Flash ESP32-S2-Touch-Devkit-1 demo project:

1. Enable ESP-IDF environment variable.

```bash
cd esp-idf
./export.sh
```

2. Build & Flash.

```bash
cd esp-dev-kits/esp32-s2-touch-devkit-1
idf.py set-target esp32s2 #Enable esp32s2 platform
idf.py build flash
```


```bash
idf.py monitor
```

Example output:

```
I (2880) Touch Demo: Slider sub-board plug in
I (22480) Touch Demo: Slider sub-board plug out
I (22480) Touch Demo: Nothing detected
I (41540) Touch Demo: Touchpad sub-board plug in
I (47700) Touchpad Board: Touchpad pressed, position: [0, 5]
I (47710) Touchpad Board: Position: [0, 5]
I (47720) Touchpad Board: Position: [0, 5]
I (47730) Touchpad Board: Position: [0, 6]
I (47740) Touchpad Board: Position: [0, 6]
I (47750) Touchpad Board: Position: [0, 6]
I (47760) Touchpad Board: Position: [0, 6]
I (47770) Touchpad Board: Position: [0, 7]
I (47780) Touchpad Board: Position: [0, 8]
I (47790) Touchpad Board: Position: [0, 9]
I (47800) Touchpad Board: Position: [0, 9]
I (47810) Touchpad Board: Position: [1, 10]
I (47820) Touchpad Board: Position: [2, 11]
I (47830) Touchpad Board: Position: [2, 12]
I (47840) Touchpad Board: Position: [3, 13]
I (47850) Touchpad Board: Position: [4, 14]
I (47860) Touchpad Board: Position: [5, 15]
I (47870) Touchpad Board: Position: [6, 16]
I (47880) Touchpad Board: Position: [7, 16]
I (47890) Touchpad Board: Position: [9, 17]
I (47900) Touchpad Board: Position: [10, 18]
I (47910) Touchpad Board: Position: [11, 18]
I (47920) Touchpad Board: Position: [11, 19]
I (47930) Touchpad Board: Position: [12, 20]
I (47940) Touchpad Board: Position: [13, 21]
I (47950) Touchpad Board: Position: [14, 21]
I (47960) Touchpad Board: Position: [14, 22]
I (47970) Touchpad Board: Position: [15, 23]
I (47980) Touchpad Board: Position: [15, 23]
I (47990) Touchpad Board: Position: [15, 24]
I (48000) Touchpad Board: Position: [15, 24]
```

(continues on next page)

1.1. Related
I (48010) Touchpad Board: Position: [15, 24]
I (48020) Touchpad Board: Position: [16, 24]
I (48030) Touchpad Board: Position: [16, 24]
I (48040) Touchpad Board: Position: [16, 24]
I (48050) Touchpad Board: Position: [16, 23]
I (48060) Touchpad Board: Position: [16, 23]
I (48070) Touchpad Board: Position: [16, 22]
I (48080) Touchpad Board: Position: [16, 22]
I (48090) Touchpad Board: Position: [16, 21]
I (48100) Touchpad Board: Touchpad released, position: [16, 21]

Notes

- Some new Touch Sensor features (Touchpad, Touch proximity) are not supported in ESP-IDF Touch Element library, so we copy Touch Element from ESP-IDF components into this demo project's components' folder and add the necessary features. They will appear in the future version of ESP-IDF.

Troubleshooting

Q1: Why Proximity-board is connected with Motherboard, they don't work or work abnormally?
A1: Though all of those subboards are hot-swappable theoretically, the Proximity-board needs to startup in an ideal environment (Far away from your hands). If it goes wrong, you can reset it mandatorily by releasing the power-switch.

Related Documents

Schematic

- ESP32-S2-Touch-Devkit-1 Motherboard Schematic
- Button Subboard Schematic
- Slider Subboard Schematic
- Matrix Button Subboard Schematic
- Touchpad Subboard Schematic
- Proximity Subboard Schematic

Other documents

- Touch Element Library Programming Guide
- ESP32-S2-MINI-1 Datasheet
- ESP32-S2 Datasheet
1.1.4 ESP32-LCDKit

ESP32-LCDKit is an HMI (Human Machine Interface) development board with the ESP32-DevKitC at its core.

Overview

ESP32-LCDKit is an HMI (Human Machine Interface) development board with the ESP32-DevKitC at its core. ESP32-LCDKit is integrated with such peripherals as SD-Card, DAC-Audio, and can be connected to an external display. The board is mainly used for HMI-related development and evaluation. Development board reserved screen interface type: SPI serial interface, 8-bit parallel interface, 16-bit parallel interface.

You may find HMI-related examples running with ESP32-LCDKit in HMI Example.

For more information on ESP32, please refer to ESP32 Series Datasheet.
Block Diagram and PCB Layout

Block Diagram

The figure below shows the block diagram for ESP32-LCDKit.

![Block Diagram](image)

Fig. 18: ESP32-LCDKit Block Diagram

PCB Layout

The figure below shows the layout of ESP32-LCDKit PCB.

Descriptions of PCB components are shown in the following table:

<table>
<thead>
<tr>
<th>Components</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display connection module</td>
<td>Allows to connect serial or parallel LCD displays (8/16 bit)</td>
</tr>
<tr>
<td>ESP32 DevKitC connection module</td>
<td>Offers connection to an ESP32 DevKitC development board</td>
</tr>
<tr>
<td>SD-Card module</td>
<td>Provides an SD-Card slot for memory expansion</td>
</tr>
<tr>
<td>DAC-Audio module</td>
<td>Features an audio power amplifier and two output ports for external speakers</td>
</tr>
</tbody>
</table>
Fig. 19: ESP32-LCDKit PCB Layout
Functional Modules

This section introduces the functional modules (interfaces) of ESP32-LCDKit and their hardware schematics.

- Schematic
- PCB Layout

ESP32 DevKitC Connection Module

For the HMI-related development with ESP32-LCDKit, you also need the ESP32 DevKitC development board. The figure below shows the schematics for the ESP32 DevKitC connection module.

Core Board Connector:

![Fig. 20: ESP32 DevKitC Connection Module](image)

Power Supply Management Module

The figure below shows the schematics for the USB power supply management module.

Display Connection Module

The display connection module supports the following interfaces:

- SPI serial interface
- 8-bit parallel interface
- 16-bit parallel interface

With this module, you can connect ESP32-LCDKit to an external display and interact with the pre-programmed GUI if the display has a touchscreen.

The figure below shows the schematics for this module.
**Power:**

![Diagram of Power Supply Module](image)

**Serial Screen Connector:**

![Diagram of Serial Screen Connector](image)

---

1.1. Related

---

Fig. 21: ESP32-LCDKit Power Supply Module

Fig. 22: ESP32-LCDKit Display Connection Module
SD-Card and DAC-Audio Modules

The SD-Card module provides an SD Card slot for memory expansion. The DAC-Audio module features the MIX3006 power amplifier and two output ports for connection of external speakers.

The figure below shows the schematics for the SD-Card and DAC-Audio modules.

Fig. 23: SD-Card and DAC-Audio Modules
Related Documents

- ESP32-LCDKit Schematic
- ESP32-LCDKit PCB Layout

1.1.5 ESP-Prog

ESP-Prog is one of Espressif’s development and debugging tools, with functions including automatic firmware downloading, serial communication, and JTAG online debugging.

ESP-Prog

This user guide will help you get started with ESP-Prog and will also provide more in-depth information.

ESP-Prog is one of Espressif’s development and debugging tools, with functions including automatic firmware downloading, serial communication, and JTAG debugging. ESP-Prog’s automatic firmware downloading and serial communication functions are supported on ESP8266, ESP32, ESP32-S2, ESP32-S3, and ESP32-C3, while the JTAG debugging is supported only on ESP32, ESP32-S2, ESP32-S3, and ESP32-C3.

ESP-Prog can be easily connected to a PC with the use of only one USB cable. Then, the PC can identify the board’s Program and JTAG interfaces (functions) by their port numbers.

Given that the power supply voltage may vary on different user boards, either of the ESP-Prog interfaces can provide 5 V or 3.3 V power supply through pin headers, in order to ensure power compatibility. Power on ESP-Prog could be toggled between 3.3 V and 5 V, but the RX/TX & JTAG signals will always be at the 3.3 V level.

The document consists of the following major sections:

- **Getting started**: Overview of the board and hardware/software setup instructions to get started.
- **Hardware Reference**: More detailed information about the board’s hardware.
- **Related Documents**: Links to related documentation.

**Getting Started**

This section provides a brief introduction of ESP-Prog on how to do the initial hardware setup.

**Description of Components**

The key components of the board are described in a clockwise direction.
Fig. 24: ESP-Prog (click to enlarge)
### Key Component

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Micro USB</td>
<td>Interface between PC and ESP-Prog.</td>
</tr>
<tr>
<td>Boot Button</td>
<td>Download button. Holding down boot and then pressing reset initiates Firmware Download mode for downloading firmware through the serial port.</td>
</tr>
<tr>
<td>IO0 On/Off</td>
<td>Pin Header to set the state of GPIO0 strapping pin.</td>
</tr>
<tr>
<td>PROG PWR SEL</td>
<td>Pin Header to select power input for the Program interface.</td>
</tr>
<tr>
<td>PROG 2.54 mm</td>
<td>Program interface with 2.54 mm (0.1&quot;) pin pitch.</td>
</tr>
<tr>
<td>PROG 1.27 mm</td>
<td>Program interface with 1.27 mm (0.05&quot;) pin pitch.</td>
</tr>
<tr>
<td>JTAG 1.27 mm</td>
<td>JTAG interface with 1.27 mm (0.05&quot;) pin pitch.</td>
</tr>
<tr>
<td>JTAG 2.54 mm</td>
<td>JTAG interface with 2.54 mm (0.1&quot;) pin pitch.</td>
</tr>
<tr>
<td>JTAG PWR SEL</td>
<td>Pin Header to select power input for the JTAG interface.</td>
</tr>
<tr>
<td>LED Indicator</td>
<td>LED to indicate ESP-Prog state. There are three LED modes: red, green, and blue. The red LED lights up when the system is connected to the 3.3 V power. The green LED lights up when ESP-Prog is downloading data. The blue LED lights up when ESP-Prog is receiving data.</td>
</tr>
<tr>
<td>USB Bridge</td>
<td>Single USB-to-UART bridge chip provides up to 3 Mbps of transfer rate.</td>
</tr>
<tr>
<td>Reset Button</td>
<td>Press this button to restart the system.</td>
</tr>
</tbody>
</table>
Start Application Development

Before powering up your board, please make sure that it is in good condition with no obvious signs of damage.

Required Hardware

- ESP-Prog
- USB 2.0 cable (Standard-A to Micro-B)
- Computer running Windows, Linux, or macOS
- Dupont lines or flat cables provided by Espressif for connecting the development board to ESP-Prog

Note: Be sure to use an appropriate USB cable. Some cables are for charging only and do not provide the needed data lines nor work for programming the boards.

Hardware Setup

1. Connect the ESP-Prog board and the PC USB port via a USB cable.
2. The PC then detects the two ports of ESP-Prog, indicating that the board is connected successfully. If the ports are not detected install the FT2232HL chip driver on your PC.
3. Select the output power voltage for the Program/JTAG interfaces, using PROG PWR SEL/JTAG PWR SEL pin headers.
4. Connect the ESP-Prog and ESP user board with the flat cables provided by Espressif.
5. Start programming (downloading) or JTAG debugging, using the official software tools or scripts provided by Espressif.

Software Setup

Please proceed to Get Started with ESP-IDF, where Section Installation Step by Step will quickly help you set up the development environment.

Contents and Packaging

Retail Orders

Each ESP-Prog board comes in an individual package.
The contents are as follows:

- Development board
  ESP-Prog
- Cables
  Two flexible flat cables:
    - One cable to connect to 2*5-PIN 2.54 mm male shrouded box header.
Fig. 26: ESP-Prog Package Contents

- One cable to connect to 2*3-PIN 1.27 mm male shrouded box header.

If you order a few samples, each board comes in an individual package in either antistatic bag or any packaging depending on your retailer.

For retail orders, please go to https://www.espressif.com/en/company/contact/buy-a-sample.

Wholesale Orders

If you order in bulk, the boards come in large cardboard boxes.

For wholesale orders, please go to https://www.espressif.com/en/contact-us/sales-questions.

Hardware Reference

Block Diagram

The block diagram below shows the components of ESP-Prog and their interconnections.
Power Supply Options

There are three mutually exclusive ways to provide power to the board:

- ESP-Prog USB Port, default power supply (recommended)
- 5 V and G (GND) pins
- 3.3 V and G (GND) pins

Header Block

The two tables below provide the Name and Function of the pins on both sides of the board (Program Interface and JTAG Interface). The pin names are shown in the front view of the ESP-Prog board. The numbering is the same as in the ESP-Prog Schematic (PDF).
Program Interface

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ESP_EN</td>
<td>Enable signal</td>
</tr>
<tr>
<td>2</td>
<td>VDD</td>
<td>Power supply</td>
</tr>
<tr>
<td>3</td>
<td>ESP_TXD</td>
<td>TX pin</td>
</tr>
<tr>
<td>4</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>5</td>
<td>ESP_RXD</td>
<td>RX pin</td>
</tr>
<tr>
<td>6</td>
<td>ESP_IO0</td>
<td>Strapping pin</td>
</tr>
</tbody>
</table>

JTAG Interface

<table>
<thead>
<tr>
<th>No.</th>
<th>Name</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>VDD</td>
<td>Power supply</td>
</tr>
<tr>
<td>2</td>
<td>ESP_TMS</td>
<td>JTAG TMS pin, mode selection</td>
</tr>
<tr>
<td>3</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>4</td>
<td>ESP_TCK</td>
<td>JTAG TCK pin, clock input</td>
</tr>
<tr>
<td>5</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>6</td>
<td>ESP_TDO</td>
<td>JTAG TDO pin</td>
</tr>
<tr>
<td>7</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>8</td>
<td>ESP_TDI</td>
<td>JTAG TDI pin</td>
</tr>
<tr>
<td>9</td>
<td>GND</td>
<td>Ground</td>
</tr>
<tr>
<td>10</td>
<td>NC</td>
<td>None</td>
</tr>
</tbody>
</table>

Related Documents

- ESP-Prog Schematic (PDF)
- ESP-Prog PCB Layout (PDF)
- ESP-Prog Dimensions (PDF)

For further design documentation for the board, please contact us at sales@espressif.com.

Reference Documentation

[Introduction to Functions]

The Working Mode of USB Bridge

ESP-Prog uses FT2232HL, which is provided by FTDI, as its USB Bridge Controller chip. The board can be configured to convert the USB 2.0 interface to serial and parallel interfaces that support a wide range of industry standards. ESP-Prog uses FT2232HL’s default dual-asynchronous serial interface mode available after installing the FT2232HL chip driver on their PCs.
Note: The PC is able to identify the ESP-Prog’s two ports by their port numbers. The bigger port number represents the Program interface, while the other one represents the JTAG interface.

Communication Interface

ESP-Prog can connect to ESP32 user boards using both the Program interface and the JTAG interface.

- **Program Interface**

  The Program interface has six pins, including the UART interface (ESP_TXD, ESP_RXD), boot mode selection pin (ESP_IO0) and reset pin (ESP_EN). The design for the Program interface on the user board should follow the reference provided in the figure below.

  ![Fig. 28: Program Interface (click to enlarge)](image)

- **JTAG Interface**

  The design for the JTAG interface on the user board should follow the reference provided in the figure below.

- **Fool-proof Design**

  The ESP-Prog board uses header connectors (DC3-6P/DC3-10P) which support reverse-current circuitry protection. In such cases, it is recommended that users also use header connectors on their user boards, such as
Fig. 29: JTAG Interface (click to enlarge)
Note: Keying of the plugs and sockets to insert the plug is in one specific orientation, which means each socket of ESP-Prog corresponds to the plugs on the cable and using a mismatched cable might lead to a wrong order of connection. Please use the cables provided by Espressif.

Automatic Downloading Function

ESP-Prog supports automatic downloading. After connecting the Program interface of ESP-Prog to the user board, the downloading program can download data or run programs automatically by controlling the states of the start-mode selection pin (ESP_IO0) and reset pin (ESP_EN), which spares the users from manually restarting the device and selecting the downloading modes. The two buttons on the ESP-Prog board enable users to reset and control the boot mode of the device manually.

The schematics of the automatic downloading circuit is displayed below.

![Fig. 30: Automatic Downloading Circuit (click to enlarge)](image)

Delay Circuit

The delay circuit of ESP-Prog includes the bus buffer, inverter, MOSFET, first-order RC circuit, and other components. This delay circuit ensures that the ESP32 chip can power up or reset itself before connecting with the JTAG signal, thus protecting the chip from the influence of JTAG on power-up or reset.
LED Status Indication

- The red LED lights up when the system is connected to the 3.3 V power.
- The green LED lights up when ESP-Prog is downloading data to ESP32.
- The blue LED lights up when ESP-Prog is receiving data from ESP32.

Pin Headers

Users can select the power supply for the Program and JTAG interfaces via the Pin Header to Select Power Supply, and select the boot modes of ESP8266 and ESP32 via the IO0 On/Off Pin.

- **Pin Header to Select Power Supply**

  The pin header in the middle is the power input pin for each interface. When this pin is connected to 5 V, the power output of the interface is 5 V. When this pin is connected to 3.3 V, the power output of the interface is 3.3 V.

- **IO0 On/Off Pin**

  Pin IO0 can be set to select ESP8266's and ESP32's boot modes. This pin can be used as a common GPIO, after the chip is powered on. By removing a jumper from the pin header, users can disconnect Pin IO0 manually to protect the operation of the user board from the influence of ESP-Prog's automatic downloading circuit.
For further design documentation for the board, please contact us at sales@espressif.com.

1.1.6 ESP32-MeshKit-Sense

ESP32-MeshKit-Sense is a development board with an ESP32 module at its core. It features peripherals, such as a temperature and humidity sensor, an ambient light sensor, etc. The board can be interfaced with screens. The board is mainly used to detect the current consumption of ESP32 modules in a normal operation state or in sleep mode, when connected to different peripherals.

ESP32-MeshKit-Sense

Overview

ESP32-MeshKit-Sense is a development board with an ESP32 module at its core. It features peripherals, such as a temperature and humidity sensor, an ambient light sensor, etc. The board can be interfaced with screens. The board is mainly used to detect the current consumption of ESP32 modules in a normal operation state or in sleep mode, when connected to different peripherals.

For more information on ESP32, please refer to ESP32 Datasheet.

Block Diagram and PCB Layout

Block Diagram

The figure below shows the block diagram of ESP32.

PCB Layout

The figure below shows the layout of ESP32-MeshKit-Sense PCB.

Functional Descriptions of PCB Layout are shown in the following table:
Fig. 34: ESP32 Block Diagram
<table>
<thead>
<tr>
<th>PCB Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>EXT5V</td>
<td>5 V input from USB</td>
</tr>
<tr>
<td>CH5V</td>
<td>input from the electrical charging chip</td>
</tr>
<tr>
<td>CHVBBA</td>
<td>output from the electrical charging chip</td>
</tr>
<tr>
<td>VBA</td>
<td>Connects to the positive electrode of the battery</td>
</tr>
<tr>
<td>SUFVCC</td>
<td>When the switch is toggled to the &quot;ON&quot; position, it is connected to the power input. When the switch is toggled to the &quot;OFF&quot; position, the power supply is disconnected.</td>
</tr>
<tr>
<td>DCVCC</td>
<td>Input from power management chip DC-DC</td>
</tr>
<tr>
<td>3.3V</td>
<td>3.3 V power output from power supply management chip</td>
</tr>
<tr>
<td>3.3V_PER</td>
<td>3.3 V power supply for all peripherals</td>
</tr>
<tr>
<td>3.3V_ESP</td>
<td>3.3 V power supply for all ESP32 modules</td>
</tr>
<tr>
<td>3.3V_SEN</td>
<td>3.3 V power supply for the three on-board sensors</td>
</tr>
<tr>
<td>3.3V_SCR</td>
<td>3.3 V power supply for the off-board screen</td>
</tr>
<tr>
<td>Charge</td>
<td>Battery charging indicator, D5 is a red light, indicating that charging is undergoing; D6 is a green light, indicating that charging is complete.</td>
</tr>
<tr>
<td>Sensor</td>
<td>Power indicator, indicating that 3.3V_Perip_Sensor is enabled</td>
</tr>
<tr>
<td>Screen</td>
<td>Power indicator, indicates that 3.3V_Perip_Screen is enabled</td>
</tr>
<tr>
<td>WiFi / IO15</td>
<td>Signal indicator, indicating that Wi-Fi connection is working properly</td>
</tr>
<tr>
<td>Network / IO4</td>
<td>Signal indicator, indicating the board is properly connected to the server</td>
</tr>
</tbody>
</table>

### Functional Modules

This chapter mainly introduces each functional module (interface) and the hardware schematics for them.

#### Power Supply Management Module

**Power Supply Management Module**

The development board can be powered by battery and the AP5056 power supply management chip can be used to charge the battery. The AP5056 is a complete constant current constant voltage linear charger for single cell lithium-ion batteries. It has 4.2 V of preset charge voltage and 1 A of programmable charge current.

When both the USB power supply and the battery power supply are available, the system selection of power supply will be: VBUS is high, Q4 is in cut-off state, VBAT (battery power) is automatically cut off from the system power supply, and the USB supplies power for the system.

The figure below shows the schematics for USB/BAT power supply management.
1.1. Related

Fig. 35: USB/BAT Power Supply Management Schematics
Power Supply Management for Peripherals

First of all, the input from the USB or BAT is converted by the power management chip into a 3.3 V voltage to power the circuit. The power management chip on the board is ETA3425, which has an output voltage of 3.3 V and a maximum output current of 600 mA.

The figure below shows the schematics for peripheral power supply.

![Peripheral Power Supply Schematics](image)

Notes:
1. $V_{out} = 1.13 \times (1 + R_1/R_2) = 3.39 V$;
2. $R_2$ is recommended to be 1M ohm for low standby current.

The main VDD33 circuit has two branches:
- ESP32_VDD33, used to power the ESP32 module
- VDD33_Perip, used to power all peripherals.

The connection between them can be controlled via the pin header and jumper cap. The figure below shows the schematics for ESP32_VDD33.

The VDD33_Perip branch circuit also has two sub-branches
- VDD33_Perip_Screen, dedicated power supply for the external screen
- VDD33_Perip_Sensor, power supply for the three sensors

The connection of the two can be controlled by the module GPIO+MOS. The figure below shows the schematics for VDD33_Perip.

Boot & UART

The development board is integrated with a PROG Header, which can be connected to an ESP-PROG development board via a cable. Users can then connect the micro USB of the ESP-PROG development board to a PC for ESP32-MeshKit-Sense firmware download and debugging.

The figure below shows the schematics for Boot & UART Circuit.
Fig. 37: ESP32_VDD33 Schematics

Fig. 38: VDD33_PeriP Schematics
Module for Wakeup from Sleep

The board has a button connected to the pin IO34, which is a pin in the RTC domain. When the chip is in sleep, pressing the button will wake up ESP32.

The figure below shows the schematics for wakeup-from-sleep module.

External Screens

The development board is integrated with a screen connector that can connect different external screens to the board via cables.

The figure below shows the schematics for external screens.

Sensors

Temperature and Humidity Sensor

The HTS221 is an ultra-compact sensor for relative humidity and temperature. A 3.3 V power supply and I2C interface on the board are dedicated to HTS221.

The figure below shows the schematics for the temperature and humidity sensor.
Switch(Wake up):

ESP32_VDD33

R171

10K(1%) R172 0R(5%) GPIO34

C65 1nF/50V(10%)

GND

Fig. 40: Wake-from-Sleep Module Schematics

Fig. 41: Schematics for External Screens
Temperature & Humidity Sensor:

![Temperature & Humidity Sensor Schematics](image)

1. I2C slave address: 0'b 101 1111 x;
2. To select I2C mode, the CS line must be tied high (i.e., connected to VDD) or left unconnected (thanks to the internal pull-up).
3. CTRL_REG3(22h) bit7:DRDY_H_L: data ready output signal active high, low(0:active-high, default;1:active low).

Ambient Light Sensor

The BH1750FVI is a digital ambient light sensor. A 3.3 V power supply and I2C interface on the board are dedicated to HTS221.

The figure below shows the schematics for the ambient light sensor.

![Ambient Light Sensor Schematics](image)

1. I2C slave address: 0'b 101 1100 x (ADDR='H'); 0'b 010 0011 x (ADDR='L');
2. DVI is I2C bus reference voltage terminal. And it is also asynchronous reset terminal. It is necessary to set to 'L' after VOC is supplied (At least 1us, DVI< ~0.4V).
**Ambient Brightness Sensor**

The APDS-9960 is a ambient brightness sensor featuring advanced gesture detection, proximity detection, digital Ambient Light Sense (ALS) and Color Sense (RGBC). It also incorporates an IR LED driver. The development board uses 3.3V power supply and I2C interface. It should be noted that this device is not surface-mounted by default.

The figure below shows the schematics for the ambient brightness sensor.

![Ambient Brightness Sensor Schematics](image)

1. I2C slave address: 0b 011 1001 X:
2. It is recommended that LEDA pin can be connected to a separate power supply from VDD. But if operating from a single supply, use a 22R resistor in series with the VDD supply line and a 1uF low ESR capacitor to filter any power supply noise.

![Fig. 44: Ambient Brightness Sensor Schematics](image)

**Example**

See esp-mdf/examples/development_kit/sense.

**Related Documents**

- ESP32-MeshKit-Sense Schematic
- ESP32-MeshKit-Sense PCB Layout

**1.1.7 ESP32-S2-HMI-DevKit-1**

ESP32-S2-HMI-DevKit-1 has been specifically designed for human-machine interfaces in smart-home automation controllers, smart speakers with display, smart alarm clocks, etc.
ESP32-S2-HMI-DevKit-1 V1.0

This user guide will help you get started with ESP32-S2-HMI-DevKit-1 and will also provide more in-depth information. ESP32-S2-HMI-DevKit-1 has been specifically designed for human-machine interfaces in smart-home automation controllers, smart speakers with display, smart alarm clocks, etc. This development kit supports rapid secondary development, since developers can take advantage of the kit’s various onboard resources and expansion interfaces, in order to develop various functions.

The main features of the board are listed below:

• **Module Embedded:** ESP32-S2-WROVER module with 4 MB flash and 2 MB PSRAM

• **Display:** 4.3-inch TFT-LCD which uses 16-bit 8080 parallel port with 480x800 resolution and 256-level hardware DC backlight adjustment circuit, connected to an I2C capacitive touch panel

• **Audio:** Audio amplifier, built-in microphone, speaker connector

• **Storage:** microSD card connector

• **Sensors:** 3-axis accelerometer, 3-axis gyroscope, ambient light sensor, temperature and humidity sensors

• **Expansion:** SPI header, TWAI interface (compatible with CAN 2.0), I2C ADC, UART/Prog header

• **LEDs:** Programmable RGB LED and IR LED

• **Buttons:** Wake Up and Reset buttons

• **USB:** 1 x USB-C OTG (DFU/CDC) port, 1 x USB-C debug port

• **Power Supply:** 5V and 3.3V power headers

• **Optional Rechargeable Battery:** 1,950 mAh single-core lithium battery with a charge IC

The document consists of the following major sections:

• **Getting started:** Overview of the board and hardware/software setup instructions to get started.

• **Hardware Reference:** More detailed information about the board’s hardware.

• **Related Documents:** Links to related documentation.

**Getting Started**

This section provides a brief introduction of ESP32-S2-HMI-DevKit-1, instructions on how to do the initial hardware setup and how to flash firmware onto it.

**Description of Components**

ESP32-S2-HMI-DevKit-1 is an HMI development board designed based on the ESP32-S2. The following figure describes its key on-board resources:

The key components of the board are described in a clockwise direction.
Fig. 45: ESP32-S2-HMI-DevKit-1 with ESP32-S2-WROVER module

Fig. 46: ESP32-S2-HMI-DevKit-1 (click to enlarge)
**Key Component** | **Description**
--- | ---
**ESP32-S2-WROVER Module** | ESP32-S2-WROVER is a powerful, generic Wi-Fi MCU module that integrates ESP32-S2. It has a PCB antenna, a 4 MB external SPI flash and an additional 2 MB PSRAM.
**Temperature & Humidity Sensor** | Temperature & humidity sensors for detecting ambient temperature and humidity. Read via the I2C bus.
**RGB LED** | Addressable RGB LED (WS2812), driven by GPIO21. Can switch between the LED and IR LED via IO expander.
**Ambient Light Sensor** | Ambient Light Sensor used to detect ambient light intensity. Read via the I2C bus.
**Wake Up Button** | Download button. Press and hold the **Boot** button while pressing the **Reset** button to initiate the “firmware download” mode to download the firmware via the serial port. This button can also be configured to wake up the device from deep sleep mode.
**Reset Button** | Press this button to restart the system.
**IR LED** | Infrared emitting diode, driven by GPIO21. Can switch between RGB LED and IR LED via IO expander.
**Charge IC** | Charge battery.
**MEMS** | 3-axis accelerometer and 3-axis gyroscope.
**1950 mAh Li-ion Battery** | An optional 1950 mAh rechargeable lithium battery.
**IO Expander** | Expand GPIO through I2C bus.
**MIC** | On-board simulated microphone.
**Audio PA** | Audio amplifier.
**4.3” 800 x 480 LCD** | A 4.3-inch TFT-LCD which uses 16-bit 8080 parallel port with 480×800 resolution, and 256-level hardware DC backlight adjustment circuit. It is connected with an I2C capacitive touch panel overlay.

### Start Application Development

Before powering up your board, please make sure that it is in good condition with no obvious signs of damage.

### Required Hardware

- 1 x PC loaded with Windows, macOS or Linux (Linux operating system is recommended)
- 1 x ESP32-S2-HMI-DevKit-1
- 1 x USB-C cable (it is recommended to prepare two USB-C cables if you want to evaluate MCU’s USB functions)
- 1 x Speaker (8 Ohm, 2 W)
- 1 x microSD card (some examples may have large storage needs)

**Note:** Be sure to use an appropriate USB cable. Some cables are for charging only and do not provide the needed data lines nor work for programming the boards.
Hardware Setup

To facilitate your quick evaluation of all examples, please follow these steps to set up the board:

1. Insert microSD card into the connector. Please make sure all the important data is backed up, as the microSD card may be formatted if its partition format is not FAT.

2. If you need to evaluate the audio playback function, please connect the speaker pad near the USB port on the bottom of the board to the supplied speaker, or to another speaker with similar size (8 Ohm, 2 W).

Software Setup

First, please make sure you have configured the ESP-IDF development environment correctly. To ensure this, please enter `idf.py --version` in your terminal and if the output is similar to ESP-IDF v4.2-dev-2084-g98d5b5dfd, it means you have installed ESP-IDF correctly. For detailed information about installation and configuration, please refer to ESP-IDF Get Started.

After configuration completed, you can switch back to the `esp-dev-kits/esp32-s2-hmi-devkit-1` directory. All code examples are placed under the examples directory, you can build projects by running `idf.py build`.

Project Options

Various examples are provided for ESP32-S2-HMI-DevKit-1 as shown below:

- Printing “Hello world!” on screen: `esp32-s2-hmi-devkit-1/examples/get-started/hello_world`
- Blinking WS2812 LED and showing the color on screen: `esp32-s2-hmi-devkit-1/examples/get-started/led_blink`
- Starting a UI to configure Wi-Fi credential: `esp32-s2-hmi-devkit-1/examples/get-started/provision`
- Acquiring audio with ADC from the output of analog MIC: `esp32-s2-hmi-devkit-1/examples/audio/audio_record`
- Playing music: `esp32-s2-hmi-devkit-1/examples/audio/music_player`
- Shutting down selected board area into a deep sleep: `esp32-s2-hmi-devkit-1/examples/examples/power`
- Using Freetype to render fonts: `esp32-s2-hmi-devkit-1/examples/examples/freetype`
- Using on-board sensors: `esp32-s2-hmi-devkit-1/examples/examples/sensors`
- Using smart panel: `esp32-s2-hmi-devkit-1/examples/examples/smart-panel`
- Viewing files on SD card: `esp32-s2-hmi-devkit-1/examples/examples/storage/sdcard_fatfs`
- USB flash disk: `esp32-s2-hmi-devkit-1/examples/examples/storage/usb_msc`

You can configure project options by entering `idf.py menuconfig` in each example directory.

Please make sure to correctly configure the following options in menuconfig:

- (Top) > HMI Board Config > HMI board: Select board version. By default, please select ESP32-S2-HMI-DevKit-V2;
- (Top) > HMI Board Config > Audio HAL: Select audio output interface, whether to use PWM or DAC;
- (Top) > HMI Board Config > LCD Drivers: Select display IC type for LCD. By default, ESP32-S2-HMI-DevKit-1 uses RM68120 as its display IC;
- In (Top) > Component config > ESP32S2-specific, please go to the Support for external, SPI-connected RAM option:
– Go to SPI RAM config > Set RAM clock speed, and set the PSRAM clock as 80 MHz clock speed;

• (Top) -> Component config -> FreeRTOS: set Tick rate (Hz) as 1000.

In each example folder, we have provided a default configuration file named \texttt{sdkconfig.defaults}, with above options configured correctly.

### ESP-IDF Version Dependencies

The \texttt{esp32-s2-hmi-devkit-1/examples/storage/usb_msc} example needs to be built in IDF v4.3, while other examples can be built in IDF v4.2 and later versions.

\textbf{Notice:} Due to strict regulation on battery export, for deliveries outside of China mainland, we are shipping ESP32-S2-HMI-DevKit-1 without the battery. As a substitute, you can use the iPhone 5 replacement battery (the connectortype is non-standard). The battery capacity is not critical.

### Contents and Packaging

#### Retail Orders

If you order one or several samples of the kit, each ESP32-S2-HMI-DevKit-1 development kit comes in an individual package.

![ESP32-S2-HMI-DevKit-1 package](image-url)
The contents are as follows:

- Development board - ESP32-S2-HMI-Devit-1
- Cables - SH1.25 to 2.54mm cables x 7

For retail orders, please go to https://www.espressif.com/en/company/contact/buy-a-sample.

**Wholesale Orders**

If you order in bulk, the boards come in large cardboard boxes.

For wholesale orders, please go to https://www.espressif.com/en/contact-us/sales-questions.

**Hardware Reference**

**Block Diagram**

The block diagram below shows the components of ESP32-S2-HMI-DevKit-1 and their interconnections.

---

![ESP32-S2-HMI-DevKit-1 block diagram](image)

**Fig. 48: ESP32-S2-HMI-DevKit-1 block diagram**
Power Supply Options

The power of the ESP32-S2-HMI-DevKit-1 development board is divided into a 5 V power domain and a 3.3 V power domain, so as to reduce power consumption, improve power efficiency and support battery charging, part of which can be controlled by software whereas the other part is fixed in the hardware design.

To reduce current consumption, the preloaded firmware will power off all controlled power domains and put all ICs to low-power mode.

For more information, please refer to Power.

Connectors

It provides multiple extended interfaces for customized development. The connectors of the board are described in a clockwise direction. Please refer to ESP32-S2-HMI-DevKit-1 key on-board resources.

<table>
<thead>
<tr>
<th>Connectors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Speaker Connector</td>
<td>To connect a speaker.</td>
</tr>
<tr>
<td>4.3” LCD FPC Connector</td>
<td>(Reserved) Connect to the supported 4.3” LCD extension board using the FPC cable.</td>
</tr>
<tr>
<td>USB DFU/CDC</td>
<td>1 x USB-C OTG (DFU/CDC) port, 1 x USB-C debug port.</td>
</tr>
<tr>
<td>3.3 V Connector</td>
<td>3.3 V power header.</td>
</tr>
<tr>
<td>5 V Connector</td>
<td>5 V power header.</td>
</tr>
<tr>
<td>I2C Connector</td>
<td>I2C connector with 5 V/3.3 V power supply options.</td>
</tr>
<tr>
<td>TWAI interface (compatible with CAN 2.0)</td>
<td>Two-wire automotive interface.</td>
</tr>
<tr>
<td>SPI</td>
<td>Can connect devices on the SPI bus if the SD card is not in use.</td>
</tr>
<tr>
<td>Prog/UART</td>
<td>This interface is used to observe log output and firmware flash.</td>
</tr>
<tr>
<td>microSD Connector</td>
<td>Insert microSD card to expand the storage space of the device.</td>
</tr>
<tr>
<td>Battery Connector</td>
<td>To connect a battery.</td>
</tr>
</tbody>
</table>

Below is the back view of the board for your reference.

Related Documents

- ESP32-S2 Datasheet (PDF)
- ESP32-S2-WROVER Datasheet (PDF)
- ESP32-S2-HMI-DevKit-1 Schematic (PDF)
- ESP32-S2-HMI-DevKit-1 PCB Layout (PDF)
- ESP32-S2-HMI-DevKit-1 Dimensions (PDF)
- ESP32-S2-HMI-DevKit-1 Dimensions Source File (DXF) - You can view it with Autodesk Viewer online

For further design documentation for the board, please contact us at sales@espressif.com.
Fig. 49: ESP32-S2-HMI-DevKit-1 - back view
Audio

The ESP32-S2-HMI-DevKit-1 development board supports audio playback and recording. You can find such examples under the `esp32-s2-hmi-devkit-1/examples/audio/` directory.

Audio Playback

The ESP32-S2-HMI-DevKit-1 development board can output audio via DAC or PWM. It is recommended to use PWM for audio output since it has lower noise and higher resolution (DAC has 8-bit resolution, while PWM can reach up to 12-bit resolution at 19.2 kHz of sampling rate).

The output signal generated through the IO port goes to the digital potentiometer TPL0401 first for lossless volume adjustment, and then passes the 100 nF isolation capacitor C33 and the 200 kOhm resistor R52. This RC circuit controls the cut-off frequency at around 8 Hz. On top of that, this signal will be sent to the 3 W class-D audio power amplifier NS4150 to set the gain to 1.5 times, thus amplifying the maximum output signal from 3.3 V to 4.95 V (slightly lower than the PA supply, 5 V) so as to maximize the output volume while minimizing saturation distortion.

Fig. 50: ESP32-S2-HMI-DevKit-1 audio playback schematic (click to enlarge)

The audio PA is powered by the 5 V power domain. Before using the audio playback function, please make sure this power domain is powered on (refer to 5 V Power Domain section).

Audio Recording

The ESP32-S2-HMI-DevKit-1 development board can record audio data from the analog microphone via an internal ADC.

The board is equipped with an analog microphone with a sensitivity of -38 dB. And it will send the output signal to the operational amplifier TLV6741 with a fixed gain to amplify the signal.

The microphone and operational amplifier mentioned above are powered by a controlled 3.3 V power domain. Before using the audio recording function, please make sure this power domain is powered on (refer to 3.3 V Power Domain section).

Please use the Timer Group interrupt to record audio data. Do not use code such as the following format in tasks for audio recording:
The above format will cause the CPU to be occupied, thus triggering the task watchdog (if it is not disabled), and make other tasks with lower priorities (e.g., IDLE Task) not able to operate normally.

When recording data via ADC with the interrupt function, you need to re-write the ADC recording function to IRAM_ATTR so as to reduce response time, and place the variables to DRAM. Also, please do not use any semaphore in this function. For more information about implementation examples, please refer to audio/audio_record under the examples directory.

**ADC Accuracy**

The ADC of ESP32-S2 has a high level of repeatability despite the fact that the lack of reference voltage and using Buck power supply may result a high overall noise.

The ADC is configured with 13-bit resolution and 11 dB attenuation, corresponding to a full-scale voltage of 2.6 V. After polling the 2.5 V voltage of the AD584T reference voltage via ADC1_CH8, we convert the 4096-time uncalibrated raw values into voltage values and get the following data:

As shown in the above figure, most of the uncalibrated data error is within the range of ±0.005 V with a standard deviation (σ) of 3.393 LSB (0.00106 V). These errors are mainly from the absolute accuracy, i.e., the bias. Therefore, the distortion and noise of sounds sampled via ADC can be kept at a relatively low level.

The AD584T has a peak output noise of 50 uV at 10 V within the range of 0.1 ~ 10 Hz, and a peak output noise divided by the internal high-precise laser-adjusted resistor at 2.5 V. And an up to 30 mA push-pull capability is provided by the transistor measured at 10 V. Its output noise at 2.5 V is lower than the resolving power of a 16-bit ADC, which therefore can be used as a testing reference.
Display and Touch Panel

The ESP32-S2-HMI-DevKit-1 has a 4.3-inch color TFT-LCD with 800×480 resolution, and a touch panel with an I2C interface. The interface type and data bit width of this display is controlled by programmable pins. This development board has been configured as 16-bit 8080 parallel communication via resistors.

This LCD uses RM68120 as its display IC and FT5436 as its touch IC.

Communication

The display IC of the LCD used in ESP32-S2-HMI-DevKit-1 has been configured for 16-bit 8080 parallel communication, with a total of 18 GPIOs used, i.e., 16 data lines (LCD_D0…LCD_D15), a bit clock signal (LCD_WR) and a data/command distinguish signal (LCD_DC/LCD_RS).

The touch IC uses the I2C interface to communicate with MCU and can share this interface with other I2C's ICs, and thus does not need to use additional GPIOs. The touch IC supports interrupt signal output. The interrupt signal is first sent to the P2 pin of the IO expander, and the falling edge from this pin will generate a low level in the interrupt output pin of the IO expander, so that the MCU receives this interrupt signal. In this case, you can read the input level register of the IO expander to check whether this interrupt is from the touch IC. Once a read operation completed, the interrupt flag will be cleared.
Fig. 53: ESP32-S2-HMI-DevKit-1 IO expander schematic
Backlight

As the LEDs are connected in series, they need to be driven by constant current via the Booster circuit. The rated current is 18 mA and the voltage is approximately 24 V (may not be accurate, only for reference). To prevent the feedback voltage of the Booster circuit always being 0 when the display is not connected, and thus causing high voltage loaded to both ends of the backlight filter capacitor C21, please make sure this capacitor can withstand 38 V.

Since PWM dimming may cause display flicker and some Booster IC do not support high-frequency PWM signal control, this development board provides an option to use DC dimming circuit to reach high performance, as shown in the figure below:

This DC dimming circuit inputs the VFB voltage to the operational amplifier TLV6741, whose gain resistor is a digital potentiometer that can be modified via the I2C bus. This digital potentiometer is CAT5171, with 256 levels of resolution and a maximum resistance value of 50 kOhm.

The EN pin of the Booster IC is controlled by the P7 pin of the IO expander in high level. If you want to keep the contents while turning off the display, please set this pin to low level so as to disable backlight.
Touch

The capacitive touch panel on the development board uses a touch IC with a resolution of 800×480 and supports up to 5-point touch and hardware gesture recognition.

The hardware of this display IC does not support screen rotation itself. Therefore, for scenarios where the panel is needed to be rotated, you may need to convert the data read by the touch IC through calculating its relative value to the resolution or via certain software. Multi-touch is supported by hardware and we provide some APIs for reading the multiple touch points. However, since the LVGL used in the GUI library does not support multi-touch processing for now, you may need to handle the data of these touch points in the application layer yourself.

Power

The power of the ESP32-S2-HMI-DevKit-1 development board is divided into a 5 V power domain and a 3.3 V power domain, so as to reduce power consumption, improve power efficiency and support battery charging. Part of the power domain can be controlled by software whereas the other part is configured as permanently enabled in hardware design.

To reduce current consumption, the preloaded firmware will power off all controlled power domains and put all ICs to low-power mode.

3.3 V Power Domain

Most of the ICs and modules are powered by the 3.3 V power domain, which can be divided into an uncontrolled 3.3 V power domain and a controlled 3.3 V power domain.

The uncontrolled 3.3 V power domain cannot be powered off via software, and provides power for the Buck circuit. When there is a power supply from USB, this power domain will obtain power from the 5 V input through the USB cable; when USB is disconnected, it will obtain 3.6 ~ 4.2 V power from the lithium battery. This power domain mainly provides power for the ESP32-S2-WROVER module and other devices which can enter low-power mode via software control.

The controlled 3.3 V power domain comes from the uncontrolled 3.3 V power domain and is turned on/off via a PMOS control switch, which is connected to the P4 pin of the IO expander. This power domain mainly provides power for ICs with higher static power consumption and cannot enter low-power mode.

5 V Power Domain

The 5 V power domain of the development board provides power for the audio amplifier and the TWAI® transceiver. It obtains power from the following resources:

- The USB port
- The power input from external 5 V power port
- The power passing through the Booster circuit from the lithium battery

The power obtained from USB and the external 5 V power input supplies power for all devices (except CP2102N) that require 5 V power and cannot be disconnected by software. When obtaining power from the lithium battery, the EN pin level of the Booster IC can be controlled via the P5 pin of the IO expander to enable 5 V power in high level.

The power input through the USB port on the bottom of the board is split into two lines: one provides power for CP2102N while the other becomes USB_5V after passing through a diode. The CP2102N will only be powered up when this USB port is connected, since it only needs to be in operation when the PC is connected. Any 5 V power input will cause the Booster IC to be powered off and charge the on-board lithium battery via the charging IC.
Fig. 56: ESP32-S2-HMI-DevKit-1 IO expander schematic
Power Dependencies

The following functions depend on the 5 V power domain:

- TWAI® (selects available power supply from USB 5 V or Booster 5 V automatically)
- Audio amplifier (gets power supply from USB 5 V, if it fails, will try from the battery)
- 5 V power output connector (the same as TWAI®)

The following functions depend on the controlled 3.3 V power domain:

- Micro-SD card
- Microphone and its bias circuit, and operational amplifier
- Display and touch function
- WS2812 RGB LED and IR LED
- IR LED

Power State

When the development board is connected via the USB cable, the 5 V power domain is powered on automatically and the charging IC outputs voltage to supply power for the battery. In this case, the controlled 3.3 V power domain is controlled by the P4 pin of the IO expander.

When the development board is powered by the battery, the controlled 3.3 V power domain is controlled by the P4 pin of the IO expander while the 5 V power domain is controlled by the P5 pin of the IO expander, and the charging IC will not work.

1.1.8 ESP32-Sense-Kit

The ESP32 touch sensor development kit, ESP32-Sense-Kit, is used for evaluating and developing ESP32 touch sensor system.

Overview

The ESP32 touch sensor development kit, ESP32-Sense-Kit, is used for evaluating and developing ESP32 touch sensor system. ESP32-Sense-Kit consists of one motherboard and multiple daughterboards. The motherboard contains a display unit, a main control unit and a debug unit. The daughterboards have touch electrodes in different combinations or shapes, such as linear slider, wheel slider, matrix buttons and spring buttons, depending on the application scenarios. Users can design and add their own daughterboards for special usage cases.

The following image shows the whole ESP32-Sense-Kit.
Fig. 57: ESP32-Sense-Kit
Preparation

- **Install overlay**
  
  If plastic is used for the overlay, the recommended thickness is 3 mm or less. Because air reduces touch sensitivity, any air gaps between the daughterboard and overlay must be eliminated. You can use double-sided adhesive tape to fill in the air gap. For the daughterboard with metal springs, 7 mm stud bolts should be used to install the overlay.

![Fig. 58: Install Overlay](image)

- **Install daughterboard**
  
  Use a connector to connect motherboard with daughterboard. You can use four 7 mm plastic stud bolts to have the daughterboard steadily parallel to the motherboard, as shown on the image below:

![Fig. 59: Install Daughterboard](image)

- **Set ESP-Prog debugger**
  
  ESP-Prog is used as the program download tool and power supply. ESP-Prog has two sets of jumpers: IO0 jumper and power supply jumper. Choose 5V power supply for the latter. IO0 can be used both for selecting boot mode
(download mode or working mode) and as a touch pin. As a result, it should be disconnected if used as a touch pin in working mode. The image below shows the settings for ESP-Prog.

![Fig. 60: Set ESP-Prog Debuggers](image)

- **Connect ESP-Prog with motherboard**
  The ESP-Prog has a Jtag interface and a Program interface. Connect ESP-Prog and the motherboard through the Program interface.

- **Download programs**
  Run `make menuconfig` to configure the config settings for ESP32-Sense Project, as the screenshot below shows. Run `make flash` to download programs into the development board.

- **Replace daughterboard**
  ESP32 will detect the divided voltage of the voltage divider on the daughterboard when it is powered on to identify different daughterboards. Re-power on the development board after replacing the daughterboard.

### Hardware Resources

#### Motherboard

- **Function Block Diagram**
  The image below shows the function block diagram of the motherboard.

- **Motherboard Components**
  The display unit includes three segment displays and an RGB circuit. The debug unit includes the ESP-Prog debugger interface. The main control unit includes the ESP32 module. The mini USB is the power supply.

- **Power Management System**
  The mini USB and ESP-Prog can both be the power supply for ESP32-Sense Kit. They do not interfere with each other thanks to the protection diode. The mini USB can only serve as the power supply, while ESP-Prog also supports automatic firmware downloading. The figure below shows the schematics of the power management system.

- **Display Unit**
  The display unit on the motherboard can intuitively feedback touch event. The three 7-segment displays show the location of the pad that is being touched and the duration of a touch event. The segment displays are driven by
Fig. 61: Connect ESP-Prog with Motherboard
Fig. 62: Download Programs

Fig. 63: Function Block Diagram
Fig. 64: Motherboard Components
Fig. 65: Power Management System
CH455G chip, and controlled through I2C interface. The RGB LED reflects the colors when a touch event occurs. When a finger moves on the slider, the RGB LED will show the change of colors.

The figure below shows the schematics of the display unit:

**Daughterboard**

- **Divided resistance**

  The touch electrodes are arranged in different combinations depending on the application scenario. Each daughterboard has a voltage divider that has a unique value. The program running on motherboard reads the divider value through ADC and thus each daughterboard can be identified. The voltage divider is shown below:

<table>
<thead>
<tr>
<th>Daughterboard</th>
<th>Divided resistance (Kohm)</th>
<th>ADC reading (Min)</th>
<th>ADC reading (Max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spring button</td>
<td>0</td>
<td>0</td>
<td>250</td>
</tr>
<tr>
<td>Linear slider</td>
<td>4.7</td>
<td>805</td>
<td>1305</td>
</tr>
<tr>
<td>Matrix button</td>
<td>10</td>
<td>1400</td>
<td>1900</td>
</tr>
<tr>
<td>Duplex slider</td>
<td>19.1</td>
<td>1916</td>
<td>2416</td>
</tr>
<tr>
<td>Wheel slider</td>
<td>47</td>
<td>2471</td>
<td>2971</td>
</tr>
</tbody>
</table>
Application Programs

ESP32-Sense Project within ESP32 IoT Solution repository contains the application programs for ESP32-Sense Kit. The directory structure is shown below:

```
    └── evb_adc.c //Identifies different daughterboards through ADC.
    └── evb.h //Configures settings for motherboard, including touch...
    └── evb_led.cpp //Initialization program of RGB LED.
    └── evb_seq_led.c //Driver for digital tube.
    └── evb_touch_button.cpp //Driver for touch button.
    └── evb_touch_wheel.cpp //Driver for wheel slider.
    └── evb_touch_matrix.cpp //Driver for matrix button.
    └── evb_touch_seq_slide.cpp //Driver for duplex slider.
    └── evb_touch_slider.cpp //Driver for linear slider.
    └── evb_touch_spring.cpp //Driver for spring button.
    └── Kconfig.projbuild
    └── main.cpp //Entry point.
    └── Makefile
    └── sdkconfig.defaults
```

Configure Settings

When using overlays of different thicknesses or materials, users need to reset the change rate of touch readings on each channel, that is, the sensitivity. This parameter is calculated from the pulse count value. The calculation formula is: \((\text{Non-touch value} - \text{Touch value}) / \text{Non-touch value}\), where “Non-touch value” refers to the pulse count value when there is no touch event, and “Touch value” refers to the pulse count value when a touch event occurs. Users need to take a measurement and obtain these two values. When the system is initialized, the touch threshold is automatically calculated from the change rate of touch readings. The touch threshold is directly proportional to the change rate.

When the change rate is set, users can write it into `evb.h` file.
The divided resistance on the motherboard is 10 KΩ. The table below lists the divided resistance on each daughterboard.
Demo

Spring Button

Matrix Button

1.1. Related
Related Resources

- **Schematic**
  - ESP32-Sense-Kit Mainboard Schematic
  - ESP32-Sense-Kit Subboard Schematic

- **PCB Layout**
  - ESP32-Sense-Kit Mainboard PCB Layout
  - ESP32-Sense-Kit Subboard PCB Layout

- **Set up Software Environment**
  - ESP-IDF is the SDK for ESP32. You can refer to Get Started for how to set up the ESP32 software environment.
  - ESP-Prog is the debugger for ESP32 that features download and debugging functions.

- **ESP32 IoT Solution**
  - ESP32 IoT Solution project is based on ESP-IDF and contains multiple projects.
  - ESP32-Sense Project contains the programs for ESP32-Sense Kit that can be downloaded to the development board to enable touch sensor function.

- **Hardware Manuals**
  - Please click ESP32-Sense Kit Reference Design to download the hardware resources including schematics, PCB reference design, BOM and other files.

- **Useful References**
  - Espressif website.
  - ESP32 programming guide: It hosts extensive documentation for ESP-IDF ranging from hardware guides to API reference.
  - ESP32 touch sensor design: It is the reference design manual for the ESP32 touch sensor system.

- **Technical Support**
  - If you need technical support regarding ESP32-Sense-Kit, please submit a new issue referring to the ESP32-Sense Project.

- **How to buy**
  - WeChat Account: espressif_systems.
  - Purchase consulting.

### 1.1.9 Contributions Guide

We welcome contributions to the esp-iot-solution project!
How to Contribute

Contributions to esp-iot-solution - fixing bugs, adding features, adding documentation - are welcome. We accept contributions via Github Pull Requests.

Before Contributing

Before sending us a Pull Request, please consider this list of points:

• Is the contribution entirely your own work, or already licensed under an Apache License 2.0 compatible Open Source License? If not then we unfortunately cannot accept it.

• Does any new code conform to the esp-idf: Style Guide?

• Does the code documentation follow requirements in Documenting-code?

• Is the code adequately commented for people to understand how it is structured?

• Are comments and documentation written in clear English, with no spelling or grammar errors?

• If the contribution contains multiple commits, are they grouped together into logical changes (one major change per pull request)? Are any commits with names like “fixed typo” squashed into previous commits?

• If you’re unsure about any of these points, please open the Pull Request anyhow and then ask us for feedback.

Pull Request Process

After you open the Pull Request, there will probably be some discussion in the comments field of the request itself. Once the Pull Request is ready to merge, it will first be merged into our internal git system for in-house automated testing. If this process passes, it will be merged onto the public github repository.

Legal Part

Before a contribution can be accepted, you will need to sign our contributor-agreement. You will be prompted for this automatically as part of the Pull Request process.

Related Documents

esp-dev-kits

1.1. Related
# ifndef _IOT_I2C_BUS_H_
# define _IOT_I2C_BUS_H_
#endif

#ifdef __cplusplus
extern "C"
{
#endif
//c code
#endif

/**
 * @brief
 *
 * @param port
 * @param conf
 * @return i2c_bus_handle_t
 */
(continues on next page)
/**
 * @brief Create an I2C bus instance then return a handle if created successfully.
 * @note Each I2C bus works in a singleton mode, which means for an i2c port only one
 * _-group parameter works. When
 * _iot_i2c_bus_create is called more than one time for the same i2c port, following_
 * _parameter will override the previous one.
 * @param[in] port I2C port number
 * @param[in] conf Pointer to I2C parameters
 * @return i2c_bus_handle_t Return the I2C bus handle if created successfully, return_
 * _NULL if failed.
 */

i2c_bus_handle_t iot_i2c_bus_create(i2c_port_t port, i2c_config_t* conf);

// Copyright 2019-2020 Espressif Systems (Shanghai) PTE LTD
// Licensed under the Apache License, Version 2.0 (the "License");
// you may not use this file except in compliance with the License.
// You may obtain a copy of the License at
//     http://www.apache.org/licenses/LICENSE-2.0
// Unless required by applicable law or agreed to in writing, software
// distributed under the License is distributed on an "AS IS" BASIS,
// WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
// See the License for the specific language governing permissions and
// limitations under the License.

1.1. Related
# Chapter 1. Supported Development Boards

<table>
<thead>
<tr>
<th>esp-dev-kits</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>iot_type_xxx</td>
<td>iot_sensor_xxx; iot_board_xxx; iot_storage_...</td>
<td>iot_***</td>
</tr>
<tr>
<td>type_xxx</td>
<td>imu_xxx; light_xxx; eeprom_xxx</td>
<td></td>
</tr>
<tr>
<td>name_xxx</td>
<td>mpu6050_xxx;</td>
<td>driver_xxx</td>
</tr>
<tr>
<td>xxx_creat / xxx_delete</td>
<td></td>
<td></td>
</tr>
<tr>
<td>xxx_read / xxx_write</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- get_set_xxx
- static_xxx
- g_xxx
- s_xxx
- i j k

Abbreviations-in-code

<table>
<thead>
<tr>
<th>addr</th>
<th>address</th>
<th>id</th>
<th>identifier</th>
<th>len</th>
<th>length</th>
<th>ptr</th>
<th>pointer</th>
</tr>
</thead>
<tbody>
<tr>
<td>buf</td>
<td>buffer</td>
<td>info</td>
<td>information</td>
<td>obj</td>
<td>object</td>
<td>ret</td>
<td>return</td>
</tr>
<tr>
<td>cfg</td>
<td>command</td>
<td>hdr</td>
<td>header</td>
<td>param</td>
<td>parameter</td>
<td>temp</td>
<td>temporary/temperature</td>
</tr>
<tr>
<td>cmd</td>
<td>command</td>
<td>init</td>
<td>initialize</td>
<td>pos</td>
<td>position</td>
<td>ts</td>
<td>timestamp</td>
</tr>
</tbody>
</table>

addr, buf, cfg, cmd, ctrl,
- typedef int signed_32_bit_t;

typedef enum {
    MODULE_FOO_ONE,
    MODULE_FOO_TWO,
    MODULE_FOO_THREE
} module_foo_t;

---

1. ESP-IDF

2. Related
### 3.

```c
if (condition) { // correct
    // ...
}

switch (n) { // correct
    case 0:
        // ...
}

for(int i = 0; i < CONST; ++i) { // INCORRECT
    // ...
}
```

```c
const int y = y0 + (x - x0) * (y1 - y0) / (x1 - x0); // correct
const int y = y0 + (x - x0)* (y1 - y0)/(x1 - x0); // also okay
int y_cur = -y;
++y_cur;
const int y = y0 + (x-x0)*(y1-y0)/(x1-x0); // INCORRECT
```

```c
gpio_matrix_in(PIN_CAM_D6, I2S0I_DATA_IN14_IDX, false);
gpio_matrix_in(PIN_CAM_D7, I2S0I_DATA_IN15_IDX, false);
gpio_matrix_in(PIN_CAM_HREF, I2S0I_H_ENABLE_IDX, false);
gpio_matrix_in(PIN_CAM_PCLK, I2S0I_DATA_IN15_IDX, false);
```

- •

### 4.

```c
// This is correct:
void function(int arg)
{
    // ...
}

// NOT like this:
void function(int arg) {
    // ...
}
```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}

5. //
// // /* */ */

•

void init_something()
{
    setup_dma();
    // load_resources(); // WHY is this thing commented, asks the reader?
    start_timer();
}

•

void init_something()
{
    setup_dma();
    // TODO: we should load resources here, but loader is not fully integrated yet.
    // load_resources();
    start_timer();
}

• #if 0 ... #endif #if 0 ... #endif

•

void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
    fill_dma_item(0);
    // end XXX add
    start_timer();
}
6. 🟢🟦🟦🟦

commit LF Unix

Windows git checkout CRLF Windows core.autocrlf commit LF
Github MSYS2 Unix ESP-IDF LF Unix

commit LF MSYS2 Unix IDF

```bash
git rebase --exec 'git diff-tree --no-commit-id --name-only -r HEAD | xargs dos2unix &
& git commit -a --amend --no-edit --allow-empty' master
```

```bash
dos2unix FILENAME
```

```bash
git commit --amend
```

7. 🟢🟦🟦🟦

```bash
astyle
```

```bash
tools/format.sh components/my_component/file.c
```

CMake

- ```bash
  ```
- ```bash
  ```
- ```bash
  ```
- ```bash
  (withunderscores)
  ```
- ```bash
  (WITH_UNDERSCORES)
  ```
- ```bash
cmake-lint
  ```