Read the Docs Template Documentation

Release

Read the Docs

January 21, 2017

This is the documentation for Espressif IoT Developement Framework (esp-idf). ESP-IDF is the official development framework for the ESP32 chip.

Contents:

CHAPTER 1

Set up of Toolchain for Windows

1.1 Step 1: Quick Steps

Windows doesn't have a built-in "make" environment, so as well as installing the toolchain you will need a GNU-compatible env
You don't need to use this environment all the time (you can use Eclipse or some other front-end), but it runs behind the scenes.

The quick setup is to download the Windows all-in-one toolchain & MSYS zip file from dl.espressif.com:

https://dl.espressif.com/dl/esp32_win32_msys2_environment_and_toolchain-20170111.zip

Unzip the zip file to C:and it will create an "msys32" directory with a pre-prepared environment.

1.2 Alternative Step 1: Configure toolchain & environment from scratch

As an alternative to getting a pre-prepared environment, you can set up the environment from scratch:

- Navigate to the MSYS2 installer page and download the `msys2-i686-xxxxxxx.exe` installer executable (we only support a 32-bit MSYS environment, it works on both 32-bit and 64-bit Windows.)
- Run through the installer steps, and accept the "Run MSYS2 now" option at the end. A window will open with a MSYS2 terminal.
- The ESP-IDF repository on github contains a script in the tools directory titled `windows_install_prerequisites.sh`. If you haven't downloaded the ESP-IDF yet, that's OK - you can just download that one file in Raw format from here: tools/windows/windows_install_prerequisites.sh. Save it somewhere on your computer.
- Type the path to the shell script into the MSYS2 terminal window. You can type it as a normal Windows path, but use forward-slashes instead of back-slashes. ie: `C:/Users/myuser/Downloads/windows_install_prerequisites.sh`. You can read the script beforehand to check what it does.
- If you use the 201602 MSYS2 installer, the first time you run `windows_install_prerequisites.sh` it will update the MSYS2 core system. At the end of this update, you will be prompted to close the MSYS2 terminal and re-open. When you re-open after the update, re-run `windows_install_prerequisites.sh`. The next version of MSYS2 (after 201602) will not need this interim step.
- The `windows_install_prerequisites.sh` script will download and install packages for ESP-IDF support, and the ESP32 toolchain.

Note: You may encounter a bug where svchost.exe uses 100% CPU in Windows after setup is finished, resulting in the ESP-IDF building very slowly. Terminating svchost.exe or restarting Windows will solve this problem.

1.3 Another Alternative Step 1: Just download a toolchain

If you already have an MSYS2 install or want to do things differently, you can download just the toolchain here:

https://dl.espressif.com/dl/xtensa-esp32-elf-win32-1.22.0-61-gab8375a-5.2.0.zip

If you followed one of the above options for Step 1, you won't need this download.

Important: Just having this toolchain is *not enough* to use ESP-IDF on Windows. You will need GNU make, bash, and sed at minimum. The above environments provide all this, plus a host compiler (required for menuconfig support).

1.4 Step 2: Getting the esp-idf repository from github

Open an MSYS2 terminal window by running `C:\msys32\msys2_shell.cmd`. The environment in this window is a bash shell.

Change to the directory you want to clone the SDK into by typing a command like this one: `cd "C:/path/to/dir"` (note the forward-slashes in the path). Then type `git clone --recursive https://github.com/espressif/esp-idf.git`

If you'd rather use a Windows UI tool to manage your git repositories, this is also possible. A wide range are available.

NOTE: While cloning submodules, the `git clone` command may print some output starting `': not a valid identifier...`. This is a known issue but the git clone still succeeds without any problems.

1.5 Step 3: Starting a project

ESP-IDF by itself does not build a binary to run on the ESP32. The binary "app" comes from a project in a different directory. Multiple projects can share the same ESP-IDF directory on your computer.

The easiest way to start a project is to download the Getting Started project from github.

The process is the same as for checking out the ESP-IDF from github. Change to the parent directory and run `git clone https://github.com/espressif/esp-idf-template.git`.

IMPORTANT: The esp-idf build system does not support spaces in paths to esp-idf or to projects.

1.6 Step 4: Configuring the project

Open an MSYS2 terminal window by running `C:\msys32\msys2_shell.cmd`. The environment in this window is a bash shell.

Type a command like this to set the path to ESP-IDF directory: `export IDF_PATH="C:/path/to/esp-idf"` (note the forward-slashes not back-slashes for the path). If you don't want to run this command every time you open an MSYS2 window, create a new file in `C:/msys32/etc/profile.d/` and paste this line in - then it will be run each time you open an MYS2 terminal.

Use `cd` to change to the project directory (not the ESP-IDF directory.) Type `make menuconfig` to configure your project, then `make` to build it, `make clean` to remove built files, and `make flash` to flash (use the menuconfig to set the serial port for flashing.)

If you'd like to use the Eclipse IDE instead of running `make`, check out the Eclipse setup guide in this directory.

CHAPTER 2

Set up of Toolchain for Linux

2.1 Step 0: Prerequisites

2.1.1 Install some packages

To compile with ESP-IDF you need to get the following packages:

- Ubuntu and Debian:

```
sudo apt-get install git wget make libncurses-dev flex bison gperf python python-serial
```

- Arch:

```
sudo pacman -S --needed gcc git make ncurses flex bison gperf python2-pyserial
```

2.2 Step 1: Download binary toolchain for the ESP32

ESP32 toolchain for Linux is available for download from Espressif website:

- for 64-bit Linux:

```
https://dl.espressif.com/dl/xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz
```

- for 32-bit Linux:

```
https://dl.espressif.com/dl/xtensa-esp32-elf-linux32-1.22.0-61-gab8375a-5.2.0.tar.gz
```

Download this file, then extract it to the location you prefer, for example:

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-linux64-1.22.0-61-gab8375a-5.2.0.tar.gz
```

The toolchain will be extracted into `~/esp/xtensa-esp32-elf/` directory.

To use it, you will need to update your `PATH` environment variable in `~/.bash_profile` file. To make `xtensa-esp32-elf` available for all terminal sessions, add the following line to your `~/.bash_profile` file:

```
export PATH=$PATH:$HOME/esp/xtensa-esp32-elf/bin
```

Alternatively, you may create an alias for the above command. This way you can get the toolchain only when you need it. To do this, add different line to your `~/.bash_profile` file:

```
alias get_esp32="export PATH=$PATH:$HOME/esp/xtensa-esp32-elf/bin"
```

Then when you need the toolchain you can type `get_esp32` on the command line and the toolchain will be added to your `PATH`.

2.2.1 Arch Linux Users

To run the precompiled gdb (xtensa-esp32-elf-gdb) in Arch Linux requires ncurses 5, but Arch uses ncurses 6. Backwards compatibility libraries are available in AUR for native and lib32 configurations: - https://aur.archlinux.org/packages/ncurses5-compat-libs/ - https://aur.archlinux.org/packages/lib32-ncurses5-compat-libs/

(Alternatively, use crosstool-NG to compile a gdb that links against ncurses 6.)

2.3 Alternative Step 1: Compile the toolchain from source using crosstool-NG

Instead of downloading binary toolchain from Espressif website (Step 1 above) you may build the toolchain yourself.

If you can't think of a reason why you need to build it yourself, then probably it's better to stick with the binary version. However, here are some of the reasons why you might want to compile it from source:

- if you want to customize toolchain build configuration
- if you want to use a different GCC version (such as 4.8.5)
- if you want to hack gcc or newlib or libstdc++
- if you are curious and/or have time to spare
- if you don't trust binaries downloaded from the Internet

In any case, here are the steps to compile the toolchain yourself.

- Install dependencies:
  - Ubuntu:

```
sudo apt-get install gawk gperf grep gettext libncurses-dev python python-dev automake bison
```

  - Debian:

```
TODO
```

  - Arch:

```
TODO
```

Download `crosstool-NG` and build it:

```
cd ~/esp
git clone -b xtensa-1.22.x https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
./bootstrap && ./configure --prefix=$PWD && make install
```

Build the toolchain:

```
./ct-ng xtensa-esp32-elf
./ct-ng build
chmod -R u+w builds/xtensa-esp32-elf
```

Toolchain will be built in `~/esp/crosstool-NG/builds/xtensa-esp32-elf`. Follow instructions given in the previous section to add the toolchain to your `PATH`.

2.4 Step 2: Getting ESP-IDF from github

Open terminal, navigate to the directory you want to clone ESP-IDF and clone it using `git clone` command:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF will be downloaded into `~/esp/esp-idf`.

Note the `--recursive` option! If you have already cloned ESP-IDF without this option, run another command to get all the submodules:

```
cd ~/esp/esp-idf
git submodule update --init
```

IMPORTANT: The esp-idf build system does not support spaces in paths to esp-idf or to projects.

2.5 Step 3: Starting a project

ESP-IDF by itself does not build a binary to run on the ESP32. The binary "app" comes from a project in a different directory. Multiple projects can share the same ESP-IDF directory.

The easiest way to start a project is to download the template project from GitHub:

```
cd ~/esp
git clone https://github.com/espressif/esp-idf-template.git myapp
```

This will download `esp-idf-template` project into `~/esp/myapp` directory.

2.6 Step 4: Building and flashing the application

In terminal, go to the application directory which was obtained on the previous step:

```
cd ~/esp/myapp
```

Type a command like this to set the path to ESP-IDF directory:

```
export IDF_PATH=~/esp/esp-idf
```

At this point you may configure the serial port to be used for uploading. Run:

```
make menuconfig
```

Then navigate to "Serial flasher config" submenu and change value of "Default serial port" to match the serial port you will use. Also take a moment to explore other options which are configurable in `menuconfig`.

Special note for Arch Linux users: navigate to "SDK tool configuration" and change the name of "Python 2 interpreter" from `python` to `python2`.

Now you can build and flash the application. Run:

```
make flash
```

This will compile the application and all the ESP-IDF components, generate bootloader, partition table, and application binaries, and flash these binaries to your development board.

2.7 Further reading

If you'd like to use the Eclipse IDE instead of running `make`, check out the Eclipse setup guide in this directory.

CHAPTER 3

Set up of Toolchain for Mac OS

3.1 Step 0: Prerequisites

- install pip:

```
sudo easy_install pip
```

- install pyserial

  sudo pip install pyserial

3.2 Step 1: Download binary toolchain for the ESP32

ESP32 toolchain for macOS is available for download from Espressif website:

https://dl.espressif.com/dl/xtensa-esp32-elf-osx-1.22.0-61-gab8375a-5.2.0.tar.gz

Download this file, then extract it to the location you prefer, for example:

```
mkdir -p ~/esp
cd ~/esp
tar -xzf ~/Downloads/xtensa-esp32-elf-osx-1.22.0-61-gab8375a-5.2.0.tar.gz
```

The toolchain will be extracted into `~/esp/xtensa-esp32-elf/` directory.

To use it, you will need to update your `PATH` environment variable in `~/.profile` file. To make `xtensa-esp32-elf` available for all terminal sessions, add the following line to your `~/.profile` file:

```
export PATH=$PATH:$HOME/esp/xtensa-esp32-elf/bin
```

Alternatively, you may create an alias for the above command. This way you can get the toolchain only when you need it. To do this, add different line to your `~/.profile` file:

```
alias get_esp32="export PATH=$PATH:$HOME/esp/xtensa-esp32-elf/bin"
```

Then when you need the toolchain you can type `get_esp32` on the command line and the toolchain will be added to your `PATH`.

3.3 Alternative Step 1: Compile the toolchain from source using crosstool-NG

Instead of downloading binary toolchain from Espressif website (Step 1 above) you may build the toolchain yourself.

If you can't think of a reason why you need to build it yourself, then probably it's better to stick with the binary version. However, here are some of the reasons why you might want to compile it from source:

- if you want to customize toolchain build configuration
- if you want to use a different GCC version (such as 4.8.5)
- if you want to hack gcc or newlib or libstdc++
- if you are curious and/or have time to spare
- if you don't trust binaries downloaded from the Internet

In any case, here are the steps to compile the toolchain yourself.

- Install dependencies:
  - Install either MacPorts or homebrew package manager. MacPorts needs a full XCode installation, while homebrew only needs XCode command line tools.
  - with MacPorts:

```
sudo port install gsed gawk binutils gperf grep gettext wget libtool autoconf automake
```

  - with homebrew:

```
brew install gnu-sed gawk binutils gperftools gettext wget help2man libtool autoconf automak
```

Create a case-sensitive filesystem image:

```
hdiutil create ~/esp/crosstool.dmg -volname "ctng" -size 10g -fs "Case-sensitive HFS+"
```

Mount it:

```
hdiutil mount ~/esp/crosstool.dmg
```

Create a symlink to your work directory:

```
cd ~/esp
ln -s /Volumes/ctng crosstool-NG
```

Download `crosstool-NG` and build it:

```
cd ~/esp
git clone -b xtensa-1.22.x https://github.com/espressif/crosstool-NG.git
cd crosstool-NG
./bootstrap && ./configure --prefix=$PWD && make install
```

Build the toolchain:

```
./ct-ng xtensa-esp32-elf
./ct-ng build
chmod -R u+w builds/xtensa-esp32-elf
```

Toolchain will be built in `~/esp/crosstool-NG/builds/xtensa-esp32-elf`. Follow instructions given in the previous section to add the toolchain to your `PATH`.

3.4 Step 2: Getting ESP-IDF from github

Open Terminal.app, navigate to the directory you want to clone ESP-IDF and clone it using `git clone` command:

```
cd ~/esp
git clone --recursive https://github.com/espressif/esp-idf.git
```

ESP-IDF will be downloaded into `~/esp/esp-idf`.

Note the `--recursive` option! If you have already cloned ESP-IDF without this option, run another command to get all the submodules:

```
cd ~/esp/esp-idf
git submodule update --init
```

3.5 Step 3: Starting a project

ESP-IDF by itself does not build a binary to run on the ESP32. The binary "app" comes from a project in a different directory. Multiple projects can share the same ESP-IDF directory.

The easiest way to start a project is to download the template project from GitHub:

```
cd ~/esp
git clone https://github.com/espressif/esp-idf-template.git myapp
```

This will download `esp-idf-template` project into `~/esp/myapp` directory.

IMPORTANT: The esp-idf build system does not support spaces in paths to esp-idf or to projects.

3.6 Step 4: Building and flashing the application

In Terminal.app, go to the application directory which was obtained on the previous step:

```
cd ~/esp/myapp
```

Type a command like this to set the path to ESP-IDF directory:

```
export IDF_PATH=~/esp/esp-idf
```

At this point you may configure the serial port to be used for uploading. Run:

```
make menuconfig
```

Then navigate to "Serial flasher config" submenu and change value of "Default serial port" to match the serial port you will use. Also take a moment to explore other options which are configurable in `menuconfig`.

If you don't know device name for the serial port of your development board, run this command two times, first with the board unplugged, then with the board plugged in. The port which appears the second time is the one you need:

```
ls /dev/tty.*
```

Now you can build and flash the application. Run:

```
make flash
```

This will compile the application and all the ESP-IDF components, generate bootloader, partition table, and application binaries, and flash these binaries to your development board.

3.7 Further reading

If you'd like to use the Eclipse IDE instead of running `make`, check out the Eclipse setup guide in this directory.

CHAPTER 4

Build and Flash with Make

4.1 Finding a project

As well as the esp-idf-template project mentioned in the setup guide, ESP-IDF comes with some example projects on github in the examples directory.

Once you've found the project you want to work with, change to its directory and you can configure and build it:

4.2 Configuring your project

make menuconfig

4.3 Compiling your project

make all

... will compile app, bootloader and generate a partition table based on the config.

4.4 Flashing your project

When *make all* finishes, it will print a command line to use esptool.py to flash the chip. However you can also do this from make by running:

make flash

This will flash the entire project (app, bootloader and partition table) to a new chip. The settings for serial port flashing can be configured with *make menuconfig*.

You don't need to run *make all* before running *make flash*, *make flash* will automatically rebuild anything which needs it.

4.5 Compiling & Flashing Just the App

After the initial flash, you may just want to build and flash just your app, not the bootloader and partition table:

- *make app* - build just the app.

- *make app-flash* - flash just the app.

make app-flash will automatically rebuild the app if it needs it.

(There's no downside to reflashing the bootloader and partition table each time, if they haven't changed.)

4.6 The Partition Table

Once you've compiled your project, the "build" directory will contain a binary file with a name like "my_app.bin". This is an ESP32 image binary that can be loaded by the bootloader.

A single ESP32's flash can contain multiple apps, as well as many different kinds of data (calibration data, filesystems, parameter storage, etc). For this reason a partition table is flashed to offset 0x4000 in the flash.

Each entry in the partition table has a name (label), type (app, data, or something else), subtype and the offset in flash where the partition is loaded.

The simplest way to use the partition table is to *make menuconfig* and choose one of the simple predefined partition tables:

- "Single factory app, no OTA"
- "Factory app, two OTA definitions"

In both cases the factory app is flashed at offset 0x10000. If you *make partition_table* then it will print a summary of the partition table.

For more details about partition tables and how to create custom variations, view the documentation.

CHAPTER 5

Build and Flash with Eclipse IDE

5.1 Installing Eclipse IDE

The Eclipse IDE gives you a graphical integrated development environment for writing, compiling and debugging ESP-IDF projects.

- Start by installing the esp-idf for your platform (see files in this directory with steps for Windows, OS X, Linux).
- Download the Eclipse Installer for your platform from eclipse.org.
- When running the Eclipse Installer, choose "Eclipse for C/C++ Development" (in other places you'll see this referred to as CDT.)

5.2 Setting up Eclipse

Once your new Eclipse installation launches, follow these steps:

5.2.1 Import New Project

- Eclipse makes use of the Makefile support in ESP-IDF. This means you need to start by creating an ESP-IDF project. You can use the skeleton project from github.
- Once Eclipse is running, choose File -> Import...
- In the dialog that pops up, choose "C/C++" -> "Existing Code as Makefile Project" and click Next.
- On the next page, enter "Existing Code Location" to be the directory of your IDF project. Don't specify the path to the ESP-IDF directory itself.
- On the same page, under "Toolchain for Indexer Settings" choose "Cross GCC". Then click Finish.

5.2.2 Project Properties

- The new project will appear under Project Explorer. Right-click the project and choose Properties from the context menu.
- Click on the "Environment" properties page under "C/C++ Build". Click "Add..." and enter name `V` and value `1`.
- Click "Add..." again, and enter name `IDF_PATH`. The value should be the full path where ESP-IDF is installed. *Windows users: Use forward-slashes not backslashes for this path, ie C:/Users/MyUser/Development/esp-idf.*

Windows users only, follow these two additional steps:

- On the same Environment property page, edit the PATH environment variable. Delete the existing value and replace it with `C:\msys32\usr\bin;C:\msys32\mingw32\bin;C:\msys32\opt\xtensa-esp32-elf\bin` (If you installed msys32 to a different directory then you'll need to change these paths to match).
- Click on the "C/C++ Build" top-level properties page then uncheck "Use default build command" and enter this for the custom build command: `bash ${IDF_PATH}/tools/windows/eclipse_make.sh`.

All users, continue with these steps:

Navigate to "C/C++ General" -> "Preprocessor Include Paths" property page:

- Click the "Providers" tab
- In the list of providers, click "CDT Cross GCC Built-in Compiler Settings". Under "Command to get compiler specs", replace the text `${COMMAND}` at the beginning of the line with `xtensa-esp32-elf-gcc`. This means the full "Command to get compiler specs" should be `xtensa-esp32-elf-gcc ${FLAGS} -E -P -v -dD "${INPUTS}"`.
- In the list of providers, click "CDT GCC Build Output Parser" and type `xtensa-esp32-elf-` at the beginning of the Compiler command pattern. This means the full Compiler command pattern should be `xtensa-esp32-elf-(g?cc)|([gc]\+\+)|(clang)`
- Click OK to close the Properties dialog, and choose Project -> Build to build your project.

5.2.3 Flash from Eclipse

You can integrate the "make flash" target into your Eclipse project to flash using esptool.py from the Eclipse UI:

- Right-click your project in Project Explorer (important to make sure you select the project, not a directory in the project, or Eclipse may find the wrong Makefile.)
- Select Make Targets -> Create from the context menu.
- Type "flash" as the target name. Leave the other options as their defaults.
- Now you can use Project -> Make Target -> Build (Shift+F9) to build the custom flash target, which will compile and flash the project.

Note that you will need to use "make menuconfig" to set the serial port and other config options for flashing. "make menuconfig" still requires a command line terminal (see the instructions for your platform.)

Follow the same steps to add `bootloader` and `partition_table` targets, if necessary.

5.2.4 Eclipse Troubleshooting

- `*** Make was invoked from ... However please do not run make from the sdk or a component directory; ...` - Eclipse will detect any directory with a Makefile in it as being a possible directory to run "make" in. All component directories also contain a Makefile (the wrong one), so it is important when using Project -> Make Target to always select the top-level project directory in Project Explorer.

CHAPTER 6

General Notes About ESP-IDF Programming

6.1 Application startup flow

This note explains various steps which happen before `app_main` function of an ESP-IDF application is called.

The high level view of startup process is as follows:

1. First-stage bootloader in ROM loads second-stage bootloader image to RAM (IRAM & DRAM) from flash offset 0x1000.
2. Second-stage bootloader loads partition table and main app image from flash. Main app incorporates both RAM segments and read-only segments mapped via flash cache.
3. Main app image executes. At this point the second CPU and RTOS scheduler can be started.

This process is explained in detail in the following sections.

6.1.1 First stage bootloader

After SoC reset, PRO CPU will start running immediately, executing reset vector code, while APP CPU will be held in reset. During startup process, PRO CPU does all the initialization. APP CPU reset is de-asserted in the `call_start_cpu0` function of application startup code. Reset vector code is located at address 0x40000400 in the mask ROM of the ESP32 chip and can not be modified.

Startup code called from the reset vector determines the boot mode by checking `GPIO_STRAP_REG` register for bootstrap pin states. Depending on the reset reason, the following takes place:

1. Reset from deep sleep: if the value in `RTC_CNTL_STORE6_REG` is non-zero, and CRC value of RTC memory in `RTC_CNTL_STORE7_REG` is valid, use `RTC_CNTL_STORE6_REG` as an entry point address and jump immediately to it. If `RTC_CNTL_STORE6_REG` is zero, or `RTC_CNTL_STORE7_REG` contains invalid CRC, or once the code called via `RTC_CNTL_STORE6_REG` returns, proceed with boot as if it was a power-on reset. **Note**: to run customized code at this point, a deep sleep stub mechanism is provided. Please see deep sleep documentation for this.
2. For power-on reset, software SOC reset, and watchdog SOC reset: check the `GPIO_STRAP_REG` register if UART or SDIO download mode is requested. If this is the case, configure UART or SDIO, and wait for code to be downloaded. Otherwise, proceed with boot as if it was due to software CPU reset.
3. For software CPU reset and watchdog CPU reset: configure SPI flash based on EFUSE values, and attempt to load the code from flash. This step is described in more detail in the next paragraphs. If loading code from flash fails, unpack BASIC interpreter into the RAM and start it. Note that RTC watchdog is still enabled when this happens, so unless any input is received by the interpreter, watchdog will reset the SOC in a few hundred

milliseconds, repeating the whole process. If the interpreter receives any input from the UART, it disables the watchdog.

Application binary image is loaded from flash starting at address 0x1000. First 4kB sector of flash is used to store secure boot IV and signature of the application image. Please check secure boot documentation for details about this.

6.1.2 Second stage bootloader

In ESP-IDF, the binary image which resides at offset 0x1000 in flash is the second stage bootloader. Second stage bootloader source code is available in components/bootloader directory of ESP-IDF. Note that this arrangement is not the only one possible with the ESP32 chip. It is possible to write a fully featured application which would work when flashed to offset 0x1000, but this is out of scope of this document. Second stage bootloader is used in ESP-IDF to add flexibility to flash layout (using partition tables), and allow for various flows associated with flash encryption, secure boot, and over-the-air updates (OTA) to take place.

When the first stage bootloader is finished checking and loading the second stage bootloader, it jumps to the second stage bootloader entry point found in the binary image header.

Second stage bootloader reads the partition table found at offset 0x8000. See partition tables documentation for more information. The bootloader finds factory and OTA partitions, and decides which one to boot based on data found in *OTA info* partition.

For the selected partition, second stage bootloader copies data and code sections which are mapped into IRAM and DRAM to their load addresses. For sections which have load addresses in DROM and IROM regions, flash MMU is configured to provide the correct mapping. Note that the second stage bootloader configures flash MMU for both PRO and APP CPUs, but it only enables flash MMU for PRO CPU. Reason for this is that second stage bootloader code is loaded into the memory region used by APP CPU cache. The duty of enabling cache for APP CPU is passed on to the application. Once code is loaded and flash MMU is set up, second stage bootloader jumps to the application entry point found in the binary image header.

Currently it is not possible to add application-defined hooks to the bootloader to customize application partition selection logic. This may be required to load different application image depending on a state of a GPIO, for example. Such customization features will be added to ESP-IDF in the future. For now, bootloader can be customized by copying bootloader component into application directory and making necessary changes there. ESP-IDF build system will compile the component in application directory instead of ESP-IDF components directory in this case.

6.1.3 Application startup

ESP-IDF application entry point is `call_start_cpu0` function found in `components/esp32/cpu_start.c`. Two main things this function does are to enable heap allocator and to make APP CPU jump to its entry point, `call_start_cpu1`. The code on PRO CPU sets the entry point for APP CPU, de-asserts APP CPU reset, and waits for a global flag to be set by the code running on APP CPU, indicating that it has started. Once this is done, PRO CPU jumps to `start_cpu0` function, and APP CPU jumps to `start_cpu1` function.

Both `start_cpu0` and `start_cpu1` are weak functions, meaning that they can be overridden in the application, if some application-specific change to initialization sequence is needed. Default implementation of `start_cpu0` enables or initializes components depending on choices made in `menuconfig`. Please see source code of this function in `components/esp32/cpu_start.c` for an up to date list of steps performed. Note that any C++ global constructors present in the application will be called at this stage. Once all essential components are initialized, *main task* is created and FreeRTOS scheduler is started.

While PRO CPU does initialization in `start_cpu0` function, APP CPU spins in `start_cpu1` function, waiting for the scheduler to be started on the PRO CPU. Once the scheduler is started on the PRO CPU, code on the APP CPU starts the scheduler as well.

Main task is the task which runs `app_main` function. Main task stack size and priority can be configured in `menuconfig`. Application can use this task for initial application-specific setup, for example to launch other tasks. Application can also use main task for event loops and other general purpose activities. If `app_main` function returns, main task is deleted.

6.2 Application memory layout

ESP32 chip has flexible memory mapping features. This section describes how ESP-IDF uses these features by default.

Application code in ESP-IDF can be placed into one of the following memory regions.

6.2.1 IRAM (instruction RAM)

ESP-IDF allocates part of *Internal SRAM0* region (defined in the Technical Reference Manual) for instruction RAM. Except for the first 64 kB block which is used for PRO and APP CPU caches, the rest of this memory range (i.e. from `0x40080000` to `0x400A0000`) is used to store parts of application which need to run from RAM.

A few components of ESP-IDF and parts of WiFi stack are placed into this region using the linker script.

If some application code needs to be placed into IRAM, it can be done using `IRAM_ATTR` define:

```
#include "esp_attr.h"

void IRAM_ATTR gpio_isr_handler(void* arg)
{
        // ...
}
```

Here are the cases when parts of application may or should be placed into IRAM.

- ISR handlers must always be placed into IRAM. Furthermore, ISR handlers may only call functions placed into IRAM or functions present in ROM. *Note 1:* all FreeRTOS APIs are currently placed into IRAM, so are safe to call from ISR handlers. *Note 1:* all constant data used by ISR handlers and functions called from ISR handlers (including, but not limited to, `const char` arrays), must be placed into DRAM using `DRAM_ATTR`.
- Some timing critical code may be placed into IRAM to reduce the penalty associated with loading the code from flash. ESP32 reads code and data from flash via a 32 kB cache. In some cases, placing a function into IRAM may reduce delays caused by a cache miss.

6.2.2 IROM (code executed from Flash)

If a function is not explicitly placed into IRAM or RTC memory, it is placed into flash. The mechanism by which Flash MMU is used to allow code execution from flash is described in the Technical Reference Manual. ESP-IDF places the code which should be executed from flash starting from the beginning of `0x400D0000 -- 0x40400000` region. Upon startup, second stage bootloader initializes Flash MMU to map the location in flash where code is located into the beginning of this region. Access to this region is transparently cached using two 32kB blocks in `0x40070000` — `0x40080000` range.

Note that the code outside `0x40000000 -- 0x40400000` region may not be reachable with Window ABI `CALLx` instructions, so special care is required if `0x40400000 -- 0x40800000` or `0x40800000 -- 0x40C00000` regions are used by the application. ESP-IDF doesn't use these regions by default.

6.2.3 RTC fast memory

The code which has to run after wake-up from deep sleep mode has to be placed into RTC memory. Please check detailed description in deep sleep documentation.

6.2.4 DRAM (data RAM)

Non-constant static data and zero-initialized data is placed by the linker into the 256 kB `0x3FFB0000 -- 0x3FFF0000` region. Note that this region is reduced by 64kB (by shifting start address to `0x3FFC0000`) if Bluetooth stack is used. Length of this region is also reduced by 16 kB or 32kB if trace memory is used. All space which is left in this region after placing static data there is used for the runtime heap.

Constant data may also be placed into DRAM, for example if it is used in an ISR handler (see notes in IRAM section above). To do that, `DRAM_ATTR` define can be used:

```
DRAM_ATTR const char[] format_string = "%p %x";
char buffer[64];
sprintf(buffer, format_string, ptr, val);
```

Needless to say, it is not advised to use `printf` and other output functions in ISR handlers. For debugging purposes, use `ESP_EARLY_LOGx` macros when logging from ISR handlers. Make sure that both `TAG` and format string are placed into `DRAM` in that case.

6.2.5 DROM (data stored in Flash)

By default, constant data is placed by the linker into a 4 MB region (`0x3F400000 -- 0x3F800000`) which is used to access external flash memory via Flash MMU and cache. Exceptions to this are literal constants which are embedded by the compiler into application code.

6.2.6 RTC slow memory

Global and static variables used by code which runs from RTC memory (i.e. deep sleep stub code) must be placed into RTC slow memory. Please check detailed description in deep sleep documentation.

CHAPTER 7

Build System

This document explains the Espressif IoT Development Framework build system and the concept of "components"

Read this document if you want to know how to organise a new ESP-IDF project.

We recommend using the esp-idf-template project as a starting point for your project.

7.1 Using the Build System

The esp-idf README file contains a description of how to use the build system to build your project.

7.2 Overview

An ESP-IDF project can be seen as an amalgamation of a number of components. For example, for a webserver that shows the current humidity, there could be:

- The ESP32 base libraries (libc, rom bindings etc)
- The WiFi drivers
- A TCP/IP stack
- The FreeRTOS operating system
- A webserver
- A driver for the humidity sensor
- Main code tying it all together

ESP-IDF makes these components explicit and configurable. To do that, when a project is compiled, the build environment will look up all the components in the ESP-IDF directories, the project directories and (optionally) in additional custom component directories. It then allows the user to configure the ESP-IDF project using a a text-based menu system to customize each component. After the components in the project are configured, the build process will compile the project.

7.2.1 Concepts

- A "project" is a directory that contains all the files and configuration to build a single "app" (executable), as well as additional supporting output such as a partition table, data/filesystem partitions, and a bootloader.

- "Project configuration" is held in a single file called sdkconfig in the root directory of the project. This configuration file is modified via `make menuconfig` to customise the configuration of the project. A single project contains exactly one project configuration.
- An "app" is an executable which is built by esp-idf. A single project will usually build two apps - a "project app" (the main executable, ie your custom firmware) and a "bootloader app" (the initial bootloader program which launches the project app).
- "components" are modular pieces of standalone code which are compiled into static libraries (.a files) and linked into an app. Some are provided by esp-idf itself, others may be sourced from other places.

Some things are not part of the project:

- "ESP-IDF" is not part of the project. Instead it is standalone, and linked to the project via the `IDF_PATH` environment variable which holds the path of the `esp-idf` directory. This allows the IDF framework to be decoupled from your project.
- The toolchain for compilation is not part of the project. The toolchain should be installed in the system command line PATH, or the path to the toolchain can be set as part of the compiler prefix in the project configuration.

7.2.2 Example Project

An example project directory tree might look like this:

```
- myProject/
             - Makefile
             - sdkconfig
             - components/ - component1/ - component.mk
                                         - Kconfig
                                         - src1.c
                           - component2/ - component.mk
                                         - Kconfig
                                         - src1.c
                                         - include/ - component2.h
             - main/       - src1.c
                           - src2.c
                           - component.mk

             - build/
```

This example "myProject" contains the following elements:

- A top-level project Makefile. This Makefile set the `PROJECT_NAME` variable and (optionally) defines other project-wide make variables. It includes the core `$(IDF_PATH)/make/project.mk` makefile which implements the rest of the ESP-IDF build system.
- "sdkconfig" project configuration file. This file is created/updated when "make menuconfig" runs, and holds configuration for all of the components in the project (including esp-idf itself). The "sdkconfig" file may or may not be added to the source control system of the project.
- Optional "components" directory contains components that are part of the project. A project does not have to contain custom components of this kind, but it can be useful for structuring reusable code or including third party components that aren't part of ESP-IDF.
- "main" directory is a special "pseudo-component" that contains source code for the project itself. "main" is a default name, the Makefile variable `SRCDIRS` defaults to this but can be set to look for pseudo-components in other directories.

- "build" directory is where build output is created. After the make process is run, this directory will contain interim object files and libraries as well as final binary output files. This directory is usually not added to source control or distributed with the project source code.

Component directories contain a component makefile - `component.mk`. This may contain variable definitions to control the build process of the component, and its integration into the overall project. See *Component Makefiles* for more details.

Each component may also include a `Kconfig` file defining the *component configuration* options that can be set via the project configuration. Some components may also include `Kconfig.projbuild` and `Makefile.projbuild` files, which are special files for *overriding parts of the project*.

7.2.3 Project Makefiles

Each project has a single Makefile that contains build settings for the entire project. By default, the project Makefile can be quite minimal.

Minimal Example Makefile

```
PROJECT_NAME := myProject

include $(IDF_PATH)/make/project.mk
```

Mandatory Project Variables

- `PROJECT_NAME`: Name of the project. Binary output files will use this name - ie myProject.bin, myProject.elf.

Optional Project Variables

These variables all have default values that can be overridden for custom behaviour. Look in `make/project.mk` for all of the implementation details.

- `PROJECT_PATH`: Top-level project directory. Defaults to the directory containing the Makefile. Many other project variables are based on this variable. The project path cannot contain spaces.
- `BUILD_DIR_BASE`: The build directory for all objects/libraries/binaries. Defaults to `$(PROJECT_PATH)/build`.
- `COMPONENT_DIRS`: Directories to search for components. Defaults to *$(IDF_PATH)/components*, *$(PROJECT_PATH)/components* and `EXTRA_COMPONENT_DIRS`. Override this variable if you don't want to search for components in the esp-idf & project `components` directories.
- `EXTRA_COMPONENT_DIRS`: Optional list of additional directories to search for components. Components themselves are in sub-directories of these directories, this is a top-level directory containing the component directories.
- `COMPONENTS`: A list of component names to build into the project. Defaults to all components found in the COMPONENT_DIRS directories.
- `SRCDIRS`: Directories under the main project directory which contain project-specific "pseudo-components". Defaults to 'main'. The difference between specifying a directory here and specifying it under `EXTRA_COMPONENT_DIRS` is that a directory in `SRCDIRS` is a component itself (contains a file "component.mk"), whereas a directory in `EXTRA_COMPONENT_DIRS` contains component directories which contain a file "component.mk". See the *Example Project* for a concrete case of this.

7.2.4 Component Makefiles

Each project contains one or more components, which can either be part of esp-idf or added from other component directories.

A component is any sub-directory that contains a *component.mk* file [1].

Minimal Component Makefile

The minimal `component.mk` file is an empty file(!). If the file is empty, the default component behaviour is set:

- All source files in the same directory as the makefile (`*.c`, `*.cpp`, `*.S`) will be compiled into the component library
- A sub-directory "include" will be added to the global include search path for all other components.
- The component library will be linked into the project app.

See *example component makefiles* for more complete component makefile examples.

Note that there is a different between an empty `component.mk` file (which invokes default component build behaviour) and no `component.mk` file (which means no default component build behaviour will occur.) It is possible for a component to have no *component.mk* file, if it only contains other files which influence the project configuration or build process.

Preset Component Variables

The following component-specific variables are available for use inside `component.mk`, but should not be modified:

- `COMPONENT_PATH`: The component directory. Evaluates to the absolute path of the directory containing `component.mk`. The component path cannot contain spaces.
- `COMPONENT_NAME`: Name of the component. Defaults to the name of the component directory.
- `COMPONENT_BUILD_DIR`: The component build directory. Evaluates to the absolute path of a directory inside *$(BUILD_DIR_BASE)* where this component's source files are to be built. This is also the Current Working Directory any time the component is being built, so relative paths in make targets, etc. will be relative to this directory.
- `COMPONENT_LIBRARY`: Name of the static library file (relative to the component build directory) that will be built for this component. Defaults to `$(COMPONENT_NAME).a`.

The following variables are set at the project level, but exported for use in the component build:

- `PROJECT_NAME`: Name of the project, as set in project Makefile
- `PROJECT_PATH`: Absolute path of the project directory containing the project Makefile.
- `COMPONENTS`: Name of all components that are included in this build.
- `CONFIG_*`: Each value in the project configuration has a corresponding variable available in make. All names begin with `CONFIG_`.
- `CC`, `LD`, `AR`, `OBJCOPY`: Full paths to each tool from the gcc xtensa cross-toolchain.
- `HOSTCC`, `HOSTLD`, `HOSTAR`: Full names of each tool from the host native toolchain.
- `IDF_VER`: Git version of ESP-IDF (produced by `git describe`)

[1] Actually, some components in esp-idf are "pure configuration" components that don't have a component.mk file, only a Makefile.projbuild and/or Kconfig.projbuild file. However, these components are unusual and most components have a component.mk file.

If you modify any of these variables inside `component.mk` then this will not prevent other components from building but it may make your component hard to build and/or debug.

Optional Project-Wide Component Variables

The following variables can be set inside `component.mk` to control build settings across the entire project:

- `COMPONENT_ADD_INCLUDEDIRS`: Paths, relative to the component directory, which will be added to the include search path for all components in the project. Defaults to `include` if not overridden. If an include directory is only needed to compile this specific component, add it to `COMPONENT_PRIV_INCLUDEDIRS` instead.
- `COMPONENT_ADD_LDFLAGS`: Add linker arguments to the LDFLAGS for the app executable. Defaults to `-l$(COMPONENT_NAME)`. If adding pre-compiled libraries to this directory, add them as absolute paths - ie $(COMPONENT_PATH)/libwhatever.a
- `COMPONENT_DEPENDS`: Optional list of component names that should be compiled before this component. This is not necessary for link-time dependencies, because all component include directories are available at all times. It is necessary if one component generates an include file which you then want to include in another component. Most components do not need to set this variable.
- `COMPONENT_ADD_LINKER_DEPS`: Optional list of component-relative paths to files which should trigger a re-link of the ELF file if they change. Typically used for linker script files and binary libraries. Most components do not need to set this variable.

The following variable only works for components that are part of esp-idf itself:

- `COMPONENT_SUBMODULES`: Optional list of git submodule paths (relative to COMPONENT_PATH) used by the component. These will be checked (and initialised if necessary) by the build process. This variable is ignored if the component is outside the IDF_PATH directory.

Optional Component-Specific Variables

The following variables can be set inside `component.mk` to control the build of that component:

- `COMPONENT_PRIV_INCLUDEDIRS`: Directory paths, must be relative to the component directory, which will be added to the include search path for this component's source files only.
- `COMPONENT_EXTRA_INCLUDES`: Any extra include paths used when compiling the component's source files. These will be prefixed with '-I' and passed as-is to the compiler. Similar to the `COMPONENT_PRIV_INCLUDEDIRS` variable, except these paths are not expanded relative to the component directory.
- `COMPONENT_SRCDIRS`: Directory paths, must be relative to the component directory, which will be searched for source files (`*.cpp`, `*.c`, `*.S`). Defaults to '.', ie the component directory itself. Override this to specify a different list of directories which contain source files.
- `COMPONENT_OBJS`: Object files to compile. Default value is a .o file for each source file that is found in `COMPONENT_SRCDIRS`. Overriding this list allows you to exclude source files in `COMPONENT_SRCDIRS` that would otherwise be compiled. See *Specifying source files*
- `COMPONENT_EXTRA_CLEAN`: Paths, relative to the component build directory, of any files that are generated using custom make rules in the component.mk file and which need to be removed as part of `make clean`. See *Source Code Generation* for an example.
- `COMPONENT_OWNBUILDTARGET` & *COMPONENT_OWNCLEANTARGET*: These targets allow you to fully override the default build behaviour for the component. See *Fully Overriding The Component Makefile* for more details.

- `CFLAGS`: Flags passed to the C compiler. A default set of `CFLAGS` is defined based on project settings. Component-specific additions can be made via `CFLAGS +=`. It is also possible (although not recommended) to override this variable completely for a component.
- `CPPFLAGS`: Flags passed to the C preprocessor (used for .c, .cpp and .S files). A default set of `CPPFLAGS` is defined based on project settings. Component-specific additions can be made via `CPPFLAGS +=`. It is also possible (although not recommended) to override this variable completely for a component.
- `CXXFLAGS`: Flags passed to the C++ compiler. A default set of `CXXFLAGS` is defined based on project settings. Component-specific additions can be made via `CXXFLAGS +=`. It is also possible (although not recommended) to override this variable completely for a component.

To apply compilation flags to a single source file, you can add a variable override as a target, ie:

```
apps/dhcpserver.o: CFLAGS += -Wno-unused-variable
```

This can be useful if there is upstream code that emits warnings.

7.2.5 Component Configuration

Each component can also have a Kconfig file, alongside `component.mk`. This contains contains configuration settings to add to the "make menuconfig" for this component.

These settings are found under the "Component Settings" menu when menuconfig is run.

To create a component KConfig file, it is easiest to start with one of the KConfig files distributed with esp-idf.

For an example, see *Adding conditional configuration*.

7.2.6 Preprocessor Definitions

ESP-IDF build systems adds the following C preprocessor definitions on the command line:

- `ESP_PLATFORM` — Can be used to detect that build happens within ESP-IDF.
- `IDF_VER` — Defined to a git version string. E.g. `v2.0` for a tagged release or `v1.0-275-g0efaa4f` for an arbitrary commit.

7.2.7 Build Process Internals

Top Level: Project Makefile

- "make" is always run from the project directory and the project makefile, typically named Makefile.
- The project makefile sets `PROJECT_NAME` and optionally customises other *optional project variables*
- The project makefile includes `$(IDF_PATH)/make/project.mk` which contains the project-level Make logic.
- `project.mk` fills in default project-level make variables and includes make variables from the project configuration. If the generated makefile containing project configuration is out of date, then it is regenerated (via targets in `project_config.mk`) and then the make process restarts from the top.
- `project.mk` builds a list of components to build, based on the default component directories or a custom list of components set in *optional project variables*.

- Each component can set some *optional project-wide component variables*. These are included via generated makefiles named `component_project_vars.mk` - there is one per component. These generated makefiles are included into `project.mk`. If any are missing or out of date, they are regenerated (via a recursive make call to the component makefile) and then the make process restarts from the top.
- *Makefile.projbuild* files from components are included into the make process, to add extra targets or configuration.
- By default, the project makefile also generates top-level build & clean targets for each component and sets up *app* and *clean* targets to invoke all of these sub-targets.
- In order to compile each component, a recursive make is performed for the component makefile.

To better understand the project make process, have a read through the `project.mk` file itself.

Second Level: Component Makefiles

- Each call to a component makefile goes via the `$(IDF_PATH)/make/component_wrapper.mk` wrapper makefile.
- The `component_wrapper.mk` is called with the current directory set to the component build directory, and the `COMPONENT_MAKEFILE` variable is set to the absolute path to `component.mk`.
- `component_wrapper.mk` sets default values for all *component variables*, then includes the *component.mk* file which can override or modify these.
- If `COMPONENT_OWNBUILDTARGET` and `COMPONENT_OWNCLEANTARGET` are not defined, default build and clean targets are created for the component's source files and the prerequisite `COMPONENT_LIBRARY` static library file.
- The `component_project_vars.mk` file has its own target in `component_wrapper.mk`, which is evaluated from `project.mk` if this file needs to be rebuilt due to changes in the component makefile or the project configuration.

To better understand the component make process, have a read through the `component_wrapper.mk` file and some of the `component.mk` files included with esp-idf.

7.2.8 Debugging The Make Process

Some tips for debugging the esp-idf build system:

- Appending `V=1` to the make arguments (or setting it as an environment variable) will cause make to echo all commands executed, and also each directory as it is entered for a sub-make.
- Running `make -w` will cause make to echo each directory as it is entered for a sub-make - same as `V=1` but without also echoing all commands.
- Running `make --trace` (possibly in addition to one of the above arguments) will print out every target as it is built, and the dependency which caused it to be built.
- Running `make -p` prints a (very verbose) summary of every generated target in each makefile.

For more debugging tips and general make information, see the *GNU Make Manual*.

7.2.9 Overriding Parts of the Project

Makefile.projbuild

For components that have build requirements that must be evaluated in the top-level project make pass, you can create a file called `Makefile.projbuild` in the component directory. This makefile is included when `project.mk` is evaluated.

For example, if your component needs to add to CFLAGS for the entire project (not just for its own source files) then you can set `CFLAGS +=` in Makefile.projbuild.

`Makefile.projbuild` files are used heavily inside esp-idf, for defining project-wide build features such as `esptool.py` command line arguments and the `bootloader` "special app".

Note that `Makefile.projbuild` isn't necessary for the most common component uses - such as adding include directories to the project, or LDFLAGS to the final linking step. These values can be customised via the `component.mk` file itself. See *Optional Project-Wide Component Variables* for details.

Take care when setting variables or targets in this file. As the values are included into the top-level project makefile pass, they can influence or break functionality across all components!

KConfig.projbuild

This is an equivalent to *Makefile.projbuild* for *component configuration* KConfig files. If you want to include configuration options at the top-level of menuconfig, rather than inside the "Component Configuration" sub-menu, then these can be defined in the KConfig.projbuild file alongside the `component.mk` file.

Take care when adding configuration values in this file, as they will be included across the entire project configuration. Where possible, it's generally better to create a KConfig file for *component configuration*.

7.2.10 Example Component Makefiles

Because the build environment tries to set reasonable defaults that will work most of the time, component.mk can be very small or even empty (see *Minimal Component Makefile*). However, overriding *component variables* is usually required for some functionality.

Here are some more advanced examples of `component.mk` makefiles:

Adding source directories

By default, sub-directories are ignored. If your project has sources in sub-directories instead of in the root of the component then you can tell that to the build system by setting `COMPONENT_SRCDIRS`:

```
COMPONENT_SRCDIRS := src1 src2
```

This will compile all source files in the src1/ and src2/ sub-directories instead.

Specifying source files

The standard component.mk logic adds all .S and .c files in the source directories as sources to be compiled unconditionally. It is possible to circumvent that logic and hard-code the objects to be compiled by manually setting the `COMPONENT_OBJS` variable to the name of the objects that need to be generated:

```
COMPONENT_OBJS := file1.o file2.o thing/filea.o thing/fileb.o anotherthing/main.o
COMPONENT_SRCDIRS := . thing anotherthing
```

Note that COMPONENT_SRCDIRS must be set as well.

Adding conditional configuration

The configuration system can be used to conditionally compile some files depending on the options selected in make menuconfig:

Kconfig:

```
config FOO_ENABLE_BAR
    bool "Enable the BAR feature."
    help
        This enables the BAR feature of the FOO component.
```

component.mk:

```
COMPONENT_OBJS := foo_a.o foo_b.o

ifdef CONFIG_FOO_BAR
COMPONENT_OBJS += foo_bar.o foo_bar_interface.o
endif
```

See the *GNU Make Manual* for conditional syntax that can be used use in makefiles.

Source Code Generation

Some components will have a situation where a source file isn't supplied with the component itself but has to be generated from another file. Say our component has a header file that consists of the converted binary data of a BMP file, converted using a hypothetical tool called bmp2h. The header file is then included in as C source file called graphics_lib.c:

```
COMPONENT_EXTRA_CLEAN := logo.h

graphics_lib.o: logo.h

logo.h: $(COMPONENT_PATH)/logo.bmp
    bmp2h -i $^ -o $@
```

In this example, graphics_lib.o and logo.h will be generated in the current directory (the build directory) while logo.bmp comes with the component and resides under the component path. Because logo.h is a generated file, it needs to be cleaned when make clean is called which why it is added to the COMPONENT_EXTRA_CLEAN variable.

Cosmetic Improvements

Because logo.h is a generated file, it needs to be cleaned when make clean is called which why it is added to the COMPONENT_EXTRA_CLEAN variable.

Adding logo.h to the graphics_lib.o dependencies causes it to be generated before graphics_lib.c is compiled.

If a a source file in another component included logo.h, then this component's name would have to be added to the other component's COMPONENT_DEPENDS list to ensure that the components were built in-order.

Embedding Binary Data

Sometimes you have a file with some binary or text data that you'd like to make available to your component - but you don't want to reformat the file as C source.

You can set a variable COMPONENT_EMBED_FILES in component.mk, giving the names of the files to embed in this way:

```
COMPONENT_EMBED_FILES := server_root_cert.der
```

Or if the file is a string, you can use the variable COMPONENT_EMBED_TXTFILES. This will embed the contents of the text file as a null-terminated string:

```
COMPONENT_EMBED_TXTFILES := server_root_cert.pem
```

The file's contents will be added to the .rodata section in flash, and are available via symbol names as follows:

```
extern const uint8_t server_root_cert_pem_start[] asm("_binary_server_root_cert_pem_start");
extern const uint8_t server_root_cert_pem_end[]   asm("_binary_server_root_cert_pem_end");
```

The names are generated from the full name of the file, as given in COMPONENT_EMBED_FILES. Characters /, ., etc. are replaced with underscores. The _binary prefix in the symbol name is added by objcopy and is the same for both text and binary files.

For an example of using this technique, see protocols/https_request - the certificate file contents are loaded from the text .pem file at compile time.

7.2.11 Fully Overriding The Component Makefile

Obviously, there are cases where all these recipes are insufficient for a certain component, for example when the component is basically a wrapper around another third-party component not originally intended to be compiled under this build system. In that case, it's possible to forego the esp-idf build system entirely by setting COMPONENT_OWNBUILDTARGET and possibly COMPONENT_OWNCLEANTARGET and defining your own targets named `build` and `clean` in `component.mk` target. The build target can do anything as long as it creates $(COMPONENT_LIBRARY) for the project make process to link into the app binary.

(Actually, even this is not strictly necessary - if the COMPONENT_ADD_LDFLAGS variable is set then the component can instruct the linker to link other binaries instead.)

7.2.12 Custom sdkconfig defaults

For example projects or other projects where you don't want to specify a full sdkconfig configuration, but you do want to override some key values from the esp-idf defaults, it is possible to create a file `sdkconfig.defaults` in the project directory. This file will be used when running `make defconfig`, or creating a new config from scratch.

To override the name of this file, set the `SDKCONFIG_DEFAULTS` environment variable.

CHAPTER 8

Debugging

8.1 OpenOCD setup for ESP32

The ESP31 and ESP32 have two powerful Xtensa cores, allowing for a great variety of program architectures. The FreeRTOS OS that comes with ESP-IDF is capable of multi-core pre-emptive multithreading, allowing for an intuitive way of writing software.

The downside of the ease of programming is that debugging without the right tools is harder: figuring out a bug that is caused by two threads, maybe even running simultaneously on two different CPU cores, can take a long time when all you have are printf statements. A better and in many cases quicker way to debug such problems is by using a debugger, connected to the processors over a debug port.

Espressif has ported OpenOCD to support the ESP32 processor and the multicore FreeRTOS that will be the foundation of most ESP32 apps, and has written some tools to help with features OpenOCD does not support natively. These are all available for free, and this document describes how to install and use them.

8.2 JTAG adapter hardware

You will need a JTAG adapter that is compatible with both the voltage levels on the ESP32 as well as with the OpenOCD software. The JTAG port on the ESP32 is an industry-standard JTAG port which lacks (and does not need) the TRST pin. The JTAG I/O pins all are powered from the VDD_3P3_RTC pin (which normally would be powered by a 3.3V rail) so the JTAG adapter needs to be able to work with JTAG pins in that voltage range. On the software side, OpenOCD supports a fair amount of JTAG adapters. See http://openocd.org/doc/html/Debug-Adapter-Hardware.html for an (unfortunately slightly incomplete) list of the adapters OpenOCD works with. This page lists SWD-compatible adapters as well; take note that the ESP32 does not support SWD.

At Espressif, we have tested the TIAO USB Multi-protocol Adapter board as well as the Flyswatter2, which are both USB2.0 high-speed devices and give a good throughput. We also tested a J-link-compatible and an EasyOpenJTAG adapter; both worked as well but are somewhat slower.

The minimal signalling to get a working JTAG connection are TDI, TDO, TCK, TMS and Gnd. Some JTAG debuggers also need a connection from the ESP32 power line to a line called e.g. Vtar to set the working voltage. SRST can optionally be connected to the CH_PD of the ESP32, although for now, support in OpenOCD for that line is pretty minimal.

8.3 Installing OpenOCD

The sources for the ESP32-enabled variant of OpenOCD are available from Espressifs Github. To download the source, use the following commands:

```
git clone --recursive https://github.com/espressif/openocd-esp32.git
cd openocd-esp32
```

For compilation of OpenOCD, please refer to the README, README.OSX and README.Windows file in the openocd-esp32 directory. You can skip the `make install` step if you want.

8.4 Configuring the ESP32 target in OpenOCD

After OpenOCD is compiled (and optionally installed) and the JTAG adapter is connected to the ESP32 board, everything is ready to invoke OpenOCD for the first time. To do this, OpenOCD needs to be told what JTAG adapter to use as well as what type of board and processor the JTAG adapter is connected to. It is the easiest to do both using a configuration file. A template configuration file (esp32.cfg) is included in the same directory as this file. A way to use this would be:

- Copy esp32.cfg to the openocd-esp32 directory
- Edit the copied esp32.cfg file. Most importantly, change the `source [find interface/ftdi/tumpa.cfg]` line to reflect the physical JTAG adapter connected.
- Open a terminal and `cd` to the openocd-esp32 directory.
- Run `./src/openocd -s ./tcl -f ./esp32.cfg` to start OpenOCD

You should now see something like this:

```
user@machine:~/esp32/openocd-esp32$ ./src/openocd -s ./tcl/ -f ../openocd-esp32-tools/esp32.cfg
Open On-Chip Debugger 0.10.0-dev-00446-g6e13a97-dirty (2016-08-23-16:36)
Licensed under GNU GPL v2
For bug reports, read
http://openocd.org/doc/doxygen/bugs.html
none separate
adapter speed: 200 kHz
Info : clock speed 200 kHz
Info : JTAG tap: esp32.cpu0 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part: 0x2003, ver:
Info : JTAG tap: esp32.cpu1 tap/device found: 0x120034e5 (mfg: 0x272 (Tensilica), part: 0x2003, ver:
Info : esp32.cpu0: Debug controller was reset (pwrstat=0x5F, after clear 0x0F).
Info : esp32.cpu0: Core was reset (pwrstat=0x5F, after clear 0x0F).
```

- If you see an error indicating permission problems, please see the 'Permissions delegation' bit in the OpenOCD README
- If you see JTAG errors (...all ones/...all zeroes) please check your connections and see if everything is powered on.

8.5 Connecting a debugger to OpenOCD

OpenOCD should now be ready to accept gdb connections. If you have compiled the ESP32 toolchain using Crosstool-NG, or if you have downloaded a precompiled toolchain from the Espressif website, you should already have xtensa-esp32-elf-gdb, a version of gdb that can be used for this. First, make sure the project you want to debug is compiled

and flashed into the ESP32's SPI flash. Then, in a different console than OpenOCD is running in, invoke gdb. For example, for the template app, you would do this like such:

```
cd esp-idf-template
xtensa-esp32-elf-gdb -ex 'target remote localhost:3333' ./build/app-template.elf
```

This should give you a gdb prompt.

8.6 FreeRTOS support

OpenOCD has explicit support for the ESP-IDF FreeRTOS; FreeRTOS detection can be disabled in esp32.conf. When enabled, gdb can see FreeRTOS tasks as threads. Viewing them all can be done using the gdb `i threads` command, changing to a certain task is done with `thread x`, with x being the number of the thread. All threads can be switched to except for a thread actually running on the other CPU, please see `ESP32 quirks` for more information.

8.7 ESP32 quirks

Normal gdb breakpoints (`b myFunction`) can only be set in IRAM, because that memory is writable. Setting these types of breakpoints in code in flash will not work. Instead, use a hardware breakpoint (`hb myFunction`). The esp32 supports 2 hardware breakpoints. It also supports two watchpoint, so two variables can be watched for change or read by the gdb command `watch myVariable`.

Connecting gdb to the APP or PRO cpu happens by changing the port gdb connects to. `target remote localhost:3333` connects to the PRO CPU, `target remote localhost:3334` to the APP CPU. Hardware-wise, when one CPU is halted because of debugging reasons, the other one will be halted as well; resuming also happens simultaneously.

Because gdb only sees the system from the point of view of the selected CPU, only the FreeRTOS tasks that are suspended and the task running on the CPU gdb is connected to, will be shown correctly. The task that was active on the other cpu can be inspected, but its state may be wildly inconsistent.

The ESP-IDF code has the option of compiling in various support options for OpenOCD: it can stop execution when the first thread is started and break the system if a panic or unhandled exception is thrown. Both options are enabled by default but can be disabled using the esp-idf configuration menu. Please see the `make menuconfig` menu for more details.

Normally, under OpenOCD, a board can be reset by entering 'mon reset' or 'mon reset halt' into gdb. For the ESP32, these commands work more or less, but have side effects. First of all, an OpenOCD reset only resets the CPU cores, not the peripherals, which may lead to undefined behaviour if software assumes the after-reset state of peripherals. Secondly, 'mon reset halt' stops before FreeRTOS is initialized. OpenOCD assumes (in the default configuration, you can change this by editing esp32.cfg) a running FreeRTOS and may get confused.

CHAPTER 9

ESP32 Core Dump

9.1 Overview

ESP-IDF provides support to generate core dumps on unrecoverable software errors. This useful technique allows post-mortem analysis of software state at the moment of failure. Upon the crash system enters panic state, prints some information and halts or reboots depending configuration. User can choose to generate core dump in order to analyse the reason of failure on PC later on. Core dump contains snapshots of all tasks in the system at the moment of failure. Snapshots include tasks control blocks (TCB) and stacks. So it is possible to find out what task, at what instruction (line of code) and what callstack of that task lead to the crash. ESP-IDF provides special script *espcoredump.py* to help users to retrieve and analyse core dumps. This tool provides two commands for core dumps analysis:

- info_corefile - prints crashed task's registers, callstack, list of available tasks in the system, memory regions and contents of memory stored in core dump (TCBs and stacks)
- dbg_corefile - creates core dump ELF file and runs GDB debug session with this file. User can examine memory, variables and tasks states manually. Note that since not all memory is saved in core dump only values of variables allocated on stack will be meaningfull

9.2 Configuration

Currently there are three options related to core dump generation which user can choose in configuration menu of the application (*make menuconfig*):

- Disable core dump generation
- Save core dump to flash
- Print core dump to UART

These options can be choosen in Components -> ESP32-specific config -> Core dump destination menu item.

9.3 Save core dump to flash

When this option is selected core dumps are saved to special partition on flash. When using default partition table files which are provided with ESP-IDF it automatically allocates necessary space on flash, But if user wants to use its own layout file together with core dump feature it should define separate partition for core dump as it is shown below:

```
# Name,   Type, SubType, Offset,  Size
# Note: if you change the phy_init or app partition offset, make sure to change the offset in Kconfig
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
coredump, data, coredump,,        64K
```

There are no special requrements for partition name. It can be choosen according to the user application needs, but partition type should be 'data' and sub-type should be 'coredump'. Also when choosing partition size note that core dump data structure introduces constant overhead of 20 bytes and per-task overhead of 12 bytes. This overhead does not include size of TCB and stack for every task. So partirion size should be at least 20 + max tasks number x (12 + TCB size + max task stack size) bytes.

The example of generic command to analyze core dump from flash is: *espcoredump.py -p </path/to/serial/port> info_corefile </path/to/program/elf/file>* or *espcoredump.py -p </path/to/serial/port> dbg_corefile </path/to/program/elf/file>*

9.4 Print core dump to UART

When this option is selected base64-encoded core dumps are printed on UART upon system panic. In this case user should save core dump text body to some file manually and then run the following command: *espcoredump.py -p </path/to/serial/port> info_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>* or *espcoredump.py -p </path/to/serial/port> dbg_corefile -t b64 -c </path/to/saved/base64/text> </path/to/program/elf/file>*

Base64-encoded body of core dump will be between the following header and footer:

```
================= CORE DUMP START =================
<body of base64-encoded core dump, save it to file on disk>
================= CORE DUMP END ===================
```

9.5 Running 'espcoredump.py'

Generic command syntax:

espcoredump.py [options] command [args]

Script Options

- –chip,-c {auto,esp32}. Target chip type. Supported values are *auto* and *esp32*.
- –port,-p PORT. Serial port device.
- –baud,-b BAUD. Serial port baud rate used when flashing/reading.

Commands

- info_corefile. Retrieve core dump and print useful info.
- dbg_corefile. Retrieve core dump and start GDB session with it.

Command Arguments

- –gdb,-g GDB. Path to gdb to use for data retrieval.
- –core,-c CORE. Path to core dump file to use (if skipped core dump will be read from flash).
- –core-format,-t CORE_FORMAT. Specifies that file passed with "-c" is an ELF ("elf"), dumped raw binary ("raw") or base64-encoded ("b64") format.

- –off,-o OFF. Ofsset of coredump partition in flash (type “make partition_table” to see it).
- –save-core,-s SAVE_CORE. Save core to file. Othwerwise temporary core file will be deleted. Ignored with “-c”.
- –print-mem,-m Print memory dump. Used only with “info_corefile”.

CHAPTER 10

Partition Tables

10.1 Overview

A single ESP32's flash can contain multiple apps, as well as many different kinds of data (calibration data, filesystems, parameter storage, etc). For this reason a partition table is flashed to offset 0x8000 in the flash.

Partition table length is 0xC00 bytes (maximum 95 partition table entries). If the partition table is signed due to *secure boot*, the signature is appended after the table data.

Each entry in the partition table has a name (label), type (app, data, or something else), subtype and the offset in flash where the partition is loaded.

The simplest way to use the partition table is to *make menuconfig* and choose one of the simple predefined partition tables:

- "Single factory app, no OTA"
- "Factory app, two OTA definitions"

In both cases the factory app is flashed at offset 0x10000. If you *make partition_table* then it will print a summary of the partition table.

10.2 Built-in Partition Tables

Here is the summary printed for the "Single factory app, no OTA" configuration:

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
```

- At a 0x10000 (64KB) offset in the flash is the app labelled "factory". The bootloader will run this app by default.
- There are also two data regions defined in the partition table for storing NVS library partition and PHY init data.

Here is the summary printed for the "Factory app, two OTA definitions" configuration:

```
# Espressif ESP32 Partition Table
# Name,   Type, SubType, Offset,  Size
nvs,      data, nvs,     0x9000,  0x4000
otadata,  data, ota,     0xd000,  0x2000
phy_init, data, phy,     0xf000,  0x1000
```

```
factory,  0,    0,       0x10000, 1M
ota_0,    0,    ota_0,   ,        1M
ota_1,    0,    ota_1,   ,        1M
```

- There are now three app partition definitions.
- The type of all three are set as "app", but the subtype varies between the factory app at 0x10000 and the next two "OTA" apps.
- There is also a new "ota data" slot, which holds the data for OTA updates. The bootloader consults this data in order to know which app to execute. If "ota data" is empty, it will execute the factory app.

10.3 Creating Custom Tables

If you choose "Custom partition table CSV" in menuconfig then you can also enter the name of a CSV file (in the project directory) to use for your partition table. The CSV file can describe any number of definitions for the table you need.

The CSV format is the same format as printed in the summaries shown above. However, not all fields are required in the CSV. For example, here is the "input" CSV for the OTA partition table:

```
# Name,   Type, SubType, Offset,   Size
nvs,      data, nvs,     0x9000,  0x4000
otadata,  data, ota,     0xd000,  0x2000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000,  1M
ota_0,    app,  ota_0,   ,         1M
ota_1,    app,  ota_1,   ,         1M
```

- Whitespace between fields is ignored, and so is any line starting with # (comments).
- Each non-comment line in the CSV file is a partition definition.
- Only the offset for the first partition is supplied. The gen_esp32part.py tool fills in each remaining offset to start after the preceding partition.

10.3.1 Name field

Name field can be any meaningful name. It is not significant to the ESP32. Names longer than 16 characters will be truncated.

10.3.2 Type field

Type field can be specified as app (0) or data (1). Or it can be a number 0-254 (or as hex 0x00-0xFE). Types 0x00-0x3F are reserved for Espressif. If your application needs to store data, please add a custom partition type in the range 0x40-0xFE.

The bootloader ignores any types other than 0 & 1.

10.3.3 Subtype

When type is "app", the subtype field can be specified as factory (0), ota_0 (0x10) ... ota_15 (0x1F) and test (0x20). Or it can be any number 0-255 (0x00-0xFF). The bootloader will execute the factory app unless there it sees a partition of type data/ota, in which case it reads this partition to determine which OTA image to boot

When type is "data", the subtype field can be specified as ota (0), phy (1), nvs (2). Or it can be a number 0x00-0xFF. The bootloader ignores all data subtypes except for ota. Subtypes 0-0x7f are reserved for Espressif use. To create custom data partition subtypes use "data" type, and choose any unused subtype in 0x80-0xFF range. If you are porting a filesystem to the ESP-IDF, consider opening a PR to add the new subtype to esp_partition.h file.

10.3.4 Offset & Size

Only the first offset field is required (we recommend using 0x10000). Partitions with blank offsets will start after the previous partition.

App partitions have to be at offsets aligned to 0x10000 (64K). If you leave the offset field blank, the tool will automatically align the partition. If you specify an unaligned offset for an app partition, the tool will return an error.

Sizes and offsets can be specified as decimal numbers, hex numbers with the prefix 0x, or size multipliers M or K (1024 and 1024*1024 bytes).

NVS data partition has to be at least 0x3000 bytes long, and OTA data parition has to be 0x2000 bytes long. If you are using NVS in your application to store a lot of data, consider using a custom partition table with larger NVS partition.

10.4 Generating Binary Partition Table

The partition table which is flashed to the ESP32 is in a binary format, not CSV. The tool bin/gen_esp32part.py is used to convert between CSV and binary formats.

If you configure the partition table CSV name in `make menuconfig` and then `make partition_table`, this conversion is done for you.

To convert CSV to Binary manually:

```
python bin/gen_esp32part.py --verify input_partitions.csv binary_partitions.bin
```

To convert binary format back to CSV:

```
python bin/gen_esp32part.py --verify binary_partitions.bin input_partitions.csv
```

To display the contents of a binary partition table on stdout (this is how the summaries displayed when running *make partition_table* are generated:

```
python bin/gen_esp32part.py binary_partitions.bin
```

`gen_esp32part.py` takes one optional argument, `--verify`, which will also verify the partition table during conversion (checking for overlapping partitions, unaligned partitions, etc.)

10.5 Flashing the partition table

- `make partition_table-flash`: will flash the partition table with esptool.py.
- `make flash`: Will flash everything including the partition table.

A manual flashing command is also printed as part of `make partition_table`.

Note that updating the partition table doesn't erase data that may have been stored according to the old partition table. You can use `make erase_flash` (or `esptool.py erase_flash`) to erase the entire flash contents.

CHAPTER 11

Flash Encryption

Flash Encryption is a feature for encrypting the contents of the ESP32's attached SPI flash. When flash encryption is enabled, physical readout of the SPI flash is not sufficient to recover most flash contents.

Flash Encryption is separate from the *Secure Boot* feature, and you can use flash encryption without enabling secure boot. However we recommend using both features together for a secure environment.

IMPORTANT: Enabling flash encryption limits your options for further updates of your ESP32. Make sure to read this document (including 'Limitations of Flash Encryption' and understand the implications of enabling flash encryption.

11.1 Background

- The contents of the flash are encrypted using AES with a 256 bit key. The flash encryption key is stored in efuse internal to the chip, and is (by default) protected from software access.
- Flash access is transparent via the flash cache mapping feature of ESP32 - any flash regions which are mapped to the address space will be transparently decrypted when read.
- Encryption is applied by flashing the ESP32 with plaintext data, and (if encryption is enabled) the bootloader encrypts the data in place on first boot.
- Not all of the flash is encrypted. The following kinds of flash data are encrypted: - Bootloader - Secure boot bootloader digest (if secure boot is enabled) - Partition Table - All "app" type partitions - Any partition marked with the "encrypt" flag in the partition table

  It may be desirable for some data partitions to remain unencrypted for ease of access, or to use flash-friendly update algorithms that are ineffective if the data is encrypted. "NVS" partitions for non-volatile storage cannot be encrypted.
- The flash encryption key is stored in efuse key block 1, internal to the ESP32 chip. By default, this key is read- and write-protected so software cannot access it or change it.
- The *flash encryption algorithm* is AES-256, where the key is "tweaked" with the offset address of each 32 byte block of flash. This means every 32 byte block (two consecutive 16 byte AES blocks) is encrypted with a unique key derived from the flash encryption key.
- Although software running on the chip can transparently decrypt flash contents, by default it is made possible for the UART bootloader to decrypt (or encrypt) data when flash encryption is enabled.

11.2 Flash Encryption Initialisation

This is the default (and recommended) flash encryption initialisation process. It is possible to customise this process for development or other purposes, see *Flash Encryption Advanced Features* for details.

IMPORTANT: Once flash encryption is enabled on first boot, the hardware allows a maximum of 3 subsequent flash updates via physical re-flashing. If secure boot is enabled, no physical re-flashes are possible. OTA updates can be used to update flash content without counting towards this limit. When enabling flash encryption in development, use a 'precalculated flash encryption key' to allow physically re-flashing an unlimited number of times with pre-encrypted data.

- The bootloader must be compiled with flash encryption support enabled. In `make menuconfig`, navigate to "Security Features" and select "Yes" for "Enable flash nencryption on boot".
- If enabling Secure Boot at the same time, you can simultaneously select those options now. See the *Secure Boot* documentation for details.
- Build and flash the bootloader, partition table and factory app image as normal. These partitions are initially written to the flash unencrypted.
- On first boot, the bootloader sees `FLASH_CRYPT_CNT` efuse is set to 0 so it generates a flash encryption key using the hardware random number generator. This key is stored in efuse. The key is read and write protected against further software access.
- All of the encrypted partitions are then encrypted in-place by the bootloader. Encrypting in-place can take some time (up to a minute for large partitions.)

IMPORTANT: Do not interrupt power to the ESP32 while the first boot encryption pass is running. If power is interrupted, the flash contents will be corrupted and require flashing with unencrypted data again. This re-flash will not count towards the flashing limit, as ''FLASH_CRYPT_CNT'' is only updated after this process finishes.

- Once flashing is complete. efuses are blown (by default) to disable encrypted flash access while the UART bootloader is running.
- If not already write-protected, the `FLASH_CRYPT_CONFIG` efuse is also burned to the maximum value (`0xF`) to maximise the number of key bits which are tweaked in the flash algorithm. See *Setting FLASH_CRYPT_CONFIG* for details of this efuse.
- Finally, the `FLASH_CRYPT_CNT` efuse is burned with the initial value 1. It is this efuse which activates the transparent flash encryption layer, and limits the number of subsequent reflashes. See the *Updating Encrypted Flash* section for details about `FLASH_CRYPT_CNT`.
- The bootloader resets itself to reboot from the newly encrypted flash.

11.3 Encrypted Flash Access

11.3.1 Reading Encrypted Flash

Whenever the `FLASH_CRYPT_CNT` efuse is set to a value with an odd number of bits set, all flash content which is accessed via the MMU's flash cache is transparently decrypted. This includes:

- Executable application code in flash (IROM).
- All read-only data stored in flash (DROM).
- Any data accessed via `esp_spi_flash_mmap`.
- The software bootloader image when it is read by the ROM bootloader.

IMPORTANT: The MMU flash cache unconditionally decrypts all data. Data which is stored unencrypted in the flash will be "transparently decrypted" via the flash cache and appear to software like random garbage.

To read data without using a flash cache MMU mapping, we recommend using the partition read function `esp_partition_read`. When using this function, data will only be decrypted when it is read from an encrypted partition. Other partitions will be read unencrypted. In this way, software can access encrypted and non-encrypted flash in the same way.

Data which is read via other SPI read APIs are not decrypted:

- Data read via `esp_spi_flash_read` is not decrypted
- Data read via ROM function `SPIRead` is not decrypted (this function is not supported in esp-idf apps).
- Data stored using the Non-Volatile Storage (NVS) API is always stored decrypted.

11.3.2 Writing Encrypted Flash

Where possible, we recommend using the partition write function `esp_partition_write`. When using this function, data will only be encrypted when writing to encrypted partitions. Data will be written to other partitions unencrypted. In this way, software can access encrypted and non-encrypted flash in the same way.

The `esp_spi_flash_write` function will write data when the write_encrypted parameter is set to true. Otherwise, data will be written unencrypted.

The ROM function `SPI_Encrypt_Write` will write encrypted data to flash, the ROM function `SPIWrite` will write unencrypted to flash. (these function are not supported in esp-idf apps).

The minimum write size for unencrypted data is 4 bytes (and the alignment is 4 bytes). Because data is encrypted in blocks, the minimum write size for encrypted data is 32 bytes (and the alignment is 32 bytes.)

11.4 Updating Encrypted Flash

11.4.1 OTA Updates

OTA updates to encrypted partitions will automatically write encrypted, as long as the `esp_partition_write` function is used.

11.4.2 Serial Flashing

Provided secure boot is not used, the `FLASH_CRYPT_CNT` registers allow the flash to be updated with new plaintext data via serial flashing (or other physical methods), up to 3 additional times. `FLASH_CRYPT_CNT` efuse is an 8-bit value, and the flash encryption enables or disables based on the number of bits which are set to "1":

- Even number (0-6) bits are set: Transparent reading of encrypted flash is disabled, any encrypted data cannot be decrypted. If the bootloader was built with "Enable flash encryption on boot" then it will see this situation and immediately re-encrypt the flash wherever it finds unencrypted data. Once done, it sets another bit in the efuse to '1' meaning an odd number of bits are now set.
- Odd number (1-7) bits are set: Transparent reading of encrypted flash is enabled.
- All 8 bits are set (valuye 0: Transparent reading of encrypted flash is disabled, any encrypted data is inaccessible. Bootloader will normally detect this condition and halt. To avoid use of this state to load unauthorised code, secure boot must be used or `FLASH_CRYPT_CNT` must be write-protected.

The espefuse.py tool can be used to manually change the number of bits set in `FLASH_CRYPT_CNT`, via serial bootloader.

11.4.3 Limited Updates

Only 4 physical flash updates (writing plaintext data which is then encrypted) are possible:

1. On first plaintext boot, bit count has brand new value 0 and bootloader changes to 1 (0x01) following encryption.
2. On next plaintext flash update, bit count is manually updated to 2 (0x03) and bootloader changes to 4 (0x07) following encryption.
3. Then bit count is manually updated to 4 (0x0F) and the bootloader changes efuse bit count to 5 (0x1F).
4. Finally bootloader is manually updated to 6 (0x3F) and bootloader changes efuse bit count to 7 (0x7F).

11.4.4 Cautions With Re-Flashing

- When reflashing via serial, reflash every partition that was previously written with plaintext (including bootloader). It is possible to skip app partitions which are not the "currently selected" OTA partition (these will not be re-encrypted unless a plaintext app image is found there.) However any partition marked with the "encrypt" flag will be unconditionally re-encrypted, meaning that any already encrypted data will be encrypted twice and corrupted.
- If secure boot is enabled, you can't reflash via serial at all unless you used chosen the "Reflashable" option for Secure Boot, pre-generated a key and burned it to the ESP32. In this case you can re-flash a plaintext secure boot digest and bootloader image at offset 0 (see *Secure Boot* documentation.) In production secure boot configuration, the secure boot digest is stored encrypted - so if `FLASH_CRYPT_CNT` is set to an even value then the ROM bootloader will read the encrypted digest as-is and therefore will fail to verify any bootloader image as valid.

11.4.5 Re-Flashing Procedure

The steps to update a device with plaintext via UART bootloader, when flash encryption is enabled are:

- Build the application as usual.
- Burn the `FLASH_CRYPT_CNT` efuse by running the command `espefuse.py burn_efuse FLASH_CRYPT_CNT`. espefuse.py will automatically increment the bit count by 1.
- Flash the device with plaintext data as usual (`make flash` or `esptool.py` commands.) Flash all previously encrypted partitions, including the bootloader. If secure boot is enabled, it must be enabled in "Reflashable" mode and a pre-generated key burned to the ESP32 - flash the bootloader-reflash-digest.bin file at offset 0x0.
- Reset the device and it will re-encrypt plaintext partitions, burn the `FLASH_CRYPT_CNT` flag to re-enable encryption.

11.4.6 Disabling Updates

To prevent further plaintext updates via physical access, use espefuse.py to write protect the `FLASH_CRYPT_CNT` efuse after flash encryption has been enabled (ie after first boot is complete):

```
espefuse.py write_protect_efuse FLASH_CRYPT_CNT
```

This prevents any further modifications to disable or re-enable flash encryption.

11.5 Limitations of Flash Encryption

Flash Encryption prevents plaintext readout of the encrypted flash, to protect firmware against unauthorised readout and modification. It is important to understand the limitations of the flash encryption system:

- Flash encryption is only as strong as the key. For this reason, we recommend keys are generated on the device during first boot (default behaviour). If generating keys off-device to burn with `esp_efuse.py burn_key`, ensure they are generated from a quality random number source, kept secure, and never shared between devices.
- Not all data is stored encrypted. If storing data on flash, check if the method you are using (library, API, etc.) supports flash encryption.
- Flash encryption does not prevent an attacker from understanding the high-level layout of the flash. This is because the same AES key is used for every two 16 byte AES blocks. When both adjacent 16 byte blocks contain identical content (such as empty or padding areas), these blocks will encrypt to produce matching pairs of encrypted blocks. This may allow an attacker to make high-level comparisons between encrypted devices (ie to tell if two devices are probably running the same firmware version).
- For the same reason, an attacker can always guess when two adjacent 16 byte blocks (32 byte aligned) contain identical content. Keep this in mind if storing sensitive data on the flash, design your flash storage so this doesn't happen (using a counter byte or some other non-identical value every 16 bytes is sufficient).
- Flash encryption alone may not prevent an attacker from modifying the firmware of the device. Always use flash encryption in combination with Secure Boot.

11.6 Flash Encryption Advanced Features

11.6.1 Encrypted Partition Flag

In the *partition table* description CSV files, there is a field for flags.

Usually left blank, if you write "encrypted" in this field then the partition will be marked as encrypted in the partition table, and data written here will be treated as encrypted (same as an app partition):

```
# Name,   Type, SubType, Offset,  Size, Flags
nvs,      data, nvs,     0x9000,  0x6000
phy_init, data, phy,     0xf000,  0x1000
factory,  app,  factory, 0x10000, 1M
secret_data, 0x40, 0x01, 0x20000, 256K, encrypted
```

- None of the default partition formats have any encrypted data partitions.
- It is not necessary to mark "app" partitions as encrypted, they are always treated as encrypted.
- The "encrypted" flag does nothing if flash encryption is not enabled.
- It is possible to mark the optional `phy` partition with `phy_init` data as encrypted, if you wish to protect this data from physical access readout or modification.
- It is not possible to mark the `nvs` partition as encrypted.

11.6.2 Precalculated Flash Encryption Key

It is possible to pre-generate a flash encryption key on the host computer and burn it into the ESP32 efuse. This allows data to be per-encrypted on the host and flashed to the ESP32 without needing a plaintext flash update.

This is useful for development, because it removes the 4 flash limit and allows reflashing with secure boot enabled.

IMPORTANT This method is intended to assist with development only, not for production devices. If pre-generating flash encryption for production, ensure the keys are generated from a high quality random number source and do not share the same flash encryption key across multiple devices.

Obtaining Flash Encryption Key

Flash encryption keys are 32 bytes of random data. You can generate a random key with espsecure.py:

```
espsecure.py generate_flash_encryption_key my_flash_encryption_key.bin
```

(The randomness of this data is only as good as the OS and it's Python installation's random data source.)

Alternatively, if you're using *secure boot* and have a secure boot signing key then you can generate a deterministic SHA-256 digest of the secure boot private key to use:

```
espsecure.py digest_private_key --keyfile secure_boot_signing_key.pem my_flash_encryption_key.bin
```

The same key is used as the secure boot digest key if you enabled "Reflashable" mode for secure boot.

This means you can always re-calculate the flash encryption key from the secure boot private signing key. This method is **not at all suitable** for production devices.

Burning Flash Encryption Key

Once you have generated a flash encryption key, you need to burn it to efuse on the device. This **must be done before first boot**, otherwise the ESP32 will generate a random key that software can't access.

To burn a key to the device (possible one time only):

```
espefuse.py burn_key flash_encryption my_flash_encryption_key.bin
```

First Flash

For the first flash, follow the same steps as for default *Flash Encryption Initialisation* and flash a plaintext image. The bootloader will enable flash encryption using the pre-burned key and encrypt all partitions.

Reflashing

To reflash an encrypted image requires an additional manual update step, to encrypt the data you wish to flash.

Suppose that this is the normal flashing non-encrypted flashing step:

```
esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash -z 0x10000 build/my-app.bin
```

The data needs to be pre-encrypted with knowledge of the address (0x10000) and the binary file name:

```
espsecure.py encrypt_flash_data --keyfile my_flash_encryption_key.bin --address 0x10000 -o build/my-a
```

This step will encrypt `my-app.bin` using the supplied key, and produce an encrypted file `my-app-encrypted.bin`. Be sure that the address argument matches the address where you plan to flash the binary.

Then, flash the encrypted binary with esptool.py:

```
esptool.py --port /dev/ttyUSB0 --baud 115200 write_flash -z 0x10000 build/my-app-encrypted.bin
```

11.6.3 Enabling UART Bootloader Encryption/Decryption

By default, on first boot the flash encryption process will burn efuses `DISABLE_DL_ENCRYPT`, `DISABLE_DL_DECRYPT` and `DISABLE_DL_CACHE`.

- `DISABLE_DL_ENCRYPT` disables the flash encryption operations when running in UART bootloader boot mode.
- `DISABLE_DL_DECRYPT` disables transparent flash decryption when running in UART bootloader mode, even if `FLASH_CRYPT_CNT` is set to enable it in normal operation.
- `DISABLE_DL_CACHE` disables the entire MMU flash cache when running in UART bootloader mode.

It is possible to burn only some of these efuses, and write-protect the rest (with unset value 0) before the first boot, in order to preserve them:

```
espefuse.py burn_efuse DISABLE_DL_DECRYPT
espefuse.py write_protect_efuse DISABLE_DL_ENCRYPT
```

(Note that all 3 of these efuses are disabled via one write protect bit, so write protecting one will write protect all of them.)

Write protecting these efuses when they are unset (0) is not currently useful, as `esptool.py` does not support flash encryption functions.

However, note that write protecting `DISABLE_DL_DECRYPT` when it is unset (0) effectively makes flash encryption useless, as an attacker with physical access can use UART bootloader mode to read out the flash.

11.7 Technical Details

11.7.1 Flash Encryption Algorithm

- AES-256 operates on 16 byte blocks of data. The flash encryption engine encrypts and decrypts data in 32 byte blocks, two AES blocks in series.
- AES algorithm is used inverted in flash encryption, so the flash encryption "encrypt" operation is AES decrypt and the "decrypt" operation is AES encrypt. This is for performance reasons and does not alter the effectiveness of the algorithm.
- The main flash encryption key is stored in efuse (BLK2) and by default is protected from further writes or software readout.
- Each 32 byte block is encrypted with a unique key which is derived from this main flash encryption key XORed with the offset of this block in the flash (a "key tweak").
- The specific tweak depends on the setting of `FLASH_CRYPT_CONFIG` efuse. This is a 4 bit efuse, where each bit enables XORing of a particular range of the key bits:
  - Bit 1, bits 0-66 of the key are XORed.
  - Bit 2, bits 67-131 of the key are XORed.
  - Bit 3, bits 132-194 of the key are XORed.
  - Bit 4, bits 195-256 of the key are XORed.

  It is recommended that `FLASH_CRYPT_CONFIG` is always left to set the default value *0xF*, so that all key bits are XORed with the block offset. See *Setting FLASH_CRYPT_CONFIG* for details.

- The high 19 bits of the block offset (bit 5 to bit 23) are XORed with the main flash encryption key. This range is chosen for two reasons: the maximum flash size is 16MB (24 bits), and each block is 32 bytes so the least significant 5 bits are always zero.
- There is a particular mapping from each of the 19 block offset bits to the 256 bits of the flash encryption key, to determine which bit is XORed with which. See the variable _FLASH_ENCRYPTION_TWEAK_PATTERN in espsecure.py for a list of these.
- For the full algorithm implemented in Python, see *_flash_encryption_operation()* in the espsecure.py source code.

11.7.2 Setting FLASH_CRYPT_CONFIG

The `FLASH_CRYPT_CONFIG` efuse determines the number of bits in the flash encryption key which are "tweaked" with the block offset. See *Flash Encryption Algorithm* for details.

First boot of the bootloader always sets this value to the maximum *0xF*.

It is possible to write these efuse manually, and write protect it before first boot in order to select different tweak values. This is not recommended.

It is strongly recommended to never write protect `FLASH_CRYPT_CONFIG` when it the value is zero. If this efuse is set to zero, no bits in the flash encryption key are tweaked and the flash encryption algorithm is equivalent to AES ECB mode.

CHAPTER 12

Secure Boot

Secure Boot is a feature for ensuring only your code can run on the chip. Data loaded from flash is verified on each reset.

Secure Boot is separate from the *Flash Encryption* feature, and you can use secure boot without encrypting the flash contents. However we recommend using both features together for a secure environment.

12.1 Background

- Most data is stored in flash. Flash access does not need to be protected from physical access in order for secure boot to function, because critical data is stored (non-software-accessible) in Efuses internal to the chip.
- Efuses are used to store the secure bootloader key (in efuse block 2), and also a single Efuse bit (ABS_DONE_0) is burned (written to 1) to permanently enable secure boot on the chip. For more details about efuse, see the (forthcoming) chapter in the Technical Reference Manual.
- To understand the secure boot process, first familiarise yourself with the standard ESP-IDF boot process.
- Both stages of the boot process (initial software bootloader load, and subsequent partition & app loading) are verified by the secure boot process, in a "chain of trust" relationship.

12.2 Secure Boot Process Overview

This is a high level overview of the secure boot process. Step by step instructions are supplied under *How To Enable Secure Boot*. Further in-depth details are supplied under *Technical Details*:

1. The options to enable secure boot are provided in the `make menuconfig` hierarchy, under "Secure Boot Configuration".
2. Secure Boot defaults to signing images and partition table data during the build process. The "Secure boot private signing key" config item is a file path to a ECDSA public/private key pair in a PEM format file.
3. The software bootloader image is built by esp-idf with secure boot support enabled and the public key (signature verification) portion of the secure boot signing key compiled in. This software bootloader image is flashed at offset 0x1000.
4. On first boot, the software bootloader follows the following process to enable secure boot:
    - Hardware secure boot support generates a device secure bootloader key (generated via hardware RNG, then stored read/write protected in efuse), and a secure digest. The digest is derived from the key, an IV, and the bootloader image contents.

- The secure digest is flashed at offset 0x0 in the flash.
- Depending on Secure Boot Configuration, efuses are burned to disable JTAG and the ROM BASIC interpreter (it is strongly recommended these options are turned on.)
- Bootloader permanently enables secure boot by burning the ABS_DONE_0 efuse. The software bootloader then becomes protected (the chip will only boot a bootloader image if the digest matches.)

5. On subsequent boots the ROM bootloader sees that the secure boot efuse is burned, reads the saved digest at 0x0 and uses hardware secure boot support to compare it with a newly calculated digest. If the digest does not match then booting will not continue. The digest and comparison are performed entirely by hardware, and the calculated digest is not readable by software. For technical details see *Hardware Secure Boot Support*.
6. When running in secure boot mode, the software bootloader uses the secure boot signing key (the public key of which is embedded in the bootloader itself, and therefore validated as part of the bootloader) to verify the signature appended to all subsequent partition tables and app images before they are booted.

12.3 Keys

The following keys are used by the secure boot process:

- "secure bootloader key" is a 256-bit AES key that is stored in Efuse block 2. The bootloader can generate this key itself from the internal hardware random number generator, the user does not need to supply it (it is optionally possible to supply this key, see *Re-Flashable Software Bootloader*). The Efuse holding this key is read & write protected (preventing software access) before secure boot is enabled.
- "secure boot signing key" is a standard ECDSA public/private key pair (see *Image Signing Algorithm*) in PEM format.
  - The public key from this key pair (for signature verificaton but not signature creation) is compiled into the software bootloader and used to verify the second stage of booting (partition table, app image) before booting continues. The public key can be freely distributed, it does not need to be kept secret.
  - The private key from this key pair *must be securely kept private*, as anyone who has this key can authenticate to any bootloader that is configured with secure boot and the matching public key.

12.4 How To Enable Secure Boot

1. Run `make menuconfig`, navigate to "Secure Boot Configuration" and select the option "One-time Flash". (To understand the alternative "Reflashable" choice, see *Re-Flashable Software Bootloader*.)
2. Select a name for the secure boot signing key. This option will appear after secure boot is enabled. The file can be anywhere on your system. A relative path will be evaluated from the project directory. The file does not need to exist yet.
3. Set other menuconfig options (as desired). Pay particular attention to the "Bootloader Config" options, as you can only flash the bootloader once. Then exit menuconfig and save your configuration
4. The first time you run `make`, if the signing key is not found then an error message will be printed with a command to generate a signing key via `espsecure.py generate_signing_key`.

   IMPORTANT A signing key generated this way will use the best random number source available to the OS and its Python installation (*/dev/urandom* on OSX/Linux and *CryptGenRandom()* on Windows). If this random number source is weak, then the private key will be weak.

   IMPORTANT For production environments, we recommend generating the keypair using openssl or another industry standard encryption program. See *Generating Secure Boot Signing Key* for more details.

5. Run `make bootloader` to build a secure boot enabled bootloader. The output of *make* will include a prompt for a flashing command, using *esptool.py write_flash*.
6. When you're ready to flash the bootloader, run the specified command (you have to enter it yourself, this step is not performed by make) and then wait for flashing to complete. **Remember this is a one time flash, you can't change the bootloader after this!**.
7. Run *make flash* to build and flash the partition table and the just-built app image. The app image will be signed using the signing key you generated in step 4.

   NOTE: *make flash* doesn't flash the bootloader if secure boot is enabled.
8. Reset the ESP32 and it will boot the software bootloader you flashed. The software bootloader will enable secure boot on the chip, and then it verifies the app image signature and boots the app. You should watch the serial console output from the ESP32 to verify that secure boot is enabled and no errors have occured due to the build configuration.

NOTE Secure boot won't be enabled until after a valid partition table and app image have been flashed. This is to prevent accidents before the system is fully configured.

9. On subsequent boots, the secure boot hardware will verify the software bootloader has not changed (using the secure bootloader key) and then the software bootloader will verify the signed partition table and app image (using the public key portion of the secure boot signing key).

12.5 Re-Flashable Software Bootloader

Configuration "Secure Boot: One-Time Flash" is the recommended configuration for production devices. In this mode, each device gets a unique key that is never stored outside the device.

However, an alternative mode "Secure Boot: Reflashable" is also available. This mode allows you to supply a 256-bit key file that is used for the secure bootloader key. As you have the key file, you can generate new bootloader images and secure boot digests for them.

In the esp-idf build process, this 256-bit key file is derived from the app signing key generated during the generate_signing_key step above. The private key's SHA-256 digest is used as the 256-bit secure bootloader key. This is a convenience so you only need to generate/protect a single private key.

NOTE: Although it's possible, we strongly recommend not generating one secure boot key and flashing it to every device in a production environment. The "One-Time Flash" option is recommended for production environments.

To enable a reflashable bootloader:

1. In the `make menuconfig` step, select "Bootloader Config" -> "Secure Boot" -> "Reflashable".
2. Follow the steps shown above to choose a signing key file, and generate the key file.
3. Run `make bootloader`. A 256-bit key file will be created, derived from the private key that is used for signing. Two sets of flashing steps will be printed - the first set of steps includes an `espefuse.py burn_key` command which is used to write the bootloader key to efuse. (Flashing this key is a one-time-only process.) The second set of steps can be used to reflash the bootloader with a pre-calculated digest (generated during the build process).
4. Resume from *Step 6<Secure Boot Process Overview>* of the one-time process, to flash the bootloader and enable secure boot. Watch the console log output closely to ensure there were no errors in the secure boot configuration.

12.6 Generating Secure Boot Signing Key

The build system will prompt you with a command to generate a new signing key via `espsecure.py generate_signing_key`. This uses the python-ecdsa library, which in turn uses Python's os.urandom() as a random number source.

The strength of the signing key is proportional to (a) the random number source of the system, and (b) the correctness of the algorithm used. For production devices, we recommend generating signing keys from a system with a quality entropy source, and using the best available EC key generation utilities.

For example, to generate a signing key using the openssl command line:

```
` openssl ecparam -name prime256v1 -genkey -noout -out my_secure_boot_signing_key.pem
`
```

Remember that the strength of the secure boot system depends on keeping the signing key private.

12.7 Remote Signing of Images

For production builds, it can be good practice to use a remote signing server rather than have the signing key on the build machine (which is the default esp-idf secure boot configuration). The espsecure.py command line program can be used to sign app images & partition table data for secure boot, on a remote system.

To use remote signing, disable the option "Sign binaries during build". The private signing key does not need to be present on the build system. However, the public (signature verification) key is required because it is compiled into the bootloader (and can be used to verify image signatures during OTA updates.

To extract the public key from the private key:

```
espsecure.py extract_public_key --keyfile PRIVATE_SIGNING_KEY PUBLIC_VERIFICATION_KEY
```

The path to the public signature verification key needs to be specified in the menuconfig under "Secure boot public signature verification key" in order to build the secure bootloader.

After the app image and partition table are built, the build system will print signing steps using espsecure.py:

```
espsecure.py sign_data --keyfile PRIVATE_SIGNING_KEY BINARY_FILE
```

The above command appends the image signature to the existing binary. You can use the –output argument to place the binary with signature appended into a separate file:

```
espsecure.py sign_data --keyfile PRIVATE_SIGNING_KEY --output SIGNED_BINARY_FILE BINARY_FILE
```

12.8 Secure Boot Best Practices

- Generate the signing key on a system with a quality source of entropy.
- Keep the signing key private at all times. A leak of this key will compromise the secure boot system.
- Do not allow any third party to observe any aspects of the key generation or signing process using espsecure.py. Both processes are vulnerable to timing or other side-channel attacks.
- Enable all secure boot options in the Secure Boot Configuration. These include flash encryption, disabling of JTAG, disabling BASIC ROM interpeter, and disabling the UART bootloader encrypted flash access.

12.9 Technical Details

The following sections contain low-level descriptions of various technical functions:

12.9.1 Hardware Secure Boot Support

The Secure Boot support hardware can perform three basic operations:

1. Generate a random sequence of bytes from a hardware random number generator.
2. Generate a digest from data (usually the bootloader image from flash) using a key stored in Efuse block 2. The key in Efuse can (& should) be read/write protected, which prevents software access. For full details of this algorithm see *Secure Bootloader Digest Algorithm*. The digest can only be read back by software if Efuse ABS_DONE_0 is *not* burned (ie still 0).
3. Generate a digest from data (usually the bootloader image from flash) using the same algorithm as step 2 and compare it to a pre-calculated digest supplied in a buffer (usually read from flash offset 0x0). The hardware returns a true/false comparison without making the digest available to software. This function is available even when Efuse ABS_DONE_0 is burned.

12.9.2 Secure Bootloader Digest Algorithm

Starting with an "image" of binary data as input, this algorithm generates a digest as output. The digest is sometimes referred to as an "abstract" in hardware documentation.

For a Python version of this algorithm, see the *espsecure.py* tool in the components/esptool_py directory.

Items marked with (^) are to fulfill hardware restrictions, as opposed to cryptographic restrictions.

1. Prefix the image with a 128 byte randomly generated IV.
2. If the image length is not modulo 128, pad the image to a 128 byte boundary with 0xFF. (^)
3. For each 16 byte plaintext block of the input image: - Reverse the byte order of the plaintext input block (^) - Apply AES256 in ECB mode to the plaintext block. - Reverse the byte order of the ciphertext output block. (^) - Append to the overall ciphertext output.
4. Byte-swap each 4 byte word of the ciphertext (^)
5. Calculate SHA-512 of the ciphertext.

Output digest is 192 bytes of data: The 128 byte IV, followed by the 64 byte SHA-512 digest.

12.9.3 Image Signing Algorithm

Deterministic ECDSA as specified by *RFC6979*.

- Curve is NIST256p (openssl calls this curve "prime256v1", it is also sometimes called secp256r1).
- Hash function is SHA256.
- Key format used for storage is PEM. - In the bootloader, the public key (for signature verification) is flashed as 64 raw bytes.
- Image signature is 68 bytes - a 4 byte version word (currently zero), followed by a 64 bytes of signature data. These 68 bytes are appended to an app image or partition table data.

12.9.4 Manual Commands

Secure boot is integrated into the esp-idf build system, so *make* will automatically sign an app image if secure boot is enabled. *make bootloader* will produce a bootloader digest if menuconfig is configured for it.

However, it is possible to use the *espsecure.py* tool to make standalone signatures and digests.

To sign a binary image:

```
espsecure.py sign_data --keyfile ./my_signing_key.pem --output ./image_signed.bin image-unsigned.bin
```

Keyfile is the PEM file containing an ECDSA private signing key.

To generate a bootloader digest:

```
espsecure.py digest_secure_bootloader --keyfile ./securebootkey.bin --output ./bootloader-digest.bin
```

Keyfile is the 32 byte raw secure boot key for the device. To flash this digest onto the device:

```
esptool.py write_flash 0x0 bootloader-digest.bin
```

CHAPTER 13

Deep Sleep Wake Stubs

ESP32 supports running a "deep sleep wake stub" when coming out of deep sleep. This function runs immediately as soon as the chip wakes up - before any normal initialisation, bootloader, or ESP-IDF code has run. After the wake stub runs, the SoC can go back to sleep or continue to start ESP-IDF normally.

Deep sleep wake stub code is loaded into "RTC Fast Memory" and any data which it uses must also be loaded into RTC memory. RTC memory regions hold their contents during deep sleep.

13.1 Rules for Wake Stubs

Wake stub code must be carefully written:

- As the SoC has freshly woken from sleep, most of the peripherals are in reset states. The SPI flash is unmapped.
- The wake stub code can only call functions implemented in ROM or loaded into RTC Fast Memory (see below.)
- The wake stub code can only access data loaded in RTC memory. All other RAM will be unintiailised and have random contents. The wake stub can use other RAM for temporary storage, but the contents will be overwritten when the SoC goes back to sleep or starts ESP-IDF.
- RTC memory must include any read-only data (.rodata) used by the stub.
- Data in RTC memory is initialised whenever the SoC restarts, except when waking from deep sleep. When waking from deep sleep, the values which were present before going to sleep are kept.
- Wake stub code is a part of the main esp-idf app. During normal running of esp-idf, functions can call the wake stub functions or access RTC memory. It is as if these were regular parts of the app.

13.2 Implementing A Stub

The wake stub in esp-idf is called `esp_wake_deep_sleep()`. This function runs whenever the SoC wakes from deep sleep. There is a default version of this function provided in esp-idf, but the default function is weak-linked so if your app contains a function named `esp_wake_deep_sleep()` then this will override the default.

If supplying a custom wake stub, the first thing it does should be to call `esp_default_wake_deep_sleep()`.

It is not necessary to implement `esp_wake_deep_sleep()` in your app in order to use deep sleep. It is only necessary if you want to have special behaviour immediately on wake.

If you want to swap between different deep sleep stubs at runtime, it is also possible to do this by calling the `esp_set_deep_sleep_wake_stub()` function. This is not necessary if you only use the default `esp_wake_deep_sleep()` function.

All of these functions are declared in the `esp_deepsleep.h` header under components/esp32.

13.3 Loading Code Into RTC Memory

Wake stub code must be resident in RTC Fast Memory. This can be done in one of two ways.

The first way is to use the `RTC_IRAM_ATTR` attribute to place a function into RTC memory:

```
void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    // Add additional functionality here
}
```

The second way is to place the function into any source file whose name starts with `rtc_wake_stub`. Files names `rtc_wake_stub*` have their contents automatically put into RTC memory by the linker.

The first way is simpler for very short and simple code, or for source files where you want to mix "normal" and "RTC" code. The second way is simpler when you want to write longer pieces of code for RTC memory.

13.4 Loading Data Into RTC Memory

Data used by stub code must be resident in RTC Slow Memory. This memory is also used by the ULP.

Specifying this data can be done in one of two ways:

The first way is to use the `RTC_DATA_ATTR` and `RTC_RODATA_ATTR` to specify any data (writeable or read-only, respectivley) which should be loaded into RTC slow memory:

```
RTC_DATA_ATTR int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    static RTC_RODATA_ATTR const char fmt_str[] = "Wake count %d\n";
    ets_printf(fmt_str, wake_count++);
}
```

Unfortunately, any string constants used in this way must be declared as arrays and marked with RTC_RODATA_ATTR, as shown in the example above.

The second way is to place the data into any source file whose name starts with `rtc_wake_stub`.

For example, the equivalent example in `rtc_wake_stub_counter.c`:

```
int wake_count;

void RTC_IRAM_ATTR esp_wake_deep_sleep(void) {
    esp_default_wake_deep_sleep();
    ets_printf("Wake count %d\n", wake_count++);
}
```

The second way is a better option if you need to use strings, or write other more complex code.

CHAPTER 14

Unit Testing in ESP32

ESP-IDF comes with a unit test app based on Unity - unit test framework. Unit tests are integrated in the ESP-IDF repository and are placed in `test` subdirectory of each component respectively.

14.1 Adding unit tests

Unit tests are added in the `test` subdirectory of the respective component. Tests are added in C files, a single C file can include multiple test cases. Test files start with the word "test".

The test file should include unity.h and the header for the C module to be tested.

Tests are added in a function in the C file as follows:

```
TEST_CASE("test name", "[module name]"
{
        // Add test here
}
```

First argument is a descriptive name for the test, second argument is an identifier in square brackets. Identifiers are used to group related test, or tests with specific properties.

There is no need to add a main function with `UNITY_BEGIN()` and `UNITY_END()` in each test case. `unity_platform.c` will run `UNITY_BEGIN()`, run the tests cases, and then call `UNITY_END()`.

Each *test* subdirectory needs to include component.mk file with at least the following line of code:

```
COMPONENT_ADD_LDFLAGS = -Wl,--whole-archive -l$(COMPONENT_NAME) -Wl,--no-whole-archive
```

See http://www.throwtheswitch.org/unity for more information about writing tests in Unity.

14.2 Building unit test app

Follow the setup instructions in the top-level esp-idf README. Make sure that IDF_PATH environment variable is set to point to the path of esp-idf top-level directory.

Change into tools/unit-test-app directory to configure and build it:

- *make menuconfig* - configure unit test app.
- *make TESTS_ALL=1* - build unit test app with tests for each component having tests in the `test` subdirectory.
- *make TEST_COMPONENTS='xxx'* - build unit test app with tests for specific components.

When the build finishes, it will print instructions for flashing the chip. You can simply run `make flash` to flash all build output.

You can also run `make flash TESTS_ALL=1` or `make TEST_COMPONENTS='xxx'` to build and flash. Everything needed will be rebuilt automatically before flashing.

Use menuconfig to set the serial port for flashing.

14.3 Running unit tests

After flashing reset the ESP32 and it will boot the unit test app.

Unit test app prints a test menu with all available tests.

Test cases can be run by inputting one of the following:

- Test case name in quotation marks to run a single test case
- Test case index to run a single test case
- Module name in square brackets to run all test cases for a specific module
- An asterisk to run all test cases

CHAPTER 15

Wi-Fi API

15.1 Wi-Fi

15.1.1 Overview

Instructions

15.1.2 Application Example

Simple code showing how to connect ESP32 module to an Access Point: esp-idf-template.

15.1.3 API Reference

Header Files

- esp32/include/esp_wifi.h

15.1.4 Macros

Warning: doxygendefine: Cannot find define "WIFI_INIT_CONFIG_DEFAULT" in doxygen xml output for project "esp32-idf" from directory: xml/

15.1.5 Type Definitions

Warning: doxygentypedef: Cannot find typedef "wifi_promiscuous_cb_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygentypedef: Cannot find typedef "esp_vendor_ie_cb_t" in doxygen xml output for project "esp32-idf" from directory: xml/

15.1.6 Functions

Warning: doxygenfunction: Cannot find function “esp_wifi_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_deinit” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_connect” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_disconnect” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_clear_fast_connect” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_deauth_sta” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_scan_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_scan_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_scan_get_ap_num” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_scan_get_ap_records” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_sta_get_ap_info” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_ps” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_ps” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_protocol” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_protocol” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_bandwidth” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_bandwidth” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_channel” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_channel” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_country” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_country” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_mac” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_get_mac” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_promiscuous_rx_cb” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_wifi_set_promiscuous” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_get_promiscuous" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_set_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_get_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_ap_get_sta_list" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_set_storage" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_set_auto_connect" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_get_auto_connect" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_set_vendor_ie" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_wifi_set_vendor_ie_cb" in doxygen xml output for project "esp32-idf" from directory: xml/

15.2 Smart Config

15.2.1 API Reference

Header Files

- esp32/include/esp_smartconfig.h

Type Definitions

Warning: doxygentypedef: Cannot find typedef "sc_callback_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_smartconfig_get_version” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_smartconfig_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_smartconfig_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_esptouch_set_timeout” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_smartconfig_set_type” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_smartconfig_fast_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Example code for this API section is provided in wifi directory of ESP-IDF examples.

CHAPTER 16

Bluetooth API

16.1 Controller && VHCI

16.1.1 Overview

Instructions

16.1.2 Application Example

Check bluetooth folder in ESP-IDF examples, which contains the following example:

bluetooth/ble_adv

> This is a BLE advertising demo with virtual HCI interface. Send Reset/ADV_PARAM/ADV_DATA/ADV_ENABLE HCI command for BLE advertising.

16.1.3 API Reference

Header Files

- bt/include/bt.h

Type Definitions

Warning: doxygentypedef: Cannot find typedef “esp_vhci_host_callback_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Structures

Warning: doxygenstruct: Cannot find class “esp_vhci_host_callback” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "esp_bt_controller_init" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vhci_host_check_send_available" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vhci_host_send_packet" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vhci_host_register_callback" in doxygen xml output for project "esp32-idf" from directory: xml/

16.2 BT COMMON

16.2.1 BT GENERIC DEFINES

Overview

Instructions

Application Example

Instructions

API Reference

Header Files

- bt/bluedroid/api/include/esp_bt_defs.h

Macros

Warning: doxygendefine: Cannot find define "ESP_DEFAULT_GATT_IF" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_CONN_PARAM_UNDEF" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_IS_VALID_PARAM" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_UUID_LEN_16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_UUID_LEN_32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_UUID_LEN_128” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_BD_ADDR_LEN” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_APP_ID_MIN” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_APP_ID_MAX” in doxygen xml output for project “esp32-idf” from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef “esp_bd_addr_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “esp_bt_status_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_bt_dev_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_bd_addr_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_addr_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Functions

16.2.2 BT MAIN API

Overview

Instructions

Application Example

Instructions

API Reference

Header Files

- bt/bluedroid/api/include/esp_bt_main.h

Macros

Type Definitions

Enumerations

Warning: doxygenenum: Cannot find enum “esp_bluedroid_status_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Functions

Warning: doxygenfunction: Cannot find function “esp_bluedroid_get_status” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_bluedroid_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_bluedroid_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_bluedroid_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_bluedroid_deinit” in doxygen xml output for project “esp32-idf” from directory: xml/

16.2.3 BT DEVICE APIs

Overview

Bluetooth device reference APIs.

Instructions

Application Example

Instructions

API Reference

Header Files

- bt/bluedroid/api/include/esp_bt_device.h

Macros

Type Definitions

Enumerations

Structures

Functions

Warning: doxygenfunction: Cannot find function “esp_bt_dev_get_address” in doxygen xml output for project “esp32-idf” from directory: xml/

16.3 BT COMMON

16.3.1 GAP API

Overview

Instructions

Application Example

Check bluetooth folder in ESP-IDF examples, which contains the following examples:

bluetooth/gatt_server, bluetooth/gatt_client

The two demos use different GAP APIs, such like advertising, scan, set device name and others.

API Reference

Header Files

- bt/bluedroid/api/include/esp_gap_ble_api.h

Macros

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_LIMIT_DISC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_GEN_DISC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_BREDR_NOT_SPT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_DMT_CONTROLLER_SPT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_DMT_HOST_SPT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_FLAG_NON_LIMIT_DISC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_ADV_DATA_LEN_MAX" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_BLE_SCAN_RSP_DATA_LEN_MAX" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "esp_gap_ble_cb_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “esp_gap_ble_cb_event_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_adv_data_type” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_adv_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_adv_channel_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_adv_filter_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_own_addr_src_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_scan_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_scan_filter_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gap_search_evt_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_ble_evt_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “esp_ble_adv_params_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_adv_data_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_scan_params_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_conn_update_params_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_adv_data_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_scan_rsp_data_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_scan_param_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_scan_result_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_adv_data_raw_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gap_cb_param_t::ble_scan_rsp_data_raw_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_ble_gap_register_callback” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_config_adv_data” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_set_scan_params” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_start_scanning” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_stop_scanning” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_start_advertising” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_stop_advertising” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_update_conn_params” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_set_pkt_data_len” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_set_rand_addr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_config_local_privacy” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_set_device_name” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_resolve_adv_data” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_config_adv_data_raw” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gap_config_scan_rsp_data_raw” in doxygen xml output for project “esp32-idf” from directory: xml/

16.3.2 GATT DEFINES

Overview

Instructions

Application Example

Instructions

API Reference

Header Files

- bt/bluedroid/api/include/esp_gatt_defs.h

Macros

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_IMMEDIATE_ALERT_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_LINK_LOSS_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_TX_POWER_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CURRENT_TIME_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_REF_TIME_UPDATE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_NEXT_DST_CHANGE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GLUCOSE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HEALTH_THERMOM_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_DEVICE_INFO_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HEART_RATE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_PHONE_ALERT_STATUS_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_BATTERY_SERVICE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_BLOOD_PRESSURE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_ALERT_NTF_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SCAN_PARAMETERS_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RUNNING_SPEED_CADENCE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CYCLING_SPEED_CADENCE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CYCLING_POWER_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_LOCATION_AND_NAVIGATION_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_USER_DATA_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_WEIGHT_SCALE_SVC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_PRI_SERVICE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SEC_SERVICE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_INCLUDE_SERVICE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_DECLARE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_EXT_PROP" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_DESCRIPTION" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_CLIENT_CONFIG" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_SRVR_CONFIG" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_PRESENT_FORMAT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_AGG_FORMAT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CHAR_VALID_RANGE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_EXT_RPT_REF_DESCR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RPT_REF_DESCR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GAP_DEVICE_NAME" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GAP_ICON" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GAP_PREF_CONN_PARAM" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GAP_CENTRAL_ADDR_RESOL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GATT_SRV_CHGD" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_ALERT_LEVEL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_TX_POWER_LEVEL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CURRENT_TIME" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_LOCAL_TIME_INFO" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_REF_TIME_INFO" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_NW_STATUS" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_NW_TRIGGER" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_ALERT_STATUS" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RINGER_CP" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RINGER_SETTING" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GM_MEASUREMENT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GM_CONTEXT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GM_CONTROL_POINT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_GM_FEATURE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SYSTEM_ID" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_MODEL_NUMBER_STR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SERIAL_NUMBER_STR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_FW_VERSION_STR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HW_VERSION_STR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SW_VERSION_STR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_MANU_NAME" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_IEEE_DATA" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_PNP_ID" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_INFORMATION" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_REPORT_MAP" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_CONTROL_POINT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_REPORT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_PROTO_MODE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_BT_KB_INPUT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_BT_KB_OUTPUT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_HID_BT_MOUSE_INPUT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_HEART_RATE_MEAS" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_BODY_SENSOR_LOCATION" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_HEART_RATE_CNTL_POINT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_BATTERY_LEVEL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SC_CONTROL_POINT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SENSOR_LOCATION" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RSC_MEASUREMENT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_RSC_FEATURE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CSC_MEASUREMENT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_CSC_FEATURE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SCAN_INT_WINDOW" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_UUID_SCAN_REFRESH" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_ILLEGAL_UUID" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_ILLEGAL_HANDLE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_ATTR_HANDLE_MAX" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_MAX_ATTR_LEN" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_RSP_BY_APP" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_AUTO_RSP" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_GATT_IF_NONE” in doxygen xml output for project “esp32-idf” from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef “esp_gatt_if_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “esp_gatt_prep_write_type” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_status_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_conn_reason_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_auth_req_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_perm_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_char_prop_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_gatt_write_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “esp_attr_desc_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_attr_control_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_gatts_attr_db_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_attr_value_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_gatts_incl_svc_desc_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_gatts_incl128_svc_desc_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_gatt_value_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_gatt_rsp_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

16.3.3 GATT SERVER API

Overview

Instructions

Application Example

Check bluetooth folder in ESP-IDF examples, which contains the following example:

bluetooth/gatt_server

> This is a GATT server demo. Use GATT API to create a GATT server with send advertising. This GATT server can be connected and the service can be discovery.

API Reference

Header Files

- bt/bluedroid/api/include/esp_gatts_api.h

Macros

Warning: doxygendefine: Cannot find define “ESP_GATT_PREP_WRITE_CANCEL” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_GATT_PREP_WRITE_EXEC” in doxygen xml output for project “esp32-idf” from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef “esp_gatts_cb_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “esp_gatts_cb_event_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_reg_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_read_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_write_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_exec_write_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_mtu_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_conf_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_create_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_add_incl_srvc_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_add_char_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_add_char_descr_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_delete_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_start_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_stop_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_connect_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_disconnect_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_congest_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_rsp_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_add_attr_tab_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gatts_cb_param_t::gatts_set_attr_val_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_register_callback” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_app_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_app_unregister” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_create_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_create_attr_tab” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_add_included_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_add_char” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_add_char_descr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_delete_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_start_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_stop_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_send_indicate” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_send_response” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_set_attr_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_get_attr_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_open” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gatts_close” in doxygen xml output for project “esp32-idf” from directory: xml/

16.3.4 GATT CLIENT API

Overview

Instructions

Application Example

Check bluetooth folder in ESP-IDF examples, which contains the following examples:

bluetooth/gatt_client

> This is a GATT client demo. This demo can scan devices, connect to the GATT server and discover the service.

API Reference

Header Files

- bt/bluedroid/api/include/esp_gattc_api.h

Macros

Warning: doxygendefine: Cannot find define "ESP_GATT_DEF_BLE_MTU_SIZE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_GATT_MAX_MTU_SIZE" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "esp_gattc_cb_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum "esp_gattc_cb_event_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class "esp_ble_gattc_cb_param_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_ble_gattc_cb_param_t::gattc_reg_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_ble_gattc_cb_param_t::gattc_open_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_close_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_cfg_mtu_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_search_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_search_res_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_read_char_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_write_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_exec_cmpl_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_notify_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_srvc_chg_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_congest_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_get_char_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_get_descr_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_get_incl_srvc_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_reg_for_notify_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_ble_gattc_cb_param_t::gattc_unreg_for_notify_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_register_callback” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_app_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_app_unregister” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_open” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_close” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_config_mtu” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_search_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_get_characteristic” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_get_descriptor” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_get_included_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_read_char” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_read_char_descr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_write_char” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_write_char_descr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_prepare_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_execute_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_register_for_notify” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_ble_gattc_unregister_for_notify” in doxygen xml output for project “esp32-idf” from directory: xml/

16.3.5 BLUFI API

Overview

BLUFI is a profile based GATT to config ESP32 WIFI to connect/disconnect AP or setup a softap and etc. Use should concern these things: 1. The event sent from profile. Then you need to do something as the event indicate. 2. Security reference. You can write your own Security functions such as symmetrical encryption/decryption and checksum functions. Even you can define the “Key Exchange/Negotiation” procedure.

Application Example

Check bluetooth folder in ESP-IDF examples, which contains the following example:

bluetooth/blufi

> This is a BLUFI demo. This demo can set ESP32’s wifi to softap/station/softap&station mode and config wifi connections.

API Reference

Header Files

- bt/bluedroid/api/include/esp_blufi_api.h

Macros

Type Definitions

Warning: doxygentypedef: Cannot find typedef “esp_blufi_event_cb_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “esp_blufi_negotiate_data_handler_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “esp_blufi_encrypt_func_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “esp_blufi_decrypt_func_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “esp_blufi_checksum_func_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “esp_blufi_cb_event_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_blufi_sta_conn_state_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_blufi_init_state_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_blufi_deinit_state_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “esp_blufi_extra_info_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_cb_param_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_cb_param_t::blufi_init_finish_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_cb_param_t::blufi_deinit_finish_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_cb_param_t::blufi_set_wifi_mode_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_cb_param_t::blufi_connect_evt_param” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_disconnect_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_sta_bssid_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_sta_ssid_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_sta_passwd_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_softap_ssid_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_softap_passwd_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_softap_max_conn_num_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_softap_auth_mode_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_softap_channel_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_username_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_ca_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_client_cert_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_server_cert_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_client_pkey_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_blufi_cb_param_t::blufi_recv_server_pkey_evt_param" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class “esp_blufi_callbacks_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_blufi_register_callbacks” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_blufi_profile_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_blufi_profile_deinit” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_blufi_send_wifi_conn_report” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_blufi_get_version” in doxygen xml output for project “esp32-idf” from directory: xml/

Example code for this API section is provided in bluetooth directory of ESP-IDF examples.

CHAPTER 17

Ethernet API

17.1 ETHERNET

17.1.1 Application Example

Ethernet example: ethernet/ethernet.

17.1.2 API Reference

Header Files

- ethernet/include/esp_eth.h

Macros

Type Definitions

Warning: doxygentypedef: Cannot find typedef “eth_phy_check_link_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_phy_check_init_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_phy_get_speed_mode_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_phy_get_duplex_mode_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_phy_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_tcpip_input_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_gpio_config_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “eth_phy_get_partner_pause_enable_func” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “eth_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “eth_speed_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “eth_duplex_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “eth_phy_base_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “eth_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “esp_eth_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_tx” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_get_mac” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_smi_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_smi_read” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_eth_free_rx_buf” in doxygen xml output for project “esp32-idf” from directory: xml/

Example code for this API section is provided in ethernet directory of ESP-IDF examples.

CHAPTER 18

Peripherals API

18.1 GPIO

18.1.1 Overview

The ESP32 chip features 40 physical GPIO pads. Some GPIO pads cannot be used or do not have the corresponding pin on the chip package(refer to technical reference manual). Each pad can be used as a general purpose I/O or can be connected to an internal peripheral signal. Note that GPIO6-11 are usually used for SPI flash. GPIO34-39 can only be set as input mode.

18.1.2 Application Example

GPIO output and input interrupt example: peripherals/gpio.

18.1.3 API Reference

Header Files

- driver/include/driver/gpio.h

Macros

Warning: doxygendefine: Cannot find define "GPIO_SEL_0" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "GPIO_SEL_1" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "GPIO_SEL_2" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "GPIO_SEL_3" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_4” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_5” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_6” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_7” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_9” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_10” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_11” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_12” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_13” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_14” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_15” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_17” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_18” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_19” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_21” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_22” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_23” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_25” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_26” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_27” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_33” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_34” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_35” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_36” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_37” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_38” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SEL_39” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_0” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_1” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_2” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_3” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_4” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_5” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_6” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_7” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_9” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_10” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_11” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_12” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_13” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_14” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_15” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_17” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_18” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_19” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_20” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_21” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_22” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_23” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_25” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_26” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_27” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_33” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_34” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_35” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_36” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_37” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_38” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_REG_39” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_APP_CPU_INTR_ENA” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_APP_CPU_NMI_INTR_ENA” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PRO_CPU_INTR_ENA” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PRO_CPU_NMI_INTR_ENA” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_SDIO_EXT_INTR_ENA” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_MODE_DEF_INPUT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_MODE_DEF_OUTPUT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_MODE_DEF_OD” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_PIN_COUNT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “GPIO_IS_VALID_GPIO” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define "GPIO_IS_VALID_OUTPUT_GPIO" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "gpio_isr_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygentypedef: Cannot find typedef "gpio_isr_handle_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum "gpio_num_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "gpio_int_type_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "gpio_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "gpio_pullup_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "gpio_pulldown_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "gpio_pull_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class "gpio_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "gpio_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_set_intr_type” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_intr_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_intr_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_set_level” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_get_level” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_set_direction” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_set_pull_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_wakeup_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_wakeup_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_pullup_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_pullup_dis” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_pulldown_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_pulldown_dis” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “gpio_install_isr_service” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "gpio_uninstall_isr_service" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "gpio_isr_handler_add" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "gpio_isr_handler_remove" in doxygen xml output for project "esp32-idf" from directory: xml/

18.2 UART

18.2.1 Overview

Instructions

18.2.2 Application Example

Configure uart settings and install uart driver to read/write using UART0 and UART1 interfaces: peripherals/uart.

18.2.3 API Reference

Header Files

- driver/include/driver/uart.h

Data Structures

Warning: doxygenstruct: Cannot find class "uart_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "uart_intr_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "uart_event_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Macros

Warning: doxygendefine: Cannot find define "UART_FIFO_LEN" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INTR_MASK” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_LINE_INV_MASK” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_BITRATE_MAX” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_PIN_NO_CHANGE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INVERSE_DISABLE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INVERSE_RXD” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INVERSE_CTS” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INVERSE_TXD” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “UART_INVERSE_RTS” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “uart_word_length_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “uart_stop_bits_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “uart_port_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “uart_parity_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “uart_hw_flowcontrol_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “uart_event_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “uart_set_word_length” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_word_length” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_stop_bits” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_stop_bits” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_parity” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_parity” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_baudrate” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_baudrate” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_line_inverse” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_hw_flow_ctrl” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_hw_flow_ctrl” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_clear_intr_status” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_enable_intr_mask” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_disable_intr_mask” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_enable_rx_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_disable_rx_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_disable_tx_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_enable_tx_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_pin” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_rts” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_set_dtr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_param_config” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_intr_config” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_driver_install” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_driver_delete” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_wait_tx_done” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_tx_chars” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_write_bytes” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_write_bytes_with_break” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_read_bytes” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_flush” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_get_buffered_data_len” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_disable_pattern_det_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “uart_enable_pattern_det_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

18.3 I2C

18.3.1 Overview

ESP32 has two I2C controllers which can be set as master mode or slave mode.

18.3.2 Application Example

I2C master and slave example: peripherals/i2c.

18.3.3 API Reference

Header Files

- driver/include/driver/i2c.h

Macros

Warning: doxygendefine: Cannot find define “I2C_APB_CLK_FREQ” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “I2C_FIFO_LEN” in doxygen xml output for project “esp32-idf” from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef “i2c_cmd_handle_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum “i2c_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “i2c_rw_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “i2c_trans_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “i2c_opmode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “i2c_port_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “i2c_addr_mode_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “i2c_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “i2c_driver_install” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_driver_delete” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_param_config” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_reset_tx_fifo” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_reset_rx_fifo” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_isr_free” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_set_pin” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_write_byte” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_read_byte” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_read” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_master_cmd_begin” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_slave_write_buffer” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “i2c_slave_read” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_set_period" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_get_period" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_set_start_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_get_start_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_set_stop_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_get_stop_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_set_data_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_get_data_timing" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_set_data_mode" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_get_data_mode" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_cmd_link_create" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "i2c_cmd_link_delete" in doxygen xml output for project "esp32-idf" from directory: xml/

18.4 SPI Master driver

18.4.1 Overview

The ESP32 has four SPI peripheral devices, called SPI0, SPI1, HSPI and VSPI. SPI0 is entirely dedicated to the flash cache the ESP32 uses to map the SPI flash device it is connected to into memory. SPI1 is connected to the same hardware lines as SPI0 and is used to write to the flash chip. HSPI and VSPI are free to use. SPI1, HSPI and VSPI all

have three chip select lines, allowing them to drive up to three SPI devices each as a master. The SPI peripherals also can be used in slave mode, driven from another SPI master.

The spi_master driver

The spi_master driver allows easy communicating with SPI slave devices, even in a multithreaded environment. It fully transparently handles DMA transfers to read and write data and automatically takes care of multiplexing between different SPI slaves on the same master

Terminology

The spi_master driver uses the following terms:

- Host: The SPI peripheral inside the ESP32 initiating the SPI transmissions. One of SPI, HSPI or VSPI. (For now, only HSPI or VSPI are actually supported in the driver; it will support all 3 peripherals somewhere in the future.)
- Bus: The SPI bus, common to all SPI devices connected to one host. In general the bus consists of the miso, mosi, sclk and optionally quadwp and quadhd signals. The SPI slaves are connected to these signals in parallel.
  - miso - Also known as q, this is the input of the serial stream into the ESP32
  - mosi - Also known as d, this is the output of the serial stream from the ESP32
  - sclk - Clock signal. Each data bit is clocked out or in on the positive or negative edge of this signal
  - quadwp - Write Protect signal. Only used for 4-bit (qio/qout) transactions.
  - quadhd - Hold signal. Only used for 4-bit (qio/qout) transactions.
- Device: A SPI slave. Each SPI slave has its own chip select (CS) line, which is made active when a transmission to/from the SPI slave occurs.
- Transaction: One instance of CS going active, data transfer from and/or to a device happening, and CS going inactive again. Transactions are atomic, as in they will never be interrupted by another transaction.

SPI transactions

A transaction on the SPI bus consists of five phases, any of which may be skipped:

- The command phase. In this phase, a command (0-16 bit) is clocked out.
- The address phase. In this phase, an address (0-64 bit) is clocked out.
- The read phase. The slave sends data to the master.
- The write phase. The master sends data to the slave.

In full duplex, the read and write phases are combined, causing the SPI host to read and write data simultaneously.

The command and address phase are optional in that not every SPI device will need to be sent a command and/or address. Tis is reflected in the device configuration: when the `command_bits` or `data_bits` fields are set to zero, no command or address phase is done.

Something similar is true for the read and write phase: not every transaction needs both data to be written as well as data to be read. When `rx_buffer` is NULL (and SPI_USE_RXDATA) is not set) the read phase is skipped. When `tx_buffer` is NULL (and SPI_USE_TXDATA) is not set) the write phase is skipped.

Using the spi_master driver

- Initialize a SPI bus by calling `spi_bus_initialize`. Make sure to set the correct IO pins in the `bus_config` struct. Take care to set signals that are not needed to -1.
- Tell the driver about a SPI slave device conencted to the bus by calling spi_bus_add_device. Make sure to configure any timing requirements the device has in the `dev_config` structure. You should now have a handle for the device, to be used when sending it a transaction.
- To interact with the device, fill one or more spi_transaction_t structure with any transaction parameters you need. Either queue all transactions by calling `spi_device_queue_trans`, later quering the result using `spi_device_get_trans_result`, or handle all requests synchroneously by feeding them into `spi_device_transmit`.
- Optional: to unload the driver for a device, call `spi_bus_remove_device` with the device handle as an argument
- Optional: to remove the driver for a bus, make sure no more drivers are attached and call `spi_bus_free`.

Transaction data

Normally, data to be transferred to or from a device will be read from or written to a chunk of memory indicated by the `rx_buffer` and `tx_buffer` members of the transaction structure. The SPI driver may decide to use DMA for transfers, so these buffers should be allocated in DMA-capable memory using `pvPortMallocCaps(size, MALLOC_CAP_DMA)`.

Sometimes, the amount of data is very small making it less than optimal allocating a separate buffer for it. If the data to be transferred is 32 bits or less, it can be stored in the transaction struct itself. For transmitted data, use the `tx_data` member for this and set the `SPI_USE_TXDATA` flag on the transmission. For received data, use `rx_data` and set `SPI_USE_RXDATA`. In both cases, do not touch the `tx_buffer` or `rx_buffer` members, because they use the same memory locations as `tx_data` and `rx_data`.

18.4.2 Application Example

Display graphics on the ILI9341-based 320x240 LCD: peripherals/spi_master.

18.4.3 API Reference

Header Files

- driver/include/driver/spi_master.h

Macros

Warning: doxygendefine: Cannot find define "SPI_DEVICE_TXBIT_LSBFIRST" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_RXBIT_LSBFIRST" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_BIT_LSBFIRST" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_3WIRE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_POSITIVE_CS" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_HALFDUPLEX" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_DEVICE_CLK_AS_CS" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_TRANS_MODE_DIO" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_TRANS_MODE_QIO" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_TRANS_MODE_DIOQIO_ADDR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_TRANS_USE_RXDATA" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SPI_TRANS_USE_TXDATA" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "spi_device_handle_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum "spi_host_device_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “spi_transaction_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “spi_bus_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “spi_device_interface_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

18.4.4 Functions

Warning: doxygenfunction: Cannot find function “spi_bus_initialize” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_bus_free” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_bus_add_device” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_bus_remove_device” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_device_queue_trans” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_device_get_trans_result” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_device_transmit” in doxygen xml output for project “esp32-idf” from directory: xml/

18.5 TIMER

18.5.1 Overview

ESP32 chip contains two hardware timer groups, each containing two general-purpose hardware timers.

They are all 64-bit generic timers based on 16-bit prescalers and 64-bit auto-reload-capable up/down counters.

18.5.2 Application Example

64-bit hardware timer example: peripherals/timer_group.

18.5.3 API Reference

Header Files

- driver/include/driver/timer.h

Macros

Warning: doxygendefine: Cannot find define "TIMER_BASE_CLK" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Enumerations

Warning: doxygenenum: Cannot find enum "timer_group_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_idx_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_count_dir_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_start_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_alarm_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_intr_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "timer_autoreload_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “timer_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “timer_get_counter_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_get_counter_time_sec” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_counter_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_pause” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_counter_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_auto_reload” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_divider” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_alarm_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_get_alarm_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_set_alarm” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_get_config” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_group_intr_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_group_intr_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_enable_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “timer_disable_intr” in doxygen xml output for project “esp32-idf” from directory: xml/

18.6 Pulse Counter

18.6.1 Overview

The PCNT (Pulse Counter) module is designed to count the number of rising and/or falling edges of an input signal. Each pulse counter unit has a 16-bit signed counter register and two channels that can be configured to either increment or decrement the counter. Each channel has a signal input that accepts signal edges to be detected, as well as a control input that can be used to enable or disable the signal input. The inputs have optional filters that can be used to discard unwanted glitches in the signal.

18.6.2 Application Example

Pulse counter with control signal and event interrupt example: peripherals/pcnt.

18.6.3 API Reference

Header Files

- driver/include/driver/pcnt.h

Macros

Type Definitions

Enumerations

Warning: doxygenenum: Cannot find enum "pcnt_ctrl_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "pcnt_count_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "pcnt_unit_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "pcnt_channel_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "pcnt_evt_type_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class "pcnt_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "pcnt_unit_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "pcnt_get_counter_value" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "pcnt_counter_pause" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "pcnt_counter_resume" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "pcnt_counter_clear" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_intr_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_intr_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_event_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_event_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_set_event_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_get_event_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_set_pin” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_filter_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_filter_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_set_filter_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_get_filter_value” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pcnt_set_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

18.7 Sigma-delta Modulation

18.7.1 Overview

ESP32 has a second-order sigma-delta modulation module. This driver configures the channels of the sigma-delta module.

18.7.2 Application Example

Sigma-delta Modulation example: peripherals/sigmadelta.

18.7.3 API Reference

Header Files

- driver/include/driver/sigmadelta.h

Macros

Type Definitions

Enumerations

Warning: doxygenenum: Cannot find enum "sigmadelta_channel_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class "sigmadelta_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "sigmadelta_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sigmadelta_set_duty" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sigmadelta_set_prescale" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sigmadelta_set_pin" in doxygen xml output for project "esp32-idf" from directory: xml/

18.8 LED Control

18.8.1 Overview

The LED control module is primarily designed to control the intensity of LEDs, although it can be used to generate PWM signals for other purposes as well. It has 16 channels which can generate independent waveforms that can be used to drive e.g. RGB LED devices. For maximum flexibility, the high-speed as well as the low-speed channels can be driven from one of four high-speed/low-speed timers. The PWM controller also has the ability to automatically increase or decrease the duty cycle gradually, allowing for fades without any processor interference.

18.8.2 Application Example

LEDC change duty cycle and fading control example: peripherals/ledc.

18.8.3 API Reference

Header Files

- driver/include/driver/ledc.h

Macros

Warning: doxygendefine: Cannot find define "LEDC_APB_CLK_HZ" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LEDC_REF_CLK_HZ" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "ledc_isr_handle_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum "ledc_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_intr_type_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_duty_direction_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_clk_src_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_timer_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_channel_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "ledc_timer_bit_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class "ledc_channel_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "ledc_timer_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "ledc_channel_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "ledc_timer_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "ledc_update_duty" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "ledc_stop" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "ledc_set_freq" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_get_freq” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_set_duty” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_get_duty” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_set_fade” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_timer_set” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_timer_rst” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_timer_pause” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_timer_resume” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_bind_channel_timer” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_set_fade_with_step” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_set_fade_with_time” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_fade_func_install” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_fade_func_uninstall” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “ledc_fade_start” in doxygen xml output for project “esp32-idf” from directory: xml/

18.9 RMT

18.9.1 Overview

The RMT (Remote Control) module driver can be used to send and receive infrared remote control signals. Due to flexibility of RMT module, the driver can also be used to generate many other types of signals.

18.9.2 Application Example

NEC remote control TX and RX example: peripherals/rmt_nec_tx_rx.

18.9.3 API Reference

Header Files

- driver/include/driver/rmt.h

Macros

Warning: doxygendefine: Cannot find define "RMT_MEM_BLOCK_BYTE_NUM" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "RMT_MEM_ITEM_NUM" in doxygen xml output for project "esp32-idf" from directory: xml/

Enumerations

Warning: doxygenenum: Cannot find enum "rmt_channel_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "rmt_mem_owner_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "rmt_source_clk_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "rmt_data_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "rmt_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "rmt_idle_level_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum “rmt_carrier_level_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Structures

Warning: doxygenstruct: Cannot find class “rmt_tx_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “rmt_rx_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “rmt_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “rmt_set_clk_div” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_clk_div” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_rx_idle_thresh” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_rx_idle_thresh” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_mem_block_num” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_mem_block_num” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_tx_carrier” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_mem_pd” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_mem_pd” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_tx_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_tx_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_rx_start” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_rx_stop” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_memory_rw_rst” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_memory_owner” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_memory_owner” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_tx_loop_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_tx_loop_mode” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_rx_filter” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_source_clk” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_source_clk” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_idle_level” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_status” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_intr_enable_mask” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_clr_intr_enable_mask” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_rx_intr_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_err_intr_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_tx_intr_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_evt_intr_en” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_set_pin” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_config” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_isr_register” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_fill_tx_items” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_driver_install” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_driver_uninstall” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_write_items” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_wait_tx_done” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “rmt_get_ringbuf_handler” in doxygen xml output for project “esp32-idf” from directory: xml/

Example code for this API section is provided in peripherals directory of ESP-IDF examples.

CHAPTER 19

System API

19.1 Memory allocation

19.1.1 Overview

The ESP32 has multiple types of RAM. Internally, there's IRAM, DRAM as well as RAM that can be used as both. It's also possible to connect external SPI flash to the ESP32; it's memory can be integrated into the ESP32s memory map using the flash cache.

In order to make use of all this memory, esp-idf has a capabilities-based memory allocator. Basically, if you want to have memory with certain properties (for example, DMA-capable, accessible by a certain PID, or capable of executing code), you can create an OR-mask of the required capabilities and pass that to pvPortMallocCaps. For instance, the normal malloc code internally allocates memory with `pvPortMallocCaps(size, MALLOC_CAP_8BIT)` in order to get data memory that is byte-addressable.

Because malloc uses this allocation system as well, memory allocated using pvPortMallocCaps can be freed by calling the standard `free()` function.

Internally, this allocator is split in two pieces. The allocator in the FreeRTOS directory can allocate memory from tagged regions: a tag is an integer value and every region of free memory has one of these tags. The esp32-specific code initializes these regions with specific tags, and contains the logic to select applicable tags from the capabilities given by the user. While shown in the public API, tags are used in the communication between the two parts and should not be used directly.

19.1.2 Special Uses

If a certain memory structure is only addressed in 32-bit units, for example an array of ints or pointers, it can be useful to allocate it with the MALLOC_CAP_32BIT flag. This also allows the allocator to give out IRAM memory; something which it can't do for a normal malloc() call. This can help to use all the available memory in the ESP32.

19.1.3 API Reference

Header Files

- esp32/include/esp_heap_alloc_caps.h
- freertos/include/freertos/heap_regions.h

Macros

Warning: doxygendefine: Cannot find define "MALLOC_CAP_EXEC" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_32BIT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_8BIT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_DMA" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID2" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID3" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID4" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID5" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID6" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_PID7" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_SPISRAM" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "MALLOC_CAP_INVALID" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "HeapRegionTagged_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “heap_alloc_caps_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pvPortMallocCaps” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “xPortGetFreeHeapSizeCaps” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “xPortGetMinimumEverFreeHeapSizeCaps” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “vPortDefineHeapRegionsTagged” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “pvPortMallocTagged” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “vPortFreeTagged” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “xPortGetMinimumEverFreeHeapSizeTagged” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “xPortGetFreeHeapSizeTagged” in doxygen xml output for project “esp32-idf” from directory: xml/

19.2 Interrupt allocation

19.2.1 Overview

The ESP32 has two cores, with 32 interrupts each. Each interrupt has a certain priority level, most (but not all) interrupts are connected to the interrupt mux. Because there are more interrupt sources than interrupts, sometimes it makes sense to share an interrupt in multiple drivers. The esp_intr_alloc abstraction exists to hide all these implementation details.

A driver can allocate an interrupt for a certain peripheral by calling esp_intr_alloc (or esp_intr_alloc_sintrstatus). It can use the flags passed to this function to set the type of interrupt allocated, specifying a specific level or trigger method. The interrupt allocation code will then find an applicable interrupt, use the interrupt mux to hook it up to the peripheral, and install the given interrupt handler and ISR to it.

This code has two different types of interrupts it handles differently: Shared interrupts and non-shared interrupts. The simplest of the two are non-shared interrupts: a separate interrupt is allocated per esp_intr_alloc call and this interrupt is solely used for the peripheral attached to it, with only one ISR that will get called. Non-shared interrupts can have multiple peripherals triggering it, with multiple ISRs being called when one of the peripherals attached signals an

interrupt. Thus, ISRs that are intended for shared interrupts should check the interrupt status of the peripheral they service in order to see if any action is required.

Non-shared interrupts can be either level- or edge-triggered. Shared interrupts can only be level interrupts (because of the chance of missed interrupts when edge interrupts are used.) (The logic behind this: DevA and DevB share an int. DevB signals an int. Int line goes high. ISR handler calls code for DevA -> does nothing. ISR handler calls code for DevB, but while doing that, DevA signals an int. ISR DevB is done, clears int for DevB, exits interrupt code. Now an interrupt for DevA is still pending, but because the int line never went low (DevA kept it high even when the int for DevB was cleared) the interrupt is never serviced.)

19.2.2 Multicore issues

Peripherals that can generate interrupts can be divided in two types: external peripherals, outside the Xtensa cores in the ESP32, and internal peripherals, inside the ESP32. Interrupt handling differs slightly between these two types of peripherals.

Each Xtensa core has its own set of internal peripherals: three timer comparators, a performance monitor and two software interrupts. These peripherals can only be configured from the core they are associated with. When generating an interrupt, the interrupt they generate is hard-wired to their associated core; it's not possible to have e.g. an internal timer comparator of one core generate an interrupt on another core. That is why these sources can only be managed using a task running on that specific core. Internal interrupt sources are still allocatable using esp_intr_alloc as normal, but they cannot be shared and will always have a fixed interrupt level (namely, the one associated in hardware with the peripheral). Internal interrupt sources are defined in esp_intr_alloc.h as ETS_INTERNAL_*_INTR_SOURCE.

The remaining interrupt slots in both cores are wired to an interrupt multiplexer, which can be used to route any external interrupt source to any of these interrupt slots. Allocating an external interrupt will always allocate it on the core that does the allocation, and freeing the interrupt should always happen on the same core. Disabling and enabling the interrupt from another core is allowed, however. External interrupts can share an interrupt slot bu passing ESP_INTR_FLAG_SHARED as a flag to esp_intr_alloc. External interrupt sources are defined in soc/soc.h as ETS_*_INTR_SOURCE.

Care should be taken when allocating an interrupt using a task not pinned to a certain core; while running code not in a critical secion, these tasks can migrate between cores at any moment, possibly making an interrupt operation fail because of the reasons mentioned above. It is advised to always use xTaskCreatePinnedToCore with a specific CoreID argument to create tasks that will handle interrupts.

19.2.3 Application Example

19.2.4 API Reference

Header Files

- esp32/include/esp_intr_alloc.h

Macros

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL1" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL2" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL3" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL4" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL5" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LEVEL6" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_NMI" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_LOWMED" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_HIGH" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_SHARED" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_EDGE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_IRAM" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_INTR_FLAG_INTRDISABLED" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Enumerations

Structures

Functions

Warning: doxygenfunction: Cannot find function "esp_intr_mark_shared" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_reserve” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_alloc” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_alloc_intrstatus” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_free” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_get_cpu” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_get_intno” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_noniram_disable” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_intr_noniram_enable” in doxygen xml output for project “esp32-idf” from directory: xml/

19.3 Watchdogs

19.3.1 Overview

Esp-idf has support for two types of watchdogs: a task watchdog as well as an interrupt watchdog. Both can be enabled using `make menuconfig` and selecting the appropriate options.

Interrupt watchdog

The interrupt watchdog makes sure the FreeRTOS task switching interrupt isn’t blocked for a long time. This is bad because no other tasks, including potentially important ones like the WiFi task and the idle task, can’t get any CPU runtime. A blocked task switching interrupt can happen because a program runs into an infinite loop with interrupts disabled or hangs in an interrupt.

The default action of the interrupt watchdog is to invoke the panic handler. causing a register dump and an opportunity for the programmer to find out, using either OpenOCD or gdbstub, what bit of code is stuck with interrupts disabled. Depending on the configuration of the panic handler, it can also blindly reset the CPU, which may be preferred in a production environment.

The interrupt watchdog is built around the hardware watchdog in timer group 1. If this watchdog for some reason cannot execute the NMI handler that invokes the panic handler (e.g. because IRAM is overwritten by garbage), it will hard-reset the SOC.

Task watchdog

Any tasks can elect to be watched by the task watchdog. If such a task does not feed the watchdog within the time specified by the task watchdog timeout (which is configurable using `make menuconfig`), the watchdog will print out a warning with information about which processes are running on the ESP32 CPUs and which processes failed to feed the watchdog.

By default, the task watchdog watches the idle tasks. The usual cause of idle tasks not feeding the watchdog is a higher-priority process looping without yielding to the lower-priority processes, and can be an indicator of badly-written code that spinloops on a peripheral or a task that is stuck in an infinite loop.

Other task can elect to be watched by the task watchdog by calling `esp_task_wdt_feed()`. Calling this routine for the first time will register the task to the task watchdog; calling it subsequent times will feed the watchdog. If a task does not want to be watched anymore (e.g. because it is finished and will call `vTaskDelete()` on itself), it needs to call `esp_task_wdt_delete()`.

The task watchdog is built around the hardware watchdog in timer group 0. If this watchdog for some reason cannot execute the interrupt handler that prints the task data (e.g. because IRAM is overwritten by garbage or interrupts are disabled entirely) it will hard-reset the SOC.

JTAG and watchdogs

While debugging using OpenOCD, if the CPUs are halted the watchdogs will keep running, eventually resetting the CPU. This makes it very hard to debug code; that is why the OpenOCD config will disable both watchdogs on startup. This does mean that you will not get any warnings or panics from either the task or interrupt watchdog when the ESP32 is connected to OpenOCD via JTAG.

19.3.2 API Reference

Header Files

- esp32/include/esp_int_wdt.h
- esp32/include/esp_task_wdt.h

19.3.3 Functions

Warning: doxygenfunction: Cannot find function “esp_int_wdt_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_task_wdt_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_task_wdt_feed" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_task_wdt_delete" in doxygen xml output for project "esp32-idf" from directory: xml/

19.4 OTA

19.4.1 Application Example

Demonstration of OTA (over the air) firmware update workflow: system/ota.

19.4.2 API Reference

Header Files

- app_update/include/esp_ota_ops.h

Macros

Warning: doxygendefine: Cannot find define "ESP_ERR_OTA_BASE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_OTA_PARTITION_CONFLICT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_OTA_SELECT_INFO_INVALID" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_OTA_VALIDATE_FAILED" in doxygen xml output for project "esp32-idf" from directory: xml/

Type Definitions

Warning: doxygentypedef: Cannot find typedef "esp_ota_handle_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function "esp_ota_begin" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_ota_write" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_ota_end" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_ota_set_boot_partition" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_ota_get_boot_partition" in doxygen xml output for project "esp32-idf" from directory: xml/

19.5 Deep Sleep

19.5.1 Overview

ESP32 is capable of deep sleep power saving mode. In this mode CPUs, most of the RAM, and all the digital peripherals which are clocked from APB_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals (including ULP coprocessor), and RTC memories (slow and fast).

Wakeup from deep sleep mode can be done using several sources. These sources can be combined, in this case the chip will wake up when any one of the sources is triggered. Wakeup sources can be enabled using `esp_deep_sleep_enable_X_wakeup` APIs. Next section describes these APIs in detail. Wakeup sources can be configured at any moment before entering deep sleep mode.

Additionally, the application can force specific powerdown modes for the RTC peripherals and RTC memories using `esp_deep_sleep_pd_config` API.

Once wakeup sources are configured, application can start deep sleep using `esp_deep_sleep_start` API. At this point the hardware will be configured according to the requested wakeup sources, and RTC controller will power down the CPUs and digital peripherals.

19.5.2 Wakeup sources

Timer

RTC controller has a built in timer which can be used to wake up the chip after a predefined amount of time. Time is specified at microsecond precision, but the actual resolution depends on the clock source selected for RTC SLOW_CLK. See chapter "Reset and Clock" of the ESP32 Technical Reference Manual for details about RTC clock options.

This wakeup mode doesn't require RTC peripherals or RTC memories to be powered on during deep sleep.

The following function can be used to enable deep sleep wakeup using a timer.

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_enable_timer_wakeup" in doxygen xml output for project "esp32-idf" from directory: xml/

External wakeup (ext0)

RTC IO module contains logic to trigger wakeup when one of RTC GPIOs is set to a predefined logic level. RTC IO is part of RTC peripherals power domain, so RTC peripherals will be kept powered on during deep sleep if this wakeup source is requested.

Because RTC IO module is enabled in this mode, internal pullup or pulldown resistors can also be used. They need to be configured by the application using `rtc_gpio_pullup_en` and `rtc_gpio_pulldown_en` functions, before calling `esp_deep_sleep_start`.

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_enable_ext0_wakeup" in doxygen xml output for project "esp32-idf" from directory: xml/

External wakeup (ext1)

RTC controller contains logic to trigger wakeup using multiple RTC GPIOs. One of the two logic functions can be used to trigger wakeup:

- wake up if any of the selected pins is low
- wake up if all the selected pins are high

This wakeup source is implemented by the RTC controller. As such, RTC peripherals and RTC memories can be powered off in this mode. However, if RTC peripherals are powered down, internal pullup and pulldown resistors will be disabled. To use internal pullup or pulldown resistors, request RTC peripherals power domain to be kept on during deep sleep, and configure pullup/pulldown resistors using `rtc_gpio_` functions, before entering deep sleep:

```
esp_deep_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_ON);
gpio_pullup_dis(gpio_num);
gpio_pulldown_en(gpio_num);
```

The following function can be used to enable this wakeup mode:

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_enable_ext1_wakeup" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "esp_ext1_wakeup_mode_t" in doxygen xml output for project "esp32-idf" from directory: xml/

ULP coprocessor wakeup

ULP coprocessor can run while the chip is in deep sleep, and may be used to poll sensors, monitor ADC or touch sensor values, and wake up the chip when a specific event is detected. ULP coprocessor is part of RTC peripherals power domain, and it runs the program stored in RTC slow memeory. Therefore RTC peripherals and RTC slow memory will be powered on during deep sleep if this wakeup mode is requested.

The following function can be used to enable this wakeup mode:

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_enable_ulp_wakeup" in doxygen xml output for project "esp32-idf" from directory: xml/

19.5.3 Power-down of RTC peripherals and memories

By default, `esp_deep_sleep_start` function will power down all RTC power domains which are not needed by the enabled wakeup sources. To override this behaviour, the following function is provided:

Note: on the first revision of the ESP32, RTC fast memory will always be kept enabled in deep sleep, so that the deep sleep stub can run after reset. This can be overriden, if the application doesn't need clean reset behaviour after deep sleep.

If some variables in the program are placed into RTC slow memory (for example, using `RTC_DATA_ATTR` attribute), RTC slow memory will be kept powered on by default. This can be overriden using `esp_deep_sleep_pd_config` function, if desired.

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_pd_config" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "esp_deep_sleep_pd_domain_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenenum: Cannot find enum "esp_deep_sleep_pd_option_t" in doxygen xml output for project "esp32-idf" from directory: xml/

19.5.4 Entering deep sleep

The following function can be used to enter deep sleep once wakeup sources are configured. It is also possible to go into deep sleep with no wakeup sources configured, in this case the chip will be in deep sleep mode indefinetly, until external reset is applied.

Warning: doxygenfunction: Cannot find function "esp_deep_sleep_start" in doxygen xml output for project "esp32-idf" from directory: xml/

19.5.5 Application Example

Implementation of basic functionality of deep sleep is shown in protocols/sntp example, where ESP module is periodically waken up to retrive time from NTP server.

19.6 Application Example

Log library is commonly used by most of esp-idf components and examples. For demonstration of log functionality check examples folder of espressif/esp-idf repository, that among others, contains the following examples:

- system/ota
- storage/sd_card
- protocols/https_request

19.7 API Reference

19.7.1 Header Files

- log/include/esp_log.h

19.7.2 Macros

Warning: doxygendefine: Cannot find define "LOG_COLOR_E" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_COLOR_W" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_COLOR_I" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_COLOR_D" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_COLOR_V" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_RESET_COLOR" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_FORMAT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "LOG_LOCAL_LEVEL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_EARLY_LOGE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_EARLY_LOGW" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_EARLY_LOGI" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_EARLY_LOGD" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_EARLY_LOGV” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_LOGE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_LOGW” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_LOGI” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_LOGD” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_LOGV” in doxygen xml output for project “esp32-idf” from directory: xml/

19.7.3 Type Definitions

Warning: doxygentypedef: Cannot find typedef “vprintf_like_t” in doxygen xml output for project “esp32-idf” from directory: xml/

19.7.4 Enumerations

Warning: doxygenenum: Cannot find enum “esp_log_level_t” in doxygen xml output for project “esp32-idf” from directory: xml/

19.7.5 Functions

Warning: doxygenfunction: Cannot find function “esp_log_level_set” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_log_set_vprintf” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_log_timestamp” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_log_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Example code for this API section is provided in system directory of ESP-IDF examples.

CHAPTER 20

Storage API

20.1 API Reference

20.1.1 Header Files

- spi_flash/include/esp_spi_flash.h
- spi_flash/include/esp_partition.h
- esp32/include/esp_flash_encrypt.h

20.1.2 Macros

Warning: doxygendefine: Cannot find define “ESP_ERR_FLASH_BASE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_ERR_FLASH_OP_FAIL” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_ERR_FLASH_OP_TIMEOUT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SPI_FLASH_SEC_SIZE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SPI_FLASH_MMU_PAGE_SIZE” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “ESP_PARTITION_SUBTYPE_OTA” in doxygen xml output for project “esp32-idf” from directory: xml/

20.1.3 Type Definitions

Warning: doxygentypedef: Cannot find typedef “spi_flash_mmap_handle_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “esp_partition_iterator_t” in doxygen xml output for project “esp32-idf” from directory: xml/

20.1.4 Enumerations

Warning: doxygenenum: Cannot find enum “spi_flash_mmap_memory_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_partition_type_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenenum: Cannot find enum “esp_partition_subtype_t” in doxygen xml output for project “esp32-idf” from directory: xml/

20.1.5 Structures

Warning: doxygenstruct: Cannot find class “esp_partition_t” in doxygen xml output for project “esp32-idf” from directory: xml/

20.1.6 Functions

Warning: doxygenfunction: Cannot find function “spi_flash_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_get_chip_size” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_erase_sector” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_erase_range” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_write_encrypted” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_read” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_read_encrypted” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_mmap” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_munmap” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “spi_flash_mmap_dump” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_find” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_find_first” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_get” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_next” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_iterator_release” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_read” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_write” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_erase_range” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_partition_mmap” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_flash_encryption_enabled" in doxygen xml output for project "esp32-idf" from directory: xml/

20.2 SDMMC Host Peripheral

20.2.1 Overview

SDMMC peripheral supports SD and MMC memory cards and SDIO cards. SDMMC software builds on top of SDMMC driver and consists of the following parts:

1. SDMMC host driver (`driver/sdmmc_host.h`) — this driver provides APIs to send commands to the slave device(s), send and receive data, and handling error conditions on the bus.
2. SDMMC protocol layer (`sdmmc_cmd.h`) — this component handles specifics of SD protocol such as card initialization and data transfer commands. Despite the name, only SD (SDSC/SDHC/SDXC) cards are supported at the moment. Support for MCC/eMMC cards can be added in the future.

Protocol layer works with the host via `sdmmc_host_t` structure. This structure contains pointers to various functions of the host. This design makes it possible to implement an SD host using SPI interface later.

20.2.2 Application Example

An example which combines SDMMC driver with FATFS library is provided in `examples/storage/sd_card` directory. This example initializes the card, writes and reads data from it using POSIX and C library APIs. See README.md file in the example directory for more information.

20.2.3 Protocol layer APIs

Protocol layer is given `sdmmc_host_t` structure which describes the SD/MMC host driver, lists its capabilites, and provides pointers to functions of the driver. Protocol layer stores card-specific information in `sdmmc_card_t` structure. When sending commands to the SD/MMC host driver, protocol layer uses `sdmmc_command_t` structure to describe the command, argument, expected return value, and data to transfer, if any.

Normal usage of the protocol layer is as follows:

1. Call the host driver functions to initialize the host (e.g. `sdmmc_host_init`, `sdmmc_host_init_slot`).
2. Call `sdmmc_card_init` to initialize the card, passing it host driver information (`host`) and a pointer to `sdmmc_card_t` structure which will be filled in (`card`).
3. To read and write sectors of the card, use `sdmmc_read_sectors` and `sdmmc_write_sectors`, passing the pointer to card information structure (`card`).
4. When card is not used anymore, call the host driver function to disable SDMMC host peripheral and free resources allocated by the driver (e.g. `sdmmc_host_deinit`).

Most applications need to use the protocol layer only in one task; therefore the protocol layer doesn't implement any kind of locking on the `sdmmc_card_t` structure, or when accessing SDMMC host driver. Such locking has to be implemented in the higher layer, if necessary (e.g. in the filesystem driver).

Warning: doxygenstruct: Cannot find class "sdmmc_host_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_HOST_FLAG_1BIT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_HOST_FLAG_4BIT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_HOST_FLAG_8BIT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_HOST_FLAG_SPI" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_FREQ_DEFAULT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_FREQ_HIGHSPEED" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "SDMMC_FREQ_PROBING" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "sdmmc_command_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "sdmmc_card_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "sdmmc_csd_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "sdmmc_cid_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "sdmmc_scr_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sdmmc_card_init" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sdmmc_write_sectors" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sdmmc_read_sectors" in doxygen xml output for project "esp32-idf" from directory: xml/

20.2.4 SDMMC host driver APIs

On the ESP32, SDMMC host peripheral has two slots:

- Slot 0 (`SDMMC_HOST_SLOT_0`) is an 8-bit slot. It uses `HS1_*` signals in the PIN MUX.
- Slot 1 (`SDMMC_HOST_SLOT_1`) is a 4-bit slot. It uses `HS2_*` signals in the PIN MUX.

Card Detect and Write Protect signals can be routed to arbitrary pins using GPIO matrix. To use these pins, set `gpio_cd` and `gpio_wp` members of `sdmmc_slot_config_t` structure when calling `sdmmc_host_init_slot`.

Of all the funtions listed below, only `sdmmc_host_init`, `sdmmc_host_init_slot`, and `sdmmc_host_deinit` will be used directly by most applications. Other functions, such as `sdmmc_host_set_bus_width`, `sdmmc_host_set_card_clk`, and `sdmmc_host_do_transaction` will be called by the SD/MMC protocol layer via function pointers in `sdmmc_host_t` structure.

Warning: doxygenfunction: Cannot find function “sdmmc_host_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_HOST_SLOT_0” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_HOST_SLOT_1” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_HOST_DEFAULT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “sdmmc_host_init_slot” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenstruct: Cannot find class “sdmmc_slot_config_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_SLOT_NO_CD” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_SLOT_NO_WP” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygendefine: Cannot find define “SDMMC_SLOT_CONFIG_DEFAULT” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “sdmmc_host_set_bus_width” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “sdmmc_host_set_card_clk” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "sdmmc_host_do_transaction" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "sdmmc_host_deinit" in doxygen xml output for project "esp32-idf" from directory: xml/

20.3 Application Example

Two examples are provided in storage directory of ESP-IDF examples:

storage/nvs_rw_value

Demonstrates how to read and write a single integer value using NVS.

The value holds the number of ESP32 module restarts. Since it is written to NVS, the value is preserved between restarts.

Example also shows how to check if read / write operation was successful, or certain value is not initialized in NVS. Diagnostic is provided in plain text to help track program flow and capture any issues on the way.

storage/nvs_rw_blob

Demonstrates how to read and write a single integer value and a blob (binary large object) using NVS to preserve them between ESP32 module restarts.

- value - tracks number of ESP32 module soft and hard restarts.
- blob - contains a table with module run times. The table is read from NVS to dynamically allocated RAM. New run time is added to the table on each manually triggered soft restart and written back to NVS. Triggering is done by pulling down GPIO0.

Example also shows how to implement diagnostics if read / write operation was successful.

20.4 API Reference

20.4.1 Header Files

- nvs_flash/include/nvs_flash.h
- nvs_flash/include/nvs.h

20.4.2 Macros

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_BASE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_NOT_INITIALIZED" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_NOT_FOUND" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_TYPE_MISMATCH" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_READ_ONLY" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_NOT_ENOUGH_SPACE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_INVALID_NAME" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_INVALID_HANDLE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_REMOVE_FAILED" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_KEY_TOO_LONG" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_PAGE_FULL" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_INVALID_STATE" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_ERR_NVS_INVALID_LENGTH" in doxygen xml output for project "esp32-idf" from directory: xml/

20.4.3 Type Definitions

Warning: doxygentypedef: Cannot find typedef "nvs_handle" in doxygen xml output for project "esp32-idf" from directory: xml/

20.4.4 Enumerations

Warning: doxygenenum: Cannot find enum "nvs_open_mode" in doxygen xml output for project "esp32-idf" from directory: xml/

20.4.5 Functions

Warning: doxygenfunction: Cannot find function “nvs_open” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_i8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_u8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_i16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_u16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_i32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_u32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_i64” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_u64” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_str” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_set_blob” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_i8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_u8” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_i16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_u16” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_i32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_u32” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_i64” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_u64” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_str” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_get_blob” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_erase_key” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_erase_all” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_commit” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_close” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “nvs_flash_init” in doxygen xml output for project “esp32-idf” from directory: xml/

20.5 Application Example

Instructions

20.6 API Reference

20.6.1 Header Files

- vfs/include/esp_vfs.h
- vfs/include/esp_vfs_dev.h

20.6.2 Macros

Warning: doxygendefine: Cannot find define "ESP_VFS_PATH_MAX" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_VFS_FLAG_DEFAULT" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygendefine: Cannot find define "ESP_VFS_FLAG_CONTEXT_PTR" in doxygen xml output for project "esp32-idf" from directory: xml/

20.6.3 Structures

Warning: doxygenstruct: Cannot find class "esp_vfs_t" in doxygen xml output for project "esp32-idf" from directory: xml/

20.6.4 Functions

Warning: doxygenfunction: Cannot find function "esp_vfs_register" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_unregister" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_write" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_lseek" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_read" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_open" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_close” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_fstat” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_stat” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_link” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_unlink” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_rename” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “esp_vfs_dev_uart_register” in doxygen xml output for project “esp32-idf” from directory: xml/

20.7 FAT Filesystem Support

ESP-IDF uses FatFs library to work with FAT filesystems. FatFs library resides in `fatfs` component. Although it can be used directly, many of its features can be accessed via VFS using C standard library and POSIX APIs.

Additionally, FatFs has been modified to support run-time pluggable disk IO layer. This allows mapping of FatFs drives to physical disks at run-time.

20.7.1 Using FatFs with VFS

`esp_vfs_fat.h` header file defines functions to connect FatFs with VFS. `esp_vfs_fat_register` function allocates a `FATFS` structure, and registers a given path prefix in VFS. Subsequent operations on files starting with this prefix are forwarded to FatFs APIs. `esp_vfs_fat_unregister` function deletes the registration with VFS, and frees the `FATFS` structure.

Most applications will use the following flow when working with `esp_vfs_fat_` functions:

1. Call `esp_vfs_fat_register`, specifying path prefix where the filesystem has to be mounted (e.g. `"/sdcard"`), FatFs drive number, and a variable which will receive a pointer to `FATFS` structure.
2. Call `ff_diskio_register` function to register disk IO driver for the drive number used in step 1.
3. Call `f_mount` function (and optionally `f_fdisk`, `f_mkfs`) to mount the filesystem using the same drive number which was passed to `esp_vfs_fat_register`. See FatFs documentation for more details.
4. Call POSIX and C standard library functions to open, read, write, erase, copy files, etc. Use paths starting with the prefix passed to `esp_vfs_register` (such as `"/sdcard/hello.txt"`).
5. Optionally, call FatFs library functions directly. Use paths without a VFS prefix in this case (`"/hello.txt"`).

6. Close all open files.
7. Call `f_mount` function for the same drive number, with NULL `FATFS*` argument, to unmount the filesystem.
8. Call `ff_diskio_register` with NULL `ff_diskio_impl_t*` argument and the same drive number.
9. Call `esp_vfs_fat_unregister` to remove FatFs from VFS, and free the `FATFS` structure allocated on step 1.

Convenience functions, `esp_vfs_fat_sdmmc_mount` and `esp_vfs_fat_sdmmc_unmount`, which wrap these steps and also handle SD card initialization, are described in the next section.

Warning: doxygenfunction: Cannot find function "esp_vfs_fat_register" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_fat_unregister" in doxygen xml output for project "esp32-idf" from directory: xml/

20.7.2 Using FatFs with VFS and SD cards

`esp_vfs_fat.h` header file also provides a convenience function to perform steps 1–3 and 7–9, and also handle SD card initialization: `esp_vfs_fat_sdmmc_mount`. This function does only limited error handling. Developers are encouraged to look at its source code and incorporate more advanced versions into production applications. `esp_vfs_fat_sdmmc_unmount` function unmounts the filesystem and releases resources acquired by `esp_vfs_fat_sdmmc_mount`.

Warning: doxygenfunction: Cannot find function "esp_vfs_fat_sdmmc_mount" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "esp_vfs_fat_sdmmc_mount_config_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "esp_vfs_fat_sdmmc_unmount" in doxygen xml output for project "esp32-idf" from directory: xml/

20.7.3 FatFS disk IO layer

FatFs has been extended with an API to register disk IO driver at runtime.

Implementation of disk IO functions for SD/MMC cards is provided. It can be registered for the given FatFs drive number using `ff_diskio_register_sdmmc` function.

Warning: doxygenfunction: Cannot find function "ff_diskio_register" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenstruct: Cannot find class "ff_diskio_impl_t" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "ff_diskio_register_sdmmc" in doxygen xml output for project "esp32-idf" from directory: xml/

Example code for this API section is provided in storage directory of ESP-IDF examples.

CHAPTER 21

Protocols API

21.1 mDNS Service

21.1.1 Overview

mDNS is a multicast UDP service that is used to provide local network service and host discovery.

mDNS is installed by default on most operating systems or is available as separate package. On `Mac OS` it is installed by default and is called `Bonjour`. Apple releases an installer for `Windows` that can be found on Apple's support page. On `Linux`, mDNS is provided by avahi and is usually installed by default.

mDNS Properties

- `hostname`: the hostname that the device will respond to. If not set, the `hostname` will be read from the interface. Example: `my-esp32` will resolve to `my-esp32.local`
- `default_instance`: friendly name for your device, like `Jhon's ESP32 Thing`. If not set, `hostname` will be used.

Example method to start mDNS for the STA interface and set `hostname` and `default_instance`:

```
mdns_server_t * mdns = NULL;

void start_mdns_service()
{
    //initialize mDNS service on STA interface
    esp_err_t err = mdns_init(TCPIP_ADAPTER_IF_STA, &mdns);
    if (err) {
        printf("MDNS Init failed: %d\n", err);
        return;
    }

    //set hostname
    mdns_set_hostname(mdns, "my-esp32");
    //set default instance
    mdns_set_instance(mdns, "Jhon's ESP32 Thing");
}
```

mDNS Services

mDNS can advertise information about network services that your device offers. Each service is defined by a few properties.

- `service`: (required) service type, prepended with underscore. Some common types can be found here.
- `proto`: (required) protocol that the service runs on, prepended with underscore. Example: `_tcp` or `_udp`
- `port`: (required) network port that the service runs on
- `instance`: friendly name for your service, like `Jhon's ESP32 Web Server`. If not defined, `default_instance` will be used.
- `txt`: `var=val` array of strings, used to define properties for your service

Example method to add a few services and different properties:

```
void add_mdns_services()
{
    //add our services
    mdns_service_add(mdns, "_http", "_tcp", 80);
    mdns_service_add(mdns, "_arduino", "_tcp", 3232);
    mdns_service_add(mdns, "_myservice", "_udp", 1234);

    //NOTE: services must be added before their properties can be set
    //use custom instance for the web server
    mdns_service_instance_set(mdns, "_http", "_tcp", "Jhon's ESP32 Web Server");

    const char * arduTxtData[4] = {
            "board=esp32",
            "tcp_check=no",
            "ssh_upload=no",
            "auth_upload=no"
    };
    //set txt data for service (will free and replace current data)
    mdns_service_txt_set(mdns, "_arduino", "_tcp", 4, arduTxtData);

    //change service port
    mdns_service_port_set(mdns, "_myservice", "_udp", 4321);
}
```

mDNS Query

mDNS provides methods for browsing for services and resolving host's IP/IPv6 addresses. Results are returned as a linked list of `mdns_result_t` objects. If the result is from host query, it will contain only `addr` and `addrv6` if found. Service queries will populate all fields in a result that were found.

Example method to resolve host IPs:

```
void resolve_mdns_host(const char * hostname)
{
    printf("mDNS Host Lookup: %s.local\n", hostname);
    //run search for 1000 ms
    if (mdns_query(mdns, hostname, NULL, 1000)) {
        //results were found
        const mdns_result_t * results = mdns_result_get(mdns, 0);
        //itterate through all results
        size_t i = 1;
        while(results) {
```

```
                //print result information
                printf("  %u: IP:" IPSTR ", IPv6:" IPV6STR "\n", i++
                    IP2STR(&results->addr), IPV62STR(results->addrv6));
                //load next result. Will be NULL if this was the last one
                results = results->next;
            }
            //free the results from memory
            mdns_result_free(mdns);
        } else {
            //host was not found
            printf("  Host Not Found\n");
        }
    }
```

Example method to resolve local services:

```
    void find_mdns_service(const char * service, const char * proto)
    {
        printf("mDNS Service Lookup: %s.%s\n", service, proto);
        //run search for 1000 ms
        if (mdns_query(mdns, service, proto, 1000)) {
            //results were found
            const mdns_result_t * results = mdns_result_get(mdns, 0);
            //itterate through all results
            size_t i = 1;
            while(results) {
                //print result information
                printf("  %u: hostname:%s, instance:\"%s\", IP:" IPSTR ", IPv6:" IPV6STR ", port:%u,
                    (results->host)?results->host:"NULL", (results->instance)?results->instance:"NUL
                    IP2STR(&results->addr), IPV62STR(results->addrv6),
                    results->port, (results->txt)?results->txt:"\r");
                //load next result. Will be NULL if this was the last one
                results = results->next;
            }
            //free the results from memory
            mdns_result_free(mdns);
        } else {
            //service was not found
            printf("  Service Not Found\n");
        }
    }
```

Example of using the methods above:

```
    void my_app_some_method(){
        //search for esp32-mdns.local
        resolve_mdns_host("esp32-mdns");

        //search for HTTP servers
        find_mdns_service("_http", "_tcp");
        //or file servers
        find_mdns_service("_smb", "_tcp"); //windows sharing
        find_mdns_service("_afpovertcp", "_tcp"); //apple sharing
        find_mdns_service("_nfs", "_tcp"); //NFS server
        find_mdns_service("_ftp", "_tcp"); //FTP server
        //or networked printer
        find_mdns_service("_printer", "_tcp");
        find_mdns_service("_ipp", "_tcp");
    }
```

21.1.2 Application Example

mDNS server/scanner example: protocols/mdns.

21.1.3 API Reference

Header Files

- mdns/include/mdns.h

Macros

Type Definitions

Warning: doxygentypedef: Cannot find typedef “mdns_server_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygentypedef: Cannot find typedef “mdns_result_t” in doxygen xml output for project “esp32-idf” from directory: xml/

Enumerations

Structures

Warning: doxygenstruct: Cannot find class “mdns_result_s” in doxygen xml output for project “esp32-idf” from directory: xml/

Functions

Warning: doxygenfunction: Cannot find function “mdns_init” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “mdns_free” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “mdns_set_hostname” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “mdns_set_instance” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function “mdns_service_add” in doxygen xml output for project “esp32-idf” from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_service_remove" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_service_instance_set" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_service_txt_set" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_service_port_set" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_service_remove_all" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_query" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_query_end" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_result_get_count" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_result_get" in doxygen xml output for project "esp32-idf" from directory: xml/

Warning: doxygenfunction: Cannot find function "mdns_result_free" in doxygen xml output for project "esp32-idf" from directory: xml/

Example code for this API section is provided in protocols directory of ESP-IDF examples.

CHAPTER 22

Espressif IoT Development Framework Style Guide

22.1 About this guide

Purpose of this style guide is to encourage use of common coding practices within the ESP-IDF.

Style guide is a set of rules which are aimed to help create readable, maintainable, and robust code. By writing code which looks the same way across the code base we help others read and comprehend the code. By using same conventions for spaces and newlines we reduce chances that future changes will produce huge unreadable diffs. By following common patterns for module structure and by using language features consistently we help others understand code behavior.

We try to keep rules simple enough, which means that they can not cover all potential cases. In some cases one has to bend these simple rules to achieve readability, maintainability, or robustness.

When doing modifications to third-party code used in ESP-IDF, follow the way that particular project is written. That will help propose useful changes for merging into upstream project.

22.2 C code formatting

22.2.1 Indentation

Use 4 spaces for each indentation level. Don't use tabs for indentation. Configure the editor to emit 4 spaces each time you press tab key.

22.2.2 Vertical space

Place one empty line between functions. Don't begin or end a function with an empty line.

```
void function1()
{
    do_one_thing();
    do_another_thing();
                                // INCORRECT, don't place empty line here
}
                                // place empty line here
void function2()
{
                                // INCORRECT, don't use an empty line here
    int var = 0;
```

```
    while (var < SOME_CONSTANT) {
        do_stuff(&var);
    }
}
```

22.2.3 Horizontal space

Always add single space after conditional and loop keywords:

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

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

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

Add single space around binary operators. No space is necessary for unary operators. It is okay to drop space around multiply and divide operators:

```
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;                                          // correct
++y_cur;

const int y = y0+(x-x0)*(y1-y0)/(x1-x0);                 // INCORRECT
```

No space is necessary around `.` and `->` operators.

Sometimes adding horizontal space within a line can help make code more readable. For example, you can add space to align function arguments:

```
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);
```

Note however that if someone goes to add new line with a longer identifier as first argument (e.g. `PIN_CAM_VSYNC`), it will not fit. So other lines would have to be realigned, adding meaningless changes to the commit.

Therefore, use horizontal alignment sparingly, especially if you expect new lines to be added to the list later.

Never use TAB characters for horizontal alignment.

Never add trailing whitespace at the end of the line.

22.2.4 Braces

- Function definition should have a brace on a separate line:

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

}

// NOT like this:
void function(int arg) {

}
```

- Within a function, place opening brace on the same line with conditional and loop statements:

```
if (condition) {
    do_one();
} else if (other_condition) {
    do_two();
}
```

22.2.5 Comments

Use `//` for single line comments. For multi-line comments it is okay to use either `//` on each line or a `/* */` block.

Although not directly related to formatting, here are a few notes about using comments effectively.

- Don't use single comments to disable some functionality:

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

- If some code is no longer required, remove it completely. If you need it you can always look it up in git history of this file. If you disable some call because of temporary reasons, with an intention to restore it in the future, add explanation on the adjacent line:

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

- Same goes for `#if 0 ... #endif` blocks. Remove code block completely if it is not used. Otherwise, add comment explaining why the block is disabled. Don't use `#if 0 ... #endif` or comments to store code snippets which you may need in the future.
- Don't add trivial comments about authorship and change date. You can always look up who modified any given line using git. E.g. this comment adds clutter to the code without adding any useful information:

```
void init_something()
{
    setup_dma();
    // XXX add 2016-09-01
    init_dma_list();
```

```
        fill_dma_item(0);
        // end XXX add
        start_timer();
    }
```

22.2.6 Formatting your code

You can use `astyle` program to format your code according to the above recommendations.

If you are writing a file from scratch, or doing a complete rewrite, feel free to re-format the entire file. If you are changing a small portion of file, don't re-format the code you didn't change. This will help others when they review your changes.

To re-format a file, run:

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

22.3 Documenting code

Please see the guide here: Documenting Code.

22.4 Structure and naming

22.5 Language features

To be written.

CHAPTER 23

Documenting Code

The purpose of this description is to provide quick summary on documentation style used in espressif/esp-idf repository and how to add new documentation.

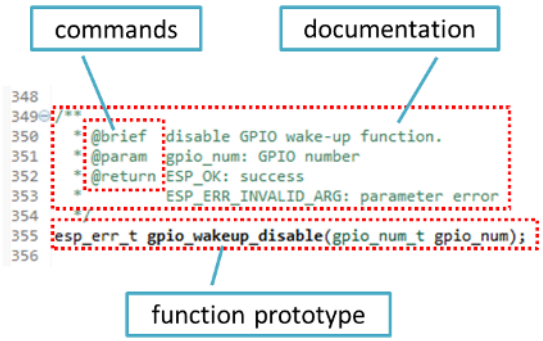
23.1 Introduction

When documenting code for this repository, please follow Doxygen style. You are doing it by inserting special commands, for instance `@param`, into standard comments blocks, for example:

```
/**
 * @param ratio this is oxygen to air ratio
 */
```

Doxygen is phrasing the code, extracting the commands together with subsequent text, and building documentation out of it.

Typical comment block, that contains documentation of a function, looks like below.

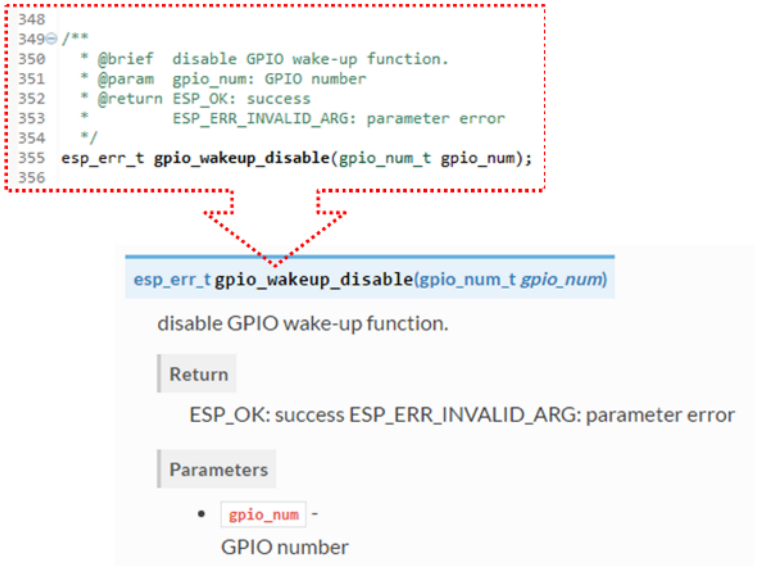


Doxygen supports couple of formatting styles. It also gives you great flexibility on level of details to include in documentation. To get familiar with available features, please check data reach and very well organized Doxygen Manual.

23.2 Why we need it?

The ultimate goal is to ensure that all the code is consistently documented, so we can use tools like Sphinx and Breathe to aid preparation and automatic updates of API documentation when the code changes.

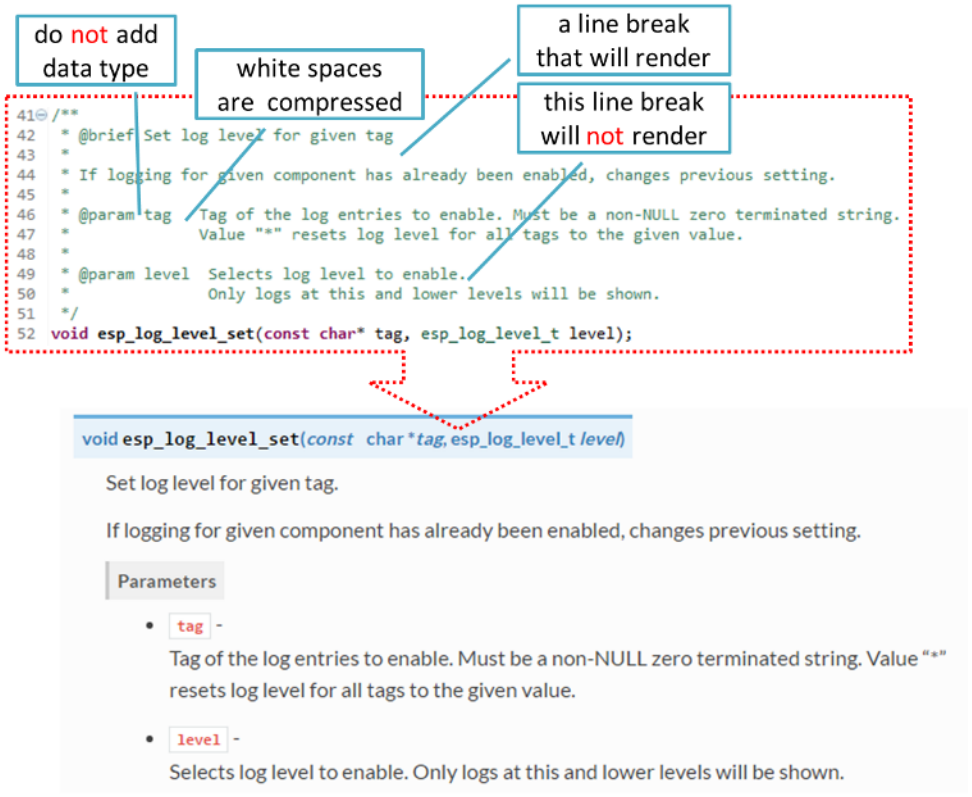
With these tools the above piece of code renders like below:



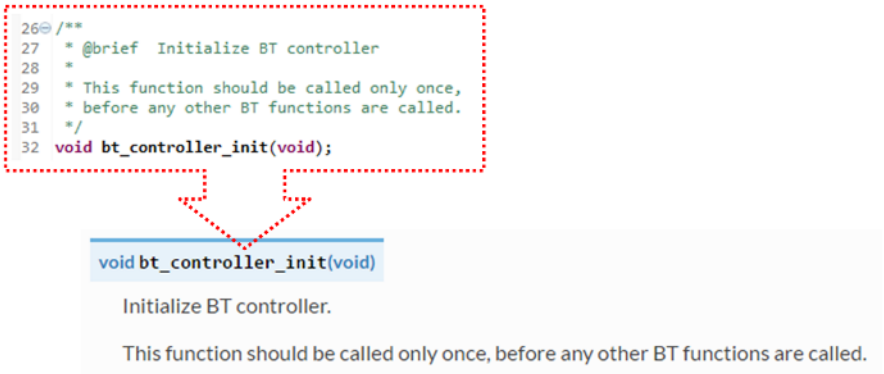
23.3 Go for it!

When writing code for this repository, please follow guidelines below.

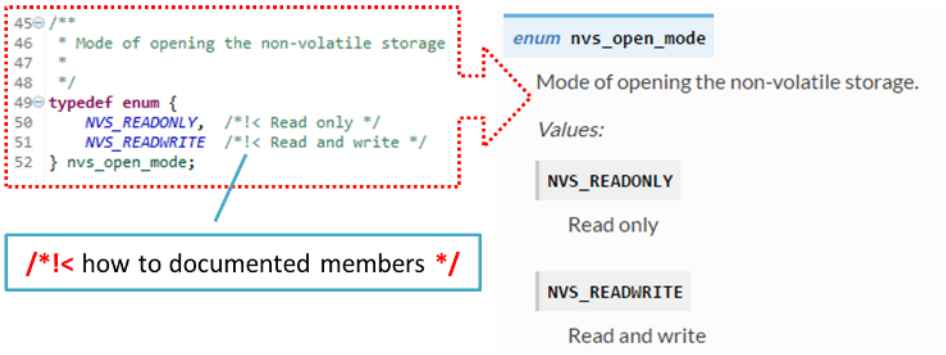
1. Document all building blocks of code: functions, structs, typedefs, enums, macros, etc. Provide enough information on purpose, functionality and limitations of documented items, as you would like to see them documented when reading the code by others.
2. Documentation of function should describe what this function does. If it accepts input parameters and returns some value, all of them should be explained.
3. Do not add a data type before parameter or any other characters besides spaces. All spaces and line breaks are compressed into a single space. If you like to break a line, then break it twice.



4. If function has void input or does not return any value, then skip `@param` or `@return`



5. When documenting a `define` as well as members of a `struct` or `enum`, place specific comment like below after each member.



6. To provide well formatted lists, break the line after command (like `@return` in example below).

```
*
* @return
*     - ESP_OK if erase operation was successful
*     - ESP_ERR_NVS_INVALID_HANDLE if handle has been closed or is NULL
*     - ESP_ERR_NVS_READ_ONLY if handle was opened as read only
*     - ESP_ERR_NVS_NOT_FOUND if the requested key doesn't exist
*     - other error codes from the underlying storage driver
*
```

7. Overview of functionality of documented header file, or group of files that make a library, should be placed in the same directory in a separate `README.rst` file. If directory contains header files for different APIs, then the file name should be `apiname-readme.rst`.

23.4 Go one extra mile

There is couple of tips, how you can make your documentation even better and more useful to the reader.

1. Add code snippets to illustrate implementation. To do so, enclose snippet using `@code{c}` and `@endcode` commands.

```
*
* @code{c}
* // Example of using nvs_get_i32:
* int32_t max_buffer_size = 4096; // default value
* esp_err_t err = nvs_get_i32(my_handle, "max_buffer_size", &max_buffer_size);
* assert(err == ESP_OK || err == ESP_ERR_NVS_NOT_FOUND);
* // if ESP_ERR_NVS_NOT_FOUND was returned, max_buffer_size will still
* // have its default value.
* @endcode
*
```

   The code snippet should be enclosed in a comment block of the function that it illustrates.

2. To highlight some important information use command `@attention` or `@note`.

```
*
* @attention
*     1. This API only impact WIFI_MODE_STA or WIFI_MODE_APSTA mode
*     2. If the ESP32 is connected to an AP, call esp_wifi_disconnect to disconnect.
*
```

   Above example also shows how to use a numbered list.

3. Use markdown to make your documentation even more readable. You will add headers, links, tables and more.

```
    *
    * [ESP32 Technical Reference](http://espressif.com/sites/default/files/documentation/esp32_techn
    *
```

Note: Code snippets, notes, links, etc. will not make it to the documentation, if not enclosed in a comment block associated with one of documented objects.

5. Prepare one or more complete code examples together with description. Place them in a separate file `example.rst` in the same directory as the API header files. If directory contains header files for different APIs, then the file name should be `apiname-example.rst`.

23.5 Put it all together

Once all the above steps are complete, follow instruction in Template and create a single file, that will merge all individual pieces of prepared documentation. Finally add a link to this file to respective `.. toctree::` in `index.rst` file located in `/docs` folder.

23.6 OK, but I am new to Sphinx!

1. No worries. All the software you need is well documented. It is also open source and free. Start by checking Sphinx documentation. If you are not clear how to write using rst markup language, see reStructuredText Primer.
2. Check the source files of this documentation to understand what is behind of what you see now on the screen. Sources are maintained on GitHub in espressif/esp-idf repository in docs folder. You can go directly to the source file of this page by scrolling up and clicking the link in the top right corner. When on GitHub, see what's really inside, open source files by clicking `Raw` button.
3. You will likely want to see how documentation builds and looks like before posting it on the GitHub. There are two options to do so:

- Install Sphinx, Breathe and Doxygen to build it locally. You would need a Linux machine for that.
- Set up an account on Read the Docs and build documentation in the cloud. Read the Docs provides document building and hosting for free and their service works really quick and great.

23.7 Wrap up

We love good code that is doing cool things. We love it even better, if it is well documented, so we can quickly make it run and also do the cool things.

Go ahead, contribute your code and documentation!

CHAPTER 24

Template

Note: *INSTRUCTIONS*

1. Use this file as a template to document API.
2. Change the file name to the name of the header file that represents documented API.
3. Include respective files with descriptions from the API folder using `..include::`
   - README.rst
   - example.rst
4. Optionally provide description right in this file.
5. Once done, remove all instructions like this one and any superfluous headers.

24.1 Overview

Note: *INSTRUCTIONS*

1. Provide overview where and how this API may be used.
2. Where applicable include code snippets to illustrate functionality of particular functions.
3. To distinguish between sections, use the following heading levels:
   - # with overline, for parts
   - * with overline, for chapters
   - =, for sections
   - -, for subsections
   - ^, for subsubsections
   - ", for paragraphs

24.2 Application Example

Note: *INSTRUCTIONS*

1. Prepare one or more practical examples to demonstrate functionality of this API.
2. Each example should follow pattern of projects located in `esp-idf/examples/` folder.
3. Place example in this folder complete with `README.md` file.
4. Provide overview of demonstrated functionality in `README.md`.
5. With good overview reader should be able to understand what example does without opening the source code.
6. Depending on complexity of example, break down description of code into parts and provide overview of functionality of each part.
7. Include flow diagram and screenshots of application output if applicable.
8. Finally add in this section synopsis of each example together with link to respective folder in `esp-idf/examples/`.

24.3 API Reference

Note: *INSTRUCTIONS*

1. Specify the names of header files used to generate this reference. Each name should be linked to the source on espressif/esp-idf repository.
2. Provide list of API members divided into sections.
3. Use corresponding `.. doxygen::` directives, so member documentation is auto updated.
   - Data Structures - `.. doxygenstruct::` together with `:members:`
   - Macros - `.. doxygendefine::`
   - Type Definitions - `.. doxygentypedef::`
   - Enumerations - `.. doxygenenum::`
   - Functions - `.. doxygenfunction::`

   See Breathe documentation for additional information.
4. Once done remove superfluous headers.
5. When changes are committed and documentation is build, check how this section rendered. Correct annotations in respective header files, if required.

24.3.1 Header Files

- *path/header-file.h*

24.3.2 Data Structures

```
.. doxygenstruct:: name_of_structure
   :members:
```

24.3.3 Macros

```
.. doxygendefine:: name_of_macro
```

24.3.4 Type Definitions

```
.. doxygentypedef:: name_of_type
```

24.3.5 Enumerations

```
.. doxygenenum:: name_of_enumeration
```

24.3.6 Functions

```
.. doxygenfunction:: name_of_function
```

CHAPTER 25

Contributor Agreement

25.1 Individual Contributor Non-Exclusive License Agreement

25.2 including the Traditional Patent License OPTION

Thank you for your interest in contributing to Espressif IoT Development Framework (esp-idf) ("We" or "Us").

The purpose of this contributor agreement ("Agreement") is to clarify and document the rights granted by contributors to Us. To make this document effective, please follow the instructions at CONTRIBUTING.rst

25.2.1 1. DEFINITIONS

"You" means the Individual Copyright owner who submits a Contribution to Us. If You are an employee and submit the Contribution as part of your employment, You have had Your employer approve this Agreement or sign the Entity version of this document.

"Contribution" means any original work of authorship (software and/or documentation) including any modifications or additions to an existing work, Submitted by You to Us, in which You own the Copyright. If You do not own the Copyright in the entire work of authorship, please contact Us at angus@espressif.com.

"Copyright" means all rights protecting works of authorship owned or controlled by You, including copyright, moral and neighboring rights, as appropriate, for the full term of their existence including any extensions by You.

"Material" means the software or documentation made available by Us to third parties. When this Agreement covers more than one software project, the Material means the software or documentation to which the Contribution was Submitted. After You Submit the Contribution, it may be included in the Material.

"Submit" means any form of physical, electronic, or written communication sent to Us, including but not limited to electronic mailing lists, source code control systems, and issue tracking systems that are managed by, or on behalf of, Us, but excluding communication that is conspicuously marked or otherwise designated in writing by You as "Not a Contribution."

"Submission Date" means the date You Submit a Contribution to Us.

"Documentation" means any non-software portion of a Contribution.

25.2.2 2. LICENSE GRANT

2.1 Copyright License to Us

Subject to the terms and conditions of this Agreement, You hereby grant to Us a worldwide, royalty-free, NON-exclusive, perpetual and irrevocable license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, under the Copyright covering the Contribution to use the Contribution by all means, including, but not limited to:

- to publish the Contribution,
- to modify the Contribution, to prepare derivative works based upon or containing the Contribution and to combine the Contribution with other software code,
- to reproduce the Contribution in original or modified form,
- to distribute, to make the Contribution available to the public, display and publicly perform the Contribution in original or modified form.

2.2 Moral Rights remain unaffected to the extent they are recognized and not waivable by applicable law. Notwithstanding, You may add your name in the header of the source code files of Your Contribution and We will respect this attribution when using Your Contribution.

25.2.3 3. PATENTS

3.1 Patent License

Subject to the terms and conditions of this Agreement You hereby grant to us a worldwide, royalty-free, non-exclusive, perpetual and irrevocable (except as stated in Section 3.2) patent license, with the right to transfer an unlimited number of non-exclusive licenses or to grant sublicenses to third parties, to make, have made, use, sell, offer for sale, import and otherwise transfer the Contribution and the Contribution in combination with the Material (and portions of such combination). This license applies to all patents owned or controlled by You, whether already acquired or hereafter acquired, that would be infringed by making, having made, using, selling, offering for sale, importing or otherwise transferring of Your Contribution(s) alone or by combination of Your Contribution(s) with the Material.

3.2 Revocation of Patent License

You reserve the right to revoke the patent license stated in section 3.1 if we make any infringement claim that is targeted at your Contribution and not asserted for a Defensive Purpose. An assertion of claims of the Patents shall be considered for a "Defensive Purpose" if the claims are asserted against an entity that has filed, maintained, threatened, or voluntarily participated in a patent infringement lawsuit against Us or any of Our licensees.

25.2.4 4. DISCLAIMER

THE CONTRIBUTION IS PROVIDED "AS IS". MORE PARTICULARLY, ALL EXPRESS OR IMPLIED WARRANTIES INCLUDING, WITHOUT LIMITATION, ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT ARE EXPRESSLY DISCLAIMED BY YOU TO US AND BY US TO YOU. TO THE EXTENT THAT ANY SUCH WARRANTIES CANNOT BE DISCLAIMED, SUCH WARRANTY IS LIMITED IN DURATION TO THE MINIMUM PERIOD PERMITTED BY LAW.

25.2.5 5. Consequential Damage Waiver

TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL YOU OR US BE LIABLE FOR ANY LOSS OF PROFITS, LOSS OF ANTICIPATED SAVINGS, LOSS OF DATA, INDIRECT, SPECIAL, INCIDENTAL, CONSEQUENTIAL AND EXEMPLARY DAMAGES ARISING OUT OF THIS AGREEMENT REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED.

25.2.6 6. Approximation of Disclaimer and Damage Waiver

IF THE DISCLAIMER AND DAMAGE WAIVER MENTIONED IN SECTION 4 AND SECTION 5 CANNOT BE GIVEN LEGAL EFFECT UNDER APPLICABLE LOCAL LAW, REVIEWING COURTS SHALL APPLY LOCAL LAW THAT MOST CLOSELY APPROXIMATES AN ABSOLUTE WAIVER OF ALL CIVIL LIABILITY IN CONNECTION WITH THE CONTRIBUTION.

25.2.7 7. Term

7.1 This Agreement shall come into effect upon Your acceptance of the terms and conditions.

7.2 In the event of a termination of this Agreement Sections 4, 5, 6, 7 and 8 shall survive such termination and shall remain in full force thereafter. For the avoidance of doubt, Contributions that are already licensed under a free and open source license at the date of the termination shall remain in full force after the termination of this Agreement.

25.2.8 8. Miscellaneous

8.1 This Agreement and all disputes, claims, actions, suits or other proceedings arising out of this agreement or relating in any way to it shall be governed by the laws of People's Republic of China excluding its private international law provisions.

8.2 This Agreement sets out the entire agreement between You and Us for Your Contributions to Us and overrides all other agreements or understandings.

8.3 If any provision of this Agreement is found void and unenforceable, such provision will be replaced to the extent possible with a provision that comes closest to the meaning of the original provision and that is enforceable. The terms and conditions set forth in this Agreement shall apply notwithstanding any failure of essential purpose of this Agreement or any limited remedy to the maximum extent possible under law.

8.4 You agree to notify Us of any facts or circumstances of which you become aware that would make this Agreement inaccurate in any respect.

You

Date:	
Name:	
Title:	
Address:	

Us

Date:	
Name:	
Title:	
Address:	

CHAPTER 26

Copyrights and Licenses

26.1 Software Copyrights

All original source code in this repository is Copyright (C) 2015-2016 Espressif Systems. This source code is licensed under the Apache License 2.0 as described in the file LICENSE.

Additional third party copyrighted code is included under the following licenses:

- Newlib (components/newlib) is licensed under the BSD License and is Copyright of various parties, as described in the file components/newlib/COPYING.NEWLIB.
- Xtensa header files (components/esp32/include/xtensa) are Copyright (C) 2013 Tensilica Inc and are licensed under the MIT License as reproduce in the individual header files.
- esptool.py (components/esptool_py/esptool) is Copyright (C) 2014-2016 Fredrik Ahlberg, Angus Gratton and is licensed under the GNU General Public License v2, as described in the file components/esptool_py/LICENSE.
- Original parts of FreeRTOS (components/freertos) are Copyright (C) 2015 Real Time Engineers Ltd and is licensed under the GNU General Public License V2 with the FreeRTOS Linking Exception, as described in the file components/freertos/license.txt.
- Original parts of LWIP (components/lwip) are Copyright (C) 2001, 2002 Swedish Institute of Computer Science and are licensed under the BSD License as described in the file components/lwip/COPYING.
- KConfig (tools/kconfig) is Copyright (C) 2002 Roman Zippel and others, and is licensed under the GNU General Public License V2.
- wpa_supplicant Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- FreeBSD net80211 Copyright (c) 2004-2008 Sam Leffler, Errno Consulting and licensed under the BSD license.

Where source code headers specify Copyright & License information, this information takes precedence over the summaries made here.

26.2 ROM Source Code Copyrights

ESP32 mask ROM hardware includes binaries compiled from portions of the following third party software:

- Newlib, as licensed under the BSD License and Copyright of various parties, as described in the file components/newlib/COPYING.NEWLIB.
- Xtensa libhal, Copyright (c) Tensilica Inc and licensed under the MIT license (see below).
- TinyBasic Plus, Copyright Mike Field & Scott Lawrence and licensed under the MIT license (see below).

- miniz, by Rich Geldreich - placed into the public domain.
- wpa_supplicant Copyright (c) 2003-2005 Jouni Malinen and licensed under the BSD license.
- TJpgDec Copyright (C) 2011, ChaN, all right reserved. See below for license.

26.3 Xtensa libhal MIT License

Copyright (c) 2003, 2006, 2010 Tensilica Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

26.4 TinyBasic Plus MIT License

Copyright (c) 2012-2013

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

26.5 TJpgDec License

TJpgDec - Tiny JPEG Decompressor R0.01 (C)ChaN, 2011 The TJpgDec is a generic JPEG decompressor module for tiny embedded systems. This is a free software that opened for education, research and commercial developments under license policy of following terms.

Copyright (C) 2011, ChaN, all right reserved.

- The TJpgDec module is a free software and there is NO WARRANTY.

- No restriction on use. You can use, modify and redistribute it for personal, non-profit or commercial products UNDER YOUR RESPONSIBILITY.
- Redistributions of source code must retain the above copyright notice.

CHAPTER 27

Indices

- genindex